Accelerated algorithms for constrained optimization and control

by

Anjali Parashar

B.Tech, Indian Institute of Technology, Indore (2020)

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2023

Author .....
Department of Mechanical Engineering
May 12, 2023

Certified by .....
Anuradha M. Annaswamy
Senior Research Scientist
Thesis Supervisor

Accepted by .....
Nicolas Hadjiconstantinou
Chairman, Department Committee on Graduate Theses

Accelerated algorithms for constrained optimization and control

by

Anjali Parashar

Submitted to the Department of Mechanical Engineering
on May 12, 2023, in partial fulfillment of the
requirements for the degree of
Master of Science in Mechanical Engineering

Abstract

Nonlinear optimization with equality and inequality constraints is a ubiquitous problem in several optimization and control problems in large-scale systems. Ensuring feasibility along with reasonable convergence to optimal solution remains an open and pressing problem in this area.

A class of high-order tuners was recently proposed in adaptive control literature with an effort to lead to accelerated convergence for the case when no constraints are present. In this thesis, we propose a new high-order tuner based algorithm that can accommodate the presence of equality and inequality constraints. We leverage the linear dependence in solution space to guarantee that equality constraints are always satisfied. We further ensure feasibility with respect to inequality constraints for the specific case of box constraints by introducing time-varying gains in the high-order tuner while retaining the attractive accelerated convergence properties. Theoretical guarantees pertaining to stability are also provided for time-varying regressors. These theoretical propositions are validated by applying them to several categories of optimization problems, in the form of academic examples, power flow optimization and neural network optimization.

We devote special attention to analyze a special case of neural network optimization, namely, linear neural network training problem, to understand the dynamics of nonconvex optimization governed by gradient flow and provide lyapunov stability guarantees for LNNs.

Thesis Supervisor: Anuradha M. Annaswamy
Title: Senior Research Scientist

Acknowledgments

Firstly, I would like to express my sincere gratitude to my thesis supervisor Dr. Anuradha Annaswamy, for her invaluable guidance and support throughout the research process. Her expertise and profundity in research helped me achieve my goal of developing a strong conceptual understanding of theoretical engineering tools. The last two years have been a great learning experience, thanks to her mentorship, encouragement and kind reminders about my strengths and skills.

I would like to extend my heartfelt thanks to the members of the Active Adaptive Control lab, for their feedback and support in advancing my research and helping me to navigate the challenges of graduate study. I am lucky to have had the opportunity of working with Dr. Priyank Srivastava for his guidance and invigorating discussions which helped me improve the quality of my work. I am also grateful to have worked with Johannes, Jules, Peter and Rabab.

I acknowledge the support of Boeing Strategic University initiative and the Siemens Fellowship in enabling me to pursue my academic goals and make a meaningful contribution to the field of control systems and learning. I am truly grateful to Dr. Joseph Gaudio and Dr. Heather Hussain for their invaluable expertise and insights in shaping the direction of my research. I would also like to thank Dr. Amit Chakraborty and Dr. Biswadip Dey, especially for helping me identify the right problems to solve.

Lastly, I wish to thank my friends and family. My special thanks to my friends Sunbochen, Kanishkar, Jai, and Simran who looked after me and cheered for me with unconditional support on a daily basis over the last two years. To my parents, brother and all of my family, who always believed in me, thank you for everything.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement

There is an immense need to study two broad areas in optimization: constrained optimization and non-convex optimization. It is quite challenging to propose theoretical guarantees of convergence and stability for constrained optimization problems in a very general context. Specifically, these optimization problems can be formulated as [11]

$$\begin{aligned} &\min\ f(x) \\ &\text{s.t. } h(x) = 0 \\ &\quad g(x) \leq 0, \end{aligned} \tag{1.1}$$

where $x \in \mathbb{R}^n$ is the decision variable, $f : \mathbb{R}^n \to \mathbb{R}$, $h : \mathbb{R}^n \to \mathbb{R}^m$, and $g : \mathbb{R}^n \to \mathbb{R}^p$ are continuously differentiable functions. Without loss of generality, we assume that problem (1.1) is not overdetermined, i.e., $m \leq n$. Unless specified, $f(x), h(x), g(x)$ are assumed to be convex.

1.2 Challenges in constrained optimization problems

By considering the case of constrained convex optimization, we face two main challenges: feasibility and speed of learning. In this section, we discuss them in detail

which provides us with the problems we have attempted to solve in this thesis.

1.2.1 Challenges in feasibility guarantees

Efforts have been made towards feasibility and convergence for specific instances of constrained optimization such as equality constrained optimization which are mathematically tractable [4]. There are known and established solutions for linear programs and quadratic optimization problems with linear constraints [34], [8]. As the complexity of the objective function and the nature of constraints becomes complex, feasibility guarantees are hard to provide. Feasibility is not just important from the perspective of constraints, but it also affects convergence. Several well known algorithms such as Alternating Direction Method of Multipliers (ADMM) [10] and Douglas-Rachford Splitting (DRS) [41], developed for composite convex optimization problems tend to diverge in the case of infeasible convex optimization.

However, even in these cases, infeasibility detection is limited to the case of linear programs (LPs), quadratic programs (QPs), second-order cone programs (SOCPs) and semidefinite programs (SDPs) constrained by closed, convex sets [6]. Overall, the set of constrained optimization problems that can be reduced to an unconstrained convex program with necessary and sufficient conditions is very limited. We attempt to solve this problem by expanding the scope of problems which can be converted to unconstrained convex optimization programs by providing a list of sufficient conditions for the same. We also make sure that the method for ensuring feasibility is modular, i.e., only depends on a set of basic assumptions regarding the objective function and is relatively agnostic to the optimization method and its order itself (i.e, valid for both first and second order methods such as GD, Netwton's, etc).

1.2.2 Speed of convergence

Most of the proposed methods in the literature are gradient based methods , that rely on first-order information of the system to make updates [7], which work well for high-dimensional computations but have drawbacks such as sensitivity to appropriate

learning rates and being very slow in learning [18]. One might argue that a way to resolve this problem is to use second-order methods such as Newton's method [20], however, these are computationally expensive, since they require calculating and storing the hessian of the loss function at every iteration. Moreover, in the case of nonconvex optimization, it is a known problem that Gradient Descent and Quasi-newton methods often get stuck at saddle points in high-dimensional spaces, since they perform local minimization [14],[45].

Recently, there have been promising numerical results with the implementation of fast learning based methods such as Nesterov's Accelerated method [35], Gradient descent with Momentum, that impart acceleration to the parameter convergence for unconstrained convex optimization. Recently, a class of algorithms called High-Order Tuner (HT) [32, 22] with guaranteed convergence and stability for unconstrained convex optimization was proposed for accelerated learning based applications in controls and optimization. In this space, constraint satisfaction remains an unaddressed problem due to the mathematical complexity of the discretized implementation of these methods. Specifically, some results pertaining to Accelerated primal and dual methods are available but restricted to linear constraints [30] or min-max problem of saddle points [24].

1.3 Contributions & Outline

We study the class of problems in (1.1) with the objective of satisfying constraints and ensuring fast convergence of parameters. As one of the first contributions of this work, we propose a framework for sufficient guarantees of convexity in the presence of equality constraints. We utilize the attractive properties of stability of HT in the presence of time-varying regressors and accelerated convergence for constant regressors to solve the stated optimization problems. We extend the idea of constraints beyond equality constraints and provide analytical tools for accommodating inequality constraints by using projection-based updates in HT.

In chapter 2, we introduce some preliminary notations, definitions and concepts

for reference.

In Chapter 3, we explore a framework for assessing the convergence and stability for a broader class of equality-constrained convex optimization and propose HT for constrained convex optimization (Algorithm-1) [40].

We provide dedicated treatment for a specific case of inequality constraints called as box-constraints in Chapter 4, which commonly occurs in many physical systems as state and input limits [39]. We propose a modified form of HT (Algorithm-2) with time-varying gains to ensure feasibility at every instance of time. We also propose a projection-based method for inequality constraint satisfaction in this chapter. We further validate the application of HT and its apparent advantages over GD in academic examples and challenging optimization problems which have been studied in the literature.

Chapter 5 is dedicated to the validation of HT in the context of optimization for real-life applications. We explore the application of HT in providing a Neural Network (NN) based method for solving DC-Optimal Power Flow problem.

Chapter 6 refers to the study of neural network training, which is currently devoid of any theoretical guarantees due to the non-convex nature of loss landscape. We share some interesting insights into the stability of neural networks using Gradient flow for a class of NNs called Linear Neural Networks (LNNs). This is meant to serve as a primer in establishing some fundamental properties of LNNs under gradient flow which can then be extended to the case of accelerated methods like Nesterov and HT with some efforts in the future.

Finally, Chapter 7 provides some interesting insights into the future directions of these ideas and presents conclusions of the work covered in the thesis.

Chapter 2

Preliminaries

This chapter provides some preliminaries relevant to the work. We introduce standard notations used in the thesis and outline some constraint satisfaction techniques utilized in optimization problems for solving equality constraints which are used in Chapter 3. We also provide definitions pertaining to Convex optimization. Lastly, we introduce High Order Tuner (HT) which has been used to provide accelerated learning features in the solutions presented.

2.1 Notations

Let $\mathbb{R}$ denote the set of real numbers. $\|.\|$ denotes the 2-norm of a vector or matrix. For a continuously differentiable function $f : \mathbb{R}^n \to \mathbb{R}$, ∇f denotes its gradient. $(.)^T$ denotes the transpose of a vector or matrix. For vectors $x, y \in \mathbb{R}^n$, $x \geq y$ implies that the inequality holds elementwise. For a vector $x \in \mathbb{R}^n$, with $1 \leq i < j \leq n$, $x_{i:j}$ denotes the subvector with elements from the i-th entry of x to the j-th entry.

Remark: By "p is convex on Ω_m," we mean that $p : \mathbb{R}^m \to \mathbb{R}^{n-m}$ is a vector function and each scalar function $p_i : \mathbb{R}^m \to \mathbb{R}$ is convex $\forall$ Ω_m

2.2 Convex Optimization

Definition of Convex Function

A function $f : \mathbb{R}^n \to \mathbb{R}$ is defined as convex function if [11]:

- **dom** f as a convex set C &
- for $0 \leq \theta \leq 1$ and $x, y \in C$ we have:

$$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta)f(y) \tag{2.1}$$

Any constrained/unconstrained optimization problem where the associated loss function satisfies the above mentioned properties and a convex set C can be identified as the domain of the function is treated as a convex optimization problem. A convex function is always continuous within its domain [11] and there are several useful properties related to convex analysis that help in parameter convergence.

The advantage of convex optimization lies in the attractive properties of relation between optimal solution and gradient, which is called as **First-Order Conditions**:

Assuming differentiability of function f, then f is convex *iff*:

- **dom** f as a convex set C &
- for $0 \leq \theta \leq 1$ and $x, y \in C$ we have:

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) \tag{2.2}$$

Additionally, if a function f is twice differentiable, then the hessian being PSD matrix, i.e, $\nabla^2 f(x) \succeq 0$, is a necessary and sufficient condition for convexity of f. This is called as **Second-Order Condition**.

Besides these basic properties, there are several inequalities specific to convex functions that depend on gradient and smoothness parameters of the convex function itself. These have been summarized in [33] and are extremely useful in the proof of Lyapunov stability theorems presented in Chapter 3,4.

The basic approach in many constrained optimization problems involves reducing it to an unconstrained optimization with guarantees of convexity, which help in establishing convergence to optimal parameter with several algorithms. In the subsequent section, we provide an overview of such constraint satisfaction approaches.

2.3 Constraint Satisfaction Techniques

In this section, we outline two important techniques utilized in reducing constrained optimization to unconstrained optimization programs, which are widely utilized in the optimization literature [16], [37], [38].

2.3.1 Nonlinear optimization via Lagrange Loss formulation

We consider optimization problems of the form

$$\begin{aligned} &\min\ f(x) \\ &\text{s.t. } h(x) = 0 \\ &\quad\ g(x) \leq 0, \end{aligned} \tag{2.3}$$

Here, $f(x), g(x), h(x)$ are differentiable and convex functions. For the equation (2.3), we define a loss function $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}$, consisting of the original objective function and soft loss terms penalizing the constraint violation [9, 1]

$$\mathcal{L}(x) = f(x) + \lambda_h \|h(x)\|^2 + \lambda_g \|\operatorname{softplus}(g(x))\|^2, \tag{2.4}$$

where $\lambda_h, \lambda_g > 0$ are design parameters. Given $y \in \mathbb{R}^p$, the function softplus : $\mathbb{R}^p \to \mathbb{R}^p$ is defined as

$$\operatorname{softplus}(y) = \log(1 + e^y),$$

and serves as a smooth approximation to ReLU (Rectified Linear Unit). Note that we use softplus to ensure differentiability of $\mathcal{L}(x)$ which is utilized in First-Order Condition for convexity.

2.3.2 Partitioning of solution-space

Equality constraints introduce linear dependencies in the feasible solution space, as explored in several works previously. Assuming that problem (2.3) is not overdetermined, x can be partitioned into an independent variable $\theta \in \mathbb{R}^m$ and a dependent variable $z \in \mathbb{R}^{n-m}$,

$$x = [\theta^T \quad z^T]^T.$$

We assume that h is such that given m entries of x, its remaining $(n-m)$ entries can be computed either in closed form or recursively. In other words, we assume that we have knowledge of the function $p : \mathbb{R}^m \to \mathbb{R}^{n-m}$ such that

$$h([\theta^T \quad p(\theta)^T]^T) = 0, \tag{2.5}$$

holds for all $\theta \in \mathbb{R}^m$. For all the points where $\frac{\partial h}{\partial z} \neq 0$, existence and uniqueness of p is guaranteed from the Implicit Function theorem. The reduction of variable dimension as explained above ensures that equality constraints are always satisfied.

Using the function $p(\cdot)$ defined as above, we now define a modified loss function $l : \mathbb{R}^m \to \mathbb{R}$ as

$$l(\theta) = \mathcal{L}([\theta^T \quad p(\theta)^T]^T).$$

The optimization problem in (2.3) is now reformulated as an unconstrained minimization problem given by

$$\min l(\theta), \tag{2.6}$$

with $\theta \in \mathbb{R}^m$ as the decision variable. Since we are also reducing the dimension of the overall optimization problem while simultaneously satisfying the equality constraints, we call this approach as *variable reduction technique*. Depending on the information about the mapping p, gradient of the modified loss function could be computed either explicitly or using the Implicit Function theorem as in [2]. In the proceeding chapter we utilise the partitioning technique for solving different kinds of constrained optimization problems.

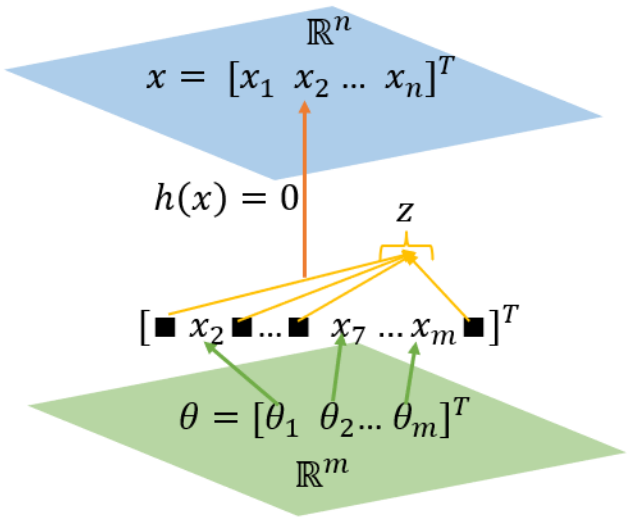


Figure 2-1: Graphical representation of one of the ways in which the partitioning technique can be utilized to construct a lower-dimension subspace in $\mathbb{R}^m$ which can be converted to an all-time feasible solution space in $\mathbb{R}^n$ using (2.5)

2.4 High Order Tuner

The emergence of gradient-based optimization update laws such as Gradient Descent (GD) has sought applications in the realm of optimization and control systems. Recently, a method emerging from the family of Bregman-Lagrangians called High-Order Tuner (HT) was proposed as an update law with accelerated convergence and stability in the presence of time-varying regressors for the purpose of unknown parameter estimation in a linear regression setting [22]. The stability property was further extended to a class of unconstrained convex optimization [32].

The second order-differential equation of a HT can be written as [21]:

$$\begin{aligned} \dot{\nu}(t) &= -\frac{\gamma}{\mathcal{N}_t}\nabla L(\theta(t)) \\ \dot{\theta}(t) &= -\beta(\theta(t) - \nu(t)) \end{aligned} \tag{2.7}$$

An equivalent discretization of the continuous HT can be expressed as [22]:

$$\begin{aligned} \bar{\theta}_k &= \theta_k - \gamma\beta\nabla\frac{\nabla l(\theta_k)}{\mathcal{N}_k} \\ \theta_{k+1} &= \bar{\theta}_k - \beta(\bar{\theta}_k - \nu_k) \\ \nu_{k+1} &= \nu_k - \gamma\nabla\frac{\nabla l(\theta_{k+1})}{\mathcal{N}_k} \end{aligned} \tag{2.8}$$

HT shares accelerated learning properties with Nesterov's method [33] for the case of linear regression. The discretized version of HT and Nesterov's method are different in the way the update steps are defined. In both methods, there is an 'averaging' action and a 'gradient' action. In HT, the 'gradient' action is performed first, generating intermediate variables $\nu_k, \bar{\theta}_k$ and then the 'averaging' is performed to obtain θ_{k+1}.

In Nesterov's method, the averaging happens first, to obtain ν_k, and then the 'gradient' action is taken to obtain θ_{k+1}. One important difference is that the general Nesterov's method is not equipped with stability guarantees in the presence of time-varying regressors. However, HT also incorporates a normalizer, $\mathcal{N}_k$, which is chosen carefully based on the optimization problem, such that a Lyapunov function can be provided for a class of convex optimization problems. Due to these advantages, we have chosen HT as the optimization algorithm for accelerated convergence in this study. The difference between the update steps in Gradient Descent and Accelerated learning methods is shown in Figure 2-2, 2-3.

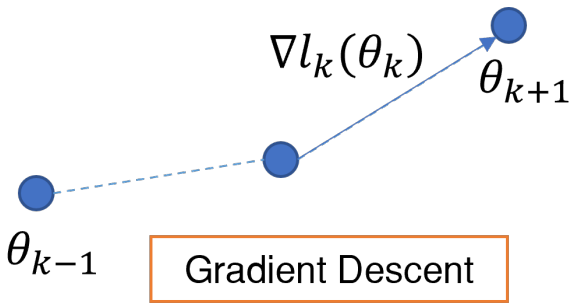


Figure 2-2: Gradient Descent Update

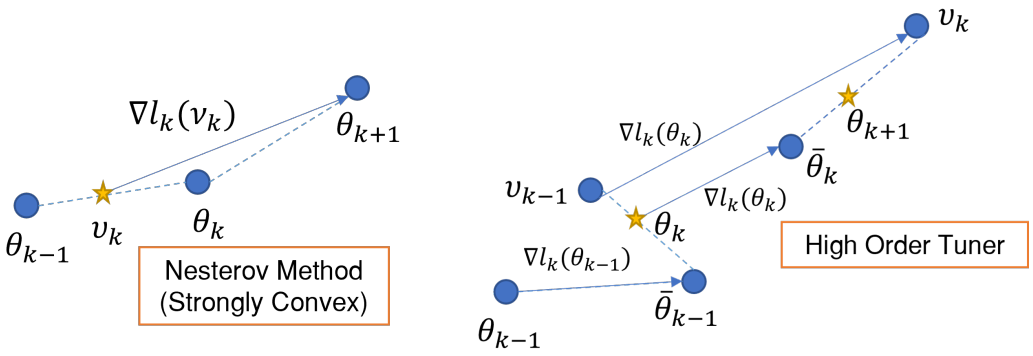


Figure 2-3: Comparison of Nesterov and HT update. Notice that in Nesterov's method, the average ν_k is calculated first and then gradient step is called to θ_{k+1}. In HT, gradient step is taken for ν_k, $\hat{\theta}_k$ and then their average is calculated as θ_{k+1}

Chapter 3

Constrained Optimization using HT: Equality Constraints

3.1 Problem Statement

In this chapter we look at two specific classes of the general optimization problem discussed in the last chapter (2.3). We begin by analyzing constrained convex optimization problems, for which the equality constraints must be linear. We then propose a HT algorithm (Algorithm 1) for solving such problems with guaranteed stability and convergence.

Formally, consider the problem:

$$\begin{aligned} &\min\ f(x) \\ &\text{s.t. } Ax = b \\ &\quad\ g(x) \leq 0, \end{aligned} \tag{3.1}$$

where $x \in \mathbb{R}^n$ is the decision variable, $f : \mathbb{R}^n \to \mathbb{R}$ and $g : \mathbb{R}^n \to \mathbb{R}^p$ are continuously differentiable (strongly) convex functions, $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$. We first provide feasibility guarantees for equality constraints and then generalize our approach to problems involving both equality and inequality constraints.

3.1.1 Equality-constrained convex optimization

Equality-constrained convex optimization problems have the general structure

$$\begin{aligned} &\min\ f(x) \\ &\text{s.t. } Ax = b. \end{aligned} \tag{3.2}$$

With $\lambda_h > 0$, the loss function $\mathcal{L}$ (2.4) and the modified loss function l (2.6) take the form:

$$\mathcal{L}(x) = f(x) + \lambda_h \|Ax - b\|^2, \tag{3.3a}$$

$$l(\theta) = f([\theta^T\ p(\theta)^T]^T). \tag{3.3b}$$

Here we have used the fact that $\lambda_h \|A[\theta^T\ p(\theta)^T]^T - b\|^2 = 0$. We can conclude that p is an affine function of θ in this case. Let

$$p(\theta) = P\theta + q, \tag{3.4}$$

where $P \in \mathbb{R}^{(n-m)\times m}$ and $q \in \mathbb{R}^{n-m}$. $\mathcal{L}$ is convex by construction. We characterize the convexity properties of l in the following result.

Proposition 3.1.1 *For the equality-constrained convex optimization problem* (3.2)*, assume f is a $\overline{L}$-smooth convex function, and let*

$$\overline{M} = \sqrt{1 + \|P\|^2}(1 + \|P\|)\overline{L}. \tag{3.5}$$

Then l is $\overline{M}$-smooth convex.

The next result extends Proposition 3.1.1 to the case of strongly convex functions.

Corollary 3.1.1 *For the equality-constrained convex optimization problem* (3.2)*, assume f is a $\overline{L}$-smooth and μ-strongly convex function. Then l is $\overline{M}$-smooth and μ-strongly convex, where $\overline{M}$ is defined in equation* (3.5)*.*

Now that we have established the convexity and smoothness properties of the modified loss function l which is unconstrained, we leverage the properties of HT [22, 32] to propose an accelerated algorithm to solve (3.2).

3.2 High Order Tuner for Equality Constrained Convex Optimization

Let $\mathcal{N}_k$ be the normalizing signal defined as

$$\mathcal{N}_k = 1 + H_k,$$

where

$$H_k = \max\{\zeta : \zeta \in \sigma(\nabla^2 l(\theta_k))\},$$

where $\sigma(\nabla^2 l(\theta_k))$ denotes the spectrum of the Hessian matrix of the loss function l evaluated at $\theta = \theta_k$. Note that it is also possible to make a more conservative selection for $\mathcal{N}_k$ such as $\overline{M}$, i.e., smoothness parameter of the loss function if accurate information about $\nabla^2 l$ is not available. Next we introduce Algorithm 1 to solve problem (3.2). The following result formally characterizes the convergence properties of Algorithm 1.

Theorem 3.2.1 *If the objective function f is $\overline{L}$-smooth convex, then with $0 < \beta < 1$ and $0 < \gamma < \frac{\beta(2-\beta)}{8+\beta}$, the sequence of iterates $\{\theta_k\}$ generated by Algorithm 1 satisfy $\lim_{k\to\infty} l(\theta_k) = l(\theta^*)$, where $l(\theta^*) = f([\theta^{*T} \quad p(\theta^*)^T]^T)$ is the optimal value of (3.2).*

Following Corollary 3.1.1 and [32, Theorem 3], a similar result exists for the strongly convex case as well.

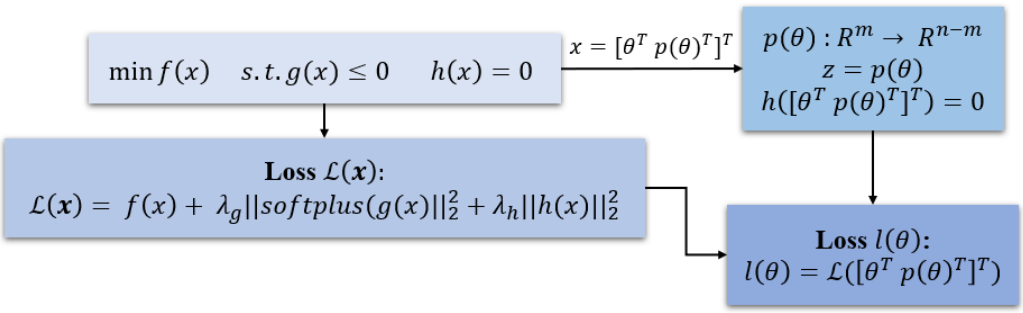


Figure 3-1: Overview of the feasibility satisfaction framework presented in Section 3.2.1

3.2.1 Convex optimization problems with equality & inequality constraints

Here we extend our approach to solve general convex optimization problems involving equality as well as the inequality constraints in the form (3.1). Redefine the loss function and the modified loss functions by including a penalty term corresponding to the inequality constraints violation as

$$\mathcal{L}(x) = f(x) + \lambda_h \|Ax - b\|^2 + \lambda_g \|\operatorname{softplus}(g(x))\|^2, \tag{3.6a}$$

$$l(\theta) = f([\theta^T\ p(\theta)^T]^T) + \lambda_g \|\operatorname{softplus}(g([\theta^T\ p(\theta)^T])^T)\|^2. \tag{3.6b}$$

Since the penalty term corresponding to the inequality constraints is convex, it follows from Proposition 3.1.1 that the modified loss function (3.6b) is convex.

Our adopted approach is based on the inequality correction procedure proposed in [16]. The method involves first implementing the HT Algorithm 1 on the loss function (3.6b) ensuring that the equality constraints are met at all times. Then we apply an additional update that drives the decision variable towards the feasible region corresponding to the inequality constraints as well. Let $\alpha > 0$ be the stepsize and define $\rho : \mathbb{R}^n \rightarrow \mathbb{R}^n$ as

$$\rho\left(\begin{bmatrix}\theta \\ p(\theta)\end{bmatrix}\right) = \begin{bmatrix}\theta - \alpha\Delta\theta \\ p(\theta) - \alpha\Delta p(\theta)\end{bmatrix}, \tag{3.7}$$

where

$$\Delta\theta = \left(\frac{d}{d\theta} \| \text{softplus} \left(g\left(\begin{bmatrix} \theta \\ p(\theta) \end{bmatrix} \right) \right) \|^2 \right)^T.$$

Note immediately that the inequality correction step above does not affect the feasibility with respect to the equality constraints (Figure 3-2). Hence by implementing the described method, we obtain Algorithm 1 (with steps 7 and 8 included as shown) that satisfies equality constraints at each step and moves closer towards satisfying the inequality constraints with each successive iteration. Constraint feasibility approach is summarized in Figure 3-1

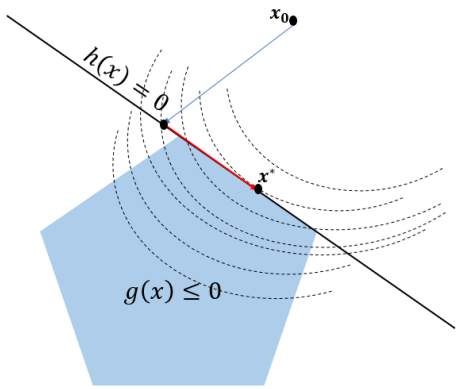


Figure 3-2: Illustration of inequality constraint satisfaction without violation of eqaulity constraints

Algorithm 1 HT optimizer for equality + inequality constrained convex optimization

1: **Initial conditions** x_0, $\overline{x}_0$, ν_0, gains α, γ, β
2: **for** k=0,1,2,... **do**
3: Compute $\nabla l(\theta_k)$ and let $\mathcal{N}_k = 1 + H_k$
4: $\nabla \overline{q}_k(\theta_k) = \dfrac{\nabla l(\theta_k)}{\mathcal{N}_k}$
5: $\overline{\theta}_k = \theta_k - \gamma\beta\nabla\overline{q}_k(\theta_k)$
6: $\theta_{k+1} \leftarrow \overline{\theta}_k - \beta(\overline{\theta}_k - \nu_k)$
7: $\boldsymbol{\overline{x}_{k+1} = [\theta_{k+1}^T \;\; p(\theta_{k+1})^T]^T}$
8: **Compute** $\boldsymbol{x_{k+1} \leftarrow \rho(\overline{x}_{k+1})}$
9: Compute $\nabla l(\theta_{k+1})$ and let
10: $\nabla \overline{q}_k(\theta_{k+1}) = \dfrac{\nabla l(\theta_{k+1})}{\mathcal{N}_k}$
11: $\nu_{k+1} \leftarrow \nu_k - \gamma\nabla\overline{q}_k(\theta_{k+1})$
12: **end for**

It is reasonable to expect that if the hypotheses of Theorem 3.2.1 are satisfied and α is properly selected, Algorithm 1 solves problem (3.1).

3.3 Convex Optimization for a Class of Nonconvex Problems

In this section, we extend our approach of achieving accelerated convergence via high-order tuners to a class of nonconvex optimization problems that arise out of certain kinds of constrained convex optimization problems. The presented approach utilizes properties of convex functions discussed in Chapter 2. We focus only on problems involving just the equality constraints which can be combined with generalized methods to problems involving inequality constraints as well. We consider a family of problems where the loss function is convex and equality constraints are convex. The arguments could be generalized easily to the strongly convex case.

3.3.1 Equality-constrained nonconvex problems

Consider the optimization problem

$$\begin{aligned} &\min\ f(x) \\ &\text{s.t. } h(x) = 0, \end{aligned} \tag{3.8}$$

The associated soft loss function can be defined as:

$$\mathcal{L}(x) = f(x) + \lambda_h \|h(x)\|^2. \tag{3.9}$$

The following lemma provides the conditions under which $\mathcal{L}$ is convex.

Lemma 3.3.1 *If f is convex, and h is an elementwise non-negative or non-positive convex function, then $\mathcal{L}$ defined in* (3.9) *is convex.*

The modified loss function in this case once again takes the form (3.3b), however the functional form of p will not necessarily be affine. Therefore, establishing the convexity of l over the entire domain as in Proposition 3.1.1 may not be feasible anymore. Additionally, the loss function $l(\theta)$ will no longer be necessarily convex.

We first search for conditions under which the modified loss function l is convex. We use these conditions to identify compact sets within the domain over which the loss is always convex to extend the existing results of stability. We summarize a set of such conditions in the following result.

Proposition 3.3.1 *Assume that there exists a convex set $\Omega_n \in \mathbb{R}^n$ such that the hypotheses of Lemma 3.3.1 are satisfied on Ω_n. Let*

$$\Omega_m = \{\theta \mid \theta = x_{1:m}, x \in \Omega_n\}. \tag{3.10}$$

If either of the following conditions is satisfied:

1. $\nabla \mathcal{L}(x) \geq 0$ *for all* $x \in \Omega_n$, *and* p *is convex on* Ω_m,

2. $\nabla \mathcal{L}(x) \leq 0$ *for all* $x \in \Omega_n$, *and* p *is concave on* Ω_m,

then l *is convex on* Ω_m.

Remark: 1. Note that these conditions are sufficient but not necessary and sufficient for identifying all possible regions of convexity for $l(\theta)$. Therefore, they serve as an analytical tool to validate convexity of $l(\theta)$ over a chosen compact set.

2. Not only is p non-affine, in several cases, p cannot be explicitly formulated. However, the above stated conditions rely on knowledge of $\nabla p(\theta), \nabla^2 p(\theta)$, which can be easily obtained using Implicit Function Theorem as explained below.

Gradient of dependent variables using Implicit Function Theorem

Recall that $h([\theta^T \quad p(\theta)^T]^T) = 0$. Therefore, by taking the derivative of the above equation:

$$\begin{aligned} \frac{\partial}{\partial \theta} h\left(\begin{bmatrix} \theta \\ p(\theta) \end{bmatrix} \right) &= \frac{\partial h}{\partial \theta} + \frac{\partial h}{\partial p(\theta)} \frac{\partial p(\theta)}{\partial \theta} = 0 \\ \frac{\partial p(\theta)}{\partial \theta} &= \left(\frac{\partial h}{\partial \theta} \right)^{-1} \frac{\partial h}{\partial \theta} \end{aligned} \tag{3.11}$$

Note that $\frac{\partial h}{\partial \theta}$ and $\frac{\partial h}{\partial p(\theta)}$ are simply $\nabla h_{[0:m]}$ and $\nabla h_{[m:n]}$ respectively. This approach has been adopted from [16]. Further, readers are encouraged to refer to [3] for construction of $\frac{\partial p(\theta)}{\partial \theta}$ in an efficient way to avoid space complexity problems in high dimensional problems. Similar ideas have been explored in [12, 29]

Additionally, using Implicit Function theorem, we can establish the following sufficient conditions on $h(\cdot)$ for which $p(\cdot)$ is convex or concave. This set of conditions is also not exhaustive but helpful in certain structures of problems, where we can comment on convexity of $p(\theta)$ directly using $h(x)$.

Proposition 3.3.2 *Following from Proposition 3.3.1, assuming f and h are convex on a given set $\bar{\Omega}_n \subseteq \mathbb{R}^n$, and h_i is twice differentiable $\forall\ i = 1, ..., n-m$, it follows that:*

1. *if $\nabla_p h(x) < 0$ for $x \in \bar{\Omega}_n$ then $p(\theta)$ is convex on Ω_m*

2. *if $\nabla_p h(x) > 0$ for $x \in \bar{\Omega}_n$ then $p(\theta)$ is concave on Ω_m*

Additionally, if h is linear in z and convex in θ, then condition (i) follows.

3.3.2 Numerical Example

The following example illustrates how the conditions of Proposition 3.3.1 and Proposition 3.3.2 can be verified. We consider the problem in (3.8) where $x \in \mathbb{R}^2$, $h(x) = {x_1}^2 + (x_2 - 4)^2 - 1 = 0$ and $f(x) = \log(\sum_{i=1}^{2} e^{x_i})$ are convex functions ($h : \mathbb{R}^2 \to \mathbb{R}$, $m = 1$). With x_1 as the independent variable, we write $x = [x_1 \ \ p(x_1)]^T$, and use the implicit function theorem to determine $p(x_1)$ explicitly. We therefore set $\lambda_h = 0$ while formulating the loss function, and hence $\mathcal{L}$ is the same as f. Now we demonstrate the application of Propostion 3.3.2 to define region Ω_n where $p(\cdot)$ is convex or concave.

Clearly, $\nabla_p h(x) = 2(p(x_1) - 4)$, and it is evident that $\nabla_p h(x) < 0$ for $x_2 < 4$ and $\nabla_p h(x) > 0$ for $x_2 > 4$ *(as $x_2 = p(x_1)$)*. Using Proposition 3.3.2, we conclude that for $x_2 \leq 4$, $p(\cdot)$ is convex and for $x_2 \geq 4$, $p(\cdot)$ is concave. Additionally, to ensure that to $p(x_1)$ evaluates to a real number, we must constrain $-1 \leq x_1 \leq 1$. Thus, following

Proposition 3.3.2, we construct sets $\bar{\Omega}_n^1$ and $\bar{\Omega}_n^2$ as:

$$\bar{\Omega}_n^1 = \{x = [x_1 \;\; x_2]^T | -1 \leq x_1 \leq 1, x_2 < 4\} \tag{3.12}$$

$$\bar{\Omega}_n^2 = \{x = [x_1 \;\; x_2]^T | -1 \leq x_1 \leq 1, x_2 > 4\} \tag{3.13}$$

Since $\mathcal{L}(x_1, x_2) = \log(e^{x_1} + e^{x_2})$ it is evident that:

$$\nabla_{x_1} \mathcal{L} = \frac{e^{x_1}}{e^{x_1} + e^{x_2}} \tag{3.14a}$$

$$\nabla_{x_2} \mathcal{L} = \frac{e^{x_2}}{e^{x_1} + e^{x_2}} \tag{3.14b}$$

Clearly, $\nabla \mathcal{L}(x_1, x_2) > 0$ for all $x \in \mathbb{R}^2$. From Proposition 3.3.1, case (ii), we can see that for $-1 \leq x_1 \leq 1$ and $x_2 \leq 4$, $\nabla \mathcal{L}(x_1, x_2) > 0$ and $p(\cdot)$ is convex. Therefore, $l(x_1)$ is convex in this region. Formally, we define $\Omega_n \equiv \bar{\Omega}_n^1$ as:

$$\Omega_n = \{x = [x_1 \;\; x_2]^T | -1 \leq x_1 \leq 1, x_2 \leq 4\} \tag{3.15}$$

Within the chosen Ω_n, Proposition-3.3.1 guarantees that $l(x_1)$ is convex. Consequently, Ω_m is automatically defined as:

$$\Omega_m = \{x_1 \in \mathbb{R} \,| -1 \leq x_1 \leq 1\} \tag{3.16}$$

Indeed, for values of $x_1 \in \Omega_m$ there are two cases for $p(x_1)$ using Implicit Function Theorem [25] given as:

$$p(x_1) = 4 - \sqrt{1 - {x_1}^2} \quad \text{for } x_2 \leq 4 \tag{3.17}$$

$$p(x_1) = 4 + \sqrt{1 - {x_1}^2} \quad \text{for } x_2 \geq 4 \tag{3.18}$$

As we seek to expand the set Ω_n where $l(\cdot)$ is convex, we note that due to the simple nature of the problem, it is easy to conclude that $l(\cdot)$ is concave for $x_2 \geq 4$, hence Ω_n for this problem is equivalent to the one in (3.15).

Hence, using Ω_n as defined in (3.15), $p(\cdot)$ takes the form mentioned in (3.17),

and by defining $l(\cdot)$ as the one in (3.3b), the optimization problem reduces to the following convex optimization problem which is much simpler to solve:

$$\begin{aligned} &\min \ \log(e^{x_1} + e^{4-\sqrt{1-x_1^2}}) \\ &\text{s.t. } \ x_1 \in \Omega_m \end{aligned} \tag{3.19}$$

The approach for feasibility of inequality constraints present in the form of $x_1 \in \Omega_m$ has been addressed in Chapter 4 in detail. We now state a HT based Algorithm which guarantees convergence to optimal solution provided the condition $\theta \in \Omega_m$ is satisfied.

3.3.3 High Order Tuner

Now that we have established sufficient conditions for the convexity of the modified loss function, we can use high-order tuners to find an optimizer of (3.8). In fact, if the sequence of iterates lie within the set Ω_n, then we can use Algorithm 1, stated earlier for convex programs, to find a solution of (3.8). The following result formalizes this.

Theorem 3.3.2 *If the objective function f and the equality constraint h are convex over a set Ω_n, In addition, with $0 < \beta < 1$, $0 < \gamma < \frac{\beta(2-\beta)}{8+\beta}$, and $\theta_0 \in \Omega_m$, where Ω_m is defined in* (3.10)*, if the sequence of iterates $\{\theta_k\}$ generated by Algorithm 1 satisfy $\{\theta_k\} \in \Omega_m$, then $\lim_{k\to\infty} l(\theta_k) = l(\theta^*)$, where $l(\theta^*) = f([\theta^{*T} \quad p(\theta^*)^T]^T)$ is the optimal value of* (3.8)*.*

Theorem 3.3.2 enables us to leverage Algorithm 1, provided that the state remains inside the set over which the modified loss function is convex. It is reasonable to argue that this is always not the case. To overcome this assumption, we use the projection operator defined as

$$\text{proj}_{\Omega_m}(\tilde{\theta}) = \text{argmin}\|\tilde{\theta} - \theta\|, \quad \forall \theta \in \Omega_m$$

to make sure that the state remains inside the set Ω_m. Algorithm 2 states this concisely.

Algorithm 2 HT Optimizer for equality-constrained nonconvex optimization

Initial conditions θ_0, ν_0, gains γ, β
$\theta_0 \leftarrow \text{proj}_{\Omega_\text{m}}(\theta_0)$
for $k = 1$ to N **do**
  Compute $\nabla l(\theta)$ and let $\mathcal{N}_k = 1 + H_k$
  $\nabla \overline{q}_k(\theta_k) = \dfrac{\nabla l(\theta_k)}{\mathcal{N}_k}$
  $\overline{\theta}_k = \theta_k - \gamma\beta\nabla \overline{q}_k(\theta_k)$
  $\theta_{k+1} \leftarrow \text{proj}_{\Omega_\text{m}}(\overline{\theta}_k - \beta(\overline{\theta}_k - \nu_k))$
  Compute $\nabla l(\theta_{k+1})$ and let
  $\nabla \overline{q}_k(\theta_{k+1}) = \dfrac{\nabla l(\theta_{k+1})}{\mathcal{N}_k}$
  $\nu_{k+1} \leftarrow \nu_k - \gamma\nabla \overline{q}_k(\theta_{k+1})$
end for

The arguments of this section show how the proposed approach could be applied to solve nonconvex problems, where a convex objective function needs to be optimized with respect to nonlinear convex equality constraints.

Note that the proposed method depends on a clearly defined projection operator $proj_{\Omega_m}(\tilde{\theta})$ which has been discussed in detail in the subsequent chapter.

Chapter 4

Constrained Optimization using HT: Inequality Constraints

Convex optimization with inequality constraints is a ubiquitous problem in several optimization and control problems in large-scale systems. In this chapter, we propose a new high-order tuner that can accommodate the presence of box inequality constraints. In order to accommodate the underlying box constraints, time-varying gains are introduced in the high-order tuner which leverage convexity and ensure anytime feasibility of the constraints. Numerical examples are provided to support the theoretical derivations. On a concluding note, some other ideas pertaining to constraint feasiblity by using a projection operator are presented with numerical simulations.

4.1 Problem Statement

Recalling the discussion in Chapter 3, Section 3.3.1, a convex optimization problem with constraints often ends up being non-convex. Since the underlying objective function $l(\theta)$ is non-convex in nature, we require the parameter θ to be constrained within a set Ω_m where Ω_m is a subset of the domain where loss $l(\theta)$ is convex.

Thus, the optimization problem can be written as:

$$\begin{aligned} &\min\ l(\theta) \\ &\text{s.t. } \theta \in \Omega_m. \end{aligned} \tag{4.1}$$

It must be noted that the set Ω_m in (4.1) is a subset of the feasible solution-space which is present in the form of inequality constraints $\theta \in \Omega_m$ due to variable reduction performed on originally prescribed equality constraints $h(x) = 0$. However, it is unlike a general convex optimization with inequality constraints, since $l(\cdot)$ is not guaranteed to be convex anywhere outside of set Ω_m. While it is tempting to apply a projection procedure for ensuring $\theta \in \Omega_m$, the lack of convexity guarantees of $l(\cdot)$ for all $\theta \in \mathbb{R}^m$ makes it very challenging to apply general projection procedures. Furthermore, lack of monotonicity of θ in accelerated learning setting inhibits us from commenting on time-invariant nature of the set Ω_m. In the next section, we delineate a general computational procedure for ensuring that the constraint $\theta \in \Omega_m$ is always satisfied.

4.2 Optimization with Box constraints

Without loss of generality, we assume Ω_m to be a compact set in $\mathbb{R}^m$. For any such Ω_m, it is always possible to find a bounded interval $I = I_1 \times I_2 ... \times I_m \subseteq \Omega_m$ where I_i is a bounded interval in $\mathbb{R}$ defined as $I_i = [\theta^i_{min},\ \theta^i_{max}]$ for all $i = 1, 2, ..., m$.

Using the above arguments, we reformulate the constrained optimization in (4.1) as:

$$\begin{aligned} &\min\ l(\theta) \\ &\text{s.t. } \theta \in I \end{aligned} \tag{4.2}$$

note that I is chosen such that $\theta^* \in I$, where θ^* is the solution of problem (4.2). To ensure that a given set containts θ^*, the condition to be satisfied is provided by the Proposition below.

Proposition 4.2.1 *For a given compact set I, if there exist θ_1, θ_2 such that $l(\theta_1) = l(\theta_2)$, then $\theta^* \in I$*

4.2.1 HT for Inequality Constraints

In this section, we present Algorithm 3 to guarantee convergence to θ^* while ensuring feasibility with respect to the inequality constraints generated by $\theta \in I$. For Algorithm 3, we first prove that $\overline{\theta}_k, \nu_k, \theta_k \in I$ for all $k \in \mathbb{N}$ through Proposition 4.2.2 and subsequently provide guarantees of convergence of the iterates $\{\theta_k\}_{k=1}^{\infty}$ generated by Algorithm 3 to θ^* using Theorem 4.2.1.

For the purpose of feasibility, we first make sure that for each k, $\bar{\theta}_k \in I$. We define define quantities $\hat{a}_k, \tilde{a}_k$ as:

$$\begin{aligned} \tilde{a}_k &= \min_{i \in \{1,\ldots,m\}} \frac{(\theta_k^i - \theta_{min}^i)\mathcal{N}_k}{\gamma\beta|\nabla_i l(\theta_k)|} \\ \hat{a}_k &= \min_{i \in \{1,\ldots,m\}} \frac{(\theta_{max}^i - \theta_k^i)\mathcal{N}_k}{\gamma\beta|\nabla_i l(\theta_k)|}. \end{aligned} \tag{4.3}$$

Similarly, we next ensure that $\nu_k \in I$.We define define quantities $\hat{b}_{k+1}, \tilde{b}_{k+1}$ as:

$$\begin{aligned} \tilde{b}_{k+1} &= \min_{i \in \{1,\ldots,m\}} \frac{(\nu_{k+1}^i - \theta_{min}^i)\mathcal{N}_k}{\gamma\beta|\nabla_i l(\theta_{k+1})|} \\ \hat{b}_{k+1} &= \min_{i \in \{1,\ldots,m\}} \frac{(\theta_{max}^i - \nu_{k+1}^i)\mathcal{N}_k}{\gamma\beta|\nabla_i l(\theta_{k=1})|}. \end{aligned} \tag{4.4}$$

Proposition 4.2.2 guarantees that Algorithm 3 generates iterates such that $\overline{\theta}_k, \nu_k \in I$, consequently $\theta_k \in I \ \forall k$

Proposition 4.2.2 *Consider Algorithm 3, for a given $k \in \mathbb{N}$, if $\theta_k, \nu_k \in I$, there exist real numbers $a_k = \min\{\hat{a}_k, \tilde{a}_k\}$ and $b_{k+1} = \min\{\hat{b}_{k+1}, \tilde{b}_{k+1}\}$ $\forall k$ (where $\hat{a}_k, \tilde{a}_k$ and $\hat{b}_{k+1}, \tilde{b}_{k+1}$ are defined in* (A.11) *and* (A.12) *respectively), such that $\overline{\theta}_k, \nu_{k+1} \in I$. Consequently, for $0 < \beta \leq 1$, and $\theta_0, \nu_0 \in I$, Algorithm 3 guarantees $\theta_k, \overline{\theta}_k, \nu_k \in I$ for all values of $k \in \mathbb{N}$.*

Proposition 4.2.2 outlines conditions under which all parameters generated by Algorithm-3 are bounded within set I, where the convexity of loss function $l(\cdot)$ is guaranteed. Theorem 4.2.1 formally establishes the stability and convergence of Algorithm 3 to an optimal solution θ^*.

Theorem 4.2.1 *For a differentiable $\bar{L}_k$-smooth convex loss function $l(.)$, Algorithm 3 with $0 \leq \beta \leq 1$, $0 < \gamma < \frac{\beta(2-\beta)}{8+\beta}$ and a_k, b_{k+1} satisfying $a_k \leq \min\{1, \bar{a}_k\}$, $b_{k+1} \leq \min\{1, \bar{b}_{k+1}\}$, where $\bar{a}_{k+1}$ and $\bar{b}_{k+1}$ are defined in* (A.11) *and* (A.12) *ensures that $V = \frac{\|\nu-\theta^*\|^2}{\gamma} + \frac{\|\nu-\theta\|^2}{\gamma}$ is a Lyapunov function. Consequently, the sequence of iterates $\{\theta_k\}$ generated by Algorithm 3 satisfy $\{\theta_k\} \in \Omega_m$, and $\lim_{k\to\infty} l(\theta_k) = l(\theta^*)$, where $l(\theta^*) = f([\theta^{*T} \quad p(\theta^*)^T]^T)$ is the optimal value of* (4.1).

Algorithm 3 HT Optimizer for localized convex optimization

1: **Initial conditions** θ_0, ν_0, gains γ, β
2: Choose $\theta_0, \nu_0 \in I$
3: **for** $k = 1$ to N **do**
4: Compute $\nabla l(\theta)$ and let $\mathcal{N}_k = 1 + H_k$
5: $\nabla \overline{q}_k(\theta_k) = \dfrac{\nabla l(\theta_k)}{\mathcal{N}_k}$
6: $\overline{\theta}_k = \theta_k - \gamma\beta a_k \nabla \overline{q}_k(\theta_k)$
7: $\theta_{k+1} = \overline{\theta}_k - \beta(\overline{\theta}_k - \nu_k)$
8: Compute $\nabla l(\theta_{k+1})$ and let
9: $\nabla \overline{q}_k(\theta_{k+1}) = \dfrac{\nabla l(\theta_{k+1})}{\mathcal{N}_k}$
10: $\nu_{k+1} \leftarrow \nu_k - \gamma b_{k+1} \nabla \overline{q}_k(\theta_{k+1})$
11: **end for**

4.2.2 An academic example

We consider the same example as before which led to the following constrained convex optimization problem (Problem 4.1):

$$\begin{aligned} &\min \ \log(e^{x_1} + e^{4-\sqrt{1-x_1^2}}) \\ &\text{s.t.} \ \ -1 \leq x_1 \leq 1 \end{aligned} \tag{4.5}$$

The box constraint in (4.5) is imposed to ensure that the objective function is always real-valued. While we can choose certain step-sizes that ensure that $-1 \leq x_1 \leq 1$ for solving (4.5), there are no guarantees that such a step-size exists to ensure this constraint. Algorithm 3 solves this problem with suitable choices of γ, β, a_k and

b_{k+1} as specified in Theorem 4.2.1. Figure 4-1 shows the convergence of parameter θ (equivalent to x_1 in (4.5)) using Algorithm 3 for a specific value of γ, β. It should be noted in Figure 4-1 that Algorithm 1 fails to lead to convergence, since the box constraints are not satisfied.

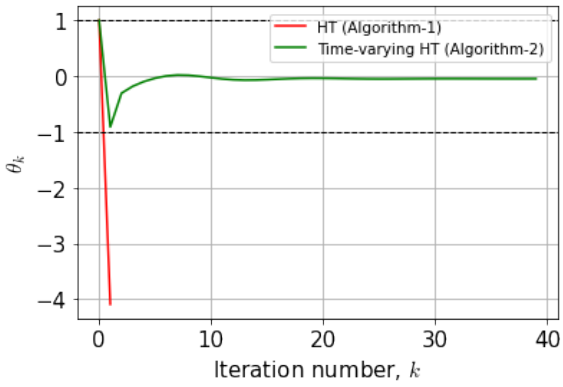


Figure 4-1: Convergence of θ using Algorithm 3 for the Problem-1 in (4.5); Algorithm 1 fails to converge

4.2.3 Provably hard problem of Nesterov

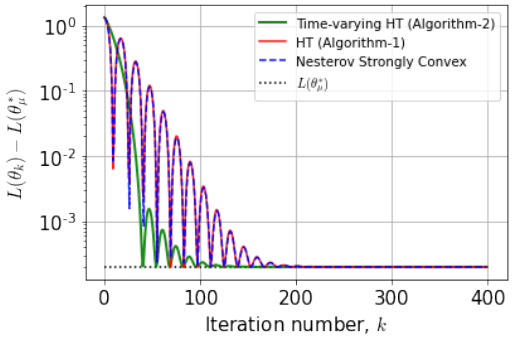
We consider a provably hard problem which corresponds to a strongly convex function (Problem 4.2) (see [32] for details)

$$\begin{aligned} l(\theta) = \log\left(c_k e^{d_k\theta} + c_k e^{-d_k\theta}\right) + \frac{\mu}{2}\|\theta - \theta_0\|^2. \\ s.t.\ \ \theta_{min} \leq \theta \leq \theta_{max} \end{aligned} \tag{4.6}$$

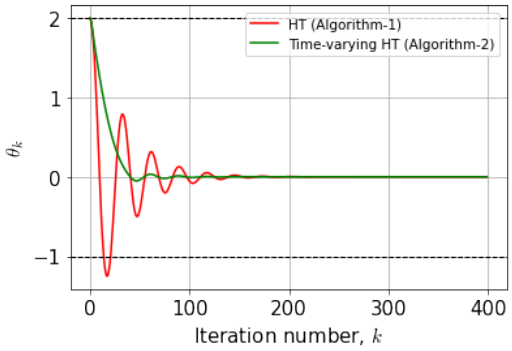
Here c_k and d_k are positive scalars chosen as $c_k = \frac{1}{2}$ and $d_k = 1$. This function has a unique minimum at $\theta^* = 0$. The objective is to show that High Order Tuner in Algorithm 3 ensures that the iterates lie within a closed interval $[\theta_{min},\ \theta_{max}]$.

Figure 4-2a shows the results with Algorithm 3. Also provided as a comparison are the results using Nesterov's algorithm [35]. In all cases, $\mu = 10^{-4}$, the intial value was chosen to be $\theta_0 = 2$ and the constraints are chosen as $\theta_{min} = -1$ and $\theta_{max} = 2$. Also included are the results using Algorithm 1. It is clear that with Algorithm 3, the parameters converge to the optimal value while remaining within the specified limits, as observed in Figure 4-2b. It can also be observed that the speed of convergence is

faster than that of Algorithm 1. Finally, it can be seen that Algorithm 3 exhibits the least amount of oscillations compared to all other algorithms, which is an attractive property. While in the first example, the box constraints were for ensuring convexity in a bounded domain, in the second example, the μ-term in (4.6) was added to ensure strong convexity.



(a) Accelerated learning properties of the HT in Algorithm 3 for the provably hard problem in comparison with Nesterov's algorithm as well as the HT in Algorithm 1. Algorithm-3 converges faster



(b) Constraint satisfaction by Algorithm-3. Algorithm-1 generates iterates that violate the box constraints [-1,2]. Algorithm-2 displays smaller amplitude oscillations compared to Algorithm-1

Figure 4-2: Feasibility and convergence comparison for HT with and without Time-varying gains for Problem 2 (4.6)

The methods outlined above work for constraint satisfaction for a non-convex optimization problem because convexity is guaranteed by virtue of feasibility. The overall approach involves tuning the hyperparameters.

In the following sections, we explore the task of constrained optimization within a compact set by using a projection operator. We motivate the problem by comparing HT for linear regression with a second order linear dynamic system and utilize this study as a basis for applying projection. We initiate the discussion with a linear regression problem and extend the underlying idea to a broader category of applications.

4.3 HT for constrained linear regression

Consider an optimization problem in linear regression setting, parameterized by $\theta \in \mathbb{R}^n$ where the gradient of the loss function is $\nabla l(\theta) = \phi\phi^T\tilde{\theta}$, $\tilde{\theta} = \theta^* - \theta$ with $\phi \in \mathbb{R}^n$ being the input regressor and loss being $l(\theta) = \|y - \theta^T\phi\|^2$. The true system is represented as $y^* = \theta^{*T}\phi$. The optimization problem is shown as a block diagram in Figure 4-3. We add additional constraints to this and the overall minimization problem can be written as:

$$\begin{aligned} &\min\ l(\theta) \\ &\text{s.t. } |\theta| \leq A \end{aligned} \tag{4.7}$$

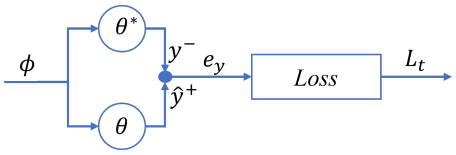


Figure 4-3: Block Diagram of Linear Regression

Where $A > 0$ is a constant used to denote the constraint on the maximum permissible amplitude of the parameters. We address this problem using HT. Due to the oscillatory behavior of parameter [21], it is challenging to apply projection schemes typically used in gradient-descent based algorithms. For the case where ϕ is time-invariant *and is a diagonal matrix*(i.e., zero correlation between different components of the input data), we can decompose the problem into simulatenous scalar linear regression problems. The consequent section evaluates this specific problem and provides comparison of the HT to the governing equation of a second order LTI system.

4.3.1 Analogy between Second order LTI systems and High Order Tuner

Equation for a damped oscillator with $f(X)$ being the potential field can be written as [52]:

$$\ddot{X}(t) + \frac{c}{m}\dot{X}(t) + \frac{1}{m}\nabla f_t(X(t)) = 0 \tag{4.8}$$

Here m is the effective mass of the system, c is the damping factor. This is equivalent to the HT in continuous time as [22]:

$$\ddot{\theta}(t) + \beta\dot{\theta}(t) + \frac{\gamma\beta}{\mathcal{N}_t}\nabla L_t(\theta(t)) = 0 \tag{4.9}$$

HT with linear regression with constant regressors for a scalar input-output problem, where $\nabla L_t(\theta(t)) = \|\phi\|_2^2\theta(t)$ can be written as:

$$\ddot{\theta}(t) + \beta\dot{\theta}(t) + \frac{\gamma\beta}{\mathcal{N}_t}\|\phi\|_2^2\theta(t) = 0 \tag{4.10}$$

For an equivalent mass-spring damper system:

$$\ddot{X}(t) + 2\omega_n\tau\dot{X}(t) + \omega_n^2 X(t) = 0 \tag{4.11}$$

Using $\mathcal{N}_t = 1 + \|\phi\|_2^2$ as established in [22]. Comparing (4.10) and (4.11) we have:

$$\begin{aligned} \beta &= 2\omega_n\tau \\ \gamma\beta\|\phi\|_2^2 &= \omega_n^2(1 + \|\phi\|_2^2) \end{aligned} \tag{4.12}$$

We would like the HT to imitate the lightly damped case, i.e., $\tau < 1$ to preserve the acceleration and oscillation. This leads to the following inequalities in addition to those required by Lyapunov stability criteria for linear regression:

$$\begin{aligned} 0 < \frac{\beta(1 + \|\phi\|_2^2)}{4\gamma\|\phi\|_2^2} < 1 \\ 0 < \beta < 1 \\ 0 < \gamma \leq \frac{\beta(2 - \beta)}{(16 + \beta^2)} \end{aligned} \tag{4.13}$$

Now we would like to tune the amplitude, in such a way that the maximum possible amplitude is contained within a box. We first find the position and time of maximum amplitude for a second order under-damped LTI using the closed form solution of a

2-dimensional LTI system:

$$\theta(t) = 2e^{-\sigma t}(\alpha cos(w_d t) + \beta sin(w_d t))$$

Here $\alpha = \frac{\theta_0}{2}$, $\beta = -\frac{\dot{\theta}_0 + \sigma\theta_0}{2w_d}$. Hence, I.C provides us with a choice in shaping the time at which maximum amplitude can be obtained.

Since $\dot{\theta(t)} = 0$ at every local maximum and minimum, we get:

$$tan(w_d t) = \frac{\alpha\tau + \beta\sqrt{1-\tau^2}}{\alpha\sqrt{1-\tau^2} + \beta\tau}$$

We want all such maximas and minimas to be contained within some set, say $|\theta(t)| \leq A$. This can be done by appropriately choosing I.C. and setting $\theta(t) \leq A$.

For instance, if we choose α, β such that $w_d t_1 = \pi/2$, (t_1 represents the first instance of maximum/minimum) then the corresponding inequality will look like:

$$e^{-\frac{\pi\tau}{2\sqrt{1-\tau^2}}} \frac{\sqrt{1-\tau^2}}{\tau}\theta_0 \leq A \tag{4.14}$$

The ideal approach would be to find one of the system parameters, i.e., τ such that (4.13) can be satisfied and thereby use (4.14) to find the right value of θ_0.

Shortcomings of this approach

The caveat of this approach is that it is very restrictive to the case when regressors are constant, can be decoupled and the system evolves under a linear regression setting. However, we can use some of the insights with the comparison of hyperparameters in HT to damping factors in a second-order dynamic system.

4.4 General Case (non-LTI)

In the general setting, we can equate the normalized gradient to the potential field.

$$\frac{\nabla L_t(\theta(t))}{\mathcal{N}_t} = \nabla f_t(X(t)) \tag{4.15}$$

Similarly, we can write $c = k\gamma$ and $\frac{1}{m} = k\gamma\beta$, here k is a scaling constant to ensure that $0 < \gamma, \beta < 1$. Clearly, for a fixed k, by increasing γ we will increase the damping in the system, thereby reducing the amplitude of oscillations, and we would have to carefully decrease β in order to preserve the effective mass of the system. The simultaneous increase and decrease of these quantities will then depend on the exact nature of the loss function. However, if we re-write equation (4.9) as a coupled linear system, we have:

$$\begin{aligned} \dot{\nu}(t) &= -\gamma \frac{\nabla L_t(\theta(t))}{\mathcal{N}_t} \\ \dot{\theta}(t) &= -\beta(\theta(t) - \nu(t)) \end{aligned} \tag{4.16}$$

We focus on a more general problem, where we want to constrain our parameter to be within a compact set C.

Inspired from the LTI comparison, we model the *effective* gain β in such a way that $\dot{\theta}(t) = 0$ at the boundary of set C.

4.4.1 Using Projection Operator for HT

Inspired from the idea of tuning $\dot{\theta}(t)$, we apply a projection operator commonly seen in adaptive systems [28, 27] for containing the parameter $\theta(t)$. The underlying concept is to control the rate $\dot{\theta}(t)$ to preserve $\|\theta\| \leq \theta_{max} + \epsilon$, for some given θ_{max}, ϵ. Thus we can re-state the optimization problem as:

$$\begin{aligned} &\min\ l(\theta) \\ &\text{s.t. } \|\theta\| \leq \theta_{max} + \epsilon \end{aligned} \tag{4.17}$$

4.4.2 Application of projection operator to $\dot{\theta}(t)$

The continuous-time equations of HT updates with projection can be written as:

$$\begin{aligned} \dot{\nu}(t) &= -\gamma \frac{\nabla L_t(\theta(t))}{\mathcal{N}_t} \\ \dot{\theta}(t) &= -\beta Proj(\theta(t), (\theta(t) - \nu(t), f) \end{aligned} \tag{4.18}$$

Here, f is a convex function such that $f(\theta(t)) = 0$ when $\|\theta(t)\| = \theta_{max}$, where $\|\theta(0)\| \leq \theta_{max}\|$, i.e, the parameters are initialized within the norm-ball θ_{max}. Additionally, $f(\theta(t)) = 1$ when $\|\theta(t)\| = \theta_{max} + \epsilon$. The projection operator can be written as:

$$proj_C(\theta, (\theta(t) - \nu(t), f) = \begin{cases} \left(I - \frac{\nabla f(\theta) \nabla f(\theta)^T}{\|\nabla f(\theta)\|^2} f(\theta)\right)(\theta(t) - \nu(t)) & \text{if } f(\theta) > 0 \bigwedge y^T f(\theta) > 0 \\ (\theta(t) - \nu(t)) & \text{otherwise} \end{cases} \tag{4.19}$$

For a given ϵ, θ_{max} we adopt a simple definition of $f(\theta)$ from [28] as:

$$f(\theta) = \frac{\|\theta(t)\|^2 - \|\theta_{max}\|^2}{2\epsilon\theta_{max} + \epsilon^2} \tag{4.20}$$

With the above-mentioned projection operator and HT updates given by (4.18), we can guarantee stability for the case of linear regression as summarized below:

Theorem 4.4.1 *For a quadratic loss function $l(\theta) = \|y - \theta^T \phi\|^2$, a compact and convex set C defined as $C = \{\theta | \|\theta\| \leq \theta_{max} + \epsilon\}$ the sequence of iterates $\{\theta_k\}$ generated by Algorithm 2 with $proj_C$ defined by (4.19) satisfy $\{\theta_k\} \in C$ and $V = \frac{\|\nu - \theta^*\|^2}{\gamma} + \frac{\|\nu - \theta\|^2}{\gamma}$ is a valid Lyapunov function*

In a discrete-time implementation of this approach, we can use Algorithm 2, with the projection operator replaced by (4.19).

4.5 Numerical Simulation

We validate the utility of the proposed projection operator first in a linear regression setting (stability guarantees in continuous time) and then in a non-convex optimization setting.

4.5.1 Linear Regression

Problem Setup

We perform simulation for a 3-dimensional linear regression problem (4.17), with $\theta_{max} = 0.7$ and $\epsilon = 1$, therefore, we need $\|\theta\| \leq 1.7$, where

$$\theta^* = [0.1965 \ -0.3835 \ -1]^T$$

We perform simulation using HT with and without projection (i.e, Algorithm 2 and Algorithm 1 respectively) to show the influence of projection in containing the parameter θ within the required norm-ball. For both algorithms, $\gamma = 0.8, \beta = 10^{-2}$ was chosen and optimization was performed for 1000 iterations each with constant regressor ϕ, which was chosen to be such that $\phi\phi^T$ is a diagonal matrix of identical terms.

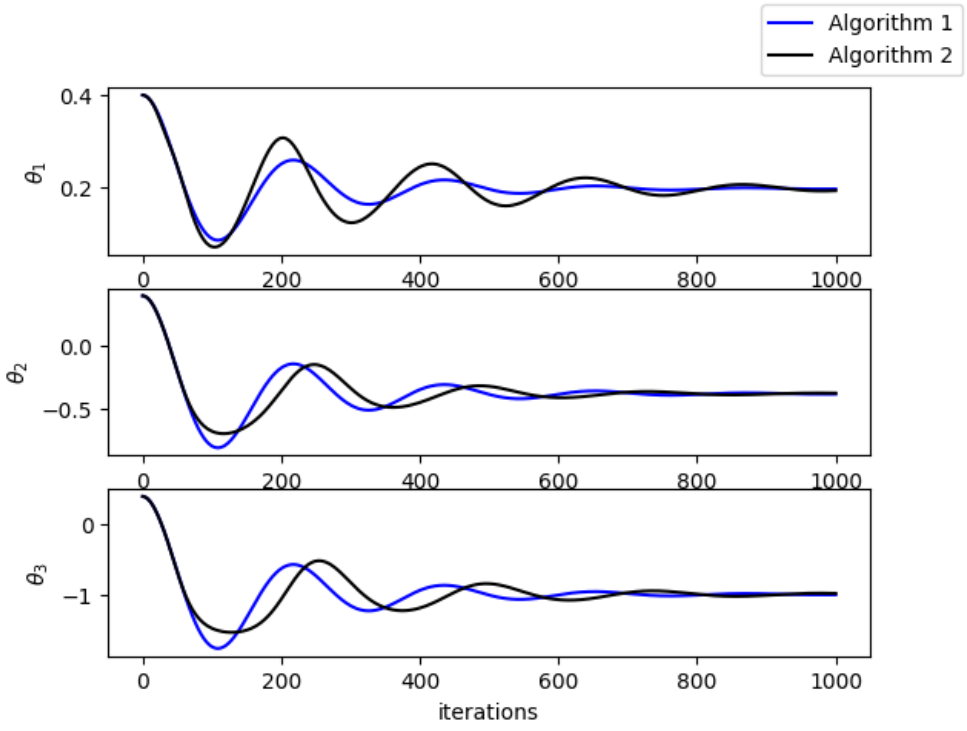


Figure 4-4: Convergence of θ to the optimal value $\theta^* = [0.1965 \ -0.3835 \ -1]^T$ for Algorithm 1 and Algorithm 2

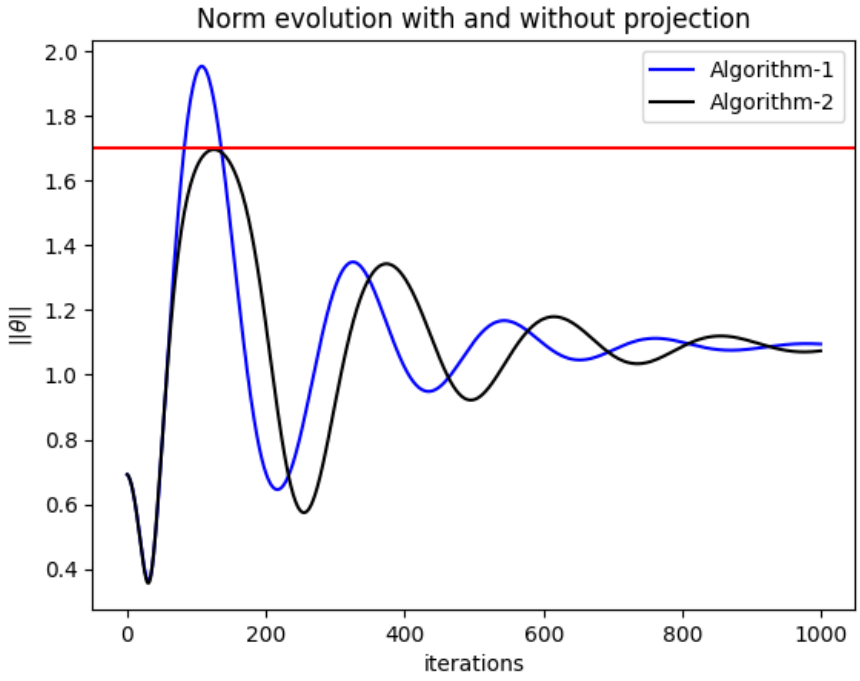


Figure 4-5: $\|\theta\|$ for Algorithm 1 and Algorithm 2. Notice that the norm constraint (norm limit denoted in red) is conserved with Algorithm 2 with projection whereas Algorithm 1 violates the constraint

Observations

Figure 4-4 shows the convergence of parameter θ to the optimal value for both algorithms. It appears that the difference between both algorithms is nominal, however, to notice the conservation of norm, we look at the evolution of $\|\theta\|$ as shown in Figure 4-5. As evident, $\|\theta\| \leq \theta_{max} + \epsilon$ i.e, $\|\theta\| \leq 1.7$ for Algorithm 2 by using the projection operator as defined in (4.19). However, $\|\theta\|$ often violates the required constraint.

4.5.2 Projection for Rosenbrock function

Problem Setup

We consider a 2-D non-convex optimization problem in this section, to strongly motivate the requirement of constraints in accelerated learning and how it can be solved.

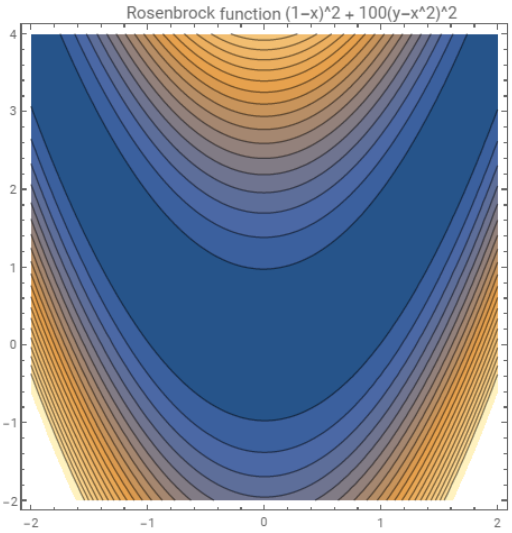


Figure 4-6: Rosbenbrock function: (vertical axis-y, horizontal axis-x) region in blue represents lesser value of loss function and yellow represents higher value of loss. Hence, the minimum lies at $(1, 1)$.

Formally, a Rosenbrock function is defined as [43]:

$$L(x, y) = (1 - x)^2 + 100(y - x^2)^2 \tag{4.21}$$

Here $\theta = [x \;\; y]^T$

This function is non-convex, except for the region where $y \leq x^2 - \frac{1}{200}$. The loss function has a global minimum at $(1, 1)$, which is also within the convex region of the function. Figure 4-6 shows the contour of the Rosenbrock function. Firstly it must be stated that GD often fails to converge for most hyperparameter settings on such a problem and converges to a sub-optimal point. Here, accelerated learning plays an important role.

Observations

In order to perform optimization to the optimal value in this case, we initialize $x_0 = -0.5, y_0 = 0.5$ for both Algorithm 1 and Algorithm 2 and choose $\gamma = 5 \times 10^{-3}, \beta = 10^{-6}$. For Algorithm 2, we constrain $\|\theta\| \leq 1.21$ to enable convergence and retain the parameters (x, y) in a set where $L(x, y)$ is convex. Figures 4-7, 4-8 illustrates

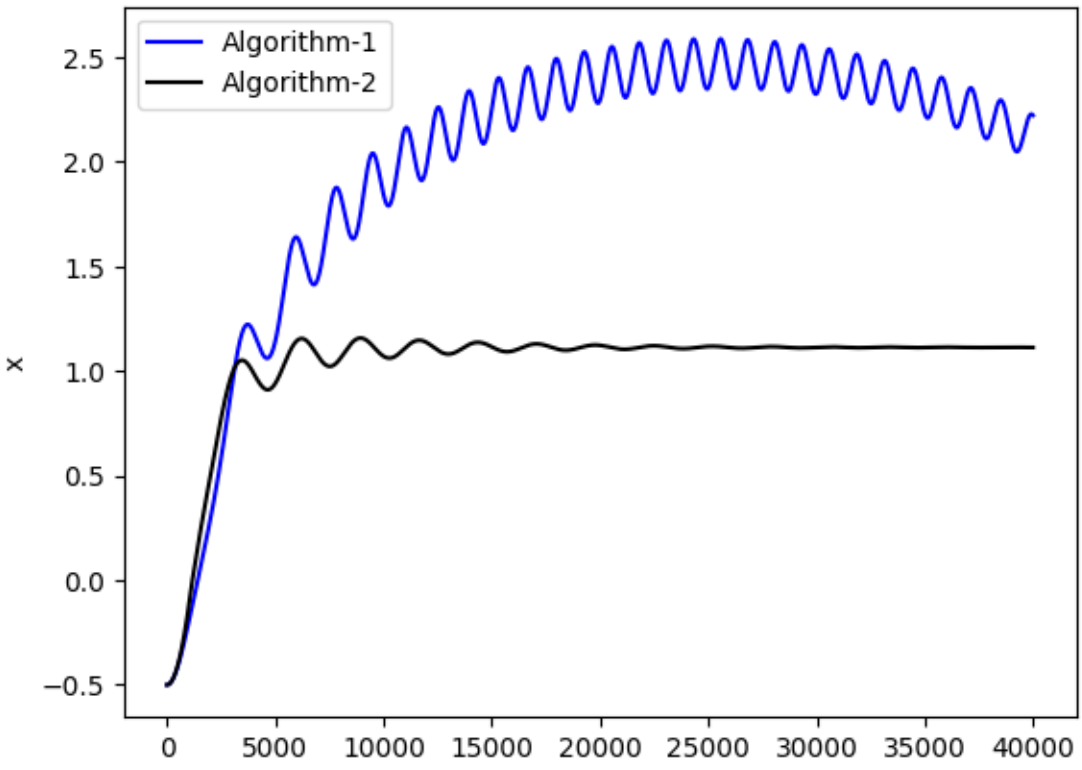


Figure 4-7: Convergence of x for Rosenbrock function: Algorithm 1 fails to converge to the global minimum $x^* = 1$ since this is a non-convex optimization problem. However, Algorithm 2 utilizes the projection operator to contain the parameter within a set where the loss is always convex and convergence is ensured, as explained in Chapter 3.

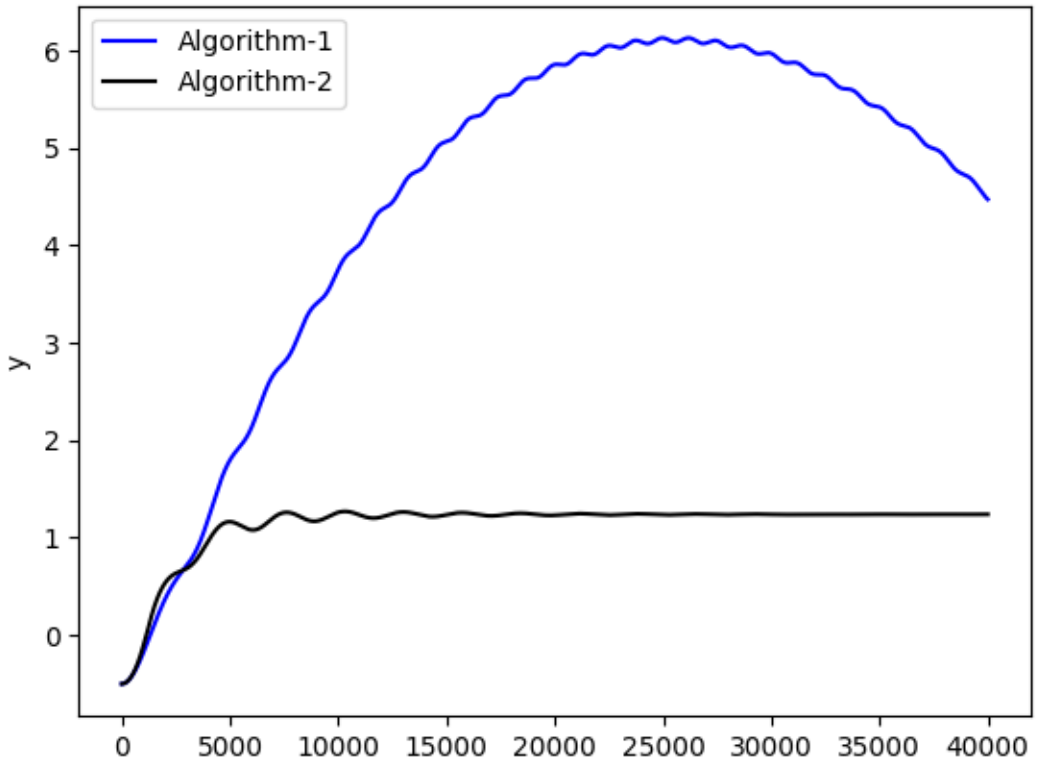


Figure 4-8: Convergence of y for Rosenbrock function: Algorithm 2 converges to the global minimum with a carefully chosen convex set and projection operator to ensure that the parameter stays within the set

the importance of projection to a convex set in the case of non-convex optimization, which can be utilized to guarantee stability and convergence.

4.6 Conclusions

In this chapter, we discuss techniques for parameter feasibility within compact set while ensuring acceleration, which is a challenging mathematical problem to solve. We present two approaches, one involves tuning the hyperparameters at every instant, and provide stability proof in discrete-time for the same. This may not always be feasible due to the choice of hyperparameters being restrictive.

Thus, as an alternative we provide a projection based approach for HT, and demonstrate its merit for two examples. This approach is intuitively inspired from the equivalence of the algorithm to a spring-mass damper system. A proof of stability in continuous time is presented for linear regression setting in this case and has potential to be extended to discrete-time setting for convex optimization, as is evident from the numerical simulations.

Chapter 5

Accelerated Methods for solving OPF problems: Numerical study

In real world applications, the underlying optimization problem is often non-convex. In this chapter, we specifically look at a widely-considered nonconvex optimization problem, i.e, Neural Network training for solving DC-Optimal Power Flow problem, which has been widely looked at [16, 37]. We utilize a combination of several techniques illustrated in the previous chapters to demonstrate their application on a real-world optimization problem.

5.1 Introduction

The emergence of learning-based optimization techniques has sought applications in the realm of optimization problems within power systems domain [48, 53, 15]. The vast majority of the work in this collaborative space has been demonstrated through numerical simulations in solving DC-OPF and AC-OPF. We examine a learning-based technique to solve DC-OPF in this Chapter using HT to illustrate the impact of acceleration in large-scale non-convex optimization. Due to the simplicity of the problem and availability of solvers, we are also able to compare the performance of our method against the baseline, which has been chosen as MATPOWER for this study [54].

The existing learning-based methods exploit the availability of abundance of data of varying nature which is already made available through solvers and real-time solving of grid, and with accurate predictions, they reduce the online solving time by 100-200 times [17], [37]. The vast majority of the work in this space focuses on ensuring feasibility of the solution process. In this work, we have two main contributions:

- We utilize the variable reduction techniques in Proposition 3.1.1 to ensure feasibility
- We address an important issue in large-scale NN training-peed of convergence and dataset requirement and how accelerated learning can help.

5.1.1 Problem formulation

A general OPF can be written as (3.1), and DC-OPF specifically takes the following form:

Objective function:

$$f(P_g, \phi) = P_g^T C P_g \tag{5.1}$$

Equality Constraints: $h(P_g, \phi)$

$$\boldsymbol{B} \cdot \phi - (P_g - P_d - P_{sh}) = 0 \tag{5.2}$$

Inequality Constraints:

$$P_{Gi}^{min} \leq P_{Gi} \leq P_{Gi}^{max},\ i = 1, 2, ..., N_{gen} \tag{5.3a}$$

$$\frac{1}{x_{ij}}(\phi_i - \phi_j) \leq P_{ij}^{max}\ i = 1, 2, ..., N_{bus} \tag{5.3b}$$

Guarantees for equality constraint satisfaction

As we can see, the equality constraints are linear, hence, as per the process described in Section 2.3.2, a convex loss function $\mathcal{L}(P_G, \phi)$ can be reduced to a convex loss function $l(P_G)$, with ϕ being the dependent variable and P_G being the independent variable (recall $x = [\theta \quad z]^T$). Hence, we can easily leverage Algorithm-1 along with

guarantees provided by Proposition 3.1.1 and Theorem 3.2.1 to solve a DC-OPF problem to generate solution which is feasible with respect to the equality constraints.

Using NN to solve this problem makes it non-convex, and devoid of the optimality guarantees. However, we can still utilize the variable reduction guarantees from Section 2.3.2 since they are agnostic to convexity of the problem. Across different instances of applications, the input load P_D is utilised as the input for training a neural network which predicts generated power P_G. The phase angle ϕ is predicted as a function of P_G using (5.2). This helps substantially in reducing the overall dimension of the problem.

Inequality constraints

While in theory, the inequality constraint completion procedure described in Chapter 2 can be used to solve a complete DC-OPF problem, however, unlike the case of equality constraints, the inequality constraints are not *guaranteed* to be satisfied. The objective is to train a neural network such that predicted value is feasible with respect to the inequality constraints. Overall, the merit of this approach is that we save substantial amount of time in online OPF computation with a well-trained neural network.

We attend the inequality constraint problem by using the NN to predict a quantity α. It has been shown that the usage of activation functions in the output layer activation greatly helps with feasibility preservation. Therefore, we use a sigmoid activation function to ensures that the output is between 0 and 1. This approach has been adopted from [37]. Hence, instead of training the neural network for P_G, it is trained to output α, which is a vector consisting of elements with values between 0 and 1. α is then transformed to P_G using:

$$P_{Gi} = \alpha_i(P_{Gi}^{max} - P_{Gi}^{min}) + P_{Gi}^{min} \quad i = 1, 2, ..., N_{gen} \tag{5.4}$$

Having described the constraint satisfaction procedure, we address the loss formulation in this section for our NN training. The proceeding sections detail the integration

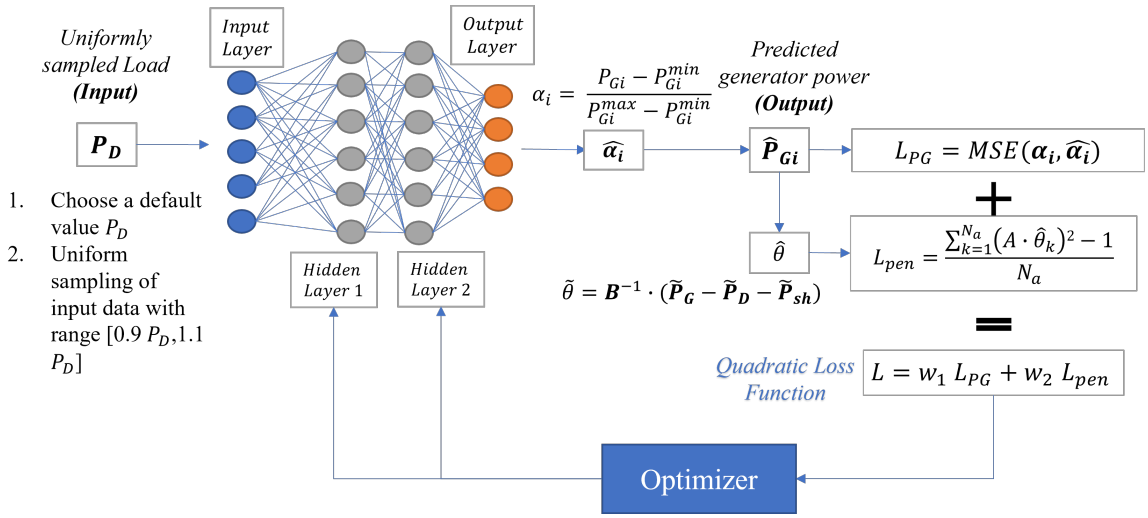


Figure 5-1: Schematic summarizing the general approach for solving DC-OPF with Neural Network [37]. The 'Optimizer' can be any optimization algorithm, and has been chosen as HT for our method

of the neural network with the variable reduction approach of Section 3.2 and the implementation of High Order Tuner an optimizer to such a problem.

HT for solving DC-OPF using learning-based methods

The underlying neural networks must approximate the mapping of an albeit large power grid extremely well. This is a well-posed problem that becomes more complex as the size of power grid increases, thereby increasing the dimension of input and output data and the size of training dataset. Despite of employing the best data pre-processing practices, under certain circumstances, training of neural network is affected by the performance of gradient-descent based optimization algorithms in converging to the global minimum. In this work, we demonstrate the efficacy of HT for training the underlying neural networks efficiently across different sizes of networks.

Figure 5-1 summarizes our approach for solving DC-OPF using Neural Network training, which is covered in detail in the next section.

5.2 Solving DC-OPF using Neural Networks

5.2.1 Approach Outline

The following section outlines the details of implementation of learning-based technique for DC-OPF. The illustrated method is inspired by several works in the literature, since they address the concept of feasibility with neural networks efficiently. For the purpose of this study, the objective was to compare the performance of High Order Tuner and Gradient Descent as optimizers under different settings. The remainder of this section outlines the numerical experiment conditions step-by-step and has been adopted from [16], [37].

Input Load & Data generation: P_D

A uniform sampling method was applied to choose input load values with a user-defined deviation (between 90-110% in our case) about the default load value. Subsequently, MATLAB based OPF solver called MATPOWER was chosen to generate the outputs P_G, ϕ corresponding to each P_D. For numerical experiments, IEEE Datasets corresponding to various cases were chosen and fed through MATPOWER for data generation.

Since the problem is a Quadratic Programming problem with constraints, MATPOWER generates optimal and feasible solutions, hence the generated data is chosen as reference data for training. Subsequently, the weights and biases of the chosen neural network are trained in a supervised setting to generate predictions.

Neural Network Design

The concept of selection of the right number of layers and size of each layer can be referred to as hyperparameter tuning, and is usually done by observing the performance of the trained network for a given set of hyperparameters, i.e., h number of neurons in each layer and N_{hid} number of hidden layers. For simplicity, we started out by choosing the same number of neurons across all layers.

Note that this was an especially challenging task, since various bodies of work in this field refer to certain hyperparameter settings specific to their study and validate their choices by showing convergence of prediction error to a desirable level. However, none of the works in the literature provide insights into the process of selection of these hyperparameters.

While there are rough heuristic guidelines one can employ, such as keeping h small to prevent the NN from becoming a memory bank of the train dataset. Quite often however, it leads to trial and error with little or no theoretical basis. To navigate through this problem, we restricted our attention to the performance of a neural network with a single hidden layer i.e., $N_{hid} = 1$. Hence, the problem of hyperparameter tuning reduces to one of finding an optimal number of neurons h. There were two main factors that led to this choice:

1. It was observed that the performance (i.e., training and validation losses) of a single-hidden layer NN was comparable to that of a multi-layer NN, so there was not a substantial loss of accuracy with this choice

2. More importantly, even though a single hidden layer NN itself leads to non-convex optimization, it can be transformed to an equivalent convex optimization problem with linear inequality constraints using dual optimization techniques [42, 49], allowing us to be closer to guarantees.

Choice of Activation Function

There are several choices of activation functions, such as ReLU, sigmoid, etc for the hidden layer. In this study, ReLU was chosen as the activation function for the single hidden layer NN. This was based on the fact that a non-decreasing activation function is required to transform the neural network training problem into a constrained convex optimization [42].

One may argue that using multiple hidden layers with ReLU activation functions in the hidden layers is a better heuristic choice. For higher dimension cases, the absence or presence of sigmoid is rather insignificant in improvement in training

performance. Ultimately, the results obtained on Single hidden layer NN with ReLU activation function can be extended to a larger and complex network easily, with carefully chosen settings that aid accuracy improvement.

This kind of analysis is beyond the scope of this work, hence, we restrict our attention to "single hidden layer neural network with ReLU activation function."

Computing dependent variable and soft loss formulation

The approach described in Section 2.3.2 is utilized to compute the predicted value of dependent variables ($\hat{\phi}$) from predicted output of Neural Network $\hat{P}_G$ using (5.2). As we can see, the relation between ϕ and P_G is linear, i.e., ϕ is an explicit function of P_G, hence easy to compute. In case of AC-OPF, it is not possible to express $\phi = f(P_G)$, hence an iterative solver has to be used, such as Newton's method [16].

Once we have the predicted values, we formulate loss function $f(x)$ similar to the one shown in Section 2.3.1, (2.4). To ensure the function is convex, a mean squared error loss function is chosen, to compare $\hat{P}_G$ with P_G, where P_G is the value obtained for the given P_D using MATPOWER. Thus, loss function can be written as:

$$f(P_G, \phi) = \frac{1}{N_{gen}} \sum_{i=1}^{N_{gen}} \|\hat{P}_G - P_G\|^2 \tag{5.5}$$

Similar to the inequality violation term in (2.4), a penalty loss function is proposed to penalize violation of (5.3b) which is modelled as a quadratic loss function and can be written as:

$$g(P_G, \phi) = \frac{1}{N_a} \sum_{k=1}^{N_a} (A\hat{\phi}_k)^2 - 1 \tag{5.6}$$

where N_a is the number of adjacent buses, and A is a $N_a \times N_{bus}$ matrix, with each row in A corresponding to an adjacent bus pair. Element A_{ij} of the matrix corresponds to a flow from i^{th} bus to j^{th} bus. This is modification of (5.3b), since directly (5.3b) cannot be utilized as a penalty loss function. The relation of the inequality constraint

with matrix A can be expressed as:

$$a_i = \frac{1}{P_{ij}^{max} \cdot x_{ij}} \text{ and } a_j = -\frac{1}{P_{ij}^{max} \cdot x_{ij}} \tag{5.7}$$

where a_i, a_j refer to the corresponding entries of matrix A.

Collectively, the overall loss function $\mathcal{L}(P_G, \phi)$ can be written as:

$$\mathcal{L}(P_G, \phi) = w_1 f(P_G, \phi) + w_2 g(P_G, \phi) \tag{5.8}$$

w_1, w_2 correspond to percentage of loss function's weightage towards penalization and optimization. In practice, it is observed that the Neural Network is able to satisfy inequality constraints and generate feasible solutions. Hence, for the purpose of training, w_2 was chosen to be much smaller than w_1, so that the training is largely centered around optimization.

Overall, $\mathcal{L}(P_G, \phi)$ is convex and reduction of $\mathcal{L}(P_G, \phi)$ to $l(P_G)$ is achieved by a linear equation. Hence, this problem satisfies all conditions of Theorem 3.2.1. Hence, theoretically, with P_G as the decision variable, High Order Tuner is guaranteed to converge to P_G^*, the optimal solution of the DC-OPF problem, from which ϕ^* can be derived easily. However, by choosing P_G as the output of a NN, the decision variable transforms to the weights of the NN as shown next.

Training a Neural Network using High Order Tuner

From optimization's perspective, Neural Network training is a non-convex optimization problem. A single hidden layer DNN problem can be formulated as:

$$f(u, \alpha) = \sum_{j=1}^{h} \sigma(x^T w_{1j}) w_{2j} \tag{5.9}$$

Here w_{1j} and w_{2j} are weights of the j^{th} neurons from input to hidden layer and hidden layer to output layer respectively with $x \in \mathbb{R}^{N_{bus}}$ being the vector corresponding to the input layer which is P_D in this case. Here, $\sigma(\cdot)$ represents the ReLU activation

function.

As $w_2 \to 0$, (5.8) approximates a standard MSE loss. While the literature does not report usage of regularization, during numerical experiments, there were instances of overfitting observed through validation loss patterns. Hence, the recommended approach was to use a l_2 norm regularization in addition to the MSE loss. However, an important distinction must be made. The decision variable for optimization is not P_G, but instead the weights w_1, w_2 associated with the NN. Thus, neglecting the inequality violation term and re-writing (5.8) with a regularization term, with updated decision variables, the loss $\mathcal{L}$ can be expressed as:

$$l(w_1, w_2) = \frac{1}{2N_{gen}} \left(\sum_{i=1}^{N_{gen}} \| \sum_{j=1}^{h} \sigma(x_i^T w_{1j}) w_{2j} - y_i \|^2 \right) + \frac{\beta}{2} \sum_{j=1}^{h} \left(\|w_{1j}\|_2^2 + w_{2j}^2 \right) \quad (5.10)$$

Neural Network training as constrained convex optimization

It can be clearly noticed, that (5.10) is a non-convex function. However, recent work done on representing neural networks as convex regularizers enables transformation of minimization of (5.10) to a convex optimization problem which is written as:

$$\begin{aligned} p^* := \min_{v,w} \frac{1}{2} \| \sum_{i=1}^{P} D_i X (v_i - w_i) - y \|_2^2 + \beta \sum_{i=1}^{P} (\|v_i\|_2 + \|w_i\|_2) \\ \text{s.t. } (2D_i - I_n) X v_i \geq 0, (2D_i - I_n) X w_i \geq 0 \quad \forall i \end{aligned} \quad (5.11)$$

Here, P is a variable dependent on the dimension of the original problem and X represents the matrix form of input vectors x_i organized column-wise for supervised training. The details of this convex formulation can be found in [42].

This gives us a constrained convex optimization problem. For a fixed input dataset and number of neurons h, we can find the optimal loss value p^* which using strong duality is equivalent to l^*. Similarly, the optimal values of weights w_1^*, w_2^* can also be computed from solution v^*, w^*. There are several solvers such as Python based CVXPY that can be used to solve such an optimization problem. While we cannot directly utilize the convex formulation in neural network training, it gives us an insight

into the global loss value that can be achieved with a given number of neurons and size of dataset, and tuning of these hyperparameters becomes easier with this knowledge. Additionally, theoretical guarantees associated with High Order Tuner can also be leveraged to solve the convex optimization problem directly, instead of using another solver, however, that has been excluded from the scope of the current work.

5.3 Numerical Experiments

For validating the utility of accelerated convergence of High Order Tuner for solving Neural Network based DC-OPF problems, we implemented the steps mentioned in Section 5.2. Pytorch framework was used for neural network implementation. IEEE Case 9, 30, 300 and 1354 sourced from [36] were used for dataset generation with the following approach:

1. IEEE Case 9 was chosen to debug the network and implemented framework.

2. IEEE Case 30 and 300 were chosen to compare results between HT and GD in test and training.

3. IEEE Case 1354 has not been discussed in the literature pertaining to learning-based techniques for DC-OPF before, and the intent behind this selection was to validate the capability of High Order Tuner in addressing challenges that arise with a network topography that resembles real-time implementation and solving.

Dataset & Implementation

Training dataset for Case 9,30 and 300 consisted of 30,000 datapoints of P_D and corresponding (P_G, ϕ) values. For Case 1354 50,000 datapoints were generated. Although not all datapoints were utilized in training, since these are benchmark values we started off with based on reference from [37]. Following the training dataset generation, 10,000 points were chosen for each case for validation dataset.

Loss formulation was done using custom loss class using Pytorch to replicate (5.8). However, in practice it was observed that the predicted values had 100% feasibility in test run, and having MSE loss instead did not make an overall difference. Additionally, there was no need for implementing inequality correction procedure similar to a projection method (Algorithm 2) for simulations. However, it is strongly recommended to have the inequality violation adjustment be present in real-time training for DC-OPF.

For the purpose of test run, feasibility was checked for every instance. 10,000 datapoints were generated for test dataset. For Case-9 and 30, the average predicted cost was also calculated to ensure that the prediction error is not extremely high. After obtaining average predicted cost with percentage errors less than 1% with respect to the reference average cost calculated over the entire dataset, and observing 100% feasibility, the model validity was ensured.

This also led to the conclusion that with proper training, the sigmoid activation function at the output layer that is typically used for constraint satisfaction is not necessarily required and can be eliminated.This does spark a discussion however, on the capability of the chosen model to generalize across different varieties of data. However, the focus of this study has been to bridge the gaps in neural network training and convex optimization using High Order Tuner. Readers are referred to [37],[38] to better understand the neural networks that accurately serve the corresponding topography of the chosen IEEE Cases.

For optimizers, custom optimizer classes for High Order Tuner and Gradient descent's implementation were chosen. Instead of using SGD, custom vanilla GD was chosen for a fair comparison between High Order Tuner and Gradient Descent's performance. The normalizing parameter for HT in each case was set to be a constant with value adjusted to suit the performance of the model.

IEEE Case-9: Testing & Debugging

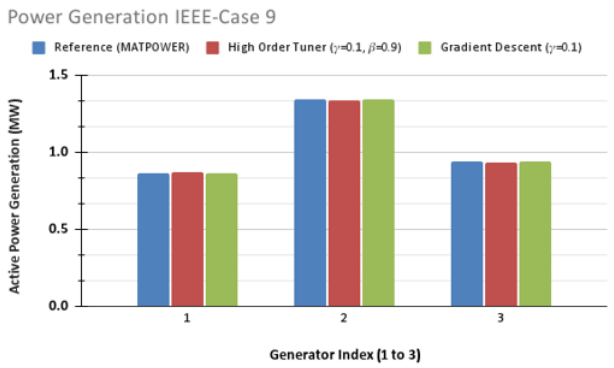


Figure 5-2: IEEE Case-9: Comparison of performance between HT and GD with baseline

- **Training Dataset**: 10,000 points
- **Test Dataset**: 5000 points
- $N_{bus} = 9$
- $N_{gen} = 3$

For this case, 2 hidden layers with 16 neurons in each was chosen as Neural Network with ReLU activation function for each layer.

Observations: Within 20 epochs, both High Order Tuner and Gradient Descent achieved desirable training loss of 8×10^{-5} and 3×10^{-4} respectively. The prediction error histograms for both algorithms have been shown in figure

This ensured that the model is working correctly. It was observed that there is no substantial difference between training performance in HT and GD, which is expected since GD trains pretty well and Case-9 is a simple case (Figure 5-2)

IEEE Case-30: Extension of Case-9

The training, test dataset and structure of neural network chosen remainded the same as Case-9. This was also because these values were taken based on results of IEEE Case-30 DC-OPF in Table-2 of [37] . Case-30 corresponds to $N_{bus} = 30, N_{gen} = 6$.

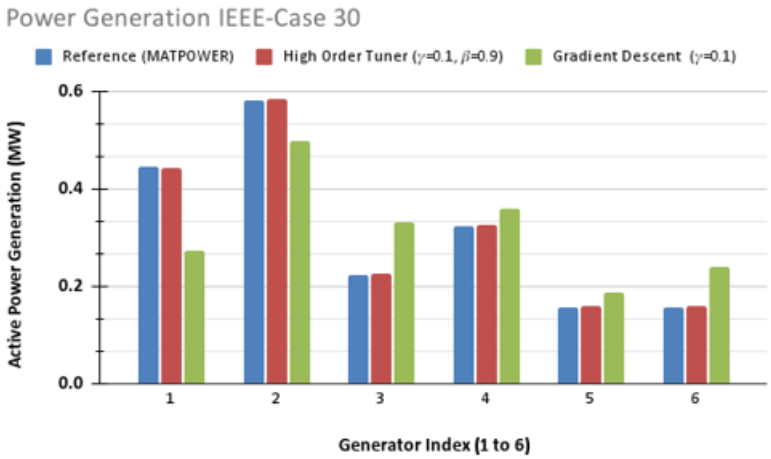


Figure 5-3: Comparison of predicted Power Generation in IEEE Case-30 by HT and GD with baseline MATPOWER. HT power prediction is more accurate than GD

Low learning rate $\gamma = 10^{-3}$ was chosen for both HT and GD. Number of epochs was set to 20 and prediction errors corresponding to Generators 1,2...6 were observed. Here, even though the training loss was satisfactory for both HT and GD, (approximately 3×10^{-4}), in prediction performance over test dataset, HT performed substantially better. This can be seen from the prediction error histograms shown in Fig 5-4 and Fig 5-3.

Observations: The results obtained are not very strongly conclusive, since GD can be trained better to obtain similar prediction performance. Moreover, the purpose of comparison between HT and GD was to observe acceleration in training, which was not substantial in Case-30 and 9 due to simplicity of these models.

Also, at this point, comparison was made between a multi-layer neural network and a single layer neural network with larger neurons, and with no substantial difference observed, latter was chosen for the subsequent cases. This was double-checked by again comparing the performance of Case-300 and Case-1354 with single and multi-layer neurons.

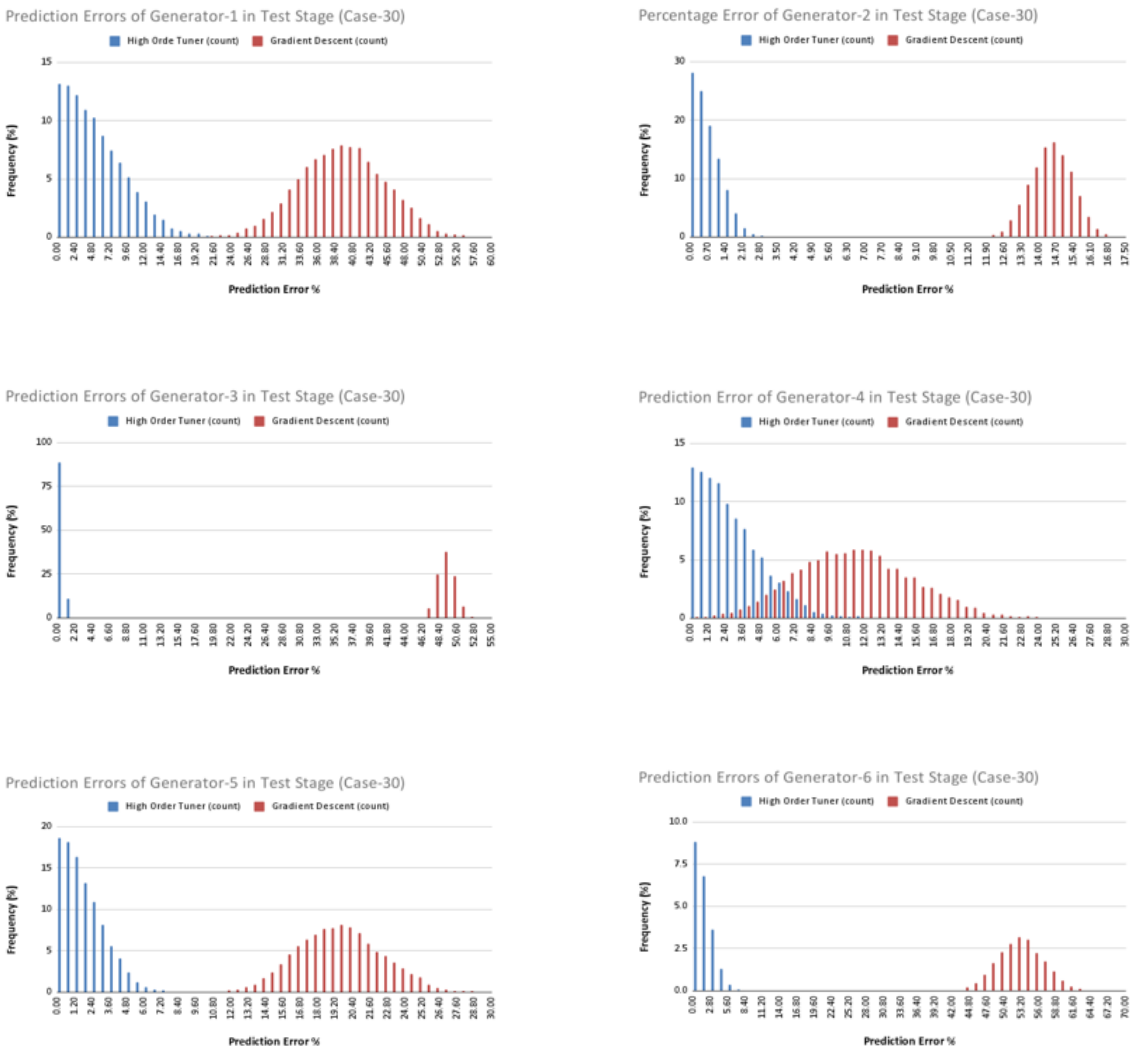


Figure 5-4: IEEE Case-30 Power Generation Prediction Error Distribution from baseline. HT performs better than GD in prediction hypothetically due to its ability to avoid local minimum

IEEE Case-300 (Summarized in Table 5.1)

- **Training Dataset**: 50,000 points, **Validation Dataset:** 10,000 points
- **Test Dataset**: 10,000 points
- $N_{bus} = 300$, $N_{gen} = 69$
- NN Model: Single hidden layer with $N_{neu} = 2048$
- Hyperparamters: HT ($\gamma = 10^{-3}, \beta = 0.09$), GD ($\gamma = 10^{-3}$)

Note here that γ for GD refers to the learning rate or step-size of the algorithm. With the above-mentioned details, the percentage deviation from average optimal reference value obtained in test run was 0.041% in case of HT and 0.043% for GD. The entire dataset consisting of $50,000$ datapoints with a batchsize of 32 was chosen for training. After observing reasonable performance similar to Case-9 and 30, the objective was to deconstruct the training process. Appendix B consists of details of the simulations pertaining to training for IEEE Case-300, that led to some important conclusions.

IEEE-300	Average cost $ per hr	Hyperparameters	No. of Epochs	Training loss
MATPOWER	706322	-	-	-
NN with HT	706612 (0.041%)	$\gamma = 0.1, \beta = 0.9$	50	3×10^{-4}
NN with GD	706625 (0.043%)	$\gamma = 0.1$	50	3×10^{-4}

Table 5.1: IEEE-Case 300 Training metrics comparison with baseline (MATPOWER). Notice that HT is marginally closer to the average baseline cost. Note that all 3 methods generate 100% feasible solutions. The NN-based methods do so because of the variable reduction technique that generates feasible solution

Inferences from simulations for Case-300:

- For varying sizes of datasets and fixed batchsize (32), the number of epochs required to converge for HT is approximately 8 times lesser than Gradient Descent
- For achieving comparable accuracy ($\sim 7 \times 10^{-3}$) in nearly the same number of epochs, (maximum limit set to 300 epochs), the dataset size required for training is 5 times larger for GD (50,000 points) than HT (10,000 points)

This has important implications in data-quality, quantity and consumption of time for training the model for Case-300. We conclude that with HT, not only do we experience an advantage in these aspects, but HT is comparatively much more beneficial than GD for re-training as well.

IEEE Case-1354

- **Training Dataset**: 50,000 points, **Validation Dataset:** 10,000 points

- **Test Dataset**: 10,000 points
- $N_{bus} = 1354$
- $N_{gen} = 260$
- NN Model: Single hidden layer with $N_{neu} = 2048$
- Hyperparamters: HT ($\gamma = 10^{-3}, \beta = 0.09$), GD ($\gamma = 10^{-3}$)

The objective of this simulation was to re-affirm the inferences drawn from previous simulation. With lower training dataset size, *plateauing effect* was observed. Although uncommon, this is referred to the phenomenon of loss saturation for a lot of epochs and then decreasing rapidly after a certain large number of epochs. It was observed that HT was able to navigate across the saturation much more easily compared to gradient descent. This observation made from single-point training and validation loss comparison graphs extends very well into the case when $10,000$ points are chosen for training. Details of this analysis can be found in Appendix B.

Due to complexity of the underlying network and large dimension of the problem, learning is extremely slow and due to practical time constraints, learning was restricted to 400 epochs. Following are some suggestions to overcome this problem:

- Using CUDA associated tensors to speed-up training and performance
- Improving the diversity profile of training dataset: Currently, the training dataset preparation mechanism chosen based on recommendations from [37] generates repetitive data-pattern. This can be avoided by generating data with larger variance than the currently chosen 10%.

Some other recommendations based on training process:

- Overcoming plateauing:A practical way of handling the plateauing effect is also to have cyclic learning rate instead of constant learning rate
- Training the model using a local loss function instead of global loss function is also suggested to overcome plateauing, and also to expedite the training

Overall, despite the observed hurdles, HT's accelerated learning is quite evidently observable compared to Gradient Descent.

5.4 Conclusion & Future Work

Experimental studies confirm the accelerated convergence of HT which can be seen in contrast to the linear convergence of Gradient Descent. This is backed by theoretical guarantees mentioned earlier for convex optimization problems. While the loss landscape is extremely unpredictable and non-convex with DNNs, using a single hidden layer NN and the dual formulation proposed in [42] helps in drawing important insights into hyperparameter tuning in a structured manner, such as impact of increasing dataset size on the problem.

The simulations conducted for Case-1354 point to several possible directions of future work in this direction of learning based methods for solving OPF problems. While DC-OPF is numerically easier to solve than AC-OPF and one might argue that learning-based solvers are an overkill for DC-OPF, structurally, the insights and observations made in DC-OPF can be very easily extended to AC-OPF too, where learning-based techniques have a strong potential. This is evident from the fact that to generate a dataset of 50,000 points for training a Neural Network for AC-OPF by solving the problem repeatedly using MATPOWER takes upto 3-4 days. Here, one can advtantage significantly from HT's accelerated performance.

Another key feature of HT which has not been tested yet in simulations is the incorporation of time-varying normalizing parameter $\mathcal{N}_k$. So far, the normalizing signal has been chosen as the smoothness parameter of the convex loss function. Presence of a normalizing signal that tunes itself based on the changing nature of the loss function has been theoretically explored in depth and can have some rewarding consequences when implemented in machine-learning based settings.

Chapter 6

Lyapunov Theory for Neural Network Optimization

6.1 Lyapunov Theory for optimization

Lyapunov theory has long existed in controls and dynamics literature as an important tool for establishing stability of the dynamic system under consideration [44]. The notion of lyapunov stability summarized in a line would be to find a function V associated to a dynamic system, which shrinks over time, then by analyzing the behavior of V we can comment about the long-term statistics of the system [47]. In the past, Lyapunov theory for optimization has been explored by treating optimization problems as deterministic and stochastic dynamic systems and finding energy-dissipative functions for the same [50, 26]. This has then been utilized for obtaining upper bounds on algorithm convergence for specific classes of functions [46].

By looking at the continuous time representation of any optimization problem solved using gradient based methods such as GD, the translation between decision variables and states can be easily achieved. By treating the decision variables as states of a dynamic system, we propose Lyapunov function for neural network optimization trained using Gradient Descent. This can be used to comment upon the convergence performance of decision variables to the optimal value or critical points regardless of the information available about input regressors (i.e,. they can be time-invariant or

not), which is currently a huge caveat in most approaches in learning theory.

6.1.1 Overview

In this chapter we focus our attention to single hidden layer Linear Neural Networks (LNNs). Linear Neural Networks are simply networks without an activation function. We first study the scalar-version of this network

6.2 Linear Neural Network optimization as a dynamic system

6.2.1 Scalar case

We first look at the scalar version of neural network optimization, to understand the system dynamics and gain information about critical points and minimas of the system. Consider a LNN with input $x(t)$, output $y(t)$ and a single hidden layer, with weights from input to hidden layer described as $u(t)$ and that from hidden to output layer described as $v(t)$. We further define u^*, v^* as the optimal values such that $y(t) = v^*u^*x(t)$ Define:

$$\tilde{u}(t) = u(t) - u^*$$
$$\tilde{v}(t) = v(t) - v^*$$

Loss function (Mean Squared Error loss):

$$L = \frac{1}{2}\|y(t) - v(t)u(t)x(t)\|_2^2 \tag{6.1}$$

Using Gradient Flow for weight update, we can write:

$$\begin{aligned} \dot{v}(t) &= (y(t) - v(t)u(t)x(t)) \cdot u(t)x(t) \\ \dot{u}(t) &= (y(t) - v(t)u(t)x(t)) \cdot v(t)x(t) \end{aligned} \tag{6.2}$$

Loss convergence analysis (scalar case): Assuming there exists atleast one (u^*, v^*) such that $y(t) = v^*u^*x(t)$

$$\begin{aligned}\dot{L}(t) &= -x(t) \cdot (y(t) - v(t)u(t)x(t)) \cdot (u\dot{v} + v\dot{u}) \\ &= -x^2(u^2 + v^2)L(t) \leq 0\end{aligned}$$

Therefore, using Gronwall's lemma, $L(t) \leq -L(0)exp(-\int_0^t x^2(u(s)^2 + v(s)^2)\, ds)$. Readers are referred to [31] for further information about the loss analysis presented above and its extensions to the vector-case.

Notice this approach depends heavily on the assumption that $x(t)$ does not have any dynamics, i.e, $\dot{x}(t) = 0$, and $x(t)$ can be treated as a constant input regressor, making it suitable for the anaysis of supervised learning problems. However, analysing the loss convergence deprives us of commenting on the stability or convergence in the case where $x(t)$ is a time-varying regressor, such as a state of a dynamic system. However, using this toy-case analysis of LNN dynamics, it becomes apparent that we are treating u, v as the states of our dynamic system and are interested in the convergence properties of these weights.

6.2.2 Phase Portait of LNN: Scalar case

Using the gradient flow update in (6.2) we analyse the dynamics of a Linear Neural Network. From the Figure 6-1, we can observe that the parameters (u, v) will converge to a point on the hyperbola $uv = u^*v^*$ everywhere in $\mathbb{R}^2$, *except* on the line $u = -v$. Note that this is under the assumption that $u^*v^* \geq 0$.

On the separatrix $u = -v$: A linear network with weights (u_0, v_0) initialized on this line will converge to the sub-optimal saddle point $(0, 0)$ instead. This explains the basis behind the popular theory of imbalance in weight initialization [31, 5], as this would amplify the chances of avoiding saddle points in a higher dimensional network. Furthermore, it can be shown that $(u^2 - v^2)$ is a time-invariant set, and the difference of square of weights is preserved throughout the training. This serves the intuition behind proposing the Lyapunov Candidate function for a scalar neural

network $V = \frac{1}{2}\|v^*u^* - vu\|^2$. For such a function V, it can be easily shown that $\dot{V} \leq 0$ and hence, V is a valid lyapunov function for this linear neural network. We now extend the analysis to a higher dimensional Linear Neural Network.

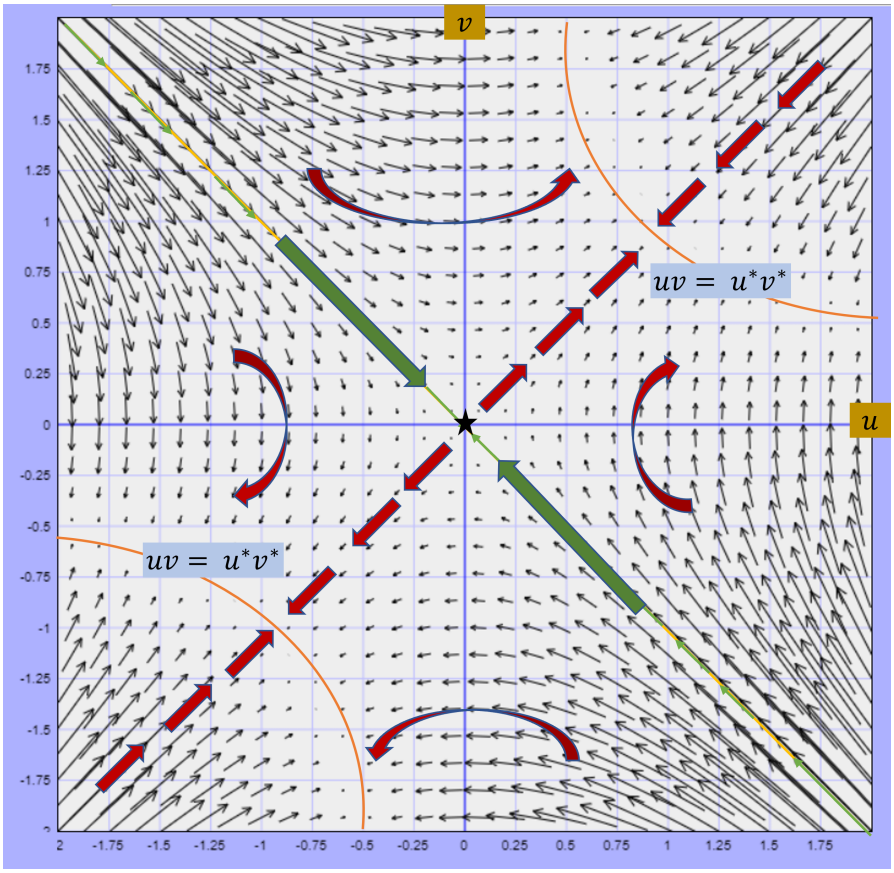


Figure 6-1: Phase portait of a scalar representation of Linear Neural Network (assuming $v^*u^* > 0$, phase portait will be a mirror image for $v^*u^* < 0$). Notice that all points eventually converge to a point lying on the hyperbola $vu = v^*u^*$ except for when initialized on the line $u = -v$, in which case it converges to the saddle point $(0, 0)$

6.3 Lyapunov Theory for Linear Neural Networks

We assume that the true system is given by:

$$y = W_2^*(W_1^*x + B_1^* \cdot 1) + B_2^* \cdot 1 \tag{6.3}$$

where $W_2^* \in \mathbb{R}^{m\times h}, W_1^* \in \mathbb{R}^{h\times d}, B_1^* \in \mathbb{R}^h, B_2^* \in \mathbb{R}^m$ are true parameters. The problem is to determine estimates for all of the parameters using a linear neural network of the form

$$\hat{y} = W_2(W_1 x + B_1 \cdot 1) + B_2 \cdot 1 \tag{6.4}$$

so that the estimated output $\hat{y}$ converges to y. We propose some transformations to accommodate the presence of biases in the network as: $\overline{x} \in \mathbb{R}^{d+1}$

$$\overline{x} = \begin{bmatrix} x \\ 1 \end{bmatrix} \tag{6.5}$$

$\overline{W_1} \in \mathbb{R}^{h\times d+1}$:

$$\overline{W_1} = \begin{bmatrix} W_1 & B_1 \end{bmatrix} \tag{6.6}$$

Thus, $W_1 x + B_1 = \overline{W}_1 x$. We further transform $\overline{W}_1$ to $\overline{W}_{1u} \in \mathbb{R}^{h+1\times d+1}$ as:

$$\overline{W}_{1u} = \begin{bmatrix} \overline{W}_1 \\ \frac{\mathbb{1}}{(\sum_{i=1}^{d} x_i)+1} \end{bmatrix} \tag{6.7}$$

Thus, $\overline{W}_{1u} \cdot \overline{x} \in \mathbb{R}^{h+1}$

$$\overline{W}_{1u} \cdot \overline{x} = \begin{bmatrix} W_1 x + B_1 \\ 1 \end{bmatrix} \tag{6.8}$$

Now we propose a transformation to W_2 as $\overline{W}_2 \in \mathbb{R}^{m\times h+1}$:

$$\overline{W}_2 = \begin{bmatrix} W_2 & B_2 \end{bmatrix} \tag{6.9}$$

The transformations $\overline{W}_{1u}, \overline{W}_2$ help us write (6.4) as:

$$\hat{y} = \overline{W}_2 \overline{W}_{1u} x \tag{6.10}$$

Similarly, true value of parameters can be written as:

$$y = \overline{W}_2^* \overline{W}_{1u}^* x \tag{6.11}$$

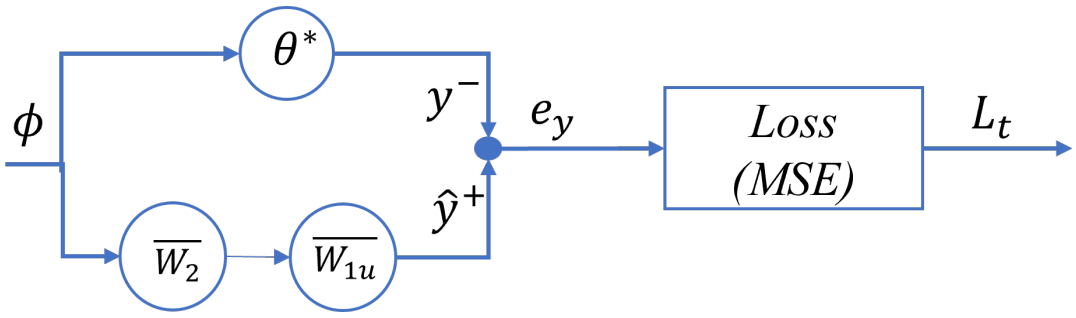


Figure 6-2: Block Diagram explaining the Linear Neural Network architecture. We assume existence of W_2^*,W_1^*, B_1^*, B_2^* such that $\theta^* = \overline{W}_2^*\overline{W}_{1u}^*$ as shown in (6.11)

This allows us to write the loss function as:

$$L = \frac{1}{2}\|y - \overline{W}_2\overline{W}_{1u}x\|^2 \tag{6.12}$$

Using the gradient update law:

$$\dot{\overline{W}}_2 = D\overline{x}\overline{x}^T\overline{W}_{1u}^T \tag{6.13}$$

$$\dot{\overline{W}}_1 = W_2^T D\overline{x}\overline{x}^T \tag{6.14}$$

Using $\dot{\overline{W}_1}$ we can write:

$$\dot{\overline{W}}_{1u} = \begin{bmatrix} \dot{\overline{W}_1} \\ 0 \end{bmatrix} \tag{6.15}$$

Therefore, $\overline{W}_2\dot{\overline{W}}_{1u} = W_2\dot{\overline{W}}_1$.

Theorem 6.3.1 *Under gradient flow given by* (6.13) *and* (6.14)*, and a mean sqaured error loss function given by* (6.12)*, the following function is a valid lyapunov function* $\mathbb{V} = \frac{1}{2}\|\overline{W}_2^*\overline{W}_{1u}^* - \overline{W}_2\overline{W}_{1u}\|_F^2$

Proof: **Note** The proof makes use of several transformations and equations mentioned in this Section, hence for ease of viewing it has been stated here directly.

Define $D = \overline{W}_2^*\overline{W}_{1u}^* - \overline{W}_2\overline{W}_{1u}$, therefore, we have:

$$\dot{\mathbb{V}} = tr(D^T\dot{D}) \tag{6.16}$$

$$\dot{D} = -(\dot{\overline{W}}_2\overline{W}_{1u} + \overline{W}_2\dot{\overline{W}}_{1u}) \tag{6.17}$$

Using, (6.17), (6.13) and (6.14), $\dot{\mathbb{V}}$ can be written as:

$$\begin{aligned}
\dot{\mathbb{V}} &= -tr(D^T(\dot{\overline{W}}_2\overline{W}_{1u} + \overline{W}_2\dot{\overline{W}}_{1u})) \\
&= -tr(D^T\dot{\overline{W}}_2\overline{W}_{1u}) - tr(D^T\overline{W}_2\dot{\overline{W}}_{1u})) \\
&= -tr(D^T\dot{\overline{W}}_2\overline{W}_{1u}) - tr(D^TW_2\dot{\overline{W}}_1)) \\
&= -tr(D^TD\overline{x}\overline{x}^T\overline{W}_{1u}^T\overline{W}_{1u}) - tr(D^TW_2W_2^TD\overline{x}\overline{x}^T)
\end{aligned} \tag{6.18}$$

$tr(D^TD\overline{x}\overline{x}^T\overline{W}_{1u}^T\overline{W}_{1u})$: Using the properties of trace operator we can write:

$$tr(D^TD\overline{x}\overline{x}^T\overline{W}_{1u}^T\overline{W}_{1u}) = \overline{x}^TD^TD\overline{W}_{1u}^T\overline{W}_{1u}\overline{x} \tag{6.19}$$

This is valid for any vector $\overline{x}$, hence, we have commutativity between $\overline{W}_{1u}^T\overline{W}_{1u}D^TD$ and $D^TD\overline{W}_{1u}^T\overline{W}_{1u}$. For 2 symmetric PSD matrices A, B, the product is PSD *iff* $(AB)^T = AB$. Applying this, we have $A = D^TD$ and $B = \overline{W}_{1u}^T\overline{W}_{1u}$ which is are symmetric PSD matrices due to construction. Therefore, $(AB)^T = \overline{W}_{1u}^T\overline{W}_{1u}D^TD$, and using $\overline{W}_{1u}^T\overline{W}_{1u}D^TD = D^TD\overline{W}_{1u}^T\overline{W}_{1u}$ we get $(AB)^T = AB$, hence the matrix $\overline{W}_{1u}^T\overline{W}_{1u}D^TD$ is a PSD.

Therefore, for any vector $\overline{x}$, using definition of PSD matrices we have:

$$\overline{x}^T\overline{W}_{1u}^T\overline{W}_{1u}D^TD\overline{x} \geq 0 \tag{6.20}$$

Similarly, we can write

$$tr(D^TW_2W_2^TD\overline{x}\overline{x}^T) \geq tr(D^TW_2W_2^TD)\lambda_{min}(\overline{x}\overline{x}^T) \tag{6.21}$$

Since $\overline{x}\overline{x}^T$ is a PSD, $\lambda_{min}(\overline{x}\overline{x}^T) \geq 0$, and $D^TW_2W_2^TD$ can be written as $(W_2^TD)^T(W_2^TD)$ and is hence a dyad, therefore a PSD. Using trace operator properties, we can write $tr(D^TW_2W_2^TD\overline{x}\overline{x}^T) = \overline{x}^TD^TW_2W_2^TD\overline{x}$

$\overline{x}^TD^TW_2W_2^TD\overline{x} \geq 0$ since $D^TW_2W_2^TD$ is a dyad. Therefore,

$$tr(D^T W_2 W_2^T D\overline{x}\overline{x}^T) = \overline{x}^T D^T W_2 W_2^T D\overline{x} \geq 0 \tag{6.22}$$

Therefore, plugging (6.20) and (6.22) in the equation (6.18), we get:

$$\dot{\mathbb{V}} = -(\overline{x}^T D^T W_2 W_2^T D\overline{x} + \overline{x}^T \overline{W}_{1u}^T \overline{W}_{1u} D^T D\overline{x}) \leq 0 \tag{6.23}$$

Remark 1: Theorem 6.3.1 implies that the product of weights $\overline{W}_2\overline{W}_{1u}$ remains bounded at all times and the weights converge to the value where $\dot{\mathbb{V}} = 0$. Note that for $h \geq d$, and $D \neq 0$, the rank$(W_2^T D) = d$, which allows us to use rank-nullity theorem to state that $W_2^T D\overline{x} = 0$ *iff* $\overline{x} = 0$. Since it was already established that $D^T W_2 W_2^T D$ is PSD, for $h \geq d$ we can conclude that $D^T W_2 W_2^T D$ is Positive Definite. Therefore, $\dot{\mathbb{V}} < 0$ if $\overline{x}^T D^T W_2 W_2^T D\overline{x} > 0$. Therefore, $\dot{\mathbb{V}} = 0$ only if $D = 0$, i.e, $\overline{W}_2\overline{W}_{1u} = \overline{W}_2^*\overline{W}_{1u}^*$. Therefore it follows that except for the degenerate case where $\overline{W}_{1u} = \overline{W}_2 = 0$, for all other initial conditions, when the weights $\overline{W}_2$ and $\overline{W}_{1u}$ are adjusted using (6.13) and (6.14), all weights will converge to the set $D = 0$. It should be noted that on $D = 0$, the loss function L given by (6.12) is at its global minimum of zero. This shows that the Lyapunov function $\mathbb{V}$ in Theorem 6.3.1 is global. Also note that from (6.13) and (6.14), $D = 0$ implies that $\dot{\overline{W}}_2 = 0, \dot{\overline{W}}_1 = 0$. Therefore $D = 0$ is an invariant set. Hence, the largest invariant set obtained from the Lyapunov function corresponds to the region of optimal product where $\overline{W}_2\overline{W}_{1u} = \overline{W}_2^*\overline{W}_{1u}^*$.

Remark 2: Note that the Lyapunov function here is $\mathbb{V} = \frac{1}{2}\|\overline{W}_2^*\overline{W}_{1u}^* - \overline{W}_2\overline{W}_{1u}\|_F^2$, which can also be written as $\frac{1}{2}tr\big((\overline{W}_2^*\overline{W}_{1u}^* - \overline{W}_2\overline{W}_{1u})^T(\overline{W}_2^*\overline{W}_{1u}^* - \overline{W}_2\overline{W}_{1u})\big)$. By defining $D = \overline{W}_2^*\overline{W}_{1u}^* - \overline{W}_2\overline{W}_{1u}$, Lyapunov function can be equivalently written as $\mathbb{V} = \frac{1}{2}tr(D^T D)$. This is similar to the scalar case where $V = \frac{1}{2}\|v^*u^* - vu\|^2$ where $tr(d^T d) = d^T d$ for $d = v^*u^* - vu$. The fairly symmetric structure suggests that Theorem 6.3.1 can be extended in a straightforward manner to a linear network with arbitrary number of layers.

6.4 Numerical Simulations

To further validate 6.3.1, we present a numerical simulation with true parameter represented by (6.3), with 2-dimensional input X, and 3-dimensional output y. Therefore, $m = 3, d = 2$. We choose $h = 6$, and aim to estimate the weights W_1 and W_2. From the above analysis, we can conclude that identifiability of the optimal weights is not guaranteed under the given setup, however, $\mathbb{V} = \frac{1}{2}\|\overline{W}_2^*\overline{W}_{1u}^* - \overline{W}_2\overline{W}_{1u}\|_F^2$ is a positive and monotonically decreasing function.

6.4.1 Dataset design

We choose a 2-dimensional dataset of 1000 points drawn from randomly between $(0, 1)$ distribution to represent x (Figure 6-3)and design the true output y as:

$$y = W_2^* W_1^* x \tag{6.24}$$

Note that for simplicity we assume biases to be zero both in design and training. The optimal weights W_2, W_1 are chosen as:

$$W_1^* = \begin{bmatrix} 0.1 & 0.2 & 0.6 & 0.3 & 0.4 & 0.5 \\ -0.1 & -0.3 & -0.6 & -0.4 & -0.8 & -10 \end{bmatrix}^T \tag{6.25}$$

$$W_2^* = \begin{bmatrix} 0.1 & 0.2 & 0.6 & 0.3 & 0.4 & 0.5 \\ -0.1 & -0.3 & -0.6 & -0.4 & -0.8 & -10 \\ 0 & -0.7 & -0.2 & -1.0 & 2 & 0 \end{bmatrix} \tag{6.26}$$

6.4.2 Design of Neural Network

We design the neural network using a single hidden layer as discussed earlier, with the estimated output $\hat{y}$ of the network given by (6.4). We perform training of the network for few different widths of the network, i.e., different values of h in the hidden

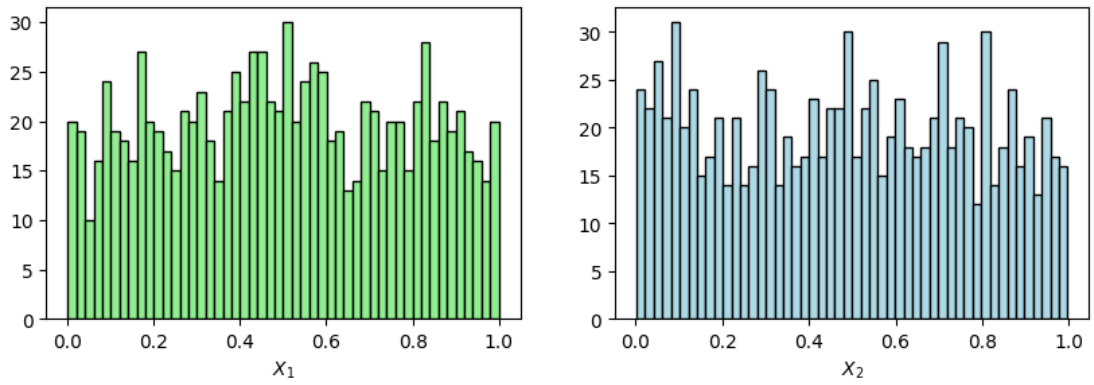


Figure 6-3: Distribution of input data x with x_1, x_2 being the vectors along the 2-dimensions of input data

layer. The batch-size is chosen as 1, to ensure consistency with the vector structure of x in Theorem 6.3.1

6.4.3 Results

Figures 6-4,6-5 show the function $\mathbb{V}(t)$, where t denotes the epochs of the simulation. Notice that the trend is monotonically decreasing, thus asserting the validity of the proposed Lyapunov function. Also note that in the construction and proof of Theorem 6.3.1, there is flexibility of working with any width of the hidden layer. Therefore, in theory, a LNN of arbitrary width will try to converge to the optimal solution $W_2^* W_1^*$.

For the specific case of $h = 6$, we observe the values that weights converge to at the end of 100 epochs, given by W_1^c and W_1^c:

$$W_1^c = \begin{bmatrix} -0.0479 & -0.0671 & -0.0787 & -0.0895 & -0.0532 & -0.0564 \\ 1.1384 & 1.1531 & 1.1190 & 1.1073 & 1.1228 & 1.1103 \end{bmatrix}^T \tag{6.27}$$

$$W_2^c = \begin{bmatrix} -9.1480 \times 10^{-2} & -9.8865 \times 10^{-2} & -7.5373 \times 10^{-2} & -6.6551 \times 10^{-2} & -8.4185 \times 10^{-2} & -6.6059 \times 10^{-2} \\ 1.4982 & 1.5004 & 1.5062 & 1.5257 & 1.4837 & 1.4878 \\ -2.5457 \times 10^{-2} & -3.1113 \times 10^{-2} & 2.6396 \times 10^{-4} & 2.0192 \times 10^{-2} & -1.4885 \times 10^{-2} & -1.1681 \times 10^{-2} \end{bmatrix} \tag{6.28}$$

Clearly, $W_1^c \neq W_1^*, W_2^c \neq W_2^*$. However, if we look at the product $W_2^c \cdot W_1^c$:

$$W_2^c \cdot W_1^c = \begin{bmatrix} 3.1108 \times 10^{-2} & -5.4405 \times 10^{-1} \\ -5.9034 \times 10^{-1} & 10.128 \\ 2.9320 \times 10^{-3} & -7.1888 \times 10^{-2} \end{bmatrix}^T \tag{6.29}$$

This is very close to the product $W_2^* W_1^*$:

$$W_2^* \cdot W_1^* = \begin{bmatrix} 0.0910 & -0.5870 \\ -0.5870 & 10.1260 \\ 0.0240 & -0.0870 \end{bmatrix}^T \tag{6.30}$$

The marginal difference in values indicates that after fine-tuning the step size and running the simulation for more epochs, the value $W_2^c W_1^c$ will converge to the optimal value $W_2^* W_1^*$.

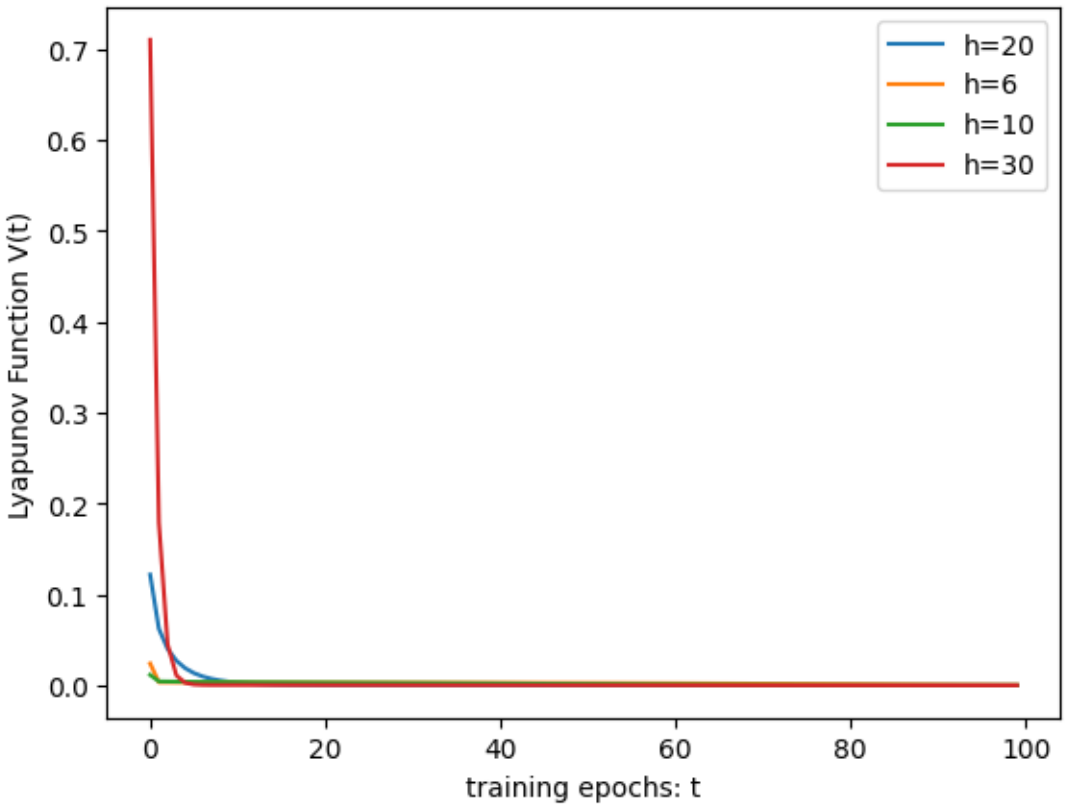


Figure 6-4: Lyapunov Function $\mathbb{V}(t)$ for different values of h under gradient update ($lr = 5 \times 10^{-3}$). Notice that increasing h, i.e, overparametrizing the network leads to smoother convergence to $W_2^* W_1^*$, as is often seen in practice [13]

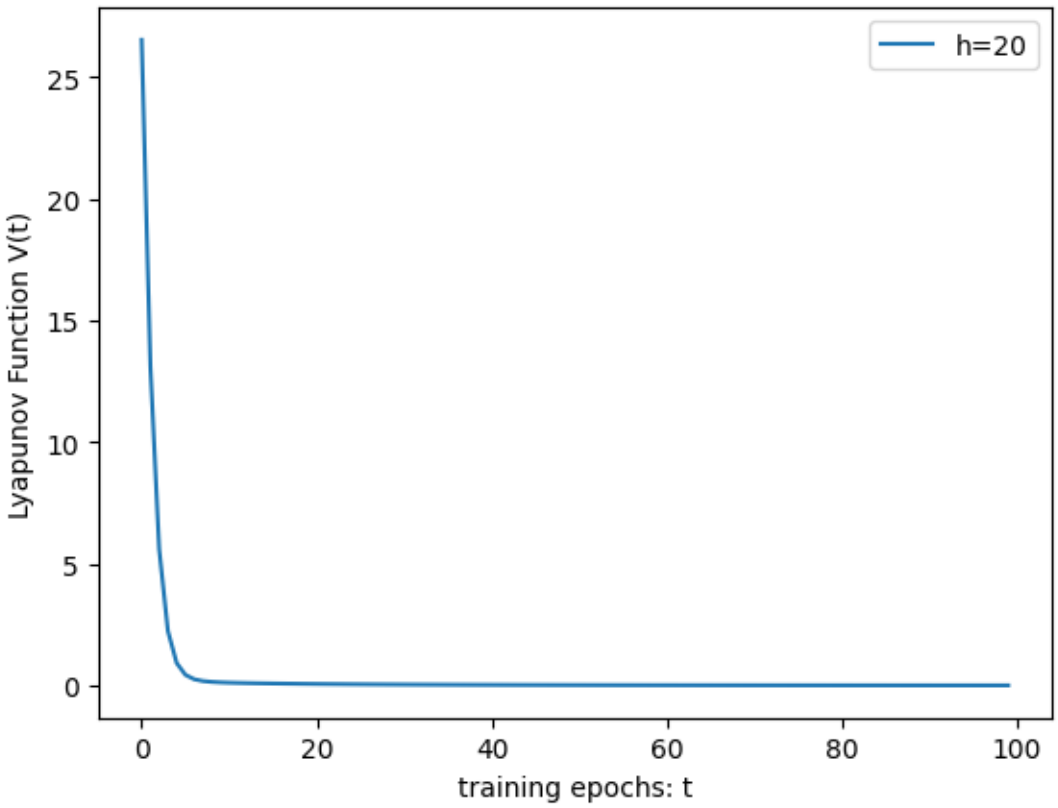


Figure 6-5: Lyapunov Function $\mathbb{V}(t)$ for $h = 20$ under gradient update with a slower learning rate to clearly show the monotonic behavior of the lyapunov function ($lr = 5 \times 10^{-4}$).

Chapter 7

Conclusions and Future Work

The central focus of this thesis is the clever adoption of accelerated learning techniques prevalent in controls and dynamics literature, and their application to open-ended problems in optimization and machine learning. We aim to contribute towards optimization and learning theory with stability guarantees by formulating lyapunov functions for different accelerated algorithms and providing feasibility guarantees wherever needed.

Specifically, we make use of a recently proposed accelerated method called High Order Tuner (HT) to impart acceleration for optimization problems, which plays a major role in avoiding saddle points in nonconvex optimization, as evident in the numerical simulations presented in Chapter 4 and 5.

In Chapter 3 and 4, we present various novel methods for constrained optimization using HT which guarantee feasibility and convergence. We also make the line between convexity and non-convexity clearer through Propositions providing analytical guarantees for feasibility of constraints while simultaneously retaining convexity of objective function. By providing Theorems which prove stability of performance of HT in these cases, we utilize fast learning to solve these problems. Scope of future research within this work pertains to establishing theoretical guarantees with the proposed projection operator which is introduced in the last section of Chapter 4.

In Chapter 5, we combine these techniques to present the application of HT to solve a constrained optimization problem, DC-OPF, which is inherently convex. We

leverage Neural network training for this purpose, and use the tools for constraint satisfaction provided in the earlier chapters. This numerical study allows us to appreciate the importance of fast learning in non-convex optimization and the challenges faced in this process. This Chapter is a dedicated study of the various selections backed by theoretical guidelines to be made in a general study involving Neural Network training. We are able to show that by appropriately tuning the Neural Network, with accelerated methods we are able to avoid local minima in loss landscape, while simultaneously avoiding overfitting. It is a challenging but extremely important direction of possible future work to consolidate these observations in the form of theoretical results.

In Chapter 5 we observe that the lack of guarantees in non-convex optimization inhibits us from fully exploiting HT's capability in learning. We therefore steer our attention in Chapter 6 to deconstruct the dynamics of neural networks.

Chapter 6 provides a novel analysis of Linear Neural Network stability in Lyapunov sense. There is an immense scope of future research in this direction by formulating a similar argument of stability for Neural Networks with activation functions and regularization. Additionally, it will be interesting to compare the weight trajectory under regularization with min-norm solutions which are widely popular in the literature [31, 51, 23].

Appendix A

Stability and Convergence proofs

A.1 Propositions and Theorems from Chapter 3

Proposition 3.1.1 For the equality-constrained convex optimization problem (3.2), assume f is a $\overline{L}$-smooth convex function, and let $\overline{M} = \sqrt{1 + \|P\|^2}(1 + \|P\|)\overline{L}$. Then l is $\overline{M}$-smooth convex.

Proof Convexity of l follows in a straightforward manner from the definitions of l and p in (3.3b) and (3.4) respectively, and the convexity of f. For the $\overline{M}$-smoothness, using the chain rule, we have

$$\frac{dl}{d\theta} = \frac{\partial l}{\partial \theta} + \frac{\partial l}{\partial p(\theta)} \frac{\partial p(\theta)}{\partial \theta}.$$

Hence

$$\nabla l = \nabla f_{1:m} + P^T \nabla f_{m+1:n}.$$

Let us consider the gradient of l at $\theta_1, \theta_2 \in \mathbb{R}^m$ and examine the Lipschitz constant

of l.

$$\begin{aligned}\frac{\|\nabla l(\theta_1) - \nabla l(\theta_2)\|}{\|\theta_1 - \theta_2\|} \leq & \frac{\|\nabla f_{1:m}(\theta_1) - \nabla f_{1:m}(\theta_2)\|}{\|\theta_1 - \theta_2\|} \\ & + \frac{\|P^T(\nabla f_{m+1:n}(\theta_1) - \nabla f_{m+1:n}(\theta_2))\|}{\|\theta_1 - \theta_2\|}\end{aligned}$$

Since $\|\nabla f_{i:j}\| \leq \|\nabla f\|$ for all i, j, we have

$$\frac{\|\nabla l(\theta_1) - \nabla l(\theta_2)\|}{\|\theta_1 - \theta_2\|} \leq \frac{(1 + \|P\|)\|\nabla f(x_1) - \nabla f(x_2)\|}{\|\theta_1 - \theta_2\|} \tag{A.1}$$

To write the denominator of the above expression in terms of x_1 and x_2, remember that

$$\|\theta_1 - \theta_2\|^2 + \|P\theta_1 - P\theta_2\|^2 = \|x_1 - x_2\|^2.$$

Using properties of the norm,

$$\begin{aligned}\|\theta_1 - \theta_2\|^2 + \|P\|^2\|\theta_1 - \theta_2\|^2 \geq & \|x_1 - x_2\|^2 \\ \|\theta_1 - \theta_2\| \geq & \frac{\|x_1 - x_2\|}{\sqrt{1 + \|P\|^2}}.\end{aligned} \tag{A.2}$$

Using (A.1) and (A.2), we have

$$\begin{aligned}\frac{\|\nabla l(\theta_1) - \nabla l(\theta_2)\|}{\|\theta_1 - \theta_2\|} \leq & \frac{\sqrt{1 + \|P\|^2}(1 + \|P\|)\|\nabla f(x_1) - \nabla f(x_2)\|}{\|x_1 - x_2\|} \\ \leq & \sqrt{1 + \|P\|^2}(1 + \|P\|)\overline{L},\end{aligned}$$

where the last inequality follows from the $\overline{L}$-smoothness property of f.

Corollary 3.1.1 For the equality-constrained convex optimization problem (3.2), assume f is a $\overline{L}$-smooth and μ-strongly convex function. Then l is $\overline{M}$-smooth and μ-strongly convex, where $\overline{M}$ is defined in equation (3.5).

Proof Consider $\theta_1, \theta_2 \in \mathbb{R}^m$. Now consider $x_1, x_2 \in \mathbb{R}^n$ defined as $x_1 = [\theta_1^T \;\; p(\theta_1)^T]^T$

and $x_2 = [\theta_2^T \;\; p(\theta_2)^T]^T$. Then from the properties of f, it follows that

$$\begin{aligned} f(\lambda x_1 + (1-\lambda)x_2) \leq & \lambda f(x_1) + (1-\lambda) f(x_2) \\ & - \frac{1}{2}\mu\lambda(1-\lambda)\|x_1 - x_2\|^2, \end{aligned}$$

for all $\lambda \in (0,1)$. Using the fact that $\|\theta_1 - \theta_2\|^2 \leq \|x_1 - x_2\|^2$, we have

$$\begin{aligned} f(\lambda x_1 + (1-\lambda)x_2) \leq & \lambda f(x_1) + (1-\lambda) f(x_2) \\ & - \frac{1}{2}\mu\lambda(1-\lambda)\|\theta_1 - \theta_2\|^2. \end{aligned}$$

Rest of the proof follows from the definition of l and the proof of Proposition 3.1.1.

Theorem 3.2.1 If the objective function f is $\overline{L}$-smooth convex, then with $0 < \beta < 1$ and $0 < \gamma < \frac{\beta(2-\beta)}{8+\beta}$, the sequence of iterates $\{\theta_k\}$ generated by Algorithm 1 satisfy $\lim_{k\to\infty} l(\theta_k) = l(\theta^*)$, where $l(\theta^*) = f([\theta^{*T} \quad p(\theta^*)^T]^T)$ is the optimal value of (3.2).

Proof From Proposition 3.1.1, $\overline{L}$-smoothness of the objective function f implies that the loss function l is $\overline{M}$-smooth and convex. Rest of the proof follows from [32, Theorem 2].

Proposition 3.3.1 Assume that there exists a convex set $\Omega_n \in \mathbb{R}^n$ such that the hypotheses of Lemma 3.3.1 are satisfied on Ω_n. Let

$$\Omega_m = \{\theta \mid \theta = x_{1:m}, x \in \Omega_n\}.$$

If either of the following conditions is satisfied:

1. $\nabla \mathcal{L}(x) \geq 0$ for all $x \in \Omega_n$, and p is convex on Ω_m,
2. $\nabla \mathcal{L}(x) \leq 0$ for all $x \in \Omega_n$, and p is concave on Ω_m,

then l is convex on Ω_m.

Proof We present here the arguments for only condition (i). The ensuing treatment easily generalizes to condition (ii). It is immediate to see that Ω_m is convex.

Consider $\theta_1, \theta_2 \in \Omega_m$. Since $p(\theta)$ is convex, we have

$$p(\lambda\theta_1 + (1-\lambda)\theta_2) \leq \lambda p(\theta_1) + (1-\lambda)p(\theta_2),$$

and it follows that

$$\begin{bmatrix} \lambda\theta_1 + (1-\lambda)\theta_2 \\ p(\lambda\theta_1 + (1-\lambda)\theta_2) \end{bmatrix} \leq \lambda \begin{bmatrix} \theta_1 \\ p(\theta_1) \end{bmatrix} + (1-\lambda) \begin{bmatrix} \theta_2 \\ p(\theta_2) \end{bmatrix}.$$

Since $\mathcal{L}$ is nondecreasing, it follows that

$$\begin{aligned} &\mathcal{L}\left(\begin{bmatrix} \lambda\theta_1 + (1-\lambda)\theta_2 \\ p(\lambda\theta_1 + (1-\lambda)\theta_2) \end{bmatrix} \right) \\ \leq &\mathcal{L}\left(\lambda \begin{bmatrix} \theta_1 \\ p(\theta_1) \end{bmatrix} + (1-\lambda) \begin{bmatrix} \theta_2 \\ p(\theta_2) \end{bmatrix} \right). \end{aligned} \tag{A.3}$$

Moreover, it follows from Lemma 3.3.1 that $\mathcal{L}$ is convex, and we have

$$\begin{aligned} &\mathcal{L}\left(\lambda \begin{bmatrix} \theta_1 \\ p(\theta_1) \end{bmatrix} + (1-\lambda) \begin{bmatrix} \theta_2 \\ p(\theta_2) \end{bmatrix} \right) \\ \leq &\lambda\mathcal{L}\left(\begin{bmatrix} \theta_1 \\ p(\theta_1) \end{bmatrix} \right) + (1-\lambda)\mathcal{L}\left(\begin{bmatrix} \theta_2 \\ p(\theta_2) \end{bmatrix} \right). \end{aligned} \tag{A.4}$$

Combining the inequalities (A.3) and (A.4), we get

$$\begin{aligned} &\mathcal{L}\begin{pmatrix} \lambda\theta_1 + (1-\lambda)\theta_2 \\ p(\lambda\theta_1 + (1-\lambda)\theta_2) \end{pmatrix} \\ \leq &\lambda\mathcal{L}\left(\begin{bmatrix} \theta_1 \\ p(\theta_1) \end{bmatrix} \right) + (1-\lambda)\mathcal{L}\left(\begin{bmatrix} \theta_2 \\ p(\theta_2) \end{bmatrix} \right). \end{aligned}$$

And from the definition of the modified loss function, it follows that

$$l(\lambda\theta_1 + (1-\lambda)\theta_2) \leq \lambda l(\theta_1) + (1-\lambda)l(\theta_2),$$

completing the proof.

Proposition 3.3.2 Following from Proposition 3.3.1, assuming f and h are convex on a given set $\bar{\Omega}_n \subseteq \mathbb{R}^n$, and h_i is twice differentiable $\forall\ i = 1, ..., n-m$, it follows that:

1. if $\nabla_p h(x) < 0$ for $x \in \bar{\Omega}_n$ then $p(\theta)$ is convex on Ω_m
2. if $\nabla_p h(x) > 0$ for $x \in \bar{\Omega}_n$ then $p(\theta)$ is concave on Ω_m

Additionally, if h is linear in z and convex in θ, then condition (i) follows.

Proof We prove condition (i), similar arguments can be extended to prove condition (ii). Noting that

$$h([\theta^T \quad p(\theta)^T]^T) = 0, \tag{A.5}$$

and applying the chain rule and differentiating (A.5) along the manifold $z = p(\theta)$ twice, for $1 \leq i \leq m$ and $1 \leq j \leq n-m$ we get:

$$\underbrace{\frac{\partial^2 h_i}{\partial \theta^2}}_{I} + \underbrace{\frac{\partial h_i}{\partial z_j}\frac{\partial^2 z_j}{\partial \theta^2}}_{II} + \underbrace{\frac{\partial^2 h_i}{\partial z_j^2}\frac{\partial z_j}{\partial \theta}\left(\frac{\partial z_j}{\partial \theta}\right)^T}_{III} = 0 \tag{A.6}$$

Since h_i is convex on $\Omega_n\ \forall\ i \in \mathbb{N}$, it follows:

1. I: $\frac{\partial^2 h_i}{\partial \theta^2}$ is a positive semi-definite matrix
2. III: $\frac{\partial z_j}{\partial \theta}\left(\frac{\partial z_j}{\partial \theta}\right)^T$ is a symmetrical dyad, i.e., PSD matrix multiplied by $\nabla^2_{z_j} h_i \geq 0$, hence III is a PSD matrix.

For (A.6) to be valid, II has to be a negative semi-definite matrix, since it is the negated sum of positive-semi definite matrices. From condition (i) we have $\frac{\partial h_i}{\partial z_j} < 0$ since $z = p(\theta)$. Thus, $\frac{\partial^2 z_j}{\partial \theta^2}$ is PSD. Therefore, $\frac{\partial^2 p_j(\theta)}{\partial \theta^2}$ is PSD, hence $p_j(\theta)$ is convex $\forall \theta \in \Omega_m, \forall j = 1, ..., n-m$.

Theorem 3.3.2 If the objective function f and the equality constraint h are convex over a set Ω_n, In addition, with $0 < \beta < 1$, $0 < \gamma < \frac{\beta(2-\beta)}{8+\beta}$, and $\theta_0 \in \Omega_m$,

where Ω_m is defined in (3.10), if the sequence of iterates $\{\theta_k\}$ generated by Algorithm 1 satisfy $\{\theta_k\} \in \Omega_m$, then $\lim_{k\to\infty} l(\theta_k) = l(\theta^*)$, where $l(\theta^*) = f([\theta^{*T} \quad p(\theta^*)^T]^T)$ is the optimal value of (3.8)

Proof Convexity of the loss function l follows from Proposition 3.3.1. Moreover, since $\{\theta_k\} \in \Omega_m$ for all k, there exists a constant $\overline{S}$ such that

$$\|\nabla l(\theta_1) - \nabla l(\theta_2)\| \leq \overline{S}\|\theta_1 - \theta_2\|,$$

for all $\theta_1, \theta_2 \in \Omega_m$. Rest of the proof follows from [32, Theorem 2].

A.2 Propositions and Theorems from Chapter 4

Proposition 4.2.1 For a given compact set I and convex loss function $l(\theta)$ if there exist θ_1, θ_2 such that $l(\theta_1) = l(\theta_2)$, then $\theta^* \in I$.

Proof For a scalar case, i.e., $\Omega_m \subset \mathbb{R}$, Rolle's theorem can be applied to the function l being continuous and differentiable. For a subset $[\theta_1, \theta_2] \subset I$, such that $l(\theta_1) = l(\theta_2)$, by Rolle's Theorem, there exists a $\hat{\theta} \in [\theta_1, \theta_2]$ such that $\nabla l(\hat{\theta}) = 0$. Since l is differentiable and convex, $\nabla l(\hat{\theta}) = 0 \iff \hat{\theta} = \theta^*$. This can be extended to the general case where $m \geq 1, \Omega_m \subset \mathbb{R}^m$, using the vector-version of Rolle's theorem, cf. [19].

Proposition 4.2.2 Consider Algorithm 3, for a given $k \in \mathbb{N}$, if $\theta_k, \nu_k \in I$, there exist real numbers $a_k > 0$ and $b_{k+1} > 0$ such that $\overline{\theta}_k, \nu_{k+1} \in I$. Consequently, for $0 < \beta \leq 1$, and $\theta_0, \nu_0 \in I$, Algorithm 3 guarantees $\theta_k, \overline{\theta}_k, \nu_k \in I$ for all values of $k \in \mathbb{N}$.

Proof We first provide conditions for the selection of a_k.

For a given $\theta_k \in \mathbb{R}^m$ and $i \in \mathbb{N}$, consider $\theta_k^i \in \mathbb{R}$ such that $\theta_k^i \in [\theta_{min}^i, \ \theta_{max}^i]$. There are two possible cases:

1. $\theta_k^i > \theta^{*i} \iff \nabla_i l(\theta_k) > 0$

2. $\theta_k^i < \theta^{*i} \iff \nabla_i l(\theta_k) < 0$

For case (ii), using **Step-6** of Algorithm 3 we have,

$$\overline{\theta}_k^i = \theta_k^i + a_k \frac{\gamma\beta|\nabla_i l(\theta_k)|}{\mathcal{N}_k} \tag{A.7}$$

For $a_k > 0$, $\overline{\theta}_k^i > \theta_{min}^i$ in (A.7). We need to ensure that $\overline{\theta}_k^i < \theta_{max}^i$ for all i. Therefore, we must ensure:

$$\theta_k^i + a_k \frac{\gamma\beta|\nabla_i l(\theta_k)|}{\mathcal{N}_k} \leq \theta_{max}^i \quad \forall i. \tag{A.8}$$

Inequality (A.8) would be true if $a_k \leq \hat{a}_k$, where

$$\hat{a}_k = \min_{i\in\{1,\ldots,m\}} \frac{(\theta_{max}^i - \theta_k^i)\mathcal{N}_k}{\gamma\beta|\nabla_i l(\theta_k)|}. \tag{A.9}$$

Similarly, for case (i), we get the following inequality criteria for a_k to ensure that $\overline{\theta}_k^i > \theta_{min}^i$ for all i:

$$a_k \leq \tilde{a}_k = \min_{i\in\{1,\ldots,m\}} \frac{(\theta_k^i - \theta_{min}^i)\mathcal{N}_k}{\gamma\beta|\nabla_i l(\theta_k)|} \tag{A.10}$$

Combining (A.9) and (A.10), we have

$$a_k \leq \bar{a}_k = \min\{\hat{a}_k, \tilde{a}_k\} \quad \forall k. \tag{A.11}$$

We now outline conditions for selection of b_{k+1}, which follows similar approach to selection of a_k, i.e., for given $\nu_k \in I$ we prescribe range of b_{k+1} such that $\nu_{k+1} \in I$. From **Step-10** of Algorithm 3, it could be deduced that for all k, b_{k+1} must satisfy

$$b_{k+1} \leq \bar{b}_{k+1} = \min\{\hat{b}_{k+1}, \tilde{b}_{k+1}\} \tag{A.12}$$

where

$$\hat{b}_{k+1} = \min_{i\in\{1,\ldots,m\}} \frac{(\theta_{max}^i - \nu_k^i)\mathcal{N}_k}{\gamma|\nabla_i l(\theta_{k+1})|}$$
$$\tilde{b}_{k+1} = \min_{i\in\{1,\ldots,m\}} \frac{(\nu_k^i - \theta_{min}^i)\mathcal{N}_k}{\gamma|\nabla_i l(\theta_{k+1})|}.$$

Note however, that (A.11), (A.12) can generate $a_k, b_{k+1} = 0$, which is undesirable. To compensate for that, we introduce an additional rule:

$$a_k = \begin{cases} -\epsilon & \min\limits_{i\in\{1,\dots,m\}}(\theta_k^i - \theta_{max}^i) = 0 \\ \epsilon & \min\limits_{i\in\{1,\dots,m\}}(\theta_k^i - \theta_{min}^i) = 0 \end{cases} \tag{A.13}$$

Here $0 < \epsilon < 1$ is a very small real number of choice. Similar update rule can be stated for b_{k+1} to avoid the case of $a_k, b_{k+1} = 0$. For a given k, by selecting a_k, b_{k+1} such that (A.11), (A.12), (A.13) are satisfied, we ensure $\overline{\theta}_k, \nu_k \in I$. From **Step-7** of Algorithm 3:

$$\theta_{k+1} = (1-\beta)\overline{\theta}_k + \beta\nu_k \tag{A.14}$$

Hence, θ_{k+1} is a convex combination of $\overline{\theta}_k$ and ν_k for a given k and $0 < \beta \leq 1$. Additionally, set I is compact, hence, if $\overline{\theta}_k, \nu_k \in I$, then $\theta_{k+1} \in I$ for a given $k \in \mathbb{N}$. By choosing $\theta_0, \nu_0 \in I$, by induction it can be shown that Proposition 4.2.2 can be applied iteratively to generate parameters that are bounded within the compact set I.

Theorem 4.2.1 For a differentiable $\bar{L}_k$-smooth convex loss function $l(.)$, Algorithm 3 with $0 < \beta \leq 1$, $0 < \gamma < \frac{\beta(2-\beta)}{8+\beta}$ and a_k, b_{k+1} satisfying $a_k \leq \min\{1, \bar{a}_k\}$, $b_{k+1} \leq \min\{1, \bar{b}_{k+1}\}$ and (A.13), where $\bar{a}_{k+1}$ and $\bar{b}_{k+1}$ are defined in (A.11) and (A.12) ensures that $V = \frac{\|\nu-\theta^*\|^2}{\gamma} + \frac{\|\nu-\theta\|^2}{\gamma}$ is a Lyapunov function. Consequently, the sequence of iterates $\{\theta_k\}$ generated by Algorithm 3 satisfy $\{\theta_k\} \in \Omega_m$, and $\lim\limits_{k\to\infty} l(\theta_k) = l(\theta^*)$, where $l(\theta^*) = f([\theta^{*T} \quad p(\theta^*)^T]^T)$ is the optimal value of (4.1).

Proof This proof follows a similar approach to the proof of stability of High Order Tuner for convex optimization, as illustrated in [32, Theorem 2].

Assuming that $\nu_k, \theta_k, \overline{\theta}_k \in I$, function $l(\cdot)$ is convex for all these parameters lying within the set I. Applying convexity and smoothness properties (ref. [32, Section II]) to $l(\cdot)$, the following upper bound is obtained:

$$l(\vartheta_k) - l(\bar{\theta}_k) = l(\vartheta_k) - l(\theta_{k+1}) + l(\theta_{k+1}) - l(\bar{\theta}_k)$$

$$
\begin{aligned}
&\leq \nabla l(\theta_{k+1})^T(\vartheta_k - \theta_{k+1}) + \frac{\bar{L}_k}{2}\|\vartheta_k - \theta_{k+1}\|^2 \\
&\quad + \nabla l(\theta_{k+1})^T(\theta_{k+1} - \bar{\theta}_k)
\end{aligned} \tag{A.15}
$$

$$
\overset{\text{Alg.3}}{\leq} \nabla l(\theta_{k+1})^T(\vartheta_k - \bar{\theta}_k) + \frac{\bar{L}_k}{2}\|\vartheta_k - (1-\beta)\bar{\theta}_k - \beta\vartheta_k)\|^2 \tag{A.16}
$$

$$
\begin{aligned}
&l(\vartheta_k) - l(\bar{\theta}_k) \\
&\leq -\nabla l(\theta_{k+1})^T(\bar{\theta}_k - \vartheta_k) + \frac{\bar{L}_k}{2}(1-\beta)^2\|\bar{\theta}_k - \vartheta_k\|^2.
\end{aligned} \tag{A.17}
$$

Similarly, we obtain:

$$
\begin{aligned}
&l(\bar{\theta}_k) - l(\vartheta_k) \\
&\quad \leq \nabla l(\theta_k)^T(\bar{\theta}_k - \vartheta_k) + \frac{a_k^2\bar{L}_k\gamma^2\beta^2}{2\mathcal{N}_k^2}\|\nabla l(\theta_k)\|^2.
\end{aligned} \tag{A.18}
$$

Using (A.17) and (A.18) we obtain:

$$
\begin{aligned}
&\nabla l(\theta_{k+1})^T(\bar{\theta}_k - \vartheta_k) \\
&- \frac{\bar{L}_k}{2}(1-\beta)^2\|\bar{\theta}_k - \vartheta_k\|^2 \\
&- \frac{a_k^2\bar{L}_k\gamma^2\beta^2}{2\mathcal{N}_k^2}\|\nabla l(\theta_k)\|^2 \leq \nabla l(\theta_k)^T(\bar{\theta}_k - \vartheta_k)
\end{aligned} \tag{A.19}
$$

Using Algorithm 3, [32, Theorem 1] and (A.19), setting $\gamma < \frac{\beta(2-\beta)}{8+\beta}$, $0 < a_k, b_{k+1} \leq 1$ and defining $\Delta V_k := V_{k+1} - V_k$, it can be shown that

$$
\begin{aligned}
\Delta V_k \leq \frac{1}{\mathcal{N}_k}\Bigg\{ &- 2b_{k+1}(l(\theta_{k+1}) - l(\theta^*)) - \left(\frac{b_{k+1}}{2\bar{L}_k} - \frac{2\gamma b_{k+1}^2}{\mathcal{N}_k}\right)\|\nabla l(\theta_{k+1})\|^2 \\
&- \left(1 - \frac{\bar{L}_k\gamma\beta a_k}{\mathcal{N}_k}\right)\frac{\gamma\beta^2 a_k^2}{\mathcal{N}_k}\|\nabla l(\theta_k)\|^2 - [\beta - a_k\beta(1-\beta)^2]\bar{L}_k\|\bar{\theta}_k - \nu_k\|^2 \\
&- \left[\frac{\sqrt{b_{k+1}}}{\sqrt{2\bar{L}_k}}\|\nabla l(\theta_{k+1})\| - 2\sqrt{2\bar{L}_k}\|\bar{\theta}_k - \nu_k\|\right]^2 - 4(\sqrt{b_{k+1}} - b_{k+1})\|\bar{\theta}_k - \nu_k\|\|\nabla l(\theta_{k+1})\| \\
&- (8+\beta)\|\bar{\theta}_k - \nu_k\|^2\Bigg\} \leq 0
\end{aligned} \tag{A.20}
$$

From (A.20), it can be seen that:

$$\Delta V_k \leq \frac{b_{k+1}}{\mathcal{N}_k}\{-2(l(\theta_{k+1}) - l(\theta^*)\} \leq 0 \tag{A.21}$$

Collecting ΔV_k terms from t_0 to T, and letting $T \to \infty$, it can be seen that $l(\theta_{k+1}) - l(\theta^*) \in \ell_1 \cap \ell_\infty$ and therefore $\lim_{k\to\infty} l(\theta_{k+1}) - l(\theta^*) = 0$.

Theorem 4.4.1 For a quadratic loss function $l(\theta) = \|y - \theta^T\phi\|^2$, a compact and convex set C defined as $C = \{\theta | \|\theta\| \leq \theta_{max} + \epsilon\}$ the sequence of iterates $\{\theta_k\}$ generated by Algorithm 2 with $proj_C$ defined by (4.19) satisfy $\{\theta_k\} \in C$ and $V = \frac{\|\nu-\theta^*\|^2}{\gamma} + \frac{\|\nu-\theta\|^2}{\gamma}$ is a valid Lyapunov function

Proof **Note**: The proof contains important steps, parts of the proof have been omitted for brevity and correspond to Lyapunov stability proof for linear regression in unconstrained setting and can be found in [33].

The projection operator applied to $\dot{\theta}(t)$ can be written as:

$$proj_C(\theta, (\theta(t) - \nu(t), f) = \begin{cases} \left(I - \frac{\nabla f(\theta)\nabla f(\theta)^T}{\|\nabla f(\theta)\|^2} f(\theta)\right)(\theta(t) - \nu(t)) & \text{if } f(\theta) > 0 \bigwedge y^T f(\theta) > 0 \\ (\theta(t) - \nu(t)) & \text{otherwise} \end{cases}$$

Thus, HT in continuous time can be written as:

$$\dot{\nu}(t) = -\gamma \frac{\nabla L_t(\theta(t))}{\mathcal{N}_t}$$
$$\dot{\theta}(t) = -\beta Proj(\theta(t), (\theta(t) - \nu(t), f)$$

We define the Lyapunov function V as:

$$V(t) = \frac{1}{2}\|\theta(t) - \nu(t)\|^2 + \frac{1}{2}\|\nu(t) - \theta^*\|^2 \tag{A.22}$$

Using the above defined Projection, $\dot{V}(t)$ can be written as:

$$\begin{aligned}
\dot{V}(t) &= (\nu(t)-\theta^*)^T\bigg(-\frac{\gamma}{\mathcal{N}(t)}\nabla L(\theta(t))\bigg)+ \\
&(\theta(t)-\nu(t))^T\bigg(-\beta proj_C(\theta,(\theta(t)-\nu(t),f)+\frac{\gamma}{\mathcal{N}(t)}\nabla L(\theta(t))\bigg) \\
&= (\nu(t)-\theta^*)^T\bigg(-\frac{\gamma}{\mathcal{N}(t)}\nabla L(\theta(t))\bigg)+(\theta(t)-\nu(t))^T\frac{\gamma}{\mathcal{N}(t)}\nabla L(\theta(t)) \\
&\quad -\beta(\theta(t)-\nu(t))^T proj_C(\theta,(\theta(t)-\nu(t),f) \\
&= (\nu(t)-\theta^*)^T\bigg(-\frac{\gamma}{\mathcal{N}(t)}\nabla L(\theta(t))\bigg)+(\theta(t)-\nu(t))^T\frac{\gamma}{\mathcal{N}(t)}\nabla L(\theta(t)) \\
&\quad -\beta(\theta(t)-\nu(t))^T(\theta(t)-\nu(t)) \\
&\quad +\beta(\theta(t)-\nu(t))^T\frac{\nabla f(\theta(t))\nabla f(\theta(t))^T}{\|\theta(t)\|^2}(\theta(t)-\nu(t))f(\theta(t))
\end{aligned} \tag{A.23}$$

It can be easily established in a linear-regression setting that [33]

$$\begin{aligned}
&(\nu(t)-\theta^*)^T\bigg(-\frac{\gamma}{\mathcal{N}(t)}\nabla L(\theta(t))\bigg)+(\theta(t)-\nu(t))^T\frac{\gamma}{\mathcal{N}(t)}\nabla L(\theta(t)) \\
&\quad -\beta(\theta(t)-\nu(t))^T(\theta(t)-\nu(t))\le 0 \\
&-(\nu(t)-\theta^*)^T\bigg(\frac{\gamma}{\mathcal{N}(t)}\nabla L(\theta(t))\bigg)+(\theta(t)-\nu(t))^T\frac{\gamma}{\mathcal{N}(t)}\nabla L(\theta(t)) \\
&\quad -\beta\|\theta(t)-\nu(t)\|^2\le 0
\end{aligned} \tag{A.24}$$

The term associated with projection:

$$\begin{aligned}
&-\beta(\theta(t)-\nu(t))^T(\theta(t)-\nu(t)) \\
&+\beta(\theta(t)-\nu(t))^T\frac{\nabla f(\theta(t))\nabla f(\theta(t))^T}{\|\theta(t)\|^2}(\theta(t)-\nu(t))f(\theta(t))
\end{aligned}$$

This can be equivalently written as:

$$\begin{aligned}
&-\beta\bigg[\|\theta(t)-\nu(t)\|^2-\|\theta(t)-\nu(t)\|^2\frac{\nabla f(\theta(t))\nabla f(\theta(t))^T}{\|\theta(t)\|^2}f(\theta(t))\bigg] \\
&\quad =\beta\|\theta(t)-\nu(t)\|^2\bigg(1-\frac{\nabla f(\theta(t))\nabla f(\theta(t))^T}{\|\theta(t)\|^2}f(\theta(t))\bigg)
\end{aligned}$$

Additionally, $\frac{\nabla f(\theta(t))\nabla f(\theta(t))^T}{\|\theta(t)\|^2} f(\theta(t)) \leq 1$, for $f(\theta(t))$ defined in Chapter 4, as shown in [28]. Therefore,

$$-\beta\left[\|\theta(t)-\nu(t)\|^2 - \|\theta(t)-\nu(t)\|^2 \frac{\nabla f(\theta(t))\nabla f(\theta(t))^T}{\|\theta(t)\|^2} f(\theta(t))\right] \leq 0 \tag{A.25}$$

By combining (A.24) with the above inequality, we get:

$$\begin{aligned}\dot{V}(t) = (\nu(t)-\theta^*)^T\left(-\frac{\gamma}{\mathcal{N}(t)}\nabla L(\theta(t))\right) + (\theta(t)-\nu(t))^T\frac{\gamma}{\mathcal{N}(t)}\nabla L(\theta(t)) \\ -\beta\left[\|\theta(t)-\nu(t)\|^2 - \|\theta(t)-\nu(t)\|^2 \frac{\nabla f(\theta(t))\nabla f(\theta(t))^T}{\|\theta(t)\|^2} f(\theta(t))\right] \leq 0\end{aligned}$$

Therefore, $V = \frac{\|\nu-\theta^*\|^2}{\gamma} + \frac{\|\nu-\theta\|^2}{\gamma}$ guarantees stability for HT with projection for the case of linear regression.

Appendix B

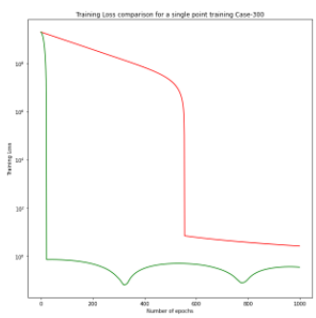
Details of IEEE Case-300 and Case-1354 simulations

B.1 IEEE Case-300

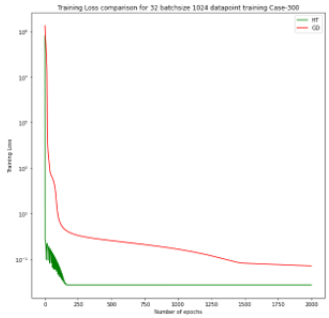
In this numerical simulation we analyze the effect of acceleration in training of a NN by comparing the performance of HT and GD. To clearly observe the acceleration effect of HT, dataset size was increased gradually, starting from a single-point training (Fig B-1a).

While it is true that increasing size alters the underlying loss function as is evident from the dual optimization problem (5.11), training with varying sizes of datasets helped in validating the hypothesis that the accelerated convergence of HT can contribute in reducing the number of epochs for training or reducing the required dataset size.

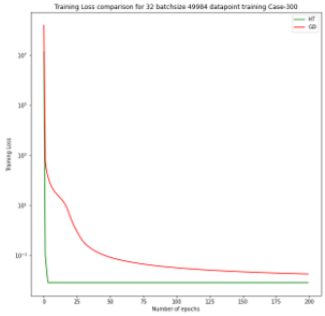
Fig B-1 shows that regardless of the size of dataset, HT converges to an extremely small loss value $\sim 10^{-4}$ in substantially less number of epochs. This is in contrast with GD, where the direct correlation between increasing training dataset size and improved training accuracy is extremely strong. Additionally, with a few more training simulations, the difference in epochs and dataset was quantized more firmly. Specifically,



(a) Single-point training, to illustrate the speed of HT



(b) 1024 datapoints for training, 32 batch-size



(c) 50,000 datapoints, 32 batch-size

Figure B-1: IEEE Case-300 Training Loss comparison: (a) HT outperforms GD significantly due to acceleration, which can be noticed in the osciallatory behavior, (b) With more training data, HT still performs better but GD's performance improves incrementally, (c)GD and HT performance converges more as dataset size increases

1. For a fixed 300 epochs, to ensure that the training loss is atmost 7×10^{-3}, training with GD required atleast 50,000 datapoints, whereas 1024 datapoints sufficed for HT. (Table B.1)

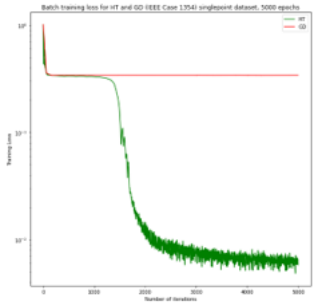
2. For a fixed dataset of 1024 points, Ht is able to reduce the training loss to atmost 7×10^{-3} in 250 epochs, whereas GD requires 2000 epochs. Intuitively, by increasing the size of training dataset to 50,000 points, the number of epochs can be brought down by 10 times for both, but is still higher for GD compared to HT (Table B.2).

Epochs	Dataset (HT)	Dataset (GD)
300	1024	50,000

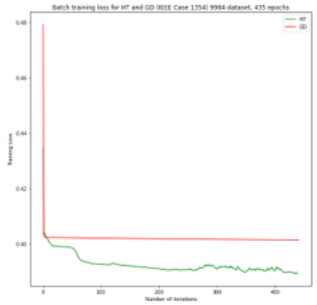
Table B.1: Fixed epochs: comparison of dataset size required to converge for HT and GD

Dataset Size	Epochs (HT)	Epochs (GD)
1024	250	2000
50,000	25	200

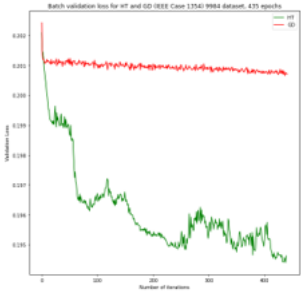
Table B.2: Fixed Dataset: comparison of epochs required to converge for HT and GD



(a) Single-point training loss comparison, to illustrate the speed of HT



(b) 50,000 datapoint, 32 batch-size: Training Loss comparison



(c) 50,000 datapoints, 32 batch-size: Validation Loss comparison

Figure B-2: IEEE Case-1354 Training and Validation Loss comparison: (a)-(b) Training Loss with single point and larger training dataset. GD gets stuck in a local minimum while HT converges. (c)-(d) Validation loss comparison illustrates similar phenomenon is observed as in Training loss

B.2 IEEE Case 1354

IEEE Case-1354 reflects how the training process would be executed on a realistic instance of DC-OPF due to sufficiently high dimension of the problem. We extend the analysis to training loss and validation loss in this case to observe the differences more clearly.

B.2.1 Analysis of Training & Validation Loss

Similar to IEEE Case-30, we first perform a dummy single-point training to compare between the peformance of HT and GD (Fig-B-2a). This shows how the acceleration of HT helps in avoiding the plateuing effect, whereas GD gets stuck in a saddle. The variance in loss is high in this case, so we increase the dataset to 50,000 points. We observe the advantage of acceleration in this case more clearly (Fig-B-2b).

To assess if the advantages of acceleration can be extended to test case as well, we analyse Validation loss over a dataset of 50,000 points, where the size of validation dataset is 10,000 points. Here, we notice that while the variance of loss is high for HT and reduces as we increase the training epochs, it is substantially lower than GD (Fig B-2c). Some methods to refine the results have been discussed in Section 5.4.

Bibliography

[1] Manya V Afonso, José M Bioucas-Dias, and Mário AT Figueiredo. An augmented lagrangian approach to the constrained optimization formulation of imaging inverse problems. *IEEE transactions on image processing*, 20(3):681–695, 2010.

[2] Brandon Amos and J. Zico Kolter. OptNet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, 2017.

[3] Brandon Amos and J. Zico Kolter. OptNet: Differentiable optimization as a layer in neural networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 136–145. PMLR, 06–11 Aug 2017.

[4] P. Armand and R. Omheni. A globally and quadratically convergent primal-dual augmented Lagrangian algorithm for equality constrained optimization. *Optimization Methods and Software*, 32(1):1–21, 2017.

[5] Sanjeev Arora, Nadav Cohen, Noah Golowich, and Wei Hu. A convergence analysis of gradient descent for deep linear neural networks. *arXiv preprint arXiv:1810.02281*, 2018.

[6] Goran Banjac, Paul Goulart, Bartolomeo Stellato, and Stephen Boyd. Infeasibility detection in the alternating direction method of multipliers for convex optimization. *Journal of Optimization Theory and Applications*, 183:490–519, 2019.

[7] Amir Beck. *First-Order Methods in Optimization*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2017.

[8] Amir Beck and Marc Teboulle. A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems *. *Society for Industrial and Applied Mathematics*, 2(1):183–202, 2009.

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

[10] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction

method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

[11] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[12] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

[13] Zixiang Chen, Yuan Cao, Difan Zou, and Quanquan Gu. How much over-parameterization is sufficient to learn deep relu networks? *arXiv preprint arXiv:1911.12360*, 2019.

[14] Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *Advances in neural information processing systems*, 27, 2014.

[15] Frederik Diehl. Warm-starting ac optimal power flow with graph neural networks. In *33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*, pages 1–6, 2019.

[16] Priya L. Donti, David Rolnick, and J. Zico Kolter. DC3: A learning method for optimization with hard constraints. In *International Conference on Learning Representations*, 2021.

[17] Priya L. Donti, David Rolnick, and J. Zico Kolter. Dc3: A learning method for optimization with hard constraints. 2021.

[18] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[19] Massimo Furi and Mario Martelli. A multidimensional version of rolle's theorem. *The American Mathematical Monthly*, 102(3):243–249, 1995.

[20] A. Galántai. The theory of newton's method. *Journal of Computational and Applied Mathematics*, 124(1):25–44, 2000. Numerical Analysis 2000. Vol. IV: Optimization and Nonlinear Equations.

[21] Joseph E. Gaudio, Anuradha M. Annaswamy, Michael A. Bolender, Eugene Lavretsky, and Travis E. Gibson. A Class of High Order Tuners for Adaptive Systems. *IEEE Control Systems Letters*, pages 1–1, 6 2020.

[22] Joseph E Gaudio, Anuradha M Annaswamy, José M Moreu, Michael A Bolender, and Travis E Gibson. Accelerated Learning with Robustness to Adversarial Regressors. *3rd L4DC Conference*, 2021.

[23] Suriya Gunasekar, Jason Lee, Daniel Soudry, and Nathan Srebro. Characterizing implicit bias in terms of optimization geometry. In *International Conference on Machine Learning*, pages 1832–1841. PMLR, 2018.

[24] Dmitry Kovalev, Alexander Gasnikov, and Peter Richtárik. Accelerated primal-dual gradient method for smooth and convex-concave saddle-point problems with bilinear coupling. *Advances in Neural Information Processing Systems*, 35:21725–21737, 2022.

[25] Steven George Krantz and Harold R Parks. *The implicit function theorem: history, theory, and applications*. Springer Science & Business Media, 2002.

[26] Maxime Laborde and Adam Oberman. A lyapunov analysis for accelerated gradient methods: from deterministic to stochastic case. In *International Conference on Artificial Intelligence and Statistics*, pages 602–612. PMLR, 2020.

[27] Gregory Larchev, Stefan Campbell, and John Kaneshige. Projection operator: A step toward certification of adaptive controllers. In *AIAA Infotech@ Aerospace 2010*, page 3366. 2010.

[28] Eugene Lavretsky and Travis E Gibson. Projection operator in adaptive systems. *arXiv preprint arXiv:1112.4232*, 2011.

[29] Andreas Look, Simona Doneva, Melih Kandemir, Rainer Gemulla, and Jan Peters. Differentiable implicit layers. *arXiv preprint arXiv:2010.07078*, 2020.

[30] Hao Luo. Accelerated primal-dual methods for linearly constrained convex optimization problems. *arXiv preprint arXiv:2109.12604*, 2021.

[31] Hancheng Min, Salma Tarmoun, René Vidal, and Enrique Mallada. On the explicit role of initialization on the convergence and implicit bias of overparametrized linear networks. In *International Conference on Machine Learning*, pages 7760–7768. PMLR, 2021.

[32] José M. Moreu and Anuradha M. Annaswamy. A stable high-order tuner for general convex functions. *IEEE Control Systems Letters*, 6:566–571, 2022.

[33] José María Moreu Gamazo. *High-order tuners for convex optimization: stability and accelerated learning*. PhD thesis, Massachusetts Institute of Technology, 2021.

[34] Hédi Nabli. An overview on the simplex algorithm. *Applied Mathematics and Computation*, 210(2):479–489, 2009.

[35] Yurii Nesterov. *Lectures on convex optimization*, volume 137. Springer International Publishing, 2018.

[36] University of Washington. Power systems test case archive. 1993.

[37] Xiang Pan, Tianyu Zhao, and Minghua Chen. Deepopf: Deep neural network for dc optimal power flow. pages 1–6, 10 2019.

[38] Xiang Pan, Tianyu Zhao, Minghua Chen, and Shengyu Zhang. Deepopf: A deep neural network approach for security-constrained dc optimal power flow, 2019.

[39] Anjali Parashar, Priyank Srivastava, and Anuradha M. Annaswamy. Accelerated algorithms for a class of optimization problems with equality and box constraints, 2023.

[40] Anjali Parashar, Priyank Srivastava, Anuradha M. Annaswamy, Biswadip Dey, and Amit Chakraborty. Accelerated algorithms for a class of optimization problems with constraints. In *2022 IEEE 61st Conference on Decision and Control (CDC)*, pages 6960–6965, 2022.

[41] Panagiotis Patrinos, Lorenzo Stella, and Alberto Bemporad. Douglas-rachford splitting: Complexity estimates and accelerated variants. In *53rd IEEE Conference on Decision and Control*, pages 4234–4239, 2014.

[42] Mert Pilanci and Tolga Ergen. Neural networks are convex regularizers: Exact polynomial-time convex optimization formulations for two-layer networks. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 7695–7705. PMLR, 13–18 Jul 2020.

[43] HoHo Rosenbrock. An automatic method for finding the greatest or least value of a function. *The computer journal*, 3(3):175–184, 1960.

[44] Jean-Jacques E Slotine, Weiping Li, et al. *Applied nonlinear control*, volume 199. Prentice hall Englewood Cliffs, NJ, 1991.

[45] Richard S Sutton. Two problems with backpropagation and other steepest-descent learning procedures for networks. In *Proc. of Eightth Annual Conference of the Cognitive Science Society*, pages 823–831, 1986.

[46] Adrien Taylor, Bryan Van Scoy, and Laurent Lessard. Lyapunov functions for first-order methods: Tight automated convergence guarantees. In *International Conference on Machine Learning*, pages 4897–4906. PMLR, 2018.

[47] Russ Tedrake. *Underactuated Robotics*. 2023.

[48] Andreas Venzke, Guannan Qu, Steven Low, and Spyros Chatzivasileiadis. Learning optimal power flow: Worst-case guarantees for neural networks. In *2020 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, pages 1–7. IEEE, 2020.

[49] Yifei Wang, Jonathan Lacotte, and Mert Pilanci. The hidden convex optimization landscape of regularized two-layer relu networks: an exact characterization of optimal solutions. In *International Conference on Learning Representations*, 2021.

[50] Ashia C. Wilson. Lyapunov arguments in optimization. 2018.

[51] Xiaoxia Wu, Edgar Dobriban, Tongzheng Ren, Shanshan Wu, Zhiyuan Li, Suriya Gunasekar, Rachel Ward, and Qiang Liu. Implicit regularization and convergence for weight normalization. *Advances in Neural Information Processing Systems*, 33:2835–2847, 2020.

[52] Lin Yang, Raman Arora, Tuo Zhao, et al. The physical systems behind optimization algorithms. *Advances in Neural Information Processing Systems*, 31, 2018.

[53] Ahmed S Zamzam and Kyri Baker. Learning optimal solutions for extremely fast ac optimal power flow. In *2020 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, pages 1–6. IEEE, 2020.

[54] Ray Daniel Zimmerman, Carlos Edmundo Murillo-Sánchez, and Robert John Thomas. Matpower: Steady-state operations, planning, and analysis tools for power systems research and education. *IEEE Transactions on Power Systems*, 26(1):12–19, 2011.