Provable and Control-Theoretic Methods for Deep Object Pose Estimation

Trevor Avant

A dissertation

submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

University of Washington

2021

Reading Committee:

Kristi Morgansen, Chair

Mehran Mesbahi

Behçet Açikmeşe

Uri Shumlak

J. Nathan Kutz

Program Authorized to Offer Degree:

Aeronautics & Astronautics

University of Washington

Abstract

Provable and Control-Theoretic Methods
for Deep Object Pose Estimation

Trevor Avant

Chair of the Supervisory Committee:
Professor and Chair Kristi Morgansen
Aeronautics & Astronautics

In this dissertation, we consider the task of object pose estimation using deep neural networks. We draw our motivation from the fact that neural networks have shown to be successful at the task of pose estimation, but are poorly theoretically understood and lack meaningful performance guarantees. As a result, our aim in this dissertation is to analyze pose estimation neural networks by developing provable performance guarantees, as well as connecting pose estimation to control theory. We take four different approaches in our analysis. First, we consider object pose estimation from the standpoint of observability in control theory, using the observability Gramian as our main tool for analysis. Next, we explore the idea of estimating the pose of a dynamic object by applying an unscented filter to pose estimates from a neural network. Next, we derive analytical bounds on the local Lipschitz constants of neural networks with ReLU activations. Finally, we consider the task of developing sensitivity bounds for pose estimation neural networks, and construct a pose estimation network with provable bounds for both the rotation and position estimates.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First, I would like to thank my advisor, Prof. Morgansen. Over the course of my graduate program, she has involved me with interesting projects, has given me the freedom to follow my interests, and has offered encouragement along the way. I am very lucky to have had her as an advisor.

I would also like to thank my committee members: Professors Mesbahi, Açikmeşe, Shumlak, and Kutz. Despite their busy schedules, they have always gladly donated their time and effort to help me complete my degree.

The staff in the Aeronautics & Astronautics department has been tremendously helpful at taking care of all of my administrative needs during my graduate program. In particular, I would like to thank graduate advisors Ed Connery, Wanda Frederick, Danyel Hacker, and Paul Neubert for their help and kindness.

Over the course of my graduate program, I have had many labmates in the Nonlinear Dynamics and Control Lab: Caleb, Atiye, Andreas, Cody, Nathan, Brian H, Natalie, Tysen, Drew, Jake, Josue, Abby, Dustin, Unsik, Alex, Karine, Burak, Ena, Sierra, John, Brian K, Brian B, Tarik, Kimber, Veevee, Carey, Nick, and Riley. I would like to thank them all for their camaraderie. Additionally, many undergraduate students have worked in the lab, and I am grateful to have had the opportunity to get to know and work with many of them.

I have also had the opportunity to get to know and work with many other graduate students at UW, both inside and outside of the A&A department. I would like to thank them for the community they have provided.

In addition to those I have mentioned, there are countless faculty, staff, and students at UW who have helped me with classes, administration, and a variety of other things. Completing my degree would not have been possible without them.

Finally, I would like to thank the NDSEG fellowship, ARCS Foundation fellowship, and ONR grant N00014-17-1-2623 for funding my graduate program and this dissertation.

DEDICATION

to my teachers

Chapter 1

INTRODUCTION

1.1 Motivation

In the last decade, the field of computer vision has been revolutionized by deep neural networks. This revolution has largely been driven by the task of image classification, in which a computer algorithm or function determines which class of objects an image belongs to. Although neural networks have existed for decades, the were an unpopular approach for image classification until around 2012, when they began significantly outperforming other techniques. In particular, in 2012 a deep neural network called "AlexNet" gained widespread recognition by winning a popular image classification competition (the ImageNet Large Scale Visual Recognition Challenge) by a large margin [23]. In the following years, neural networks exploded in popularity, became the standard approach for image classification, and have been applied to many other tasks inside and outside the realm of computer vision. In the last few years, deep neural networks have also become the standard approach for object pose estimation, which is the task of estimating the pose (position and orientation) of a known object, often solely from a single camera image [53, 36, 46, 49].

Despite their successes, one of the major weaknesses of deep neural networks is that, due to their high dimensionality and complex nonlinear functions, they are poorly theoretically understood and lack meaningful performance guarantees. Nevertheless, deep neural networks are currently being applied to real-world robotics applications such as autonomous driving. The fact that deep neural networks lack theoretical understanding, but are being applied in safety-critical applications, raises serious concerns. For example, failure of a deep neural

network in an autonomous driving application could have catastrophic consequences.

With these concerns in mind, the aim of this dissertation will be to develop a better theoretical understanding of deep learning, with a specific focus on object pose estimation applications. To do so, we will develop new techniques to analyze and implement deep neural networks, and create new connections between deep learning and control theory. Our work will employ tools from the areas of deep learning, control theory, mechanics, 3D rotations, estimation theory, and mathematical analysis.

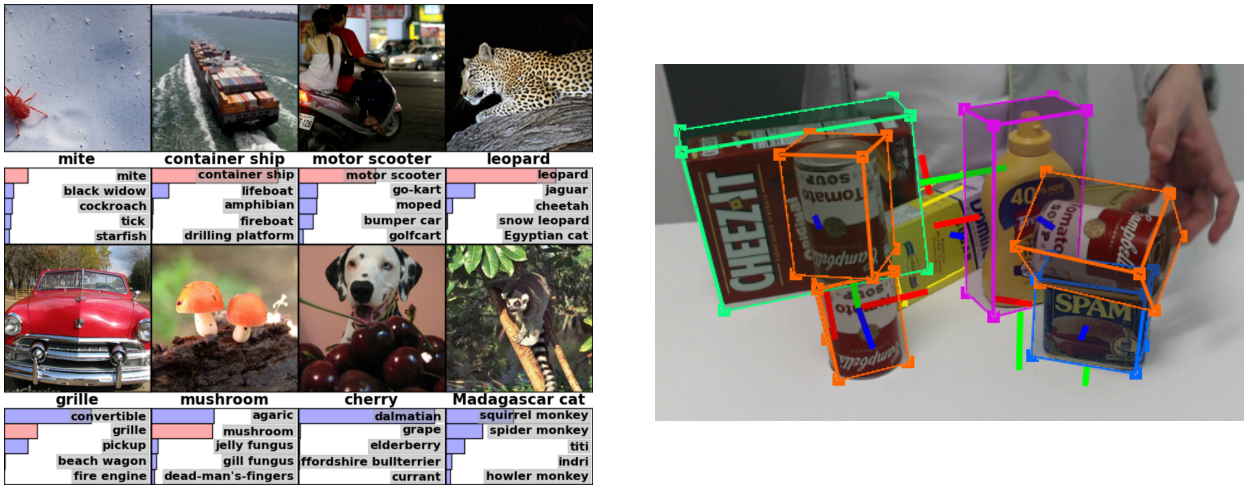
1.2 History of Neural Networks

Although neural networks have recently had a large explosion in popularity, they can trace their origins back to the middle of the twentieth century. A computational model of the brain was proposed as early as the 1940s, and the first brain-inspired models were implemented in computers as early as the 1950s [28, 37]. In the following decades, research continued on implementing brain-inspired mathematical models on computers. These types of models came to be known as "artificial neural networks" or simply "neural networks".

Although neural networks have been applied to solve machine learning problems for decades, they have not always enjoyed high popularity. However, in the past decade, there has been a large revival in neural networks, which is largely a result of their effectiveness at solving image classification problems. Consequently, neural networks have also recently been successfully applied to many other areas, such as natural language processing, medicine, and theoretical physics [25].

1.3 Basic Mathematics of Deep Neural Networks

We will now provide a brief mathematical discussion of the basic mathematics of neural networks. A neural network can be mathematically described as a function $\mathbf{f}$ with parameters



(a) AlexNet (image classification network) (b) DOPE (pose estimation network)

Figure 1.1: (a) Image classification results from the 2012 paper by Krizhevsky et al. [23], and (b) object pose estimation results from the 2018 paper by Tremblay et al. [49].

$\mathbf{a}$, which maps an input $\mathbf{x}$ to an output $\mathbf{y}$:

$$\mathbf{y} = \mathbf{f}(\mathbf{x}; \mathbf{a}). \tag{1.1}$$

Although a neural network can be designed in many possible ways, many networks are feed-forward, which means the input is passed through a composition of functions to arrive at the output. Common functions used in neural networks include fully-connected, convolutional, and max pooling functions. Each of these functions is often followed by an "activation" function, which is usually an elementwise operation such as a sigmoid or rectified linear unit (ReLU). A function and its activation function are often called a "layer" of the network. Assuming the network has n layers, we define the i^{th} function for $i = 1, ..., n$ as $\mathbf{f}_i$ with parameters $\mathbf{a}_i$. Not showing the dependence of $\mathbf{f}_i$ on $\mathbf{a}_i$ for clarity, the feedforward neural network can be written as

$$\mathbf{f}(\mathbf{x}; \mathbf{a}) = \mathbf{f}_n \circ \ldots \circ \mathbf{f}_2 \circ \mathbf{f}_1(\mathbf{x}). \tag{1.2}$$

A neural network is often called a "deep" neural network if it has a large number of layers. The number of layers a network must have to qualify as "deep" is not precise, but any network with three or more layers is commonly called deep.

In the case of image classification, the input $\mathbf{x}$ to the network will be an image, and the output $\mathbf{y}$ will be a classification vector. The parameters $\mathbf{a}$ are tuned to achieve a good fit with a dataset of known pairs of inputs and outputs. This dataset is called "training data", and the fitting process is called "training". Training is performed by first creating a scalar error function $E(\mathbf{a})$ which measures the difference between the training data's true output and the network's prediction of the output. Then, the gradient of the error is calculated with respect to the parameters $\mathbf{a}$ (i.e., $\partial E/\partial \mathbf{a}$), and the cost is minimized by gradient descent. Often times, a specific variety of gradient descent called "stochastic gradient descent" is used, which involves using only a portion of the training data for each step of the descent. Note that training neural networks is a computationally expensive process that has been made possible by using GPUs, and computing the error gradient by judiciously using the chain rule via an algorithm called backpropagation.

1.4 Contributions

In this dissertation, we present several new methods of analysis for pose estimation neural networks and neural networks in general. We present four different research topics, each of which corresponds to a chapter of the dissertation. Each topic/chapter contains both theoretical and practical contributions, where the practical contributions can be viewed as tools that a practitioner can use. We will now describe the theoretical and practical contributions for each topic/chapter.

In Chapter 2, we consider the task of object pose estimation using the framework of observability in control theory. A theoretical contribution of this chapter is that we develop

a new connection between pose estimation and the topic of observability in control theory. A practical contribution is that we develop a tool, based on the observability Gramian, which quantifies which poses of an object are most favorable for pose estimation.

In Chapter 3, we develop an unscented filter to estimate the state of an object with rigid body dynamics, in the scenario in which the pose estimates come from a pose estimation neural network. This chapter provides a new connection between pose estimation neural networks and unscented filtering, as well as a new application of unscented filtering. The practical contribution of this chapter is the unscented filter we develop, which is shown to improve the neural network's pose estimates by incorporating knowledge about the physics of the object.

In Chapter 4, we devise a method to determine sensitivity bounds for neural networks with ReLU activation functions. The theoretical contributions of this chapter include the derivation of Lipschitz constants and bounds for several functions commonly used in neural networks, as well as a method to use these bounds to compute the local Lipschitz bound of a full network. The practical contribution of this chapter is a computationally efficient method to compute sensitivity bounds for large neural networks.

In Chapter 5, we develop a provable bound on the sensitivity of pose estimation neural networks. The theoretical contributions of this chapter include a result describing how rotational sensitivity bounds for pose estimation neural networks can be related to Lipschitz constants, as well as the derivation of several sensitivity-related quantities for various rotation parameterizations. The practical contribution of this chapter is a pose estimation network of our own design for which provable rotational sensitivity bounds can be calculated.

1.5 Dissertation Overview

The remainder of this dissertation is organized as follows. In Chapter 2, we apply the notion of observability to image-based object pose estimation. In Chapter 3, we consider the task of visually estimating the state of a rigid body by applying an unscented filter to estimates from a pose estimation neural network. In Chapter 4, we determine bounds on the local Lipschitz constants of neural networks with ReLU activation functions. In Chapter 5, we approach the task of deriving sensitivity bounds for pose estimation neural networks. Finally, in Chapter 6, we provide some additional insights, tie together some of the various research topics, and discuss some directions of future work.

Chapter 2

OBSERVABILITY OF POSE ESTIMATION

2.1 Introduction

Estimating the pose of a 3D object is useful for many robotics tasks, such as object manipulation and autonomous driving. While using images is an appealing approach to estimate pose, it is challenging to extract relevant information from high-dimensional images. This task is further complicated by the high variability of images, which is the result of factors such as lighting, occlusion, and variations in background.

To address these challenges, many different techniques have been developed for the problem of pose estimation using monocular images, including feature and edge detection, and template methods [26]. In recent years, the success of using neural networks for image processing applications has given rise to pose estimation neural networks [34, 36, 46, 53]. Although neural networks have had successful results, their high-dimensionality and highly nonlinear functions has prevented them from being understood from a theoretical viewpoint. As a result, they lack guarantees for purposes of provable performance.

In control theory, the problem of estimating the state of a system based on measurements is fundamental and well-studied. A key tool for addressing this problem is the system observability, which relates the system state to its measurement function. One means of characterizing observability is through the construction of the observability Gramian. In essence, the observability Gramian quantifies how well each state can be inferred from the given measurements, based on a calculation of output energy. Different expressions of the observability Gramian can be employed depending on the system structure being analyzed,

with the most common being for linear systems. For nonlinear systems, a local representation can be used, and in the case of unknown or highly complex systems, an empirical local version can be constructed. Here, we will be leveraging the empirical approach. The empirical observability Gramian is an approximation of the observability Gramian, and is constructed from a particular set of perturbed system trajectories. This version of the Gramian has been applied to problems involving PDEs [18], aeronautics [15], power systems [35], and networks [1]. In this chapter, we will consider the system state to be the pose (position and orientation) of an object, and the measurement function to be an RGB image.

By applying the observability Gramian to the problem of pose estimation, we can characterize what makes a certain view of an object "good" in the sense that it can be distinguished from other views. We believe that this type of observability analysis will help in developing a better theoretical understanding of pose estimation. As the observability Gramian is fundamental and simple in its nature, we believe it serves as a useful tool for any method of pose estimation.

The remainder of this chapter is organized as follows. In Section 2.2, we present the observability Gramian and empirical observability Gramian. In Section 2.3, we apply the observability Gramian to the task of pose estimation. In Section 2.4, we consider some special cases in which our analysis and the observability Gramian simplify. In Section 2.5, we connect the task of pose estimation to the concept of indistinguishability of nonlinear systems. We apply our methodology and present some static examples in Section 2.6, and some dynamic examples in Section 2.7. In Section 2.8, we discuss how our methods could be applied to pose estimation methods such as learning-based methods. Finally, we provide a chapter summary in Section 2.9.

2.2 The observability Gramian

2.2.1 Definition of the observability Gramian

The observability Gramian matrix plays a fundamental role in the analysis of state-space systems with measurements. Assume we have a state $\mathbf{x}(t) \in \mathbb{R}^n$, control $\mathbf{u}(t) \in \mathbb{R}^p$, and measurement $\mathbf{y}(t) \in \mathbb{R}^q$. For linear time-variant systems $\dot{\mathbf{x}}(t) = \mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)\mathbf{u}(t)$ with measurements $\mathbf{y}(t) = \mathbf{C}(t)\mathbf{x}(t) + \mathbf{D}(t)\mathbf{u}(t)$, the observability Gramian admits the closed-form solution

$$\mathbf{W}(t_f) = \int_0^{t_f} \mathbf{\Phi}(t,0)^T \mathbf{C}(t)^T \mathbf{C}(t) \mathbf{\Phi}(t,0) dt \tag{2.1}$$

where $\mathbf{\Phi}(t, t_0)$ is the state-transition matrix satisfying

$$\dot{\mathbf{\Phi}}(t, t_0) = \mathbf{A}(t)\mathbf{\Phi}(t, t_0)$$
$$\mathbf{\Phi}(t, t) = \mathbf{I}.$$

The state-transition matrix describes how the unforced state evolves, that is $\mathbf{x}(t) = \mathbf{\Phi}(t, t_0)\mathbf{x}(t_0)$ when $\mathbf{u}(t) = 0$. If $\mathbf{W}(t_f)$ is full-rank, then the system is observable [30].

For nonlinear systems, the observability Gramian can no longer be used as a global measure, as global observability analysis requires using tools from differential geometry. We can, however, linearize the nonlinear system about a trajectory to create an LTV system. We can then apply (2.1), which allows us to calculate the *local* observability Gramian matrix.

For the sake of clarity, we will now drop the explicit dependence on time from $\mathbf{x}(t)$, $\mathbf{u}(t)$ and $\mathbf{y}(t)$. Throughout the chapter, we will make this simplification for other variables as

well. We assume the nonlinear system is in the following form

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}, \mathbf{u}) \\ \mathbf{y} &= \mathbf{h}(\mathbf{x}). \end{aligned} \tag{2.2}$$

To calculate the local observability Gramian, we first denote our nominal reference trajectory as $\mathbf{x} = \overline{\mathbf{x}}$, which is produced by a nominal reference control $\mathbf{u} = \overline{\mathbf{u}}$. We define the error state and control as $\widetilde{\mathbf{x}} \coloneqq \mathbf{x} - \overline{\mathbf{x}}$ and $\widetilde{\mathbf{u}} \coloneqq \mathbf{u} - \overline{\mathbf{u}}$, and the measurement error as $\widetilde{\mathbf{y}} \coloneqq \mathbf{h}(\mathbf{x}) - \mathbf{h}(\overline{\mathbf{x}})$. We can linearize around the error system by computing

$$\mathbf{A}(t) = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{\substack{\mathbf{x}=\overline{\mathbf{x}} \\ \mathbf{u}=\overline{\mathbf{u}}}}, \qquad \mathbf{B}(t) = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{u}}\right|_{\substack{\mathbf{x}=\overline{\mathbf{x}} \\ \mathbf{u}=\overline{\mathbf{u}}}}, \qquad \mathbf{C}(t) = \left.\frac{\partial \mathbf{h}}{\partial \mathbf{x}}\right|_{\mathbf{x}=\overline{\mathbf{x}}}$$

so that $\dot{\widetilde{\mathbf{x}}} \approx \mathbf{A}\widetilde{\mathbf{x}} + \mathbf{B}\widetilde{\mathbf{u}}$ and $\widetilde{\mathbf{y}} \approx \mathbf{C}\widetilde{\mathbf{x}}$ for small errors. This gives us an LTV system, and we use (2.1) to can calculate the local observability Gramian. Since the linearization was performed with respect to the trajectory $\overline{\mathbf{x}}$, we can denote the local observability Gramian as $\mathbf{W}(\overline{\mathbf{x}}, \overline{\mathbf{u}}, t_f)$. Local observability of the error system along the trajectory is equivalent to there being a $t_f > 0$ such that this matrix is nonsingular.

We can also use the observability Gramian to compute the "energy" of output $\mathbf{y}$. Since the unforced state of the linearized system evolves as $\widetilde{\mathbf{x}}(t) = \mathbf{\Phi}(t, 0)\widetilde{\mathbf{x}}(0)$, and the corresponding output is $\widetilde{\mathbf{y}} = \mathbf{C}\widetilde{\mathbf{x}}$, the norm squared of $\widetilde{\mathbf{y}}$ integrated over t_f is thus

$$\int_0^{t_f} \|\widetilde{\mathbf{y}}\|^2 dt = \int_0^{t_f} \widetilde{\mathbf{x}}(0)^T \mathbf{\Phi}(t, 0)^T \mathbf{C}(t)^T \mathbf{C}(t) \mathbf{\Phi}(t, 0) \widetilde{\mathbf{x}}(0) dt \tag{2.3}$$

$$= \widetilde{\mathbf{x}}(0)^T \mathbf{W}(t_f) \widetilde{\mathbf{x}}(0). \tag{2.4}$$

The equation above is very helpful because it allows us to think of the observability Gramian as a ellipsoidal mapping, which we will discuss in Section 2.2.3.

2.2.2 The empirical observability Gramian

For some systems, the matrix $\mathbf{W}$ in (2.1) will be too expensive to compute. For other systems, the system equations may be unknown, so $\mathbf{W}$ cannot be computed. In these cases, an approximation of $\mathbf{W}$, the *empirical* local observability Gramian, can be computed as an alternative [22]. Calculating the empirical local observability Gramian involves a finite-differencing approach in which the system state is perturbed and the corresponding output is measured. This method is advantageous because it does not require knowing the exact equation of the measurement function.

The empirical local observability Gramian is calculated by computing $2n$ trajectories of the system, which we define as $\mathbf{x}^{\pm i}(t)$ for $i = 1, 2, ..., n$. These trajectories are determined from $2n$ initial states, which are calculated by independently perturbing each element of the nominal initial state $\overline{\mathbf{x}}(0)$ by an amount ϵ in both the positive and negative directions. Each of these initial conditions are then integrated based on (2.2) using the nominal control $\overline{\mathbf{u}}$. Mathematically, we define the $2n$ initial conditions as

$$\mathbf{x}^{\pm i}(0) = \overline{\mathbf{x}}(0) \pm \epsilon \mathbf{e}_i$$

for $i = 1, 2, ..., n$ where $\mathbf{e}_i \in \mathbb{R}^n$ is the i^{th} standard basis vector. We define the corresponding measurements as

$$\mathbf{y}^{\pm i}(t) = \mathbf{h}(\mathbf{x}^{\pm i}(t)) \in \mathbb{R}^q.$$

We define the difference of perturbations associated with the i^{th} state as

$$\Delta \mathbf{y}^i(t) = \mathbf{y}^{+i}(t) - \mathbf{y}^{-i}(t)$$

and the matrix formed by stacking these column vectors as

$$\Delta\mathbf{y}(t) = [\Delta\mathbf{y}^1(t) \;\; \Delta\mathbf{y}^2(t) \;\; \cdots \;\; \Delta\mathbf{y}^n(t)] \in \mathbb{R}^{q\times n}. \tag{2.5}$$

The empirical local observability Gramian can now be computed as

$$\begin{aligned}\widehat{\mathbf{W}}(\overline{\mathbf{x}}, \overline{\mathbf{u}}, t_f) &= \frac{1}{4\epsilon^2}\int_0^{t_f} (\Delta\mathbf{y})^T \Delta\mathbf{y} dt \\ &= \frac{1}{4\epsilon^2}\int_0^{t_f} \begin{bmatrix} (\Delta\mathbf{y}^1)^T\Delta\mathbf{y}^1 & (\Delta\mathbf{y}^1)^T\Delta\mathbf{y}^2 & \dots \\ (\Delta\mathbf{y}^2)^T\Delta\mathbf{y}^1 & (\Delta\mathbf{y}^2)^T\Delta\mathbf{y}^2 & \dots \\ \vdots & \vdots & \end{bmatrix} dt \in \mathbb{R}^{n\times n}.\end{aligned} \tag{2.6}$$

2.2.3 Ellipsoidal interpretation

Although we described the observability Gramian mathematically, it may not be intuitively obvious what this matrix represents. A helpful interpretation is that the observability Gramian measures how much changing the initial state changes the output measurement. This fact can be observed by noting that $\mathbf{W} \succeq 0$, and rewriting (2.4) as

$$\int_0^{t_f} \|\widetilde{\mathbf{y}}\| dt = \|\mathbf{W}^{1/2}\widetilde{\mathbf{x}}(0)\|. \tag{2.7}$$

If we consider all initial states in the unit n-sphere ($\|\widetilde{\mathbf{x}}(0)\| \leq 1$) then $\mathbf{W}^{1/2}\widetilde{\mathbf{x}}(0)$ maps this n-sphere to an ellipsoid. We can write this ellipsoid as $\mathcal{E} = \{\mathbf{W}^{1/2}\widetilde{\mathbf{x}}(0) \mid \|\widetilde{\mathbf{x}}(0)\| \leq 1\}$. To put it simply, the farther an initial state $\widetilde{\mathbf{x}}(0)$ is mapped by $\mathbf{W}^{1/2}\widetilde{\mathbf{x}}(0)$, the more that initial state will change the output function. Note that for $\mathcal{E}$, the lengths of the semi-axes are the eigenvalues of $\mathbf{W}^{1/2}$, and the volume equals $\text{vol}(\mathcal{S}^n)(\det(\mathbf{W}))^{1/2}$, where $\text{vol}(\mathcal{S}^n)$ denotes the volume of the unit n-sphere.

2.2.4 Measuring the Gramian

As the observability Gramian $\mathbf{W}$ is a matrix, we want to map it to a scalar which gives us some measure of how observable the system is. Common ways of measuring observability are as follows:

- $\det(\mathbf{W})$: (determinant) related to the volume of the ellipsoid
- $\mathrm{tr}(\mathbf{W})$: (trace) related to the average axis length of the ellipsoid
- $\lambda_{\min}(\mathbf{W})$: (minimum eigenvalue) related to the shortest axis length of the ellipsoid
- $\lambda_{\max}(\mathbf{W})/\lambda_{\min}(\mathbf{W})$: (condition number) related to the "sphericalness" of the ellipsoid

For determinant, trace, and minimum eigenvalue, larger measures are preferable as they correspond to making the ellipsoid larger (i.e., all elements of $\widetilde{\mathbf{x}}(0)$ have a large effect on the output energy). For the condition number, a smaller measure is preferable as it corresponds to making the ellipsoid more spherical (i.e., all elements of $\widetilde{\mathbf{x}}(0)$ affect the output energy similarly).

2.3 The observability Gramian for pose estimation

2.3.1 Pose of an object

Consider a rigid object in three-dimensional space. The object can be uniquely described by its position and orientation. Its position can be described by the three coordinates $x(t)$, $y(t)$, and $z(t)$, and its orientation is defined by a 3D rotation matrix belonging to the group SO(3). We define the rotation matrix as $\mathbf{R}(t) \in \mathbb{R}^{3\times3}$, which is in reference to an inertial frame (i.e., where $\mathbf{R}$ equals the identity matrix). We define a pose $\mathscr{P}(t)$ as a tuple of position and orientation,

$$\mathscr{P}(t) = (x(t), y(t), z(t), \mathbf{R}(t)).$$

We will describe the orientation $\mathbf{R}$ as a sequence of two rotations: a nominal rotation, and a rotation offset from nominal. This will allow us to define a set of orthogonal, body-fixed rotations, regardless of the nominal orientation. We define $\overline{\mathbf{R}}(t)$ as the nominal orientation, and $\mathbf{R}_B(t)$ as the body-fixed rotation from nominal, so we have $\mathbf{R} = \overline{\mathbf{R}}\mathbf{R}_B$. We will express $\mathbf{R}_B$ in terms of $\theta_x(t)$, $\theta_y(t)$, and $\theta_z(t)$, which we define as rotation angles around the body-fixed x, y and z axes. This definition will be helpful later when we compute the empirical local observability Gramian. We denote the rotation matrices around the x, y and z axes as

$$\mathbf{R}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}, \mathbf{R}_y(\alpha) = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix}, \mathbf{R}_z(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

We will apply the rotations θ_x, θ_y, and θ_z sequentially, essentially treating them as Euler angles. For simplicity, we define $\mathbf{R}_B = \mathbf{R}_x(\theta_x)\mathbf{R}_y(\theta_y)\mathbf{R}_z(\theta_z)$, although we could use any sequence of distinct angles (Tait-Bryan angles).

Now we will define the state of a pose, with respect to a nominal rotation $\overline{\mathbf{R}}$, as

$$\mathbf{x} = [x \;\; y \;\; z \;\; \theta_x \;\; \theta_y \;\; \theta_z]^T \in \mathbb{R}^6. \tag{2.8}$$

2.3.2 Camera images

We now consider camera images of the object. We consider the images to be taken from a single camera. We assume each image is n_w pixels wide, n_h pixels high, and has three color channels: red, green and blue. We will assume each color channel of each pixel is between 0 and 1. We write the image as a function of the object's coordinates in the following way: $\mathbf{h}(\mathscr{P}) \in [0,1]^{n_w \times n_h \times 3}$. Since $\mathbf{y}$ is a 3-dimensional array, it will be convenient to first define a mapping $\mathbf{vec} : \mathbb{R}^{n_w \times n_h \times 3} \to \mathbb{R}^{3 n_w n_h}$ which assembles all of the elements of the array into

a vector (for our purposes, the order of the elements in the resulting vector is irrelevant as long we use it consistently). Our vectorized image is thus

$$\mathbf{y} = \mathbf{vec}\left(\mathbf{h}(\mathscr{P})\right) \in \mathbb{R}^{3n_w n_h}. \tag{2.9}$$

The full state of a dynamic rigid body object will usually consist of $\mathbf{x}$ and its time derivative. However, since an image measurement will only depend on position and orientation, we will treat $\mathbf{x}$ as the state variable, which yields a 6-by-6 Gramian matrix. If velocity states were to be included, it would result in an augmented 12-by-12 Gramian matrix, in which the upper 6-by-6 block is the Gramian we use. With respect to a given nominal rotation $\overline{\mathbf{R}}$, it will also be helpful to denote a pose as $\mathscr{P}(\mathbf{x})$.

To compute the Gramian, we will eventually need to find the derivative of $\mathbf{h}$. Note that if we are considering photos taken from an actual camera, it is unlikely that the function $\mathbf{h}$ can be expressed mathematically. If we are considering photos rendered by a computer (which is the case in this chapter), $\mathbf{h}$ will be the result of a complex rendering engine. In this case, although it may be possible to express $\mathbf{h}$ mathematically, we will not attempt to do so. Since we are using the empirical observability Gramian, the derivative will be approximated. By using this approximation we are making the assumption that $\mathbf{h}$ is a smooth function.

Now we can apply the empirical local observability Gramian described in Section 2.2.2. In our case, the state is (2.8), the position and orientation of the object (with respect to the nominal orientation $\overline{\mathbf{R}}$). The nominal state to compute the local observability Gramian is

$$\overline{\mathbf{x}} = [\overline{x} \;\; \overline{y} \;\; \overline{z} \;\; 0 \;\; 0 \;\; 0]^T.$$

The measurement is the vectorized camera image (2.9).

We now note that we have considered a continuous-time system, but our measurement

$\mathbf{y}(t)$ is actually a video, which consists of discrete samples (image frames). With this in mind, our next step will be to fit the discrete set of measurements into our continuous framework. We do so by representing the video measurement as a zero-order hold. More specifically, at each time t we consider $\mathbf{y}(t)$ to be equal to the most recent frame in the video. Note that since $\mathbf{y}(t)$ is a function of $\mathbf{x}(t)$, this zero-order hold implicitly places the motion of the object $\mathbf{x}(t)$ under a zero-order hold as well.

We assume the video consists of n_f frames, which are acquired at times $t_0, t_1, ..., t_{n_f-1}$. We assume the video has a constant frame rate, and we define the period of time between subsequent frames as Δt. Using the zero-order hold, the Gramian in (2.1) becomes

$$\widehat{\mathbf{W}}(\overline{\mathbf{x}}, \overline{\mathbf{u}}, t_f) = \frac{1}{4\epsilon^2} \sum_{i=0}^{n_f-1} (\Delta \mathbf{y}(t_i))^T \Delta \mathbf{y}(t_i) \Delta t. \tag{2.10}$$

An illustration of how the initial perturbation matrices (2.5) of the Gramian are computed from images is shown in Fig. 2.1.

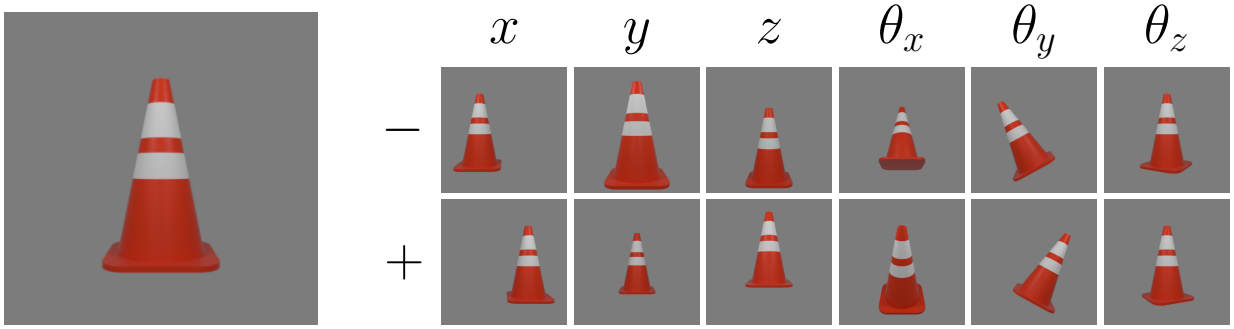


Figure 2.1: Image farthest left: nominal pose. Right 12 images: Initial perturbations ($\mathbf{x}^{\pm i}(0)$) of the nominal state to compute the Gramian. The 6 columns represent the 6 states, and the 2 rows represent negative/positive perturbation directions. The two images in each column are subtracted to form the difference vectors $\Delta \mathbf{y}^i$. The perturbation amount ϵ used to create these images is very large for illustrative purposes.

2.4 Empirical observability Gramian, special cases

We will now consider some special cases which simplify the empirical observability Gramian. More specifically, we consider cases in which the object has zero dynamics. These cases allow us to simplify the Gramian, which simplifies our analysis and allows us to build intuition.

The first case we consider is when the object has zero dynamics, but the camera is free to move. In this case, although the object does not move, the view of the object may change. Additionally, the initial perturbation of the object (with respect to the inertial frame) remains the same throughout the dynamics. An example of a camera moving around a static object is presented in Section 2.7.2.

Another case is when both the camera and object have zero dynamics. In this case, the state does not change, so the state-transition matrix simplifies to $\mathbf{\Phi}(t,0) = \mathbf{I}$. Furthermore, the measurement matrix $\mathbf{C}(t)$ becomes constant, so the Gramian simplifies to

$$\mathbf{W} = t_f \mathbf{C}^T \mathbf{C}. \tag{2.11}$$

For the empirical Gramian, the perturbation matrices $\Delta\mathbf{y}(t_i)$ are all the same, so the empirical Gramian simplifies to

$$\widehat{\mathbf{W}}(\overline{\mathbf{x}}, \overline{\mathbf{u}}, t_f) = \frac{1}{4\epsilon^2} n_f (\Delta\mathbf{y}(t_0))^T \Delta\mathbf{y}(t_0) \Delta t. \tag{2.12}$$

Although there are no dynamics in this case, the simplicity of this result is still very useful. Since the object and camera dynamics are no longer a factor, the Gramian is solely a function of a single view of the object. As we will see in Section 2.6, this allows us to examine which states of the object are better observable from a certain view, or compare observability among views.

2.5 Indistinguishable images and object symmetries

We will now make some standard definitions related to the observability of nonlinear systems, and relate them to the problem of pose estimation. In this section, we will use subscript A and subscript B to denote static quantities.

Definition 2.1. *[31] Two states $\mathbf{x}_A, \mathbf{x}_B$ are said to be **indistinguishable** (denoted $\mathbf{x}_A I \mathbf{x}_B$) for (2.2) if for every admissible input function $\mathbf{u}$, the output functions $\mathbf{h}$ of the systems with initial states $\mathbf{x}(0) = \mathbf{x}_A$ and $\mathbf{x}(0) = \mathbf{x}_B$ are identical on their common domain of definition. The system is called **observable** if $\mathbf{x}_A I \mathbf{x}_B$ implies $\mathbf{x}_A = \mathbf{x}_B$.*

Definition 2.2. *[31] The system (2.2) is called **locally observable at** $\mathbf{x}_A$ if there exists a neighborhood $\boldsymbol{W}$ of $\mathbf{x}_A$ such that for every neighborhood $\boldsymbol{V} \subset \boldsymbol{W}$ of $\mathbf{x}_A$, the relation $\mathbf{x}_A I^{\boldsymbol{V}} \mathbf{x}_B$ implies that $\mathbf{x}_B = \mathbf{x}_A$. If the system is locally observable at each $\mathbf{x}_A$ then it is called **locally observable**.*

Definition 2.1 describes the concept of indistinguishability for general nonlinear systems in terms of identical output functions. We can now apply the idea of indistinguishability to the system we are considering.

Definition 2.3. *Consider a camera in a non-dynamic environment with a fixed calibration, denoted $\mathscr{C}$ (i.e., focal length, zoom, number of pixels, etc.). Suppose this camera takes photographs of an object at two different poses, $\mathscr{P}_A$ and $\mathscr{P}_B$. We say that these poses are **image indistinguishable**, denoted $\mathscr{P}_A I^{\mathscr{C}} \mathscr{P}_B$ if the image measurements of both poses are the same, i.e., $\mathbf{h}(\mathscr{P}_A) = \mathbf{h}(\mathscr{P}_B)$. Additionally, for a given $\sigma > 0$, we say that these poses are **$\boldsymbol{\sigma}$ image indistinguishable** if they are norm-bounded by σ, i.e., $\|\mathbf{h}(\mathscr{P}_A) - \mathbf{h}(\mathscr{P}_B)\| < \sigma$.*

Definition 2.4. *Consider a pose $\mathscr{P}(\overline{\mathbf{x}}_A)$ and its associated measurement $\mathbf{h}(\mathscr{P}(\overline{\mathbf{x}}_A))$. Now consider a perturbation of that pose, $\mathbf{x}_A = \overline{\mathbf{x}}_A + [\delta x \quad \delta y \quad \delta z \quad \delta\theta_x \quad \delta\theta_y \quad \delta\theta_z]^T$. If the*

measurements of the original pose and its perturbation are image indistinguishable (i.e., $\mathscr{P}(\overline{\mathbf{x}}_A) I^{\mathscr{C}} \mathscr{P}(\mathbf{x}_A)$), then we call this perturbation an ***image indistinguishable perturbation****. Additionally, for a given $\sigma > 0$, if $\|\mathbf{h}(\mathscr{P}(\overline{\mathbf{x}}_A)) - \mathbf{h}(\mathscr{P}(\mathbf{x}_A))\| < \sigma \|\overline{\mathbf{x}}_A - \mathbf{x}_A\|$ then we call this perturbation a* $\boldsymbol{\sigma}$ ***image indistinguishable perturbation****.*

Lemma 2.1. *In the case of a static camera and object, an image indistinguishable perturbation corresponds to a vector in the null space of both the measurement matrix* $\mathbf{C}$ *and observability Gramian* $\mathbf{W}$*.*

Proof. Consider the image indistinguishable perturbation to be $\widetilde{\mathbf{x}}_A = [\delta x \ \ \delta y \ \ \delta z \ \ \delta\theta_x \ \ \delta\theta_y \ \ \delta\theta_z]^T$. We know the measurement will not change as a result, therefore $\mathbf{h}(\mathscr{P}(\overline{\mathbf{x}}_A)) - \mathbf{h}(\mathscr{P}(\overline{\mathbf{x}}_A + \widetilde{\mathbf{x}}_A)) = \mathbf{0}$. Since the perturbation is small, we have $\mathbf{h}(\mathscr{P}(\overline{\mathbf{x}}_A)) - \mathbf{h}(\mathscr{P}(\overline{\mathbf{x}}_A + \widetilde{\mathbf{x}}_A)) \approx \mathbf{C}\overline{\mathbf{x}}_A - \mathbf{C}(\overline{\mathbf{x}}_A + \widetilde{\mathbf{x}}_A) = -\mathbf{C}\widetilde{\mathbf{x}}_A$. So $\mathbf{C}\widetilde{\mathbf{x}}_A \approx \mathbf{0}$.

To show that $\widetilde{\mathbf{x}}_A$ is in the null space of the static observability Gramian, we note that from (2.11) we have $\mathbf{W} = t_f \mathbf{C}^T \mathbf{C}$. Since $\mathbf{C}\widetilde{\mathbf{x}}_A \approx \mathbf{0}$, then $\mathbf{W}\widetilde{\mathbf{x}}_A \approx 0$. □

Theorem 2.1. *In the case of a static camera and object, a rotational perturbation about an object's axis of symmetry corresponds to a vector in the null space of both the measurement matrix* $\mathbf{C}$ *and the observability Gramian* $\mathbf{W}$*.*

Proof. A rotational perturbation along an axis of symmetry is an image indistinguishable perturbation, since the resulting image measurement does not change. This vector can be denoted as $\widetilde{\mathbf{x}}_A = [0 \ \ 0 \ \ 0 \ \ \delta\theta_x \ \ \delta\theta_y \ \ \delta\theta_z]^T$. By Lemma 2.1, we know this vector is in the null space of the static matrices $\mathbf{C}$ and $\mathbf{W}$. □

Due to noise and various imperfections, two images are unlikely to be *exactly* the same, so we have defined the notion of a σ image indistinguishable perturbation. This allows us to create a numerical threshold σ which defines a degree of indistinguishability. This consideration leads to another useful description of objects: "near symmetries". A near

symmetry can be thought of as an axis of rotation for which the image of an object changes by only a small amount.

As an example, consider the traffic cone shown in Fig. 2.2. The cone technically does not have an axis of symmetry due to its square base. So when it is rotated around its z-axis, the resulting image measurement does change, but only by a small amount. As a result, the (6,6) element of the Gramian matrix (which corresponds to rotations about the z-axis) is much smaller than the other diagonal elements. Also note that while the traffic cone is not symmetric about the z axis, it is "discretely symmetric" in that it is symmetric for 90° rotations. However, it is important to note that the empirical local observability Gramian does not capture these types of discrete symmetries because it is a local measure.

2.6 Examples

We now present some simple examples which relate views of objects to their associated Gramian matrix.

2.6.1 Poses of different objects

The first example we will present considers views of different static objects and their associated Gramian matrices. In these examples, the Gramian matrices are computed by (2.12). Fig. 2.2 shows views of sphere, lamp, and traffic cone. The results show how symmetry and near-symmetry is captured by the Gramian matrix. For example, since the sphere is symmetric about all axes, the vectors $\Delta\mathbf{y}^4$, $\Delta\mathbf{y}^5$ and $\Delta\mathbf{y}^6$ in (2.5) will be approximately zero. As a result, every entry of the Gramian matrix except the upper-left 3×3 block will be zero. The lamp in Fig. 2.2 is symmetric about the z axis, so the vector $\Delta\mathbf{y}^6$ should be approximately zero, meaning the last row and column of the Gramian should be all zero. Similarly, the cone is *almost* symmetric about the z axis, which is why the last row and

column are nearly zero.

2.6.2 Comparing poses of the same object

We can also compare different views of the same object using the Gramian. Such comparison can help inform us for which view the object is the most or least observable. Figure 2.3 shows the simulation results of a model of a chair, which was viewed by a camera at different angles. More specifically, we positioned the camera at 200 orientations by considering 20 azimuthal angles on the interval $[0, 2\pi)$, and 10 elevation angles per azimuthal angle on the interval $[0, \pi/2]$. This example illustrates that that the most optimal view will depend on the measure of observability being used. For example, the most optimal view based on the determinant of the Gramian is not necessarily the same as that based on the minimum eigenvalue.

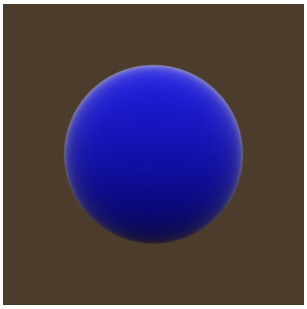
2.7 Dynamic examples

2.7.1 Dynamic object, static camera

We will now consider an example of an object with dynamics. We use the example of a cube with rigid body dynamics, which follows a projectile trajectory. The cube begins with an initial translational and angular velocities, and descends under gravity in the z-direction. We consider a trajectory of 2 seconds, and capture $n_f = 60$ frames. The results are shown in Fig. 2.4.

2.7.2 Static object, dynamic camera

In our final example, we consider a camera moving around an object along different trajectories, and calculate the Gramian along each trajectory. By doing so, we can determine which trajectories are better or worse for pose estimation. We will consider our candidate trajectories to lie on the surface of an imaginary sphere which surrounds the object. More



$$\widehat{\mathbf{W}} = 10^3 \times \begin{bmatrix} 285 & -1 & 1 & 0 & 0 & 0 \\ -1 & 37 & -75 & 0 & 0 & 0 \\ 1 & -75 & 330 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\widehat{\mathbf{W}} = 10^4 \times \begin{bmatrix} 98 & 0 & 0 & 0 & -16 & 0 \\ 0 & 20 & 13 & 7 & 0 & 0 \\ 0 & 13 & 259 & -17 & 0 & 0 \\ 0 & 7 & -17 & 25 & 0 & 0 \\ -16 & 0 & 0 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\widehat{\mathbf{W}} = 2 \times 10^3 \times \begin{bmatrix} 104 & 0 & 0 & 0 & 60 & 1 \\ 0 & 8 & 3 & -10 & 0 & 0 \\ 0 & 3 & 262 & -63 & 0 & 0 \\ 0 & -10 & -63 & 33 & 0 & 0 \\ 60 & 0 & 0 & 0 & 87 & -2 \\ 1 & 0 & 0 & 0 & -2 & 1 \end{bmatrix}$$

Figure 2.2: (Left column) Views of a sphere, lamp [10], and traffic cone, and (right column) the associated Gramian matrix for each view. In all of these examples, the camera and object are static.

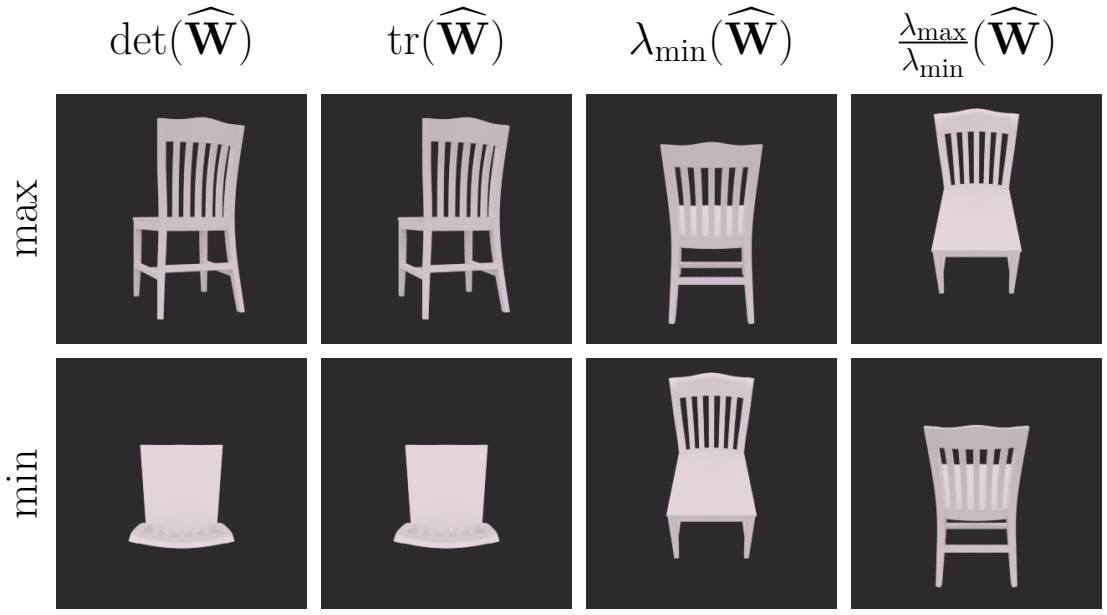


Figure 2.3: Best and worst views of a chair [21] with respect to certain measures of the observability Gramian. Note that "better" observability corresponds to larger measures of the determinant, trace and minimum eigenvalue, and smaller measures of the condition number.

specifically, we consider the trajectories to be semi-circular paths which lie on the top-half of this sphere, and have the same radius as the sphere. Using geometric terminology, we could call these paths the "great semicircles" of the upper half of the sphere. To sample these trajectories, we take the semicircle which lies on the $x-z$ plane and rotate it around the x-axis for 10 angles on the interval $[-\pi/2, \pi/2]$, then around the z-axis for 10 angles on the interval $[0, \pi]$. This yields a total of 100 trajectories. Note that all of these trajectories have equal length.

We assume the camera travels at a constant velocity along each trajectory, taking $n_f = 10$ images at a constant frame rate. The results are shown in Fig. 2.5.

$$\widehat{\mathbf{W}} = 10^3 \times \begin{bmatrix} 39 & 0 & 0 & -2 & -1 & 6 \\ 0 & 2 & -4 & 1 & 0 & 0 \\ 0 & -4 & 40 & -8 & 2 & -2 \\ -2 & 1 & -8 & 9 & 0 & 1 \\ -1 & 0 & 2 & 0 & 8 & -1 \\ 6 & 0 & -2 & 1 & -1 & 7 \end{bmatrix}$$

Figure 2.4: The observability of a cube following a projectile trajectory. The cube has rigid body dynamics with initial translational and rotational velocities, and is subject to zero torque and only the force of gravity. The image on the left was constructed by superimposing some of the frames of the nominal trajectory. The initial state is at the far bottom-left in the image. The Gramian was computed by (2.10).

2.8 Applications to pose estimation methods

While there have been many proposed methods to estimate an object's pose from an image, there is currently a lot of interest in using learning-based methods (e.g., neural networks). These methods essentially attempt to estimate pose by fitting a function to a large amount of training data. In these cases, the accuracy of the estimator will depend on how well two poses can be distinguished. With this in mind, we believe the Gramian may be a helpful metric in analyzing these techniques, as it describes a relationship between how much changing an object's pose will change the corresponding image.

So, for example, if an object has a certain axis of rotation for which the image does not change very much, it is conceivable that it would be harder to estimate rotations along that axis. Additionally, as generating synthetic data for neural networks is currently a popular idea, the observability-based metric we have presented may help inform us which poses should be more heavily included in training data.

We note that many learning-based methods such as neural networks have complex nonlin-

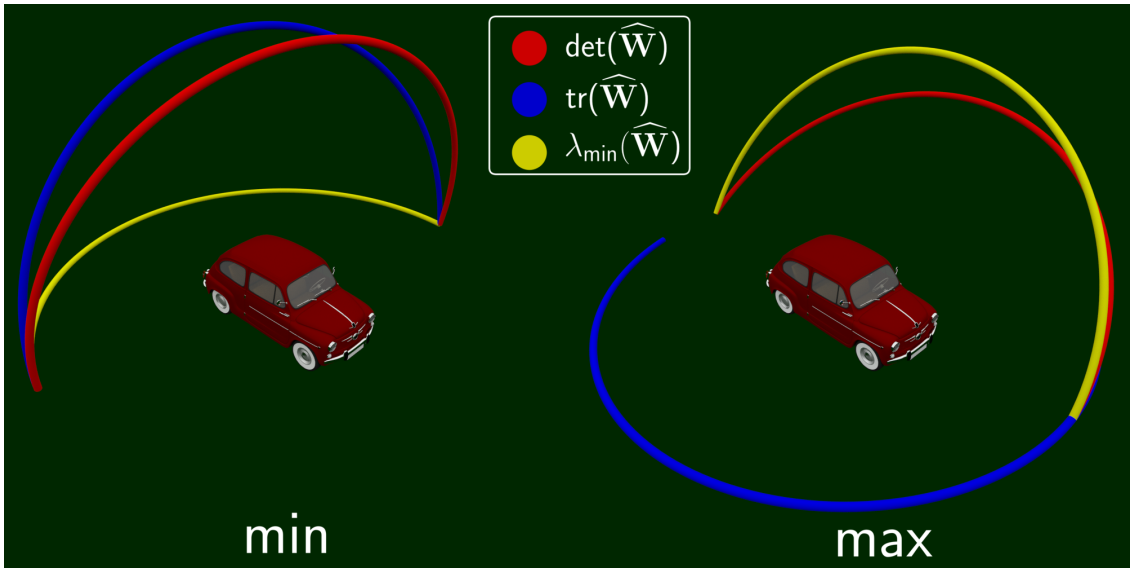


Figure 2.5: Best and worst trajectories for estimating a car's [40] pose. Each curve represents the trajectory of a camera in which the camera's view stays centered on the car. Trajectories with the lowest measures of the observability Gramian are shown on the left, and those with the highest are shown on the right.

ear functions, so the observability Gramian cannot quantify every aspect of these methods. But we do believe the Gramian's simple and fundamental nature can serve as a useful tool.

2.9 Summary

We have applied the notion of observability to the problem of estimating the pose of a known object from a camera image. As a camera image is a complex nonlinear function of an object's pose, we analyzed observability using the empirical local observability Gramian, which quantifies how perturbing an object's initial pose will change image frames of the object. By connecting observability to pose estimation, we were able to relate the notion of object symmetries to the unobservable subspace the pose vector. Additionally, as the observability Gramian gives a relative degree of observability, we were also able to express the idea of axes of "near symmetry". Using computer renderings of 3D objects, we showed

examples of how the Gramian matrix quantifies an image's sensitivity to an object's pose, and how this information can be used to help inform us which views of an object may be better for estimation. Lastly, we discussed how these results could benefit pose estimation techniques such as learning-based methods.

In the next chapter, we will remain focused on the task of image-based object pose estimation, but we will consider the case in which we are using a pose estimation neural network to estimate pose. More specifically, we will consider the scenario in which we are estimating the pose of a dynamic object using a pose estimation network. We will then apply an unscented filter to the system to mitigate inaccuracies in the network's estimates.

Chapter 3

UNSCENTED FILTERING ON POSE ESTIMATION NEURAL NETWORKS

3.1 Introduction

Over the last ten years, the area of image processing has advanced tremendously due to the use of deep neural networks [23]. These advancements have largely been driven by the task of image classification, in which a neural network attempts to determine which objects appear in an image. Part of the power of these networks is their ability to generalize. In other words, they can accurately classify images they have not seen before, and objects which have slightly different visual forms from those on which a network was trained. However, a well-known downside of neural networks is that, due to their high dimensionality and complex functions, they are poorly theoretically understood and lack any meaningful performance guarantees.

In the last few years, largely due to the success of classification networks, deep neural networks have also been applied to the task of pose estimation [53, 36, 46, 49]. Pose estimation networks usually consider the problem of estimating the pose (position and orientation) of a known object from a single camera image. Current approaches are able to achieve high accuracy, especially in favorable environments for recognition (e.g., bright lighting, and objects which are visually similar to those in the training data). However, these networks can often fail in less favorable environments, and like classification networks, lack performance guarantees.

As pose estimation is often motivated by the application of robotic manipulation, the objects of interest for pose estimation neural networks are usually non-moving. In this

chapter, we will explore a slightly different goal of estimating the pose of a dynamic object with a known dynamical model. Using this model, and estimates from the neural network, our goal is then to employ a filter to improve the network's estimates. In other words, we consider the neural network to be analogous to a sensor, and the inaccuracies of the network to be analogous to sensor noise.

We believe that our approach is advantageous because it separates the neural network and filter, and therefore preserves the standard mathematical framework of filtering. A main challenge of our approach is that, in order to apply the filter, we must characterize a highly complex neural network in terms of simple properties such as bias and covariance.

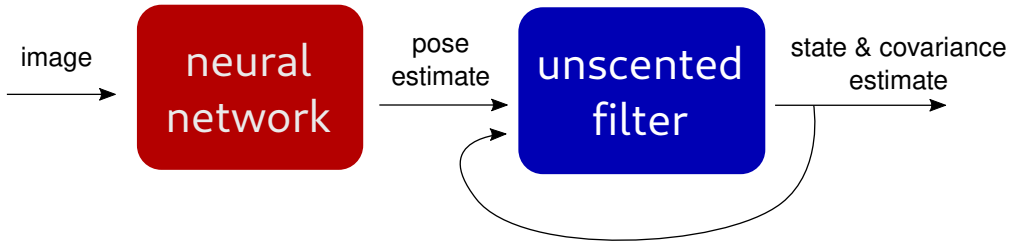


Figure 3.1: Diagram showing our estimation technique. From left to right: An image of an object is processed by a neural network to generate a pose estimate, which is then used as a measurement in an unscented filter which uses a known model of the object's dynamics.

To the best of our knowledge, no one has considered our proposed approach of applying a filter to the output of a pose estimation neural network. Other works [13, 5] have explored a related idea of combining a neural network and Kalman filter together. However, in these works, the filter is represented as layers in the network with parameters that are trained.

The remainder of this chapter is organized as follows. Section 3.2 provides background information on the group of rotation matrices SO(3), while Section 3.3 provides background information on pose estimation neural networks. In Section 3.4 we develop the unscented filter. In Section 3.5 we discuss our simulation methodology, and then characterize the

network in Section 3.6, and show example filtering simulations in Section 3.7. Finally, we present a chapter summary in Section 3.8.

3.2 Background on the group of rotation matrices: SO(3)

3.2.1 Rotation matrices

In Section 3.4 we will develop an unscented filter to estimate the state of a rigid body. In the filter, we will parameterize the rotational component of the rigid body by using the notion of the tangent space of the group of rotation matrices SO(3). Accordingly, we will now present background information on SO(3).

The group of all orthogonal 3×3 matrices is denoted as O(3). All matrices in O(3) have determinant equal to ±1. The subgroup of O(3) which has determinant equal to +1 is denoted as the "special" orthogonal group, SO(3). The group SO(3) corresponds to all 3×3 rotation matrices. In addition to being a group, SO(3) is also a Lie group, due to the fact that it is a smooth manifold for which the group operations of multiplication and inversion are smooth [29, App. A].

We consider a rotation matrix $\mathbf{R}(t) \in \mathrm{SO}(3)$ which is parameterized by t. Note that t will often represent time, but for now we will leave it as a general parameter. Dropping the explicit dependence on t, we can write the orthogonality condition as

$$\mathbf{R}\mathbf{R}^T = \mathbf{I}.$$

Differentiating with respect to t yields

$$\mathbf{R}\dot{\mathbf{R}}^T + \dot{\mathbf{R}}\mathbf{R}^T = \mathbf{0}.$$

If we move the second term in the equation above to the right hand side, we can see that the

two terms in the equation are skew-symmetric: $\mathbf{R}\dot{\mathbf{R}}^T = -(\mathbf{R}\dot{\mathbf{R}}^T)^T$. Furthermore, when $\mathbf{R}$ equals the identity matrix, then we can see that $\dot{\mathbf{R}}$ itself is skew-symmetric. In other words, the set of 3×3 skew-symmetric matrices represents the tangent space of $\mathbf{R}$ at the identity matrix $\mathbf{I}$ (which is defined as the identity element of SO(3)). With this concept in mind, the set of 3×3 skew-symmetric matrices is referred to as so(3), the Lie algebra of SO(3).

Note that a 3×3 skew-symmetric matrix is composed of three unique elements. By defining

$$\mathbf{E}_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}, \quad \mathbf{E}_2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}, \quad \mathbf{E}_3 = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

we can write any skew-symmetric matrix as the scaled sum of these three matrices: $s_1\mathbf{E}_1 + s_2\mathbf{E}_2 + s_3\mathbf{E}_3$ for scalars s_1, s_2, and s_3. In other words, $\mathbf{E}_1$, $\mathbf{E}_2$ and $\mathbf{E}_3$ define a basis for so(3). It is convenient to condense these three elements into a vector $\mathbf{s}$. In this chapter we will use lowercase $\mathbf{s}$ to denote this vector, and capital $\mathbf{S}$ to denote the corresponding skew-symmetric matrix:

$$\mathbf{s} \coloneqq \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix}, \quad \mathbf{S} \coloneqq \begin{bmatrix} 0 & -s_3 & s_2 \\ s_3 & 0 & -s_1 \\ -s_2 & s_1 & 0 \end{bmatrix}. \tag{3.1}$$

3.2.2 Exponential and logarithmic maps

We next define the exponential map, which maps an element of the tangent space $\mathbf{S} \in$ so(3) to a rotation matrix $\mathbf{R} \in$ SO(3). Note that SO(3) is a subgroup of the group of invertible matrices, for which the exponential map is just the matrix exponential, which is given by

the power series $\exp(\mathbf{X}) = \mathbf{I} + \sum_{p=1}^{\infty} \mathbf{X}^p/p!$. However, when $\mathbf{S} \in \mathrm{so}(3)$, this expansion can be reduced to a closed form, which is known as Rodrigues' formula:

$$\begin{aligned} \mathbf{R} = \exp(\mathbf{S}) &= \mathbf{I} + \frac{\sin\theta}{\theta}\mathbf{S} + \frac{1-\cos\theta}{\theta^2}\mathbf{S}^2 \\ \theta &= \|\mathbf{s}\|. \end{aligned} \tag{3.2}$$

The logarithmic map is the inverse of the exponential map, and maps a matrix $\mathbf{R} \in \mathrm{SO}(3)$ to a matrix $\mathbf{S} \in \mathrm{so}(3)$. Like the exponential map, the logarithmic map reduces to a closed form when the input is restricted to SO(3) [8]:

$$\begin{aligned} \mathbf{S} = \log(\mathbf{R}) &= \frac{\theta}{2\sin\theta}(\mathbf{R} - \mathbf{R}^T) \\ \theta &= \arccos\left(\frac{\mathrm{tr}(\mathbf{R}) - 1}{2}\right). \end{aligned}$$

Note that $\sin\theta/\theta$ and $(1-\cos\theta)/\theta^2$ in the exponential map, and $\theta/(2\sin\theta)$ in the logarithmic map are all indeterminant when $\theta = 0$. So, when θ is close to zero, care must be taken when numerically evaluating both maps.

3.2.3 Axis-angle interpretation

The components of $\mathbf{s}$ are often called exponential coordinates [29]. Additionally, the vector $\mathbf{s}$ corresponds to an axis-angle representation, which can be written as the product of an angle θ and a unit-length axis $\mathbf{e}_a \in \mathbb{R}^3$: $\mathbf{s} = \theta\mathbf{e}_a$. Note that this interpretation corresponds to our use of the of the variable "θ" in (3.2). This angle can be written as

$$\theta = \|\mathbf{s}\| = \frac{1}{\sqrt{2}}\|\mathbf{S}\|_F. \tag{3.3}$$

The angle θ represents the angle of the most direct rotation from $\mathbf{I}$ to $\exp(\mathbf{S})$. We

can also apply (3.3) to express the angular "distance" between any two rotation matrices $\mathbf{R}_1, \mathbf{R}_2 \in \mathrm{SO}(3)$. To do so, we first note that the matrix $\mathbf{R}_1\mathbf{R}_2^T$ denotes the net rotation from the $\mathbf{R}_2$ frame to the $\mathbf{R}_1$ frame. So, we can calculate the angular distance by calculating the tangent space matrix of the net rotation $\mathbf{R}_1\mathbf{R}_2^T$, and then applying (3.3):

$$\mathrm{dist}(\mathbf{R}_1, \mathbf{R}_2) = \frac{1}{\sqrt{2}}\|\log(\mathbf{R}_1\mathbf{R}_2^T)\|_F.$$

3.3 Pose estimation neural networks

In the past decade, there have been tremendous advances in the area of image processing using neural networks. These developments have largely been driven by the task of image classification, in which the goal is to determine which objects are visible in an image. More recently, interest has also been extended to not just classify objects, but to determine where in the image those objects are. This task is usually approached by determining a 2D bounding box (in pixel space) around the object.

Inspired by these advances in image classification, neural networks have recently been applied to the task of pose estimation. Many recent pose estimation networks have leveraged the architectures of classification networks in their design. Instead of determining a 2D bounding box, many pose estimation networks seek to determine the location of a 3D bounding box around the object, projected into the 2D image plane [49, 46, 36]. This box is usually described by 8 vertices in the image plane. The box's 3D location can then be found by mapping the 8 projected 2D vertices into the 3D world using a Perspective-n-Point (PnP) algorithm (which requires knowing the intrinsics of the camera which produced the image). It is important to note that many pose estimation techniques which use neural networks are not purely neural networks, and often consist the neural network itself, a PnP algorithm, and possibly other post-processing steps.

A pose estimator network outputs the pose of an object with respect to the camera's coordinate frame. We will define the position component of the pose as $\mathbf{p}$:

$$\mathbf{p} \coloneqq [x \quad y \quad z]^T$$

and rotation component as the rotation matrix $\mathbf{R}$. Thus, the pose can be denoted as $(\mathbf{p}, \mathbf{R})$.

The pose estimator we will use in this chapter is the "DOPE" (Deep Object Pose Estimation) estimator [49]. We chose to use this estimator due to its accuracy and well-supported code (which the authors have made available online). DOPE uses a neural network in combination with an algorithm to determine the pose estimate. More specifically, DOPE uses a neural network to process an image into a series of belief maps and vector fields, which are then processed by an algorithm to determine the 3D bounding box vertices. The pose can then be determined from these vertices using a PnP algorithm. More specific details regarding how we applied DOPE are described in Section 3.5.2.

3.4 Unscented filter

3.4.1 Nonlinear filters

Considering the pose estimates from a pose estimation neural network to be measurements, our next step will be to develop a filter to estimate the state of a dynamic object. In other words, the dynamics equations used in the filter will be those which describe the object's motion, and the sensor measurements will be the pose estimates provided by the neural network. As most object dynamics are nonlinear, we will use a nonlinear filtering technique.

There are several approaches for filtering nonlinear systems. One of the most popular is the extended Kalman filter (EKF), which is based on applying a standard Kalman filter to a linearization of both the system dynamics and measurement function. While this approach

is straightforward to apply, it suffers from poor accuracy and/or divergence when the system is far from the point of linearization, or when the nonlinearities are large [6, 41].

Another approach to nonlinear filtering is the unscented filter [17]. This filter is based on the unscented transform, which attempts to approximate a probability distribution by passing a set of points ("sigma points") with known mean and covariance through a nonlinear function, and then using the output points to estimate the mean and covariance of the output. The unscented filter is generally more accurate than the EKF, but involves a more complicated procedure, and may still diverge when the system's nonlinearities are large.

Another popular nonlinear filter is the particle filter. Compared to the EKF and unscented filter, the particle filter uses more of a brute force approach in which a distribution is approximated using a large number of samples ("particles"). The particle filter is appealing because, with a large enough number of particles, it can approximate an arbitrary distribution. However, applying a particle filter to systems with large number of states requires a very large number of particles, which can quickly render this technique computationally intractable.

Since we are considering a rigid object with 6 degrees of freedom, the corresponding state vector is relatively large, which makes using a standard particle filter intractable. So, we will use an unscented filter.

3.4.2 Filtering on the tangent space of SO(3)

In order to implement the unscented filter, we have to define our state vector $\mathbf{x}$. Our state will correspond to the object's position and orientation, as well as the translational and rotational velocities. While the position $\mathbf{p}$ is easy to incorporate into $\mathbf{x}$, the orientation is not. This is due to the manifold structure of the set of 3D rotations. One option to incorporate rotation into the filter would be to use an attitude parameterization such as Euler angles or

quaternions. While it would be possible to develop a filter using these parameterizations, any attitude parameterization would have inherent challenges such as angle wrap, singularities, and/or constraints. For example, Euler angles have singularities in which the three angles lose independence (i.e., gimbal lock), while quaternions require a constraint to be imposed.

We will approach the problem of parameterizing attitude by using the tangent space of SO(3). Note that other works have considered the idea of constructing an unscented filter on Riemannian manifolds [14] and Lie groups [4]. Our approach will be similar to that of [14].

In our approach, at each iteration of the filter we will keep track of our current estimate of rotation using the matrix $\widehat{\mathbf{R}}$. We will express the rotation of each sigma point with respect to the coordinate frame of $\widehat{\mathbf{R}}$, using a tangent space matrix $\mathbf{S}$. Therefore, letting $\mathbf{R}$ represent a matrix in the camera frame, we can write

$$\mathbf{R} = \widehat{\mathbf{R}} \exp(\mathbf{S}) \tag{3.4}$$

$$\mathbf{S} = \log(\widehat{\mathbf{R}}^T \mathbf{R}). \tag{3.5}$$

There are several notes to make about our approach. First, note that the tangent space parameter $\mathbf{s}$ wraps under the exponential map. In other words, $\exp(\mathbf{S}) = \exp(2\pi n \mathbf{S})$ for integers n (this can be observed from the interpretation of $\mathbf{s}$ as an axis-angle representation). However, the wrapping will not cause any difficulties in our approach, due to the fact that we define $\mathbf{s}$ with respect to $\widehat{\mathbf{R}}$, which effectively "zeros" our estimate $\hat{\mathbf{s}}$ at each iteration of the filter. Also, one advantage of using the tangent space of SO(3) is that $\mathbf{s}$ naturally corresponds to our parameterization of angular velocity, which we will define to be in the body-fixed frame.

We can now define our state as the position $\mathbf{p}$, the tangent space parameter $\mathbf{s}$, the velocity

$\dot{\mathbf{p}}$, and the angular velocity in the body-fixed frame $\boldsymbol{\omega}$:

$$\begin{aligned} \mathbf{x} &= [\mathbf{p}^T \;\; \mathbf{s}^T \;\; \dot{\mathbf{p}}^T \;\; \boldsymbol{\omega}^T]^T \\ \mathbf{y} &= [\mathbf{p}^T \;\; \mathbf{s}^T]^T. \end{aligned} \tag{3.6}$$

Again, note that the parameters $\mathbf{s}$ are defined with respect to the estimate $\widehat{\mathbf{R}}$ of rotation at each iteration of the filter. Also note that the values $\mathbf{p}$, $\dot{\mathbf{p}}$, and $\widehat{\mathbf{R}}$ are defined with respect to a camera-fixed coordinate frame. Finally, note that we will avoid using the variable $\mathbf{v}$ to denote velocity as we will use it to denote measurement noise.

3.4.3 Details of the unscented filter

The unscented filter requires a model of the measurement function and measurement noise. Since our measurement is provided by the pose estimation network, it is likely that this model is very complex and difficult to express mathematically. For this reason, we will not try to model the full complexity of the noise, but instead try to capture the noise using a generic and simple model.

We define our measurement noise using two components: $\mathbf{v}_p \in \mathbb{R}^3$ which represents the noise in the position, and $\mathbf{v}_s \in \mathbb{R}^3$ which represents noise in the rotation. Similar to our definition of $\mathbf{s}$ as the state of the filter, we will let $\mathbf{v}_s$ correspond to tangent space parameters, with a skew-symmetric representation $\mathbf{V}_S$ (as in (3.1)), and corresponding rotation matrix $\exp(\mathbf{V}_S)$. We define the position measurement $\mathbf{h}_p$ and orientation measurements $\mathbf{h}_R$ as

$$\begin{aligned} \mathbf{h}_p(\mathbf{x}) &= \mathbf{p} + \mathbf{v}_p \\ \mathbf{h}_R(\mathbf{x}, \widehat{\mathbf{R}}) &= \mathbf{R} \exp(\mathbf{V}_S). \end{aligned} \tag{3.7}$$

We define the total measurement noise as

$$\mathbf{v} \coloneqq [\mathbf{v}_p^T \quad \mathbf{v}_s^T]^T. \tag{3.8}$$

There are several variations of the unscented filter. We will use the approach of combining the state, process noise, and measurement noise into an augmented state [17, 6, 41]. Fig. 3.2 shows the steps in the unscented filter we used. Fig. 3.2 includes the following notes, which describe how the filter has to be modified in order to incorporate our parameterization of rotation:

- **note 1**: We define the expected value of $\mathbf{R}_0$ to be a function that returns a rotation matrix.

- **note 2**: At each iteration of the filter, the measurement of rotation (from the pose estimator) comes in as rotation matrix expressed in the camera frame. We need to convert this matrix into $\mathbf{s}$, so it can be incorporated into our measurement vector $\mathbf{y}$. This conversion is done using (3.5).

- **note 3**: The sigma points have been computed with respect to the parameters of the tangent space $\mathbf{s}$. However, to evaluate the dynamics and measurement functions, we need to write the rotation described by each sigma point ($\hat{\mathbf{s}}_k^{(i)}$) in the camera frame. This conversion is done using (3.4). Next, since the outputs of the dynamics and measurement function are also expressed in the camera frame, we need to convert those back into tangent parameters $\hat{\mathbf{s}}_{k+1}^{(i)}$. This conversion is done using (3.5).

- **note 4**: These are extra steps which are required for our filter. The first equation updates our rotational estimate using (3.4). Since our rotation has been updated, the second equation resets the corresponding estimate $\hat{\mathbf{s}}_{k+1}$ to zero.

3.5 Simulation methodology

3.5.1 Image rendering

In the remainder of the chapter, we will present examples using synthetically generated images. We will generate images using image rendering software, which will then be evaluated by a pre-trained pose estimation network.

We generate images using the open source image rendering program Blender. This approach is advantageous as it allows us generate realistic images, and precisely control the object's position and orientation, as well as the lighting and background of the environment.

3.5.2 Pose estimation network

As mentioned in Section 3.3, in this chapter we use the DOPE estimator. We use the code and trained checkpoints which the authors have made available online.

Note that in cases in which it is difficult to recognize the object, DOPE will not produce a pose estimate at all. For this reason, the only modification we will make to the network is reducing the thresholding parameter for object detection ("`thresh_points`" in the code). Furthermore, in cases in which the object is not detected, we will assume estimates take on default values of: $\mathbf{p} = [0 \;\; 0 \;\; 0]^T$ and $\mathbf{R} = \mathbf{I}$. This approach is disadvantageous as it gives the pose estimator a bias when many images can not be recognized. An alternative approach would be to consider the measurement to be "null" in these cases. However, with that approach, it would be unclear how to apply the unscented filter and perform the covariance calculations.

We note that this network was trained on several household objects from the YCB dataset [43] such as a cracker box, mustard bottle, and soup can. In our simulations we will use the soup can as our object of interest. To generate images, we will use the 3D model from the

constants

$$L = n + n_{\mathrm{w}} + n_{\mathrm{v}} \qquad \lambda = \alpha^2(L+\kappa) - L$$
$$\gamma = \sqrt{L+\lambda}$$

weights

$$W^{(0)}_{\mathrm{mean}} = \frac{\lambda}{L+\lambda} \qquad W^{(0)}_{\mathrm{cov}} = \frac{\lambda}{L+\lambda} + (1-\alpha^2+\beta)$$
$$W^{(i)}_{\mathrm{mean}} = W^{(i)}_{\mathrm{cov}} = \frac{1}{2(L+\lambda)}, \quad i = 1, ..., 2L$$

initialization[note 1]

$$\hat{\mathbf{x}}_0^+ = E[\mathbf{x}_0] \qquad \widehat{\mathbf{R}}_0 = E[\mathbf{R}_0]$$
$$\mathbf{P}_0^{\mathrm{xx},+} = E[(\mathbf{x}_0 - \hat{\mathbf{x}}_0^+)(\mathbf{x}_0 - \hat{\mathbf{x}}_0^+)^T]$$

covariance[note 2]

$$\widehat{\boldsymbol{\chi}}_k^+ = \begin{bmatrix} \hat{\mathbf{x}}_k^+ \\ \hat{\mathbf{w}}_k^+ \\ \hat{\mathbf{v}}_k^+ \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{x}}_k^+ \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \qquad \mathbf{P}_k^{\chi} = \begin{bmatrix} \mathbf{P}_k^{\mathrm{xx},+} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{P}_k^{\mathrm{ww}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{P}_k^{\mathrm{vv}} \end{bmatrix}$$
$$\mathbf{M}_k = \gamma \times \mathrm{cholesky}(\mathbf{P}_k^{\chi}) \qquad \boldsymbol{\sigma}_k = [\mathbf{0}_{1\times L} \quad \mathbf{M}_k \quad -\mathbf{M}_k]$$
$$\widehat{\boldsymbol{\chi}}_k^{(i)} = \begin{bmatrix} \hat{\mathbf{x}}_k^{(i)} \\ \hat{\mathbf{w}}_k^{(i)} \\ \hat{\mathbf{v}}_k^{(i)} \end{bmatrix} = \widehat{\boldsymbol{\chi}}_k^+ + \boldsymbol{\sigma}_k^{(i)}$$

iteration[note 3]

$$\hat{\mathbf{x}}_{k+1}^{(i)} = \mathbf{f}(\hat{\mathbf{x}}_k^{(i)}, \widehat{\mathbf{R}}_k, \mathbf{u}_k, \hat{\mathbf{w}}_k^{(i)}) \qquad \hat{\mathbf{y}}_{k+1}^{(i)} = \mathbf{h}(\hat{\mathbf{x}}_{k+1}^{(i)}, \widehat{\mathbf{R}}_k, \mathbf{u}_{k+1}, \hat{\mathbf{v}}_{k+1}^{(i)})$$

predictions

$$\hat{\mathbf{x}}_k^- = \sum_{i=0}^{2L} W^{(i)}_{\mathrm{mean}} \hat{\mathbf{x}}_k^{(i)} \qquad \mathbf{P}_k^{\mathrm{xx},-} = \sum_{i=0}^{2L} W^{(i)}_{\mathrm{cov}} (\hat{\mathbf{x}}_k^{(i)} - \hat{\mathbf{x}}_k^-)(\hat{\mathbf{x}}_k^{(i)} - \hat{\mathbf{x}}_k^-)^T$$
$$\hat{\mathbf{y}}_k^- = \sum_{i=0}^{2L} W^{(i)}_{\mathrm{mean}} \mathbf{y}_k^{(i)}$$

covariances

$$\mathbf{P}_k^{\mathrm{yy}} = \sum_{i=0}^{2L} W^{(i)}_{\mathrm{cov}} (\hat{\mathbf{y}}_k^{(i)} - \hat{\mathbf{y}}_k^-)(\hat{\mathbf{y}}_k^{(i)} - \hat{\mathbf{y}}_k^-)^T \qquad \mathbf{P}_k^{\mathrm{xy}} = \sum_{i=0}^{2L} W^{(i)}_{\mathrm{cov}} (\hat{\mathbf{x}}_k^{(i)} - \hat{\mathbf{x}}_k^-)(\hat{\mathbf{y}}_k^{(i)} - \hat{\mathbf{y}}_k^-)^T$$

update

$$\mathbf{e}_k^- = \mathbf{y}_k - \hat{\mathbf{y}}_k^- \qquad \mathbf{K}_k = \mathbf{P}_k^{\mathrm{xy}} (\mathbf{P}_k^{\mathrm{yy}})^{-1}$$
$$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + \mathbf{K}_k \mathbf{e}_k^- \qquad \mathbf{P}_k^{\mathrm{xx},+} = \mathbf{P}_k^{\mathrm{xx},-} - \mathbf{K}_k \mathbf{P}_k^{\mathrm{yy}} \mathbf{K}_k^T$$

rotation[note 4]

$$\widehat{\mathbf{R}}_{k+1} = \widehat{\mathbf{R}}_k \exp(\hat{\mathbf{S}}_k^+) \qquad \hat{\mathbf{s}}_{k+1}^+ = \mathbf{0}$$

Figure 3.2: Steps in our unscented filter. The accompanying notes describe extra modification steps to incorporate our parameterization of rotation.

YCB dataset.

3.6 Characterizing the pose estimation neural network

3.6.1 Approach

One of the main challenges of our approach is that we are considering pose estimates from a neural network-based estimator to be sensor measurements. Due to the high dimensionality of images, and the complex functions involved with the neural network, it is unlikely that measures such as bias and covariance can characterize the pose estimation network for all scenarios. In reality, the network's performance will depend on many factors, such as lighting, object/background contrast, and the apparent size of the object. Also, as with any neural network, the performance will be highly dependent on the training data. For these reasons, we will attempt to characterize the network by considering simple 1D motions, as well as using a Monte Carlo approach to characterize the network's mean and covariance.

3.6.2 Error and bias in 1D motions

We will first consider simple 1D motions of the object in the x (left-to-right), y (forward-to-backward), and z (down-to-up) directions. We will generate several images in each direction, run the images through the DOPE estimator to generate predictions, and compare the predictions to the truth values. We generated these images using bright lighting so that the network could generate accurate pose estimates. The results are shown in the upper three plots of Fig. 3.3.

The results show that the error in the estimator is state-dependent. This has important implications for the task of filtering. If, for example, we were to attempt to filter motion in the x direction, then we would be using positively biased estimates, which would make it difficult or impossible for the filter to improve the measurements.

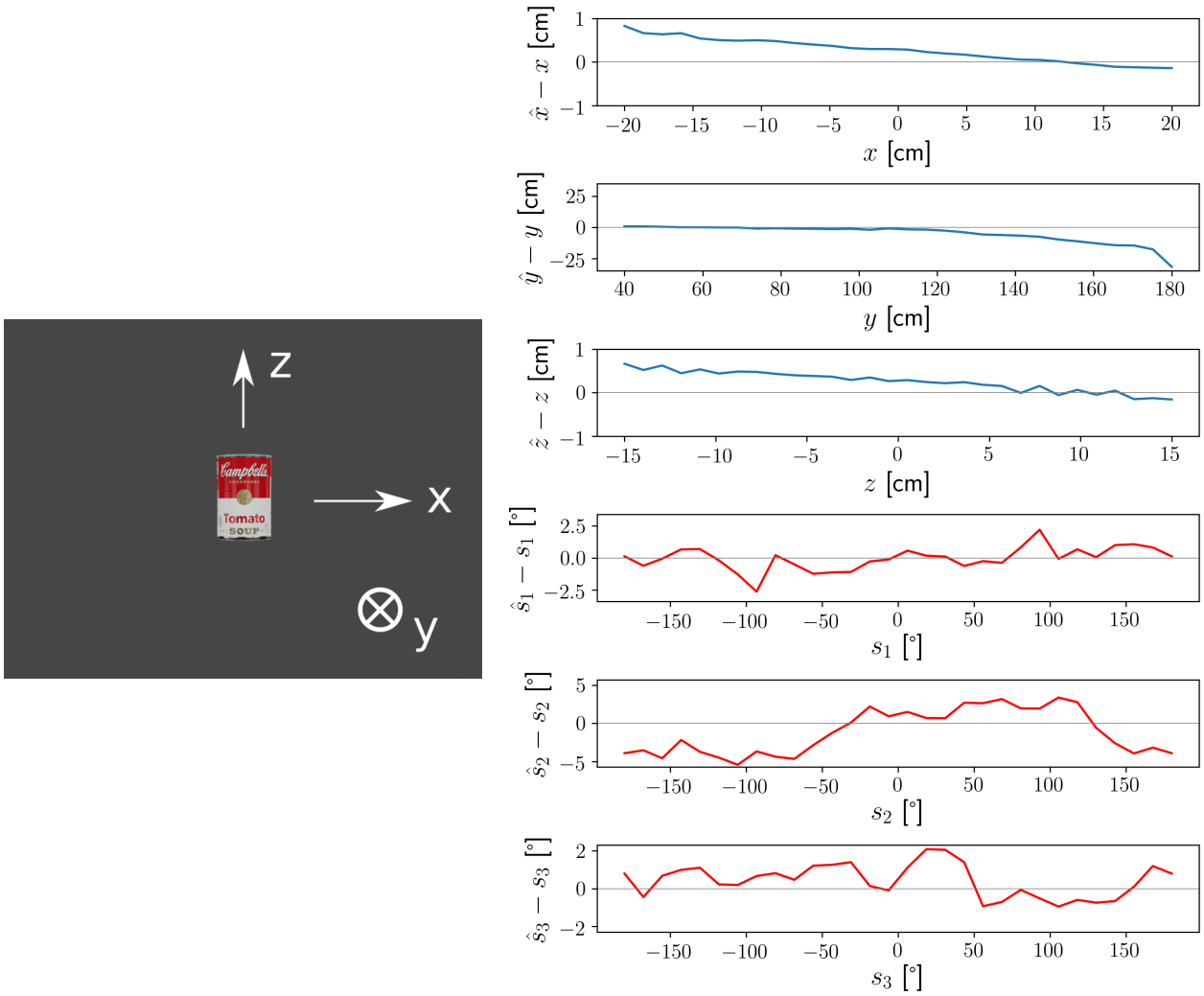


Figure 3.3: Image on the left: The default view of the soup can and coordinate axes. Plots on the right: Error for 1-dimensional translations and rotations of the soup can in the x, y and z-directions.

Next, we will consider rotating the object around each axis. In this case, the error is computed with respect to the tangent space parameters $\mathbf{s}$ in the body-fixed frame. The results, shown in the lower three plots of Fig. 3.3, show that the estimation errors are more evenly distributed around zero than the translational errors. This would lead us to expect that the filter is more likely to achieve greater gains in accuracy when filtering these rotational motions.

3.6.3 Mean and covariance of different environments

Next, we will attempt to characterize the mean and covariance of the noise in the pose estimation network with respect to certain "environments", which can be described by a specific lighting condition, background, and 3D object model. For each distinct environment, we will characterize the mean and covariance via a Monte Carlo approach using 1,000 renderings of the object at random poses. These images will then be evaluated by the DOPE estimator to generate estimates, which will then be compared to the truth values. We can then calculate the sample mean and covariance of the measurement noise $\mathbf{v}$ from (3.8) as:

$$\bar{\mathbf{v}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{v}_i$$
$$\mathbf{P}^{\mathrm{vv}} = \frac{1}{N-1}\sum_{i=1}^{N}(\mathbf{v}_i - \bar{\mathbf{v}})(\mathbf{v}_i - \bar{\mathbf{v}})^T.$$

The first example is intended to show circumstances which are conducive to good estimation: bright lighting, solid background color, high object/background contrast. The second example is the same as the first, except with dimmer lighting. The means and covariances of the measurement noise are shown in Fig. 3.4. As expected, the results show that dimmer lighting leads to worse predictions and therefore higher covariances. Also note that second diagonal element of the covariance is much larger than the first and third. This suggests that

the network has more error in detecting object depth, which corresponds to the y coordinate.

$$\bar{\mathbf{v}} = \begin{bmatrix} .20 \\ -1.20 \\ -.07 \\ -.21 \\ -1.31 \\ -.48 \end{bmatrix}, \quad \text{diag}(\mathbf{P}^{\text{vv}}) = \begin{bmatrix} 2 \\ 27 \\ 1 \\ 65 \\ 62 \\ 70 \end{bmatrix}$$

$$\bar{\mathbf{v}} = \begin{bmatrix} .13 \\ .44 \\ -.16 \\ -.04 \\ -1.46 \\ .29 \end{bmatrix}, \quad \text{diag}(\mathbf{P}^{\text{vv}}) = \begin{bmatrix} 14 \\ 979 \\ 9 \\ 198 \\ 586 \\ 230 \end{bmatrix}$$

Figure 3.4: Means and covariances of the measurement noise of the DOPE pose estimator for two different environments, computed using a Monte Carlo approach with 1,000 images per environment. Images on the left show one of the 1,000 images. The vector $\bar{\mathbf{v}}$ is the mean, and diag($\mathbf{P}^{\text{vv}}$) represents the diagonal elements of the covariance matrix. These vectors are computed with respect to the units of centimeters for translational states and degrees for rotational states.

3.7 Unscented filter simulations

We will now apply the filter to images of the soup can traveling along a trajectory. We will simulate the soup can translating and rotating through the air which would, for example, simulate if it had been tossed. Note that a full can of soup would have complex dynamics due to the interaction between the fluid and the can. However, we will consider the can to be a rigid object, which neglects the effects of the fluid, or considers a scenario in which the can is frozen or empty. The rigid body dynamics of the can object can be derived using the

Newton-Euler or Euler-Lagrange methods.

We generate a sequence of 30 images along the object's trajectory. In the filter, we use the dim lighting covariances calculated in Fig 3.4, and initial translational and angular velocity estimates which are slightly offset from the truth. Also, to make a fair comparison between the DOPE measurements and filter estimates, we set the initial pose estimate of the filter to be equal to that of the first DOPE measurement. Also, in an attempt to focus on filtering measurement noise rather than process noise, we will not include any process noise in the filter. The results are shown in Figures 3.5 and 3.6.

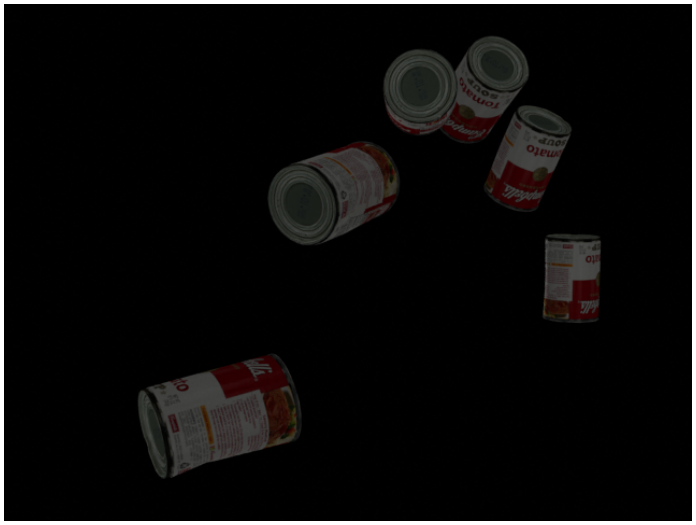


Figure 3.5: Several frames (of 30 total frames) showing a can along a projectile trajectory, superimposed into one image. The trajectory starts from the bottom-left.

The filter is able to improve the accuracy of both the position and rotation. Note that the filter does not produce lower error for every time point, but does produce a lower average error. Also, note that the 3σ bounds are relatively large for the rotational components of the state. Finally, the filter seems to have a larger error in its estimates of velocity.

As a final test, we re-ran the previous simulation 100 times with randomized initial conditions. As it is difficult to generate random trajectories that stay in frame, for each trial

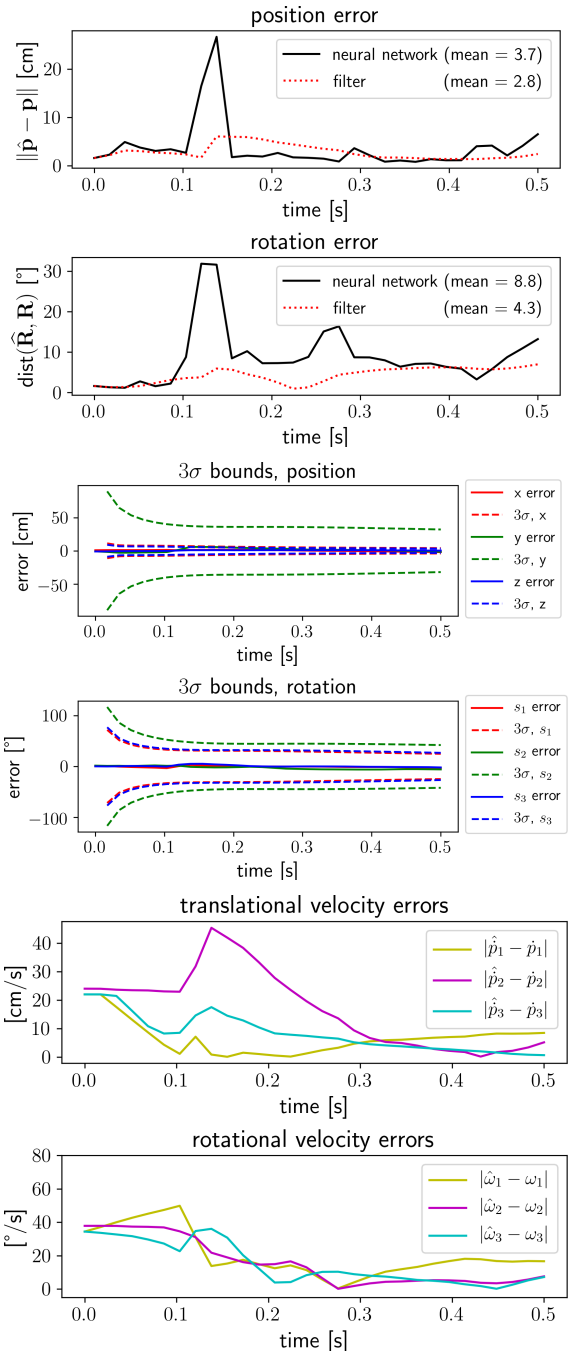


Figure 3.6: Filtering results corresponding to the trajectory shown in Fig. 3.5. Top two plots: The errors in the pose estimates of the neural network compared to those of the unscented filter. Center two plots: The 3σ bounds computed for the filter errors (computed as three times the square root of the diagonal elements of $\mathbf{P}$ in the filter). Bottom two plots: Errors in the filter's estimates of translational and rotational velocity.

we only randomized the initial orientation and initial velocity estimates (slightly offset from the truth). The results are shown in Table 3.1.

% of trials in which filter improves position estimate	**% of trials in which filter improves orientation estimate**
83%	85%

Table 3.1: Unscented filtering results for 100 randomized trajectories.

The results show that the filter improves the results most of the time. However, we note that these simulations still do not explore all possible trajectories and environmental conditions. Also, we observed that these percentages go up when the initial velocity estimates are better, and down when these estimates are worse.

3.8 Summary

We have proposed the idea of estimating the state of a dynamic rigid body by applying an unscented filter to measurements from a neural network pose estimator. By considering simple scenarios and using a Monte Carlo approach, we have shown that the noise in the network exhibits complex behavior. Despite this complexity, we have shown that even with a simple noise model, the filter still can improve the pose estimates.

Finally, we make note of some challenges of our proposed method of estimation. First, the filter generally had trouble producing accurate estimates of velocity, especially rotational velocity. Next, we noticed that making small changes to the parameters of a simulation sometimes had large effects on the filter's estimates. We attribute this high variability to both the neural network as well as the filter. Finally, we note that the covariances of our filter are computed with respect to a certain environment. So further investigation is needed if our approach is to be applied to more dynamic environments.

In the next chapter, we will mathematically analyze the sensitivity of neural networks. Our analysis will be general in the sense that it can be applied to a wide variety of neural networks, not just pose estimation networks. A main outcome of this chapter will be a method to compute a bound on the input-output sensitivity of a wide class of neural networks.

Chapter 4

ANALYTICAL BOUNDS ON THE LOCAL LIPSCHITZ CONSTANTS OF RELU NETWORKS

4.1 Introduction

Although neural networks have proven to be very adept at handling image processing tasks, they are also often very sensitive. For many networks, a small perturbation of the input can produce a huge change in the output [45]. Due to neural networks' high-dimensionality and complex constitutive functions, sensitivity is difficult to analyze, and as a result, is still not theoretically well-understood. Nevertheless, neural networks are currently being applied to a wide range of tasks, including safety-critical applications such as autonomous driving. In order to safely incorporate neural networks into the physical world, it is necessary to develop a better theoretical understanding of their sensitivity.

The high sensitivity of deep neural networks has been noted as early as [45]. This work also conceived the idea of adversarial examples, which are small perturbations to an input that cause a network to misclassify (and have since become a popular area of research in their own right [11]). Sensitivity can be characterized in a variety of ways, one of which being the input-output Jacobian [32, 44]. Although the Jacobian gives a local estimate of sensitivity, it generally cannot be used to provide any meaningful guarantees.

Another characterization of sensitivity is the Lipschitz constant, which describes how much the output of a function can change with respect to changes in the input. Lipschitz constants can be computed as a global measure which applies to any input, or as a local measure which applies only to a specific set of possible inputs. Furthermore, the Lipschitz

constant can take several forms depending on which norm is used to define it (e.g., the 1, 2, or ∞-norm).

Regardless of whether the measure is global or local, or which norm is used to define it, analytically computing the exact Lipschitz constant is challenging due to the complexity and high-dimensionality of neural networks. Although this task was approached in [16] using mixed integer programming, the resulting method can only be applied to very small networks, and only works with respect to the 1 and ∞-norms.

As exact computation of the Lipschitz constant is formidable, the next best option is to determine an upper bound. A standard upper bound on the global Lipschitz constant was described in [45], and is computed by taking the product of the Lipschitz bounds of each function in a network. This bound is simple and can be computed for larger networks, but it has the downside of being very conservative.

Recently, several studies have explored using optimization-based approaches to bound or approximate the Lipschitz constant of neural networks. The work of [38] presents two algorithms to bound the Lipschitz constant: AutoLip and SeqLip. AutoLip reduces to the global Lipschitz bound, while SeqLip is an algorithm which requires a greedy approximation for larger networks. The work of [24] presents a sparse polynomial optimization method (LiPopt) to compute bounds on Lipschitz constants, but relies on the network being sparse which often requires the network to be pruned. A semidefinite programming technique (LipSDP) is presented in [9] to compute Lipschitz bounds, but in order to apply it to larger networks, a relaxation must be used which invalidates the guarantee. Another approach is that of [54], in which linear programming is used to estimate Lipschitz constants. The downside to all of these approaches is that they usually can only be applied to small networks, and also often have to be relaxed, which invalidates any guarantee on the bound. Also of note are several other works that have considered constraining Lipschitz constants as a means to

regularize a network [12, 47, 3].

Network sensitivity is also often analyzed in regards to the robustness of classification networks against adversarial examples [48, 33, 50, 51]. This area of research is often closely related to Lipschitz analysis, but since the focus is on the specific task of adversarial examples for classification networks, it is often unclear how or if these techniques can be adapted to provide Lipschitz bounds. However, we do note that the method we present in this chapter involves a few key insights that have also been utilized in [51, 48], specifically, bounding the Lipschitz constants of the ReLU, and determining that certain output elements of a layer are always zero.

In summary, few techniques are available which can provide guaranteed Lipschitz bounds, and the ones that do only work for small networks. In this chapter, we present a method which provides guaranteed local Lipschitz bounds which can be computed for large networks. Furthermore, as Lipschitz constants are directly related to adversarial bounds, we also show how our method can produce guaranteed bounds on the minimum magnitude of adversarial examples. Our method produces a bound that is orders of magnitude tighter than the bound derived from the global Lipschitz constant (the only other bound which can be applied to large networks), so our method represents a significant improvement in certifying adversarial bounds.

Our analysis focuses on two types of functions, affine-ReLU and max pooling, which serve as the building blocks of many networks such as AlexNet [23] and the VGG networks [42].

The remainder of this chapter is organized as follows. In Section 4.2 we make a short note about which norm we consider in the chapter. In Section 4.3 we discuss global and local Lipschitz constants. In Sections 4.4, 4.5, and 4.6 we derive Lipschitz constants or bounds for ReLU, affine-ReLU, and max pooling functions, respectively. In Section 4.7 we describe how to combine our bounds to calculate the local Lipschitz constant of an entire

feedforward neural network. In Section 4.8 we discuss computational techniques which make it possible to apply our results to large networks. In Section 4.9 we apply our method to various networks, and show how our Lipschitz bounds can be used to determine bounds on adversarial examples for classification networks. The chapter concludes in Section 4.10 with a summary and possible next steps.

4.2 Note about norms

Note that in this chapter, we let $\|\cdot\|$ denote the 2-norm. However, many of our results, such as those in Section 4.3, hold for any norm. Extending our results to other norms is an area of future work.

4.3 Lipschitz constants

4.3.1 Global Lipschitz constants

In this chapter, we will analyze sensitivity using Lipschitz constants, which measure how much the output of a function can change with respect to changes in the input.

Definition 4.1. *The **global Lipschitz constant** of a function* $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ *is the minimal* $L \geq 0$ *such that*

$$\|\mathbf{f}(\mathbf{x}_2) - \mathbf{f}(\mathbf{x}_1)\| \leq L\|\mathbf{x}_2 - \mathbf{x}_1\|, \quad \forall \mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^n. \tag{4.1}$$

Note that some authors define any L that satisfies the inequality above as "a Lipschitz constant", but we will define "the Lipschitz constant" as the minimal L for which this inequality holds, and we refer to any larger value as an upper bound. We can solve for L in (4.1) as

$$L = \sup_{\mathbf{x}_1 \neq \mathbf{x}_2} \frac{\|\mathbf{f}(\mathbf{x}_2) - \mathbf{f}(\mathbf{x}_1)\|}{\|\mathbf{x}_2 - \mathbf{x}_1\|}. \tag{4.2}$$

Note that excluding points such that $\mathbf{x}_1{=}\mathbf{x}_2$ does not affect the supremization above since these points satisfy (4.1) for any L.

4.3.2 Local Lipschitz constants

While the global Lipschitz constant is computed with respect to all possible inputs in $\mathbb{R}^n$, we can also compute a local Lipschitz constant with respect to only a specific set of inputs. In this chapter, we will define the local Lipschitz constant with respect to a nominal input $\mathbf{x}_0 \in \mathbb{R}^n$ and set of all possible inputs $\mathcal{X} \subset \mathbb{R}^n$. Note that we have used the symbol "L" to denote the global Lipschitz constant, and will overload our notation and use the symbol "$L(\mathbf{x}_0, \mathcal{X})$" to denote the local Lipschitz constant.

Definition 4.2. *The **local Lipschitz constant** of a function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$, with respect to nominal input $\mathbf{x}_0 \in \mathbb{R}^n$ and set of all possible inputs $\mathcal{X} \subset \mathbb{R}^n$, is the minimal $L(\mathbf{x}_0, \mathcal{X})$ such that*

$$\|\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{x}_0)\| \leq L(\mathbf{x}_0, \mathcal{X})\|\mathbf{x} - \mathbf{x}_0\|, \qquad \forall \mathbf{x} \in \mathcal{X}. \tag{4.3}$$

As we did in (4.2), we can solve for $L(\mathbf{x}_0, \mathcal{X})$ in (4.3) which yields

$$L(\mathbf{x}_0, \mathcal{X}) \coloneqq \sup_{\substack{\mathbf{x} \in \mathcal{X} \\ \mathbf{x} \neq \mathbf{x}_0}} \frac{\|\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{x}_0)\|}{\|\mathbf{x} - \mathbf{x}_0\|}. \tag{4.4}$$

As with the global Lipschitz constant, excluding points such that $\mathbf{x}{=}\mathbf{x}_0$ does not affect the result since these points satisfy (4.3) for any $L(\mathbf{x}_0, \mathcal{X})$. In this chapter we will often leave the "$\mathbf{x}{\neq}\mathbf{x}_0$" out of the subscript (4.4) to avoid clutter.

Additionally, we note that in the special case of $\mathcal{X} = \{\mathbf{x}_0\}$, then (4.4) cannot be used in place of (4.3). In this case, we can determine that $L(\mathbf{x}_0, \mathcal{X}) = 0$ from (4.3). So in (4.4) we are implicitly assuming that $\mathcal{X} \neq \{\mathbf{x}_0\}$.

4.3.3 Properties of local Lipschitz constants

There are three properties of local Lipschitz constants that will come in handy in our analysis. The first is that the local Lipschitz constant taken with respect to set $\mathcal{X}$ is upper bounded by the local Lipschitz constant taken with respect to a superset $\mathcal{S}$ of $\mathcal{X}$. We will use this property in our analysis as we will often determine the local Lipschitz constant with respect to a bound around $\mathcal{X}$ rather than with respect to $\mathcal{X}$ itself.

Proposition 4.1. *The local Lipschitz constant taken with respect to the set $\mathcal{X}$ is upper bounded by the local Lipschitz constant taken with respect to a superset $\mathcal{S}$ of $\mathcal{X}$:*

$$L(\mathbf{x}_0, \mathcal{X}) \leq L(\mathbf{x}_0, \mathcal{S}). \tag{4.5}$$

Proof. The local Lipschitz constant $L(\mathbf{x}_0, \mathcal{S})$ is a supremization over $\mathcal{S}$ which is a superset of $\mathcal{X}$. Since $\mathcal{S}$ contains all elements of $\mathcal{X}$, the supremization over $\mathcal{S}$ results in a value at least as large as the supremization over $\mathcal{X}$, which implies (4.5). □

The second property of local Lipschitz constants is that if all possible inputs of the function $\mathbf{f}$ are within a distance ϵ of the nominal input $\mathbf{x}_0$, then all possible outputs of $\mathbf{f}$ will be within a distance $\epsilon L(\mathbf{x}_0, \mathcal{X})$ of the nominal output $\mathbf{f}(\mathbf{x}_0)$. We will use this property in our analysis to transfer input bounds from one layer of a network to the next.

Proposition 4.2. *Consider a function $\mathbf{f}$, with nominal input $\mathbf{x}_0$ and input set $\mathcal{X}$. If there exists an $\epsilon \geq 0$ such that $\|\mathbf{x} - \mathbf{x}_0\| \leq \epsilon, \forall \mathbf{x} \in \mathcal{X}$, then the deviation of $\mathbf{f}(\mathbf{x})$ from $\mathbf{f}(\mathbf{x}_0)$ is norm-bounded by $\epsilon L(\mathbf{x}_0, \mathcal{X})$ for all $\mathbf{x} \in \mathcal{X}$:*

$$\|\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{x}_0)\| \leq \epsilon L(\mathbf{x}_0, \mathcal{X}), \qquad \forall \mathbf{x} \in \mathcal{X}.$$

Proof. From (4.3) we have $\|\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{x}_0)\| \leq L(\mathbf{x}_0, \mathcal{X})\|\mathbf{x} - \mathbf{x}_0\|$. Since $\|\mathbf{x} - \mathbf{x}_0\| \leq \epsilon$, then

$\|\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{x}_0)\| \leq \epsilon L(\mathbf{x}_0, \mathcal{X})$. □

The third property of local Lipschitz constants is that a composite function is upper bounded by the product of the local Lipschitz constants of each of the composing functions. This property is often applied to global Lipschitz constants to determine global bounds of feedforward networks [45]. We now present this result for the local case.

Proposition 4.3. *Consider a function,* $\mathbf{f}$*, with nominal input* $\mathbf{x}_0$ *and input set* $\mathcal{X}$*. Assume* $\mathbf{f}$ *is the composition of functions* $\mathbf{g}$ *and* $\mathbf{h}$*, i.e.,* $\mathbf{f} = \mathbf{h} \circ \mathbf{g}$*. Define* $\mathbf{y}_0 = \mathbf{g}(\mathbf{x}_0)$ *and let* $\mathcal{Y}$ *denote the range of* $\mathbf{g}$*, i.e.,* $\mathcal{Y} = \{\mathbf{g}(\mathbf{x}) \mid \mathbf{x} \in \mathcal{X}\}$*. Let* $L_{\mathbf{f}}$*,* $L_{\mathbf{g}}$ *and* $L_{\mathbf{h}}$ *denote the local Lipschitz constants of function* $\mathbf{f}$*,* $\mathbf{g}$ *and* $\mathbf{h}$*, respectively. The following inequality holds*

$$L_{\mathbf{f}}(\mathbf{x}_0, \mathcal{X}) \leq L_{\mathbf{g}}(\mathbf{x}_0, \mathcal{X}) L_{\mathbf{h}}(\mathbf{y}_0, \mathcal{Y}). \tag{4.6}$$

Proof. We have

$$\begin{aligned}
L_{\mathbf{f}}(\mathbf{x}_0, \mathcal{X}) &= \sup_{\substack{\mathbf{x} \in \mathcal{X} \\ \mathbf{x} \neq \mathbf{x}_0}} \frac{\|\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{x}_0)\|}{\|\mathbf{x} - \mathbf{x}_0\|} \\
&= \sup_{\substack{\mathbf{x} \in \mathcal{X} \\ \mathbf{x} \neq \mathbf{x}_0}} \frac{\|\mathbf{h}(\mathbf{g}(\mathbf{x})) - \mathbf{h}(\mathbf{g}(\mathbf{x}_0))\|}{\|\mathbf{x} - \mathbf{x}_0\|} \\
&= \sup_{\substack{\mathbf{x} \in \mathcal{X} \\ \mathbf{x} \neq \mathbf{x}_0 \\ \mathbf{g}(\mathbf{x}) \neq \mathbf{g}(\mathbf{x}_0)}} \frac{\|\mathbf{g}(\mathbf{x}) - \mathbf{g}(\mathbf{x}_0)\|}{\|\mathbf{x} - \mathbf{x}_0\|} \frac{\|\mathbf{h}(\mathbf{g}(\mathbf{x})) - \mathbf{h}(\mathbf{g}(\mathbf{x}_0))\|}{\|\mathbf{g}(\mathbf{x}) - \mathbf{g}(\mathbf{x}_0)\|} \\
&\leq \sup_{\substack{\mathbf{x} \in \mathcal{X} \\ \mathbf{x} \neq \mathbf{x}_0}} \frac{\|\mathbf{g}(\mathbf{x}) - \mathbf{g}(\mathbf{x}_0)\|}{\|\mathbf{x} - \mathbf{x}_0\|} \sup_{\substack{\mathbf{y} \in \mathcal{Y} \\ \mathbf{y} \neq \mathbf{y}_0}} \frac{\|\mathbf{h}(\mathbf{y}) - \mathbf{h}(\mathbf{y}_0)\|}{\|\mathbf{y} - \mathbf{y}_0\|} \\
&= L_{\mathbf{g}}(\mathbf{x}_0, \mathcal{X}) L_{\mathbf{h}}(\mathbf{y}_0, \mathcal{Y}).
\end{aligned}$$

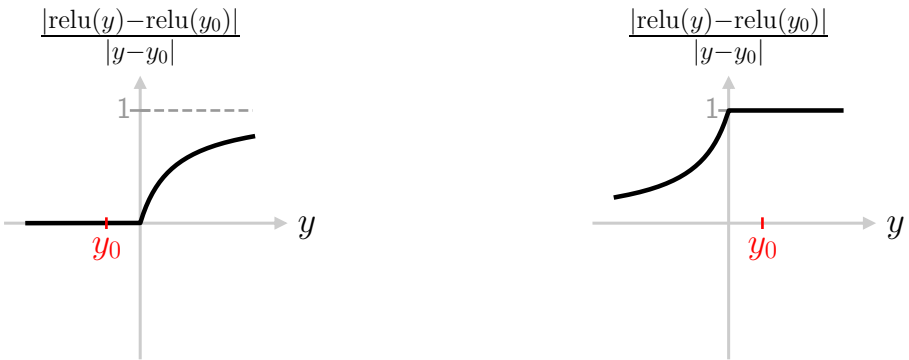
In the derivation above, points such that $\mathbf{x} \neq \mathbf{x}_0$, $\mathbf{g}(\mathbf{x}) \neq \mathbf{g}(\mathbf{x}_0)$, and $\mathbf{y} \neq \mathbf{y}_0$ can be excluded because these points can only achieve the supremum when $L_{\mathbf{f}}(\mathbf{x}_0, \mathcal{X}) = 0$, in which

case (4.6) always holds. Additionally, in the special cases that $\mathcal{X} = \{\mathbf{x}_0\}$ or $\mathcal{Y} = \{\mathbf{y}_0\}$, then $L_{\mathbf{f}}(\mathbf{x}_0, \mathcal{X}) = 0$ and (4.6) always holds. □

Using induction, the relationship in Proposition 4.3 can be applied to compositions of more than two functions. As a result, since a feedforward neural network is a composition of functions, we can bound the local Lipschitz constant of the network by the product of the local Lipschitz constants of each layer.

4.4 Local Lipschitz constants of ReLUs

The rectified linear unit (ReLU) is widely used as an activation function in deep neural networks. The ReLU is simply the maximum of an input and zero: $\text{relu}(y) = \max(0, y)$ where $\text{relu} : \mathbb{R} \to \mathbb{R}$. The ReLU can be applied to a vector by taking the ReLU of each element: $\mathbf{relu}(\mathbf{y}) = \mathbf{max}(\mathbf{0}, \mathbf{y})$ where $\mathbf{relu} : \mathbb{R}^m \to \mathbb{R}^m$. Note that in this section we will write the ReLU as a function of y rather than x to match with our notation in Section 4.5.



(a) $y_0 \leq 0$ (b) $y_0 > 0$

Figure 4.1: Diagram showing the ReLU function and its local Lipschitz fraction for (a) negative and (b) positive nominal inputs y_0. In both cases, the fraction is non-decreasing, and is therefore maximized at the largest possible y. This property is formalized in Theorem 4.1.

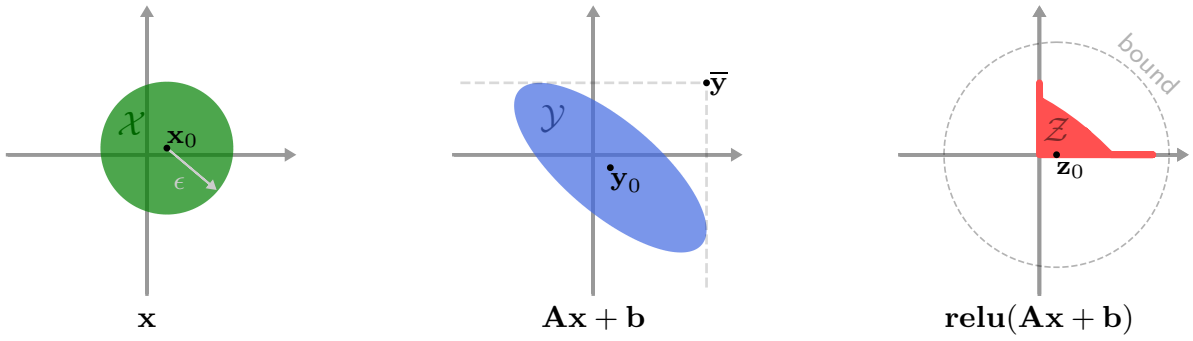


Figure 4.2: Geometric visualization of the set $\mathcal{X}$ (a ball of size ϵ) transformed through affine and ReLU functions. The set $\mathcal{X}$ is transformed by the affine transformation into the set $\mathcal{Y}$ which is transformed by the ReLU into the set $\mathcal{Z}$. The variable $\mathbf{x}_0$ is the nominal input which is transformed into $\mathbf{y}_0$ by the affine function and then into $\mathbf{z}_0$ by the ReLU function. The "bound" refers to a norm bound on the output of the ReLU, which in this chapter is computed by Theorem 4.2 and has magnitude $\epsilon\|\mathbf{RAD}\|$.

We will now determine the local Lipschitz constant of the scalar ReLU function. This result will be used in bounding the local Lipschitz constant of the affine-ReLU function.

Theorem 4.1. *Consider the ReLU function of a scalar value $y \in \mathbb{R}$. Let $y_0 \in \mathbb{R}$ denote the nominal input, let $\mathcal{Y} \subset \mathbb{R}$ denote the set of permissible inputs, and let $\bar{y} \in \mathbb{R}$ denote the largest element of $\mathcal{Y}$ such that $\bar{y} \neq y_0$. The local Lipschitz constant of the ReLU function is*

$$L(y_0, \mathcal{Y}) = \begin{cases} 0, & \mathcal{Y} = \{y_0\} \\ \frac{\text{relu}(\bar{y}) - \text{relu}(y_0)}{\bar{y} - y_0}, & \textit{otherwise.} \end{cases} \tag{4.7}$$

Proof. The first case in (4.7) follows from the fact that the local Lipschitz constant of any function is zero if the nominal input is the only permissible input. In all other cases, applying the definition of the local Lipschitz constant from (4.4) to this scenario yields

$$L(y_0, \mathcal{Y}) = \sup_{y \in \mathcal{Y}} \frac{|\text{relu}(y) - \text{relu}(y_0)|}{|y - y_0|}. \tag{4.8}$$

We refer to the fraction in the RHS of (4.8) as the "local Lipschitz fraction". The table below shows what the local Lipschitz fraction evaluates to for different signs of y and y_0:

y_0	y	$\text{relu}(y_0)$	$\text{relu}(y)$	$\frac{\lvert\text{relu}(y)-\text{relu}(y_0)\rvert}{\lvert y-y_0\rvert}$
≤ 0	≤ 0	0	0	0
≤ 0	> 0	0	y	$y/(y-y_0)$
> 0	≤ 0	y_0	0	$y_0/(y_0-y)$
> 0	> 0	y_0	y	1

Using this table, we can break this problem down into the following two cases, and show that for each case the local Lipschitz fraction is non-decreasing in y, which implies that it is maximized at $y = \bar{y}$.

Case 1: $y_0 \leq 0$ *(Fig. 4.1a)*: The local Lipschitz fraction equals 0 for non-positive y and equals $y/(y-y_0)$ for positive y. The derivative of the latter expression with respect to y is $-y_0/(y-y_0)^2$ which is non-negative since $y_0 \leq 0$. So the local Lipschitz fraction is non-decreasing in y which implies it is maximized at the largest possible $y \neq y_0$ (i.e., $\bar{y}$), so (4.7) holds in this case.

Case 2: $y_0 > 0$ *(Fig. 4.1b)*: The local Lipschitz fraction equals 1 for positive y and equals $y_0/(y_0-y)$ for non-positive y. The derivative of the latter expression with respect to y is $y_0/(y_0-y)^2$ which is positive since $y_0 > 0$. So, the local Lipschitz fraction is non-decreasing in y which implies it is maximized at the largest possible $y \neq y_0$ (i.e., $\bar{y}$), so (4.7) holds in this case as well.

□

4.5 Local Lipschitz bounds for affine-ReLU functions

4.5.1 Affine-ReLU functions

Affine functions are ubiquitous in neural networks as convolution, fully-connected, and normalization operations are all affine. An affine function can be written as

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{b} \tag{4.9}$$

where $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{x} \in \mathbb{R}^n$, and $\mathbf{b} \in \mathbb{R}^m$. The inputs and outputs of affine functions in neural networks are often multi-dimensional arrays, but it is mathematically equivalent to consider them to be 1D vectors. So in this chapter, $\mathbf{x}$ will often represent a multi-dimensional array that has been been reshaped into a 1D vector. Note that the global Lipschitz constant of an affine function is $\|\mathbf{A}\|$.

We define an affine-ReLU function as a ReLU composed with an affine function, which can be written as

$$\mathbf{z} = \mathbf{relu}(\mathbf{A}\mathbf{x} + \mathbf{b}).$$

In a neural network, affine-ReLU functions represent one layer (e.g., convolution with a ReLU activation). Note that although they are commonly used in neural networks, we are not aware of any work that has directly analyzed affine-ReLU functions except for [7].

We denote $\mathbf{x}_0$ as the nominal input to the affine-ReLU function, $\mathbf{y}_0$ as the affine transformation of $\mathbf{x}_0$, and $\mathbf{z}_0$ as the ReLU transformation of $\mathbf{y}_0$:

$$\begin{aligned} \mathbf{y}_0 &\coloneqq \mathbf{A}\mathbf{x}_0 + \mathbf{b} \\ \mathbf{z}_0 &\coloneqq \mathbf{relu}(\mathbf{y}_0) = \mathbf{relu}(\mathbf{A}\mathbf{x}_0 + \mathbf{b}). \end{aligned} \tag{4.10}$$

Recall that $\mathcal{X}$ denotes the domain of inputs, so we denote the range of the affine function as

$\mathcal{Y}$, and the range of the ReLU function as $\mathcal{Z}$ (see Fig. 4.2):

$$\begin{aligned} \mathcal{Y} &\coloneqq \{\mathbf{A}\mathbf{x} + \mathbf{b} \;\mid\; \mathbf{x} \in \mathcal{X}\} \\ \mathcal{Z} &\coloneqq \{\mathbf{relu}(\mathbf{y}) \;\mid\; \mathbf{y} \in \mathcal{Y}\}. \end{aligned} \tag{4.11}$$

4.5.2 Input set

In this chapter, we will consider the input set $\mathcal{X}$ to be in a specific mathematical form. We will use this form not just for affine-ReLU functions, but for other layers (e.g., max pooling) of a network as well.

More specifically, we consider the set $\mathcal{X}$ to be a ball of perturbations centered at the nominal input $\mathbf{x}_0$, for which certain entries of all vectors $\mathbf{x} \in \mathcal{X}$ are equal to zero. Knowledge of these entries will come from having determined that specific ReLUs of previous layers cannot be activated. Since certain dimensions of $\mathbf{x} \in \mathcal{X}$ are zero, we can think of $\mathcal{X}$ as a lower dimensional Euclidean ball embedded in a higher dimensional space (e.g., a 2D disk in three-dimensional space). We also denote perturbations about $\mathbf{x}_0$ as $\Delta\mathbf{x} \in \mathbb{R}^n$, and the maximum magnitude of these perturbations as $\epsilon \in \mathbb{R}$.

We will use a diagonal binary matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$ to ensure the zero elements of $\mathcal{X}$ are enforced. We will often refer to this matrix as a "domain-restriction matrix". Letting x_i denote the i^{th} entry of $\mathbf{x}$, we have

$$d_i \coloneqq \begin{cases} 0, & x_i = 0 \quad \forall \mathbf{x} \in \mathcal{X} \\ 1, & \text{otherwise} \end{cases} \tag{4.12}$$

$$\mathbf{D} \coloneqq \mathbf{diag}(d_1, \cdots, d_n) \tag{4.13}$$

where $\mathbf{diag} : \mathbb{R}^n \to \mathbb{R}^{n \times n}$ forms a diagonal matrix from its inputs. Note that if we do not

know any input indices to be zero, then $\mathbf{D}$ will be the identity matrix.

Using $\mathbf{D}$ and the ϵ norm constraint, we can write the input $\mathbf{x}$ and input set $\mathcal{X}$ as

$$\begin{aligned} \mathbf{x} &= \mathbf{x}_0 + \mathbf{D}\Delta\mathbf{x} \\ \mathcal{X} &= \{\mathbf{x}_0 + \mathbf{D}\Delta\mathbf{x} \;\mid\; \|\Delta\mathbf{x}\| \leq \epsilon\}. \end{aligned} \tag{4.14}$$

4.5.3 Upper bound on the affine function

We can now substitute $\mathbf{x}$ and $\mathcal{X}$ from (4.14) into the equations for $\mathbf{y}$ and $\mathcal{Y}$ from (4.9) and (4.11). Letting y_i and $y_{0,i}$ denote the i^{th} elements of $\mathbf{y}$ and $\mathbf{y}_0$, respectively, and letting $\mathbf{a}_i^T \in \mathbb{R}^n$ denote the i^{th} row of $\mathbf{A}$, we have

$$\begin{aligned} \mathbf{y} &= \mathbf{A}\mathbf{D}\Delta\mathbf{x} + \mathbf{y}_0 \\ y_i &= \mathbf{a}_i^T\mathbf{D}\Delta\mathbf{x} + y_{0,i} \\ \mathcal{Y} &= \{\mathbf{A}\mathbf{D}\Delta\mathbf{x} + \mathbf{y}_0 \;\mid\; \|\Delta\mathbf{x}\| \leq \epsilon\}. \end{aligned} \tag{4.15}$$

We now discuss how to compute an elementwise upper bound on $\mathcal{Y}$ (see center of Fig. 4.2). First, for each i, we define the set of all y_i values as

$$\mathcal{Y}_i \coloneqq \{y_i \;\mid\; \mathbf{y} \in \mathcal{Y}\}. \tag{4.16}$$

Next, define the upper bound $\bar{y}_i$ and vector $\overline{\mathbf{y}}$ as follows:

$$\bar{y}_i \coloneqq \max_{y_i \in \mathcal{Y}_i} y_i \tag{4.17}$$

$$\overline{\mathbf{y}} \coloneqq [\bar{y}_1 \quad \cdots \quad \bar{y}_m]^T \in \mathbb{R}^m. \tag{4.18}$$

The following proposition describes how to compute $\overline{\mathbf{y}}$.

Proposition 4.4. *Consider the upper bound $\bar{y}_i$ from* (4.17) *where $\mathcal{Y}_i$ is given by* (4.16)*, and y_i and $\mathcal{Y}$ are given by* (4.15)*. The equation for $\bar{y}_i$ is*

$$\bar{y}_i = \epsilon \|\mathbf{a}_i^T \mathbf{D}\| + y_{0,i}. \tag{4.19}$$

Proof. Substituting (4.15) into (4.17) yields

$$\bar{y}_i = \max_{\|\Delta \mathbf{x}\| \leq \epsilon} \mathbf{a}_i^T \mathbf{D} \Delta \mathbf{x} + y_{0,i}. \tag{4.20}$$

In this maximization, we can ignore $y_{0,i}$ because it is constant. The remaining term is the dot product of $\mathbf{D}\mathbf{a}_i$ and $\Delta \mathbf{x}$, which will be maximized when $\Delta \mathbf{x}$ points in the direction of $\mathbf{D}\mathbf{a}_i$ and has maximum magnitude of ϵ. Therefore, the value of $\Delta \mathbf{x}$ that maximizes (4.20) is $\epsilon \mathbf{D}\mathbf{a}_i / \|\mathbf{D}\mathbf{a}_i\|$. Plugging this expression into the RHS of (4.20) yields (4.19). □

4.5.4 Local Lipschitz constant upper bound

We can now derive a bound on the local Lipschitz constant of an affine-ReLU function.

Theorem 4.2. *Consider the affine-ReLU function* $\mathbf{relu}(\mathbf{A}\mathbf{x}+\mathbf{b})$ *with nominal input* $\mathbf{x}_0$*, and input set $\mathcal{X}$ from* (4.14) *with domain-restriction matrix* $\mathbf{D}$ *from* (4.13)*. Let* $\mathbf{y}$ *and* $\mathbf{y}_0$ *denote the output and nominal output of the affine function as in* (4.9) *and* (4.10)*, respectively. Let y_i and $y_{0,i}$ denote the i^{th} element of* $\mathbf{y}$ *and* $\mathbf{y}_0$*, respectively. Let $\mathcal{Y}_i$ denote the set of all y_i values for $\mathbf{y} \in \mathcal{Y}$ as in* (4.16)*, and let $\bar{y}_i$ denote the maximum of all $y_i \in \mathcal{Y}_i$ as in* (4.17)*.*

Define r_i and $\mathbf{R}$ as follows:

$$r_i \coloneqq \begin{cases} 0, & \mathcal{Y}_i = \{y_{0,i}\} \\ \frac{\text{relu}(\bar{y}_i) - \text{relu}(y_{0,i})}{\bar{y}_i - y_{0,i}}, & \text{otherwise} \end{cases} \tag{4.21}$$

$$\mathbf{R} \coloneqq \mathbf{diag}(r_1, \cdots, r_m).$$

The following is an upper bound on the affine-ReLU function's local Lipschitz constant:

$$L(\mathbf{x}_0, \mathcal{X}) \leq \|\mathbf{RAD}\|. \tag{4.22}$$

Proof. We start by considering the output of the affine function on an elementwise basis. We can use y_i, $y_{0,i}$, $\mathcal{Y}_i$, and $\bar{y}_i$ in place of y, y_0, $\mathcal{Y}$, and $\bar{y}$ (respectively) in Theorem 4.1. Applying the theorem, we can see that r_i in (4.21) is the local Lipschitz constant from (4.7), so using (4.3) we have

$$r_i |y_i - y_{0,i}| \geq |\text{relu}(y_i) - \text{relu}(y_{0,i})|. \tag{4.23}$$

Note that in (4.17) we have not assumed that $y_i \neq y_{0,i}$. We do not have to make this assumption because using the expression for y_i from (4.15) we can see that since $\|\Delta\mathbf{x}\| \leq \epsilon$, then $\mathcal{Y}_i$ will always be an interval centered at $y_{0,i}$, so the only way the maximum y_i will be $y_{0,i}$ is when $\mathcal{Y}_i = \{y_{0,i}\}$, in which case r_i is computed without $\bar{y}_i$.

Taking (4.23) and stacking it into an elementwise vector equation for all i yields

$$|\mathbf{relu}(\mathbf{Ax} + \mathbf{b}) - \mathbf{relu}(\mathbf{Ax}_0 + \mathbf{b})| \leq \mathbf{R}|\mathbf{Ax} + \mathbf{b} - (\mathbf{Ax}_0 + \mathbf{b})|.$$

Noting that the inequality above holds elementwise, and that $\mathbf{R}$ is a diagonal matrix with

non-negative entries, we have

$$\|\mathbf{relu}(\mathbf{A}\mathbf{x}+\mathbf{b})-\mathbf{relu}(\mathbf{A}\mathbf{x}_0+\mathbf{b})\| \leq \|\mathbf{R}(\mathbf{A}\mathbf{x}+\mathbf{b}-(\mathbf{A}\mathbf{x}_0+\mathbf{b}))\|.$$

Next, we substitute the equation above into the definition of the local Lipschitz constant from (4.4) as follows:

$$\begin{aligned}
L\left(\mathbf{x}_0, \mathcal{X}\right) &= \sup_{\mathbf{x}\in\mathcal{X}} \frac{\|\mathbf{relu}(\mathbf{A}\mathbf{x}+\mathbf{b})-\mathbf{relu}(\mathbf{A}\mathbf{x}_0+\mathbf{b})\|}{\|\mathbf{x}-\mathbf{x}_0\|} \\
&\leq \sup_{\mathbf{x}\in\mathcal{X}} \frac{\|\mathbf{R}(\mathbf{A}\mathbf{x}+\mathbf{b}-(\mathbf{A}\mathbf{x}_0+\mathbf{b}))\|}{\|\mathbf{x}-\mathbf{x}_0\|} \\
&= \sup_{\mathbf{x}\in\mathcal{X}} \frac{\|\mathbf{R}(\mathbf{A}\mathbf{x}-\mathbf{A}\mathbf{x}_0)\|}{\|\mathbf{x}-\mathbf{x}_0\|} \\
&= \max_{\|\Delta\mathbf{x}\|\leq\epsilon} \frac{\|\mathbf{RAD}\Delta\mathbf{x}\|}{\|\mathbf{D}\Delta\mathbf{x}\|} \\
&= \max_{\|\Delta\mathbf{x}\|\leq\epsilon} \frac{\|\mathbf{RADD}\Delta\mathbf{x}\|}{\|\mathbf{D}\Delta\mathbf{x}\|} \\
&\leq \max_{\|\Delta\mathbf{x}\|\leq\epsilon} \frac{\|\mathbf{RAD}\|\|\mathbf{D}\Delta\mathbf{x}\|}{\|\mathbf{D}\Delta\mathbf{x}\|} \\
&= \|\mathbf{RAD}\|.
\end{aligned}$$

In the derivation above, we have used (4.14) and the fact that $\mathbf{D} = \mathbf{DD}$ since $\mathbf{D}$ is a diagonal binary matrix □

Note that since $\mathbf{R}$ and $\mathbf{D}$ are both diagonal binary matrices, it follows from Theorem 4.2 that $L(\mathbf{x}_0, \mathcal{X}) \leq \|\mathbf{A}\|$. This result provides a global Lipschitz bound that was mentioned as early as [45]. In this chapter, we will often refer to network-wide bounds found using $\|\mathbf{A}\|$ as the "global" bound.

4.6 Lipschitz constants of max pooling functions

4.6.1 Max pooling functions

Max pooling is ubiquitous in neural networks, so in order to compute the local Lipschitz constant of a full network, we need to compute the local or global Lipschitz constant of the max pooling function.

$$\mathbf{M}(\mathbf{x})\mathbf{x} = \overbrace{\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}}^{\substack{\text{kernel} \\ \text{size}}\quad\quad\substack{\text{stride} \\ \text{size}}} \begin{bmatrix} 1 \\ 1 \\ 4 \\ 2 \\ 3 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 4 \\ 3 \end{bmatrix}$$

Figure 4.3: A 1D max pooling operation with kernel size $k = 3$ and stride size $s = 2$, shown in matrix form. Note that max pooling can be mathematically described as a piecewise linear matrix operation. Each row in $\mathbf{M}(\mathbf{x})$ corresponds to one pooling region (and one output element), and each column corresponds to one input element. The red elements in this diagram represent the elements in each pooling region.

Max pooling is a downsampling operation which involves sliding a small window, which we will call a "kernel", across an input array, and taking the maximum value within the kernel every time the kernel is in place. The amount that the kernel is shifted in each direction is called the "stride". For each placement of the kernel, the elements of the input array within the kernel are called the "pooling region". If the stride size is less than the kernel size in any dimension, then some input elements will be part of more than one pooling region, and the max pooling function is called "overlapping". Otherwise, each input element will appear in a maximum of one pooling region, and the function is called "non-overlapping".

Although max pooling functions typically operate on multi-dimensional arrays, as we did

with affine functions in Section 4.5.1, without loss of generality we can consider the input and output of the max pooling function to be vectors. Max pooling is piecewise linear operation and can be written as $\mathbf{f}(\mathbf{x}) = \mathbf{M}(\mathbf{x})\mathbf{x}$ where $\mathbf{M}(\mathbf{x}) \in \mathbb{R}^{m \times n}$ is a binary matrix (see Fig. 4.3).

4.6.2 Lipschitz constants of piecewise linear functions

Computing the Lipschitz constant of a max pooling function requires the following result for the Lipschitz constant of a vector-valued continuous piecewise linear function.

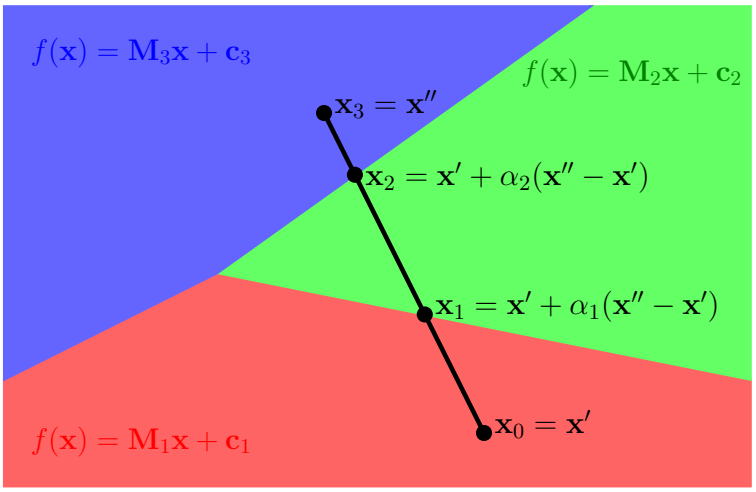


Figure 4.4: Diagram of the domain of a piecewise linear function in 2D. The different colors represent different linear regions, each of which has a constant scaling matrix, $\mathbf{M}$ and bias vector, $\mathbf{c}$. The points $\mathbf{x}'$ and $\mathbf{x}''$ are the start and end points of the line segment, and $\mathbf{x}_1$ and $\mathbf{x}_2$ are the internal points on the line segment for which the linear regions change.

Lemma 4.1. *Consider a vector-valued, continuous piecewise linear function* $\mathbf{f}(\mathbf{x}) = \mathbf{M}(\mathbf{x})\mathbf{x} + \mathbf{c}(\mathbf{x})$ *where* $\mathbf{M}(\mathbf{x}) \in \mathbb{R}^{m \times n}$ *and* $\mathbf{c}(\mathbf{x}) \in \mathbb{R}^m$*. The global Lipschitz constant of* $\mathbf{f}$ *is the maximum norm of all matrices* $\mathbf{M}(\mathbf{x})$*:*

$$L = \max_{\mathbf{x}} \|\mathbf{M}(\mathbf{x})\|.$$

Proof. Let $\mathbf{x}' \in \mathbb{R}^n$ and $\mathbf{x}'' \in \mathbb{R}^n$ denote two arbitrary points. Using Fig. 4.4 as a guide, a

line segment from $\mathbf{x}'$ to $\mathbf{x}''$ will travel through some number p different linear regions, which we will reference using indices $i = 1, ..., p$. Denote the associated scaling matrix and bias of linear region i as $\mathbf{M}_i$ and $\mathbf{c}_i$ respectively. Define $\Delta\mathbf{x} \coloneqq \mathbf{x}'' - \mathbf{x}'$ and denote the points on the boundary of the linear regions as $\mathbf{x}_i \coloneqq \mathbf{x}' + \alpha_i \Delta\mathbf{x}$ where $0 \leq \alpha_i \leq 1$. Let $\alpha_0 = 0$ and $\alpha_p = 1$ so that $\mathbf{x}_0 = \mathbf{x}'$ and $\mathbf{x}_p = \mathbf{x}''$, and also define $\Delta\alpha_i \coloneqq \alpha_i - \alpha_{i-1}$ for $i = 1, .., p$. Note that $\sum_{i=1}^{p} \Delta\alpha_i = 1$. The difference in the function $\mathbf{f}$ across the i^{th} linear region of the line segment can then be written as

$$\begin{aligned}
\mathbf{f}(\mathbf{x}_i) - \mathbf{f}(\mathbf{x}_{i-1}) &= \mathbf{M}_i\mathbf{x}_i + \mathbf{c}_i - (\mathbf{M}_i\mathbf{x}_{i-1} + \mathbf{c}_i) \\
&= \mathbf{M}_i(\mathbf{x}_i - \mathbf{x}_{i-1}) \\
&= \Delta\alpha_i \mathbf{M}_i \Delta\mathbf{x}.
\end{aligned}$$

We can write $\mathbf{f}(\mathbf{x}'') - \mathbf{f}(\mathbf{x}')$ as the sum of differences in $\mathbf{f}$ across each linear region. Noting that $\mathbf{f}(\mathbf{x}_0) = \mathbf{f}(\mathbf{x}')$ and $\mathbf{f}(\mathbf{x}_p) = \mathbf{f}(\mathbf{x}'')$ we have

$$\begin{aligned}
\mathbf{f}(\mathbf{x}'') - \mathbf{f}(\mathbf{x}') &= \sum_{i=1}^{p} \mathbf{f}(\mathbf{x}_i) - \mathbf{f}(\mathbf{x}_{i-1}) \\
&= \sum_{i=1}^{p} \Delta\alpha_i \mathbf{M}_i \Delta\mathbf{x}.
\end{aligned}$$

Next, we plug the equation above into the definition of the Lipschitz constant in (4.2). Note that in (4.2), the points $\mathbf{x}_1$ and $\mathbf{x}_2$ denote any points in $\mathbb{R}^n$, but in this theorem we are using $\mathbf{x}'$ and $\mathbf{x}''$ to denote any points in $\mathbb{R}^n$, and $\mathbf{x}_1$ and $\mathbf{x}_2$ to denote points on the line segment

between $\mathbf{x}'$ and $\mathbf{x}''$:

$$\begin{aligned}
L &= \sup_{\mathbf{x}'\neq\mathbf{x}''} \frac{\|\mathbf{f}(\mathbf{x}'') - \mathbf{f}(\mathbf{x}')\|}{\|\mathbf{x}'' - \mathbf{x}'\|} \\
&= \sup_{\mathbf{x}'\neq\mathbf{x}''} \frac{\|\sum_{i=1}^{p} \Delta\alpha_i \mathbf{M}_i \Delta\mathbf{x}\|}{\|\Delta\mathbf{x}\|} \\
&\leq \sup_{\mathbf{x}'\neq\mathbf{x}''} \frac{\sum_{i=1}^{p} \|\Delta\alpha_i \mathbf{M}_i \Delta\mathbf{x}\|}{\|\Delta\mathbf{x}\|} \\
&\leq \sup_{\mathbf{x}'\neq\mathbf{x}''} \frac{\sum_{i=1}^{p} \Delta\alpha_i \|\mathbf{M}_i\| \|\Delta\mathbf{x}\|}{\|\Delta\mathbf{x}\|} \\
&= \sup_{\mathbf{x}'\neq\mathbf{x}''} \sum_{i=1}^{p} \Delta\alpha_i \|\mathbf{M}_i\| \\
&\leq \max_{\mathbf{x}} \|\mathbf{M}(\mathbf{x})\|.
\end{aligned}$$

In the last step we used the fact that $\sum_{i=1}^{p} \Delta\alpha_i = 1$.

We have shown that $L \leq \max_{\mathbf{x}} \|\mathbf{M}(\mathbf{x})\|$ so we can complete the proof by showing that $L \geq \max_{\mathbf{x}} \|\mathbf{M}(\mathbf{x})\|$. Let $\mathbf{M}^*$ denote the $\mathbf{M}$ matrix of the linear region associated with $\max_{\mathbf{x}} \|\mathbf{M}(\mathbf{x})\|$, i.e., $\max_{\mathbf{x}} \|\mathbf{M}(\mathbf{x})\| = \|\mathbf{M}^*\|$. Let $\mathbf{x}'^*$ and $\mathbf{x}''^*$ denote any two points in the linear region of the $\mathbf{M}^*$ such that $\|\mathbf{M}^*\| = \|\mathbf{M}(\mathbf{x}''^* - \mathbf{x}'^*)\| / \|\mathbf{x}''^* - \mathbf{x}'^*\|$. We have

$$\begin{aligned}
L &= \sup_{\mathbf{x}'\neq\mathbf{x}''} \frac{\|\mathbf{f}(\mathbf{x}'') - \mathbf{f}(\mathbf{x}')\|}{\|\mathbf{x}'' - \mathbf{x}'\|} \\
&\geq \frac{\|\mathbf{f}(\mathbf{x}''^*) - \mathbf{f}(\mathbf{x}'^*)\|}{\|\mathbf{x}''^* - \mathbf{x}'^*\|} \\
&= \frac{\|\mathbf{M}^*(\mathbf{x}''^* - \mathbf{x}'^*)\|}{\|\mathbf{x}''^* - \mathbf{x}'^*\|} \\
&= \|\mathbf{M}^*\| \\
&= \max_{\mathbf{x}} \|\mathbf{M}(\mathbf{x})\|.
\end{aligned}$$

We have shown that $L \leq \max_{\mathbf{x}} \|\mathbf{M}(\mathbf{x})\|$ and $L \geq \max_{\mathbf{x}} \|\mathbf{M}(\mathbf{x})\|$ so $L = \max_{\mathbf{x}} \|\mathbf{M}(\mathbf{x})\|$. □

We will use this lemma to determine the Lipschitz constant of a max pooling function. Note that ReLUs are also piecewise linear, so we could use this result to determine the global Lipschitz constant of the ReLU (which equals one).

4.6.3 Lipschitz constants of max pooling functions

If a max pooling function is non-overlapping, then it will have a Lipschitz constant of one. However, if it is overlapping, the Lipschitz constant will be larger due to the fact that an input element can map to multiple places in the output (note that this fact is sometimes overlooked in the literature). We present the global Lipschitz constant of a general overlapping or non-overlapping max pooling function in the following theorem.

Theorem 4.3. *Consider a max pooling function. Let $n_{\max}$ denote the maximum number of pooling regions that any input element is part of. The global Lipschitz constant of the max pooling function is:*

$$L = \sqrt{n_{\max}}. \tag{4.24}$$

Proof. Without loss of generality, we can consider the input and output of the max pooling function to be vectors in $\mathbb{R}^n$ and $\mathbb{R}^m$, respectively, so we can represent max pooling as a matrix operation. The max pooling function is piecewise linear and can be written as $\mathbf{f}(\mathbf{x}) = \mathbf{M}(\mathbf{x})\mathbf{x}$ where $\mathbf{M}(\mathbf{x}) \in \mathbb{R}^{m\times n}$ is a binary matrix (see Fig. 4.3).

Let $\mathbf{m}_j(\mathbf{x}) \in \mathbb{R}^m$ denote the j^{th} column of $\mathbf{M}$ and let $m_{ij}(\mathbf{x}) \in \mathbb{R}$ denote the $(i,j)^{th}$ entry of $\mathbf{M}(\mathbf{x})$. Dropping the explicit dependence on $\mathbf{x}$, we have $\mathbf{M} = [\mathbf{m}_1 \ \cdots \ \mathbf{m}_n]$. Each row of $\mathbf{M}$ represents one pooling region, and will only have a single 1, with all other values being zero. Each column of $\mathbf{M}$ represents a particular input, and the number of occurrences of the value 1 in any column represents the number of pooling regions that input is the maximum for. Since any input can be the maximum for all of its pooling regions, the maximum possible

number occurrences of the value 1 in any column of all matrices $\mathbf{M}(\mathbf{x})$ can be expressed as

$$n_{\max} = \max_{\mathbf{x}} \left(\max_{j} \mathbf{m}_j^T \mathbf{m}_j \right). \tag{4.25}$$

Since each row contains a single 1, the columns $\mathbf{m}_j$ are orthogonal so $\mathbf{M}^T\mathbf{M} = \mathbf{diag}(\mathbf{m}_1^T\mathbf{m}_1, \cdots, \mathbf{m}_n^T\mathbf{m}_n)$. Since $\mathbf{M}^T\mathbf{M}$ is diagonal, its singular values are $\mathbf{m}_j^T\mathbf{m}_j$ which implies $\|\mathbf{M}^T\mathbf{M}\| = \max_j \mathbf{m}_j^T\mathbf{m}_j$ and $\|\mathbf{M}\| = \max_j \sqrt{\mathbf{m}_j^T\mathbf{m}_j}$. Using (4.25) and Lemma 4.1, we have $L = \max_{\mathbf{x}} \|\mathbf{M}(\mathbf{x})\| = \sqrt{n_{\max}}$.

□

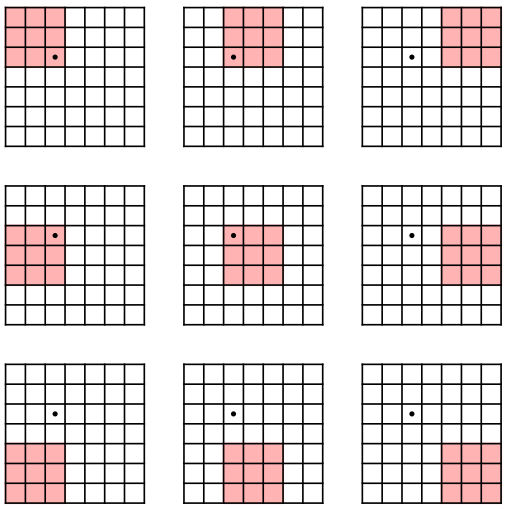


Figure 4.5: Illustration of a 2D max pooling function. Each of the nine grids represent the same input array, but with the kernel (red square) placed in a different location. The dot represents one particular input, which is covered the maximum number of times by the kernel. In this example, the kernel size is $k = 3$, the stride size is $s = 2$, and the number of strides required to move the kernel to a completely new set of inputs is $c = \text{ceil}(k/s) = 2$ (see Proposition 4.5). The maximum number of pooling regions that any input can be a part of is $n_{\max} = c^2 = 4$.

Next, we show how the value $n_{\max}$ can be computed.

Proposition 4.5. *Consider a 2D max pooling function with a dilation of one, and with kernel size k and stride size s in each dimension. Let $n_{\max}$ denote the maximum number of pooling regions that any input can be part of. The value of $n_{\max}$ is given by the following equation:*

$$n_{\max} = \text{ceil}(k/s)^2. \tag{4.26}$$

Proof. We start by considering this problem in each of the two dimensions independently. Let c denote the maximum number of times any input can be covered by different placements of the kernel, which also corresponds to the number of the strides required to move the kernel to an entirely different set of inputs (see Fig. 4.5). The values k, s, and c are related by the equation $cs \geq k$. Since c must be an integer, it can be determined with the equation $c = \text{ceil}(k/s)$.

Note that c represents the maximum number of kernel placements in each dimension that can cover a particular input. Since the kernel moves in strides along a 2D grid, $c^2 = \text{ceil}(k/s)^2$ represents the maximum number of kernel placements over both dimensions that can cover a particular input, which is equivalent to $n_{\max}$.

□

Note that this result can easily be generalized to the case in which the max pooling function has different kernel sizes and/or stride sizes in each dimension.

4.7 Network-wide bounds

4.7.1 Summary of Lipschitz constants and bounds

We have derived Lipschitz constants and Lipschitz bounds for several functions, most of which describes a single layer of a network. These bounds are summarized in Table 4.1. In this section we will describe how to combine these bounds to determine a network-wide

bound.

function	**global/local**	**exact/ upper bound**	**value**
affine	global	exact	$\|\mathbf{A}\|$
ReLU	global	exact	1
ReLU	local	exact	$\frac{\text{relu}(\bar{y})-\text{relu}(y_0)}{\bar{y}-y_0}$
affine-ReLU	global	exact	$\|\mathbf{A}\|$
affine-ReLU	local	upper bound	$\|\mathbf{RAD}\|$
max pooling	global	exact	$\sqrt{n_{\max}}$

Table 4.1: Summary of the Lipschitz constants and bounds derived and discussed in this chapter, whether they are global or local measures, and whether they are exact Lipschitz constants or upper bounds. The equation for the local Lipschitz constant of ReLU functions assumes $\mathcal{Y} \neq \{y_0\}$.

4.7.2 Determining the zero output indices of a layer

As we mentioned in Section 4.5.2, we consider the input set $\mathcal{X}$ of each layer of a network to be a function of a bound ϵ on the inputs, and a domain-restriction matrix $\mathbf{D}$. Given the current layer of a network, we now describe how to determine $\mathbf{D}$ for the following layer.

For affine-ReLU functions, our goal is to determine which entries of the output set $\mathcal{Z}$ are zero. The zero elements of the vectors $\mathbf{z} \in \mathcal{Z}$ can easily be determined from the upper bound vector $\overline{\mathbf{y}}$. Since $\overline{\mathbf{y}}$ is an upper bound on $\mathbf{y} \in \mathcal{Y}$, if $\bar{y}_i \leq 0$ then $y_i \leq 0$ for all $\mathbf{y} \in \mathcal{Y}$, which implies $z_i = \text{relu}(y_i) = 0$ for all $\mathbf{z} \in \mathcal{Z}$. Therefore, given an affine-ReLU function, we can

form the domain-restriction matrix $\mathbf{D}^{\text{next}} \in \mathbb{R}^{m \times m}$ of the next layer as follows:

$$d_i^{\text{next}} = \begin{cases} 0, & \bar{y}_i \leq 0 \\ 1, & \bar{y}_i > 0 \end{cases}$$
$$\mathbf{D}^{\text{next}} = \mathbf{diag}(d_1^{\text{next}}, \cdots, d_m^{\text{next}}). \tag{4.27}$$

We can also determine the zero output indices of max pooling layers by noting that if all inputs to a particular pooling region are known to be zero, then the output must be zero. We can efficiently determine these indices by letting $\mathbf{d} \in \mathbb{R}^n$ denote the diagonal elements of the input domain-restriction matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$. Each of these elements is a binary value indicating whether the i^{th} element of all inputs equals zero. Therefore, if we plug this vector into the max pooling function, then each output will equal zero if and only if all inputs in its pooling region are zero, and will equal one otherwise. Therefore, we can use these outputs to form the diagonal elements of the domain-restriction matrix for the next layer:

$$\begin{aligned} \mathbf{d}^{\text{next}} &= \mathbf{maxpool}(\mathbf{d}) \in \mathbb{R}^m \\ \mathbf{D}^{\text{next}} &= \mathbf{diag}(\mathbf{d}^{\text{next}}) \in \mathbb{R}^{m \times m}. \end{aligned} \tag{4.28}$$

Finally, for affine layers, we let the domain-restriction matrix equal the identity matrix:

$$\mathbf{D}^{\text{next}} = \mathbf{I}. \tag{4.29}$$

Note that affine layers are usually the final layer of a network, so we usually do not have to use this equation.

Algorithm 1 Steps in our method to compute a bound on the local Lipschitz constant of a feedforward network

initialize nominal input $\mathbf{x}_0$
initialize input perturbation size ϵ
initialize domain-restriction matrix $\mathbf{D}$ as identity matrix
initialize network local Lipschitz bound: $L^{\text{net}} \leftarrow 1$
for each layer **in** network **do**
  if layer **is** affine-ReLU **then**
    determine $\mathbf{y}_0$ using (4.10): $\mathbf{y}_0 \leftarrow \mathbf{A}\mathbf{x}_0 + \mathbf{b}$
    compute $\overline{\mathbf{y}}$ using (4.19)
    compute $\mathbf{R}$ using (4.21)
    compute Lipschitz bound using (4.22): $L \leftarrow \|\mathbf{RAD}\|$
    set nominal input for next layer: $\mathbf{x}_0 \leftarrow \mathbf{relu}(\mathbf{A}\mathbf{x}_0 + \mathbf{b})$
  else if layer **is** max pooling **then**
    compute $n_{\max}$ using (4.26)
    compute Lipschitz constant using (4.24): $L \leftarrow \sqrt{n_{\max}}$
    set nominal input for next layer: $\mathbf{x}_0 \leftarrow \mathbf{maxpool}(\mathbf{x}_0)$
  else if layer **is** affine **then**
    compute Lipschitz constant: $L \leftarrow \|\mathbf{A}\|$
    set nominal input for next layer: $\mathbf{x}_0 \leftarrow \mathbf{A}\mathbf{x}_0 + \mathbf{b}$
  compute $\mathbf{D}$ for next layer using (4.27), (4.28), or (4.29)
  compute ϵ for next layer: $\epsilon \leftarrow \epsilon L$ (see Proposition 4.2)
  update L^{net}: $L^{\text{net}} \leftarrow L^{\text{net}} L$ (see Proposition 4.3)

4.7.3 Network-wide bounds

We now have all of the tools to compute a local Lipschitz bound on a feedforward neural network. The steps are shown in Algorithm 1. In summary, we start with a nominal input $\mathbf{x}_0$ and a bound ϵ on the set of inputs. We iterate through each layer of the network, and calculate the Lipschitz constant or bound of the layer. Then, we determine the nominal input $\mathbf{x}_0$, domain-restriction matrix $\mathbf{D}$, and input perturbation bound ϵ for the next layer. We then update the network Lipschitz bound, and continue iterating through the layers of the network.

4.7.4 Relationship between local Lipschitz constants and adversarial bounds

One useful application of local Lipschitz constants is that they can be used to bound adversarial examples. Since the local Lipschitz constant represents how much a network's output can change with respect to changes in the input, it can be used to determine a bound on input perturbations that can change the classification of a classification network. The following proposition describes how input perturbations can be related to adversarial bounds.

Proposition 4.6. *Consider a feedforward classification neural network* $\mathbf{f}$ *with nominal input* $\mathbf{x}_0$ *and input set* $\mathcal{X}$*. Assume the input set is norm bounded by* ϵ*, i.e.,* $\|\mathbf{x}-\mathbf{x}_0\| \leq \epsilon,\ \forall \mathbf{x} \in \mathcal{X}$*. Let* $\delta \geq 0$ *denote the difference between the largest and second-largest values of* $\mathbf{f}(\mathbf{x}_0)$ *(i.e., the top two classes). Any* ϵ *that satisfies the following equation is a lower bound on the minimum adversarial perturbation (i.e., such a perturbation cannot change the network classification):*

$$\epsilon L(\mathbf{x}_0, \mathcal{X}) < \frac{\delta}{\sqrt{2}}. \tag{4.30}$$

Proof. Let $\mathbf{y} = \mathbf{f}(\mathbf{x})$ and $\mathbf{y}_0 = \mathbf{f}(\mathbf{x}_0)$, and let y_i and $y_{0,i}$ denote the i^{th} entries of $\mathbf{y}$ and $\mathbf{y}_0$, respectively. Let a and b denote the indices i of the largest and second-largest elements of $\mathbf{y}_0$,

respectively (i.e., the indices of the first and second classes of the nominal input). The top-1 classification will change when y_i is greater than or equal to y_a for some $i \neq a$. Therefore, the smallest value of $\|\mathbf{y} - \mathbf{y}_0\|$ for which the classification can change can be derived as follows

$$\min_{\substack{i \neq a \\ y_i \geq y_a}} \|\mathbf{y} - \mathbf{y}_0\| = \min_{\substack{i \neq a \\ y_i = y_a}} \|\mathbf{y} - \mathbf{y}_0\| \tag{4.31}$$

$$= \min_{\substack{i \neq a \\ y_i = y_a}} \sqrt{(y_1 - y_{0,1})^2 + \cdots + (y_m - y_{0,m})^2} \tag{4.32}$$

$$= \min_{\substack{i \neq a \\ y_i}} \sqrt{(y_i - y_{0,a})^2 + (y_i - y_{0,i})^2} \tag{4.33}$$

$$= \min_{i \neq a} (y_{0,a} - y_{0,i})/\sqrt{2} \tag{4.34}$$

$$= (y_{0,a} - y_{0,b})/\sqrt{2} \tag{4.35}$$

$$= \delta/\sqrt{2}. \tag{4.36}$$

The RHS of equation (4.31) comes from the fact that we can associate any case for which $y_i \geq y_a$ with the case for which $y_i = y_a$, which will have a lower value of $\|\mathbf{y} - \mathbf{y}_0\|$. Equation (4.33) comes from noting that for all scenarios in which $y_i = y_a$, the value $\|\mathbf{y} - \mathbf{y}_0\|$ will be the lowest when all entries of $\mathbf{y}$ and $\mathbf{y}_0$ are equal, except for those corresponding to indices i and a. Equation (4.34) comes from noting that the expression in the square root of (4.33) is minimized when $y_i = (y_{0,a} + y_{0,i})/2$. Equation (4.35) comes from noting that the minimum value of $y_{0,a} - y_{0,i}$ occurs when $i = b$. Equation (4.36) comes from the definition of δ.

In summary, for any $\mathbf{y}$, if $\|\mathbf{y} - \mathbf{y}_0\| < \delta/\sqrt{2}$, then the network classification cannot change. Therefore, the network classification will not change if

$$\|\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{x}_0)\| < \frac{\delta}{\sqrt{2}}. \tag{4.37}$$

Next, from Proposition 4.2, if $\|\mathbf{x} - \mathbf{x}_0\| \leq \epsilon$ for all $\mathbf{x} \in \mathcal{X}$, then $\|\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{x}_0)\| \leq \epsilon L(\mathbf{x}_0, \mathcal{X})$

for all $\mathbf{x} \in \mathcal{X}$. Therefore, if $\epsilon L(\mathbf{x}_0, \mathcal{X}) < \delta/\sqrt{2}$ then $\|\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{x}_0)\| < \delta/\sqrt{2}$, which using (4.37) tells us that the classification cannot change.

□

In practice, we will use Proposition 4.6 to find the largest ϵ which is a lower bound on the minimum adversarial perturbation (i.e., that satisfies (4.30)). Note that $L(\mathbf{x}_0, \mathcal{X})$ is a non-decreasing function of ϵ (a consequence of Proposition 4.1), so $\epsilon L(\mathbf{x}_0, \mathcal{X})$ is also non-decreasing in ϵ. As a result, we can determine the largest possible ϵ by increasing ϵ until (4.30) is no longer satisfied. Section 4.9.2 shows simulations in which we apply this technique.

4.8 Computational techniques

We will now discuss two computational insights that make it possible to apply our method to large layers and networks.

Our first computational insight concerns efficiently calculating $\overline{\mathbf{y}}$, which is the upper bound of the set $\mathcal{Y}$. The equation for $\overline{\mathbf{y}}$ is shown in (4.19), and requires determining $\mathbf{a}_i^T$, the i^{th} row of the $\mathbf{A}$ matrix. For large convolutional layers, the $\mathbf{A}$ matrices are usually too large to store in random-access memory. So instead of determining the entire $\mathbf{A}$ matrix, we can obtain the i^{th} row of $\mathbf{A}$ by noting that $\mathbf{a}_i^T = \mathbf{A}^T\mathbf{e}_i$ where $\mathbf{e}_i \in \mathbb{R}^m$ is the i^{th} standard basis vector. To perform the $\mathbf{A}^T$ transformation we can use a transposed convolution function based on the original convolution function (making sure to reshape $\mathbf{e}_i$ into the appropriate input size). Furthermore, to reduce computation time, we can use a batch of standard basis vectors in the transposed convolution function to obtain multiple rows of $\mathbf{A}$.

Our second computational insight concerns efficiently computing $\|\mathbf{RAD}\|$, which is the affine-ReLU local Lipschitz constant bound from Theorem 4.2. The matrix $\mathbf{RAD}$ is usually too large to be stored in memory, so we use a power iteration to compute it. Note that the largest singular value of a matrix $\mathbf{M}$ is the square root of the largest eigenvalue of $\mathbf{M}^T\mathbf{M}$. So,

we can find the spectral norm of $\mathbf{M}$ by applying a power iteration to the operator $\mathbf{M}^T\mathbf{M}$. In our case, our matrix is $\mathbf{M} = \mathbf{RAD}$, so we have $\mathbf{M}^T\mathbf{M} = \mathbf{D}^T\mathbf{A}^T\mathbf{R}^T\mathbf{RAD} = \mathbf{DA}^T\mathbf{R}^2\mathbf{AD}$. For convolutional layers, we can perform the $\mathbf{A}$ transformation using the convolution function (with zero bias) and the $\mathbf{A}^T$ transformation using transposed convolution (with zero bias). Furthermore, since both $\mathbf{R}^2$ and $\mathbf{D}$ are diagonal matrices, we can apply these transformations using elementwise vector multiplication.

4.9 Simulations

4.9.1 Local Lipschitz constants for various networks

CIFAR-10 Net Architecture										
C3-32	C3-32	MP-2	D	C3-64	C3-64	MP-2	D	FC-512	D	FC-10
MNIST Net Architecture										
C5-6	MP-2	C5-16	MP-2	FC-120	FC-84	FC-10				

Figure 4.6: Architectures of the networks we constructed for this chapter (in sequence left-to-right). "Cα-β" denotes a convolution layer with kernel size α and β output channels, "MP-α" denotes a max pooling layer with kernel size α, "FC-α" denotes a fully-connected layer with α output features, and "D" denotes dropout layers. All convolution layers are followed by a ReLU and have a stride of 1. All fully-connected layers are followed by a ReLU unless it is the last layer.

We compare four different networks in this chapter: a seven-layer network trained on MNIST (which we refer to as "MNIST Net"), an eight-layer network trained on CIFAR-10 (which we refer to as "CIFAR-10 Net"), AlexNet [23] (11-layers, trained on ImageNet), and VGG-16 (21 layers including 16 affine-ReLU layers, trained on ImageNet) [42]. The architectures of MNIST Net and CIFAR-10 Net are shown in Fig. 4.6. We constructed MNIST Net ourselves and trained it to 99% top-1 test accuracy in 100 epochs. We also constructed CIFAR-10 Net ourselves, and trained it to 84% top-1 test accuracy in 500 epochs. We used the trained versions of AlexNet and VGG-16 from Pytorch's Torchvision package.

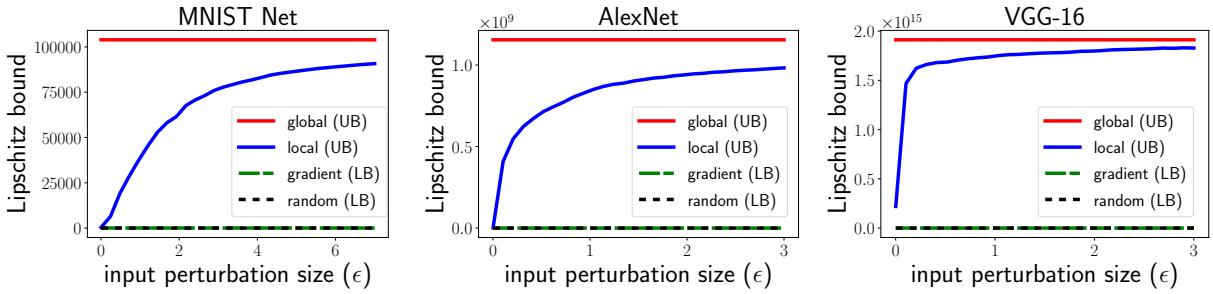
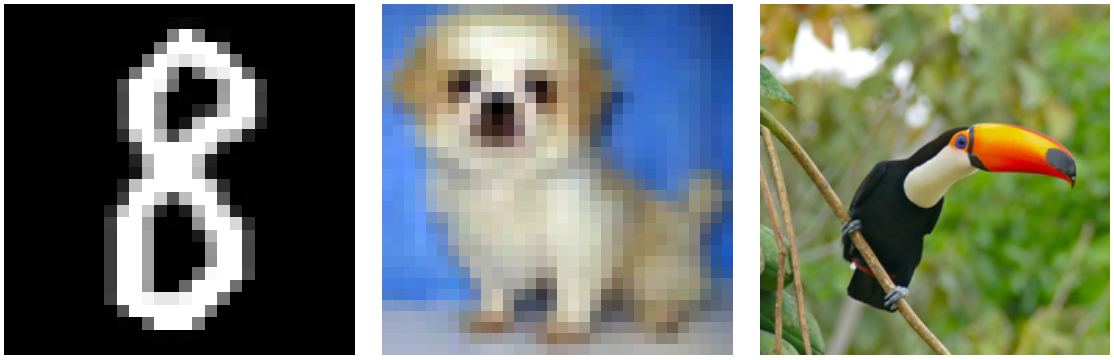


Figure 4.7: Upper bounds (UB) and lower bounds (LB) on the local Lipschitz constants of the MNIST Net, AlexNet, and VGG-16 networks for various perturbation sizes (the plot for CIFAR-10 Net is not shown, but has similar trends). These results are computed with respect to the nominal input images in Fig. 4.8. The term "global" represents to global bounds computed using Theorem 4.2 for affine-ReLU functions, "local" refers to local bounds computed using $\|\mathbf{A}\|$ for affine-ReLU functions, "gradient" refers to lower bounds determined using gradient ascent, and "random" refers to lower bounds determined by sampling random input perturbations.

Note these networks both have an "adaptive average pooling" layer which has no effect when the network inputs are the default size of $3\times224\times224$.

All simulations were performed using Pytorch, and were run on an Nvidia GTX 1080 Ti card. In our simulations we used the nominal input images shown in Fig. 4.8, which all classify correctly. In each simulation we determined global and local upper bounds, as well as lower bounds computed using both gradient ascent and random sampling methods. Fig. 4.7 shows the full-network local Lipschitz bounds, and Table 4.2 shows the computation times.

The results show that our Lipschitz bounds increase with the size of the perturbation ϵ, and approach the global bound for large ϵ. For small perturbations, the bound is significantly lower than the global bound.



MNIST Net
28×28
8

CIFAR-10 Net
3×32×32
dog

AlexNet/VGG-16
3×224×224
toucan

Figure 4.8: Images, networks, image sizes, and true class names of the nominal input images used in our simulations.

	MNIST Net	**CIFAR-10 Net**	**AlexNet**	**VGG-16**
time	.1 sec	1 sec	16 sec	52 min

Table 4.2: Times to compute the local Lipschitz constant upper bound for one input perturbation of size ϵ for various networks (using Algorithm 1), based on the nominal input images in Fig. 4.8. All computations were performed on a desktop computer using Pytorch and an Nvidia GTX 1080 Ti card.

4.9.2 Bounds on adversarial examples

Next, we apply our local Lipschitz bounds to determine lower bounds on adversarial perturbations, as described in Section 4.7.4. We calculated these bounds with respect to the same networks and nominal input images in Fig. 4.8 in Section 4.9. The results are shown in Table 4.3.

For each network, our method provides an orders of magnitude improvement over the global bound. We are unaware of any other method that can improve upon the global bound for networks such as AlexNet and VGG-16, so our bounds represent a significant

		MNIST Net	CIFAR-10 Net	AlexNet	VGG-16
upper bound	FGSM	$1.8 \cdot 10^{1}$	$6.7 \cdot 10^{0}$	$7.8 \cdot 10^{0}$	$4.7 \cdot 10^{1}$
	gradient	$4.0 \cdot 10^{0}$	$2.6 \cdot 10^{0}$	$4.6 \cdot 10^{0}$	$2.7 \cdot 10^{0}$
lower bound	local	$5.2 \cdot 10^{-2}$	$4.2 \cdot 10^{-3}$	$1.0 \cdot 10^{-5}$	$1.8 \cdot 10^{-8}$
	global	$5.1 \cdot 10^{-4}$	$3.9 \cdot 10^{-4}$	$8.6 \cdot 10^{-9}$	$7.4 \cdot 10^{-15}$

Table 4.3: Bounds on the minimum Euclidean perturbation required to change the top classification of a classification network, based on the nominal input images shown in Fig. 4.8. The "global" row refers to bounds computed using the global Lipschitz constant, "local" refers to bounds computed using the local Lipschitz constant, "gradient" refers to lower bounds determined by finding adversarial examples using gradient ascent, and "FGSM" refers to lower bounds determined using the Fast Gradient Sign Method [11].

improvement in certifying adversarial bounds to Euclidean perturbations.

Also shown in Table 4.3 are upper bounds computed using gradient methods. Having both lower and upper bounds allows us to identify the range in which the true minimum perturbation resides.

4.9.3 Comparison with other methods

As mentioned in Section 4.1, there are several methods which provide Lipschitz estimates or bounds, but many only work for small networks. We did not apply the methods in [16, 24] as they have only been shown to be applicable to networks smaller than the smallest network we considered (MNIST Net). We were able to implement the method in [9] to MNIST Net, but it ran out of memory for the larger networks. As this method is not designed to incorporate max pooling functions, we used it estimate the global Lipschitz constant of each affine-ReLU sequence, and combined the results with the max pooling global bound, which resulted in an estimate of 67×10^3. We also were able to apply the estimation method in [38]. We note

that [38] presents two bounds: AutoLip which is equivalent to the global bound, and SeqLip. SeqLip produced estimates of 72×10^3, 7×10^3, and 174×10^6 for MNIST Net, CIFAR-10 Net, and AlexNet, respectively, and took longer than 48 hours to produce an estimate for VGG-16 so we aborted the operation.

4.10 Summary

We have presented a method to determine guaranteed upper bounds on the local Lipschitz constant of neural networks with ReLU activations. Our approach is based on determining Lipschitz constants and bounds of ReLU, affine-ReLU and max pooling functions. We then showed how we can calculate these Lipschitz constants/bounds in a sequential fashion for each layer of a feedforward network, which allows us to compute a network-wide bound.

We calculated our bounds for small MNIST and CIFAR-10 networks, as well as large networks such as AlexNet and VGG-16. The results show that our bounds are especially tight for small perturbations. We then showed how we can use our method to determine lower bounds on Euclidean adversarial perturbations. To the best of our knowledge, our method produces the tightest known bounds for larger networks.

Potential future work includes reducing the computation time of our method, as well as further mathematical analysis to obtain even tighter bounds. Note that for larger layers, the main computational bottleneck comes from computing $\overline{\mathbf{y}}$, which requires evaluating each row of the $\mathbf{A}$ matrix.

Finally, there are several ways in which this work could be extended. For example, we could consider activation functions other than ReLU, and we could consider other types of layers. Additionally, as we only considered the 2-norm, we could generalize our results to other norms. Note that many of our results hold for general matrix norms, and we believe many of the other results could be generalized without too much trouble.

Similar to the work done in this chapter, in the next chapter we will consider the task of quantifying the sensitivity of pose estimation neural networks. This task is particularly challenging as it requires characterizing the sensitivity of 3D rotations. One outcome of this chapter will be a specialized Lipschitz constant which measures the output of a pose estimation neural network using a rotational distance function rather than the Euclidean distance.

Chapter 5

SENSITIVITY BOUNDS FOR POSE ESTIMATION NEURAL NETWORKS

5.1 Introduction

In the past few years, deep neural networks have achieved remarkable success at the task of object pose estimation [36, 49, 53, 46]. However, a major downside of deep learning-based methods is that deep neural networks are not well-understood mathematically and lack provable performance guarantees. This lack of theoretical understanding is problematic as pose estimation networks are often intended to be applied to real-world applications which require safe and reliable operation. Therefore, it is necessary to develop better tools to analyze pose estimation networks in order to use them in the real world.

One major disadvantage of deep neural networks is that they are often highly sensitive, and small changes in a network's input can result in huge changes in its output [45]. One of the primary tools for analyzing neural network sensitivity is the Lipschitz constant, which measures the maximum distance between two outputs of a function with respect to the distance between the corresponding inputs. Although exact computation of a network's Lipschitz constant is too formidable for all but very small networks [16], an upper bound can be computed instead. We also note that other work has considered estimating Lipschitz constants [38, 24, 9, 54], as well as specifically analyzing the sensitivity of classification networks (often with regard to adversarial examples) [48, 33, 50, 52].

While sensitivity analysis of general neural networks and of classification neural networks are active areas of research, to the best of our knowledge, the sensitivity of pose estimation

neural networks has not yet been studied. This task is particularly challenging as sensitivity must be considered using a measure of rotational distance rather than a norm-based metric such as the Euclidean distance. As such, we measure network sensitivity as the maximum rotational distance between two outputs of a network with respect to the Euclidean distance between the two corresponding inputs. We show that this measure is a type of Lipschitz constant, and that it can be bounded by the product of two values: the network's Euclidean Lipschitz constant, and a measure we call the "distance ratio constant" which is the maximum ratio of the rotational and Euclidean distances between all pairs of rotation parameters.

Analyzing the sensitivity of a pose estimation network will depend on which specific rotation parameterization is used, and in this chapter we consider five types of rotation parameterizations: exponential coordinates, unconstrained exponential coordinates, quaternions, unconstrained quaternions, and 2D projection parameterizations. We show how the structure of most of these parameterizations raise difficulties in constructing a pose estimation network with provable rotational sensitivity bounds. However, we further show that such bounds can be determined for networks which parameterize rotation using unconstrained exponential coordinates. We then construct such a network and calculate rotational sensitivity bounds for it.

The remainder of this chapter is organized as follows. In Section 5.2 we discuss mathematical properties of 3D rotations and pose. In Section 5.3 we discuss Lipschitz constants, construct a Lipschitz constant that measures the sensitivity of a pose estimation neural network, and derive a bound on it. In Sections 5.4, 5.5, and 5.6, we derive the distance ratio constant for exponential coordinates, quaternions and 2D projected methods, respectively. In Section 5.7, we summarize our findings from the previous sections, and discuss their implications in creating a pose estimation neural network with provable bounds. In Section 5.8, we construct a pose estimation neural network which outputs rotation in the form of uncon-

strained exponential coordinates, and calculate sensitivity bounds for the network. Finally, we discuss areas of future work and provide concluding remarks in Section 5.9.

name	**authors**	**year**	**rot. param.**	**ref**
PoseNet	Kendall et al.	2015	quaternion	[20]
BB8	Rad et al.	2017	2D proj	[36]
DOPE	Tremblay et al.	2018	2D proj	[49]
PoseCNN	Xiang et al.	2018	quaternion	[53]
-	Tekin et al.	2018	2D proj	[46]

Figure 5.1: Several of the earliest pose estimation neural networks, along with the rotation parameterization used in each method. The term "2D proj" denotes 2D projection parameterizations, in which pose is represented by projecting 3D points on an object into the 2D image plane (see Section 5.6).

5.2 Pose and 3D rotations

5.2.1 Pose and 3D rotations

The pose of a rigid object is defined as its position and orientation. The set of all 3D positions and orientations is topologically characterized by SE(3), the "special Euclidean group" in three dimensional Euclidean space. Pose is often parameterized by treating position and rotation separately, in which case position can be described by a 3D vector, and rotation can be described by a rotation parameterization such as exponential coordinates or quaternions. In other representations of pose, such as 2D projected points (see Section 5.6), position and rotation are coupled.

The set of all rotations about the origin in three-dimensional Euclidean space is termed the special Euclidean group SO(3). Each element of SO(3) can be described by an orthogonal

3×3 matrix with determinant equal to one:

$$\mathrm{SO}(3) = \{\mathbf{R} \in \mathbb{R}^{3\times 3} \quad | \quad \mathbf{R}^T\mathbf{R} = \mathbf{I}, \ \ \det(\mathbf{R}) = 1\}$$

where $\det(\cdot)$ denotes the determinant operation, and $\mathbf{I} \in \mathbb{R}^{3\times 3}$ denotes the identity matrix.

There are many different parameterizations of the group of 3D rotations, including quaternions, exponential coordinates, and Euler angles. In this chapter, we will use the symbol $\mathcal{P} \subseteq \mathbb{R}^m$ to denote a set of rotation parameters, and the symbol $\mathbf{p} \in \mathcal{P}$ to denote a particular parameter in that set. The symbols $\mathcal{P}$ and $\mathbf{p}$ are generic in that they do not correspond to any specific parameterization, and they will be replaced with specific symbols when considering a specific rotation parameterization. For example, when considering quaternions, $\mathcal{P}$ will be replaced with $\mathcal{S}^3 \subset \mathbb{R}^4$ (see (5.26)), and $\mathbf{p}$ will be replaced with $\mathbf{q} \in \mathbb{R}^4$.

5.2.2 Rotational distance

We now consider the task of quantifying the "distance" between two 3D rotations. As the set of 3D rotations is structured as a Riemannian manifold, Euclidean distance cannot be used to properly measure distance for rotations. A proper measure of rotational distance is the angle of direct rotation between two 3D rotations, which is equivalent to the geodesic distance on SO(3). This measure provides a consistent measure of rotational distance, can be applied to any rotation parameterization, and is intuitive as it corresponds to an angle of rotation.

Definition 5.1. *The **rotational distance** between two 3D rotations is the angle of direct rotation between the two rotations using an axis-angle representation. Letting $\mathcal{P}$ denote a set of rotation parameters, we denote the rotational distance function as* $\mathrm{dist} : \mathcal{P} \times \mathcal{P} \to [0, \pi]$.

Note that the rotational distance is not a metric as it does not generally satisfy the

property that $\text{dist}(\mathbf{p}_1, \mathbf{p}_2) = 0$ implies $\mathbf{p}_1 = \mathbf{p}_2$ [19, Ch. 4]. However, it is a pseudometric as it satisfies the remaining properties of a metric (i.e., non-negativity, symmetry, and the triangle inequality).

The function $\text{dist}(\cdot, \cdot)$ is overloaded notation as its explicit expression will depend on which rotation parameterization is being considering. For example, the rotational distance between two rotation matrices $\mathbf{R}_1, \mathbf{R}_2 \in \text{SO}(3)$ is given by the following equation:

$$\text{dist}(\mathbf{R}_1, \mathbf{R}_2) = \cos^{-1}\left(\tfrac{1}{2}(\text{tr}(\mathbf{R}_1\mathbf{R}_2^T) - 1)\right)$$

where $\text{tr}(\cdot)$ denotes the trace operation. Later in the chapter, we consider the distance between two quaternions, $\text{dist}(\mathbf{q}_1, \mathbf{q}_2)$, in (5.27), and the distance between two pairs of exponential coordinates, $\text{dist}(\mathbf{s}_1, \mathbf{s}_2)$, in (5.8).

5.3 Lipschitz constants and measures of sensitivity for pose estimation neural networks

5.3.1 Lipschitz constants

Lipschitz constants are a tool to quantify the sensitivity of mathematical functions, and are one of the main tools used to analyze the sensitivity of neural networks. Given a function $\mathbf{f}$, the Lipschitz constant describes the maximum amount the output of the function can change with respect to changes in the input. Lipschitz constants are useful due to their applicability to a wide array of functions, including vector-valued, nonlinear, and non-differentiable functions.

Consider a function $\mathbf{f} : \mathcal{X} \to \mathcal{Y}$, and metrics $d_{\mathcal{X}} : \mathcal{X} \times \mathcal{X} \to [0, \infty)$ and $d_{\mathcal{Y}} : \mathcal{Y} \times \mathcal{Y} \to [0, \infty)$. The Lipschitz constant, $L \in \mathbb{R}$, of $\mathbf{f}$ with respect to $d_{\mathcal{X}}$ and $d_{\mathcal{Y}}$ is the minimal $L \geq 0$ such that [39, Ch. 9]

$$d_{\mathcal{Y}}(\mathbf{f}(\mathbf{x}_1), \mathbf{f}(\mathbf{x}_2)) \leq L d_{\mathcal{X}}(\mathbf{x}_1, \mathbf{x}_2), \quad \forall \mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}. \tag{5.1}$$

Note that some authors define any L that satisfies the inequality above as "a Lipschitz constant", but we will define "the Lipschitz constant" as the minimal L for which this inequality holds, and we refer to any larger value as an upper bound. We can solve for L in (5.1) as

$$L = \sup_{\substack{\mathbf{x}_1,\mathbf{x}_2 \in \mathcal{X} \\ \mathbf{x}_1 \neq \mathbf{x}_2}} \frac{d_{\mathcal{Y}}(\mathbf{f}(\mathbf{x}_1), \mathbf{f}(\mathbf{x}_2))}{d_{\mathcal{X}}(\mathbf{x}_1, \mathbf{x}_2)}.$$

Note that excluding points such that $\mathbf{x}_1{=}\mathbf{x}_2$ does not affect the supremization above since these points satisfy (5.1) for any L.

5.3.2 Euclidean Lipschitz constants

In this chapter, we are interested in the case in which the function $\mathbf{f}$ is a neural network that maps an image or other type of input to a rotation parameter. We can write this function as $\mathbf{f} : \mathbb{R}^n \to \mathcal{P}$. Lipschitz constants are typically applied to neural networks using the metrics induced by the 1, 2, and ∞-norms for $d_{\mathcal{X}}$ and $d_{\mathcal{Y}}$. We now present the Lipschitz constant in which both metrics are defined to be the Euclidean distance.

Definition 5.2. *Consider a function $\mathbf{f} : \mathbb{R}^n \to \mathcal{P}$ which outputs a rotation parameter. The* ***Euclidean Lipschitz constant****, L_e, of $\mathbf{f}$ is the Lipschitz constant of the function using the Euclidean distance as the metric on both the input and output sets:*

$$L_e \coloneqq \sup_{\mathbf{x}_1 \neq \mathbf{x}_2} \frac{\|\mathbf{f}(\mathbf{x}_2) - \mathbf{f}(\mathbf{x}_1)\|}{\|\mathbf{x}_2 - \mathbf{x}_1\|}. \tag{5.2}$$

5.3.3 Rotational Lipschitz constants

Considering a neural network $\mathbf{f}$ that maps an image to a rotation parameter, the Euclidean Lipschitz constant does not provide a meaningful measure of sensitivity, due to the fact that the Euclidean distance between two rotation parameters is not a meaningful measure

of rotational distance. So, to properly measure the sensitivity of these functions, we will create a Lipschitz constant that measures the distance between inputs using the Euclidean distance, and the distance between outputs using the rotational distance from Definition 5.1. We present this measure in the following definition.

Definition 5.3. *Consider a function* $\mathbf{f} : \mathbb{R}^n \to \mathcal{P}$ *that outputs a rotation parameter. The* ***rotational Lipschitz constant****,* L_r*, of* $\mathbf{f}$ *is the Lipschitz constant of the function using the Euclidean distance as the metric on the input set, and the rotational distance as the metric on the output set:*

$$L_r \coloneqq \sup_{\mathbf{x}_1 \neq \mathbf{x}_2} \frac{\text{dist}(\mathbf{f}(\mathbf{x}_1), \mathbf{f}(\mathbf{x}_2))}{\|\mathbf{x}_2 - \mathbf{x}_1\|}. \tag{5.3}$$

Technically speaking, since the rotational distance function is not a metric (although it is a pseudometric), the rotational Lipschitz constant is not a true Lipschitz constant as described in Section 5.3.1. However, this discrepancy is minor and does not have any negative effects on using the rotational Lipschitz constant to measure the sensitivity of a neural network.

As with Euclidean Lipschitz constants, it is too difficult to exactly calculate the rotational Lipschitz constant of complex functions such as neural networks, so instead we will derive an upper bound. This bound will require defining the following measure which relates the Euclidean and rotational distances between pairs of rotation parameters.

Definition 5.4. *Consider a rotation parameterization with set of parameters* $\mathcal{P}$*. The* ***distance ratio constant****,* μ*, of the parameterization is the maximum ratio of the rotational distance to the Euclidean distance over all pairs of parameters:*

$$\mu \coloneqq \sup_{\substack{\mathbf{p}_1, \mathbf{p}_2 \in \mathcal{P} \\ \mathbf{p}_1 \neq \mathbf{p}_2}} \frac{\text{dist}(\mathbf{p}_1, \mathbf{p}_2)}{\|\mathbf{p}_2 - \mathbf{p}_1\|}. \tag{5.4}$$

Note this definition is equivalent to applying the rotational Lipschitz constant from Def-

inition 5.3 to the identity map $\mathbf{f}(\mathbf{p}) = \mathbf{p}$.

5.3.4 Bounds on the rotational Lipschitz constant

Using L_e and μ, we can now derive a bound on a rotational Lipschitz constant L_r.

Theorem 5.1. *Consider a function $\mathbf{f} : \mathbb{R}^n \to \mathcal{P}$ that outputs a rotation parameter. Let L_r denote the rotational Lipschitz constant of $\mathbf{f}$, let L_e denote the Euclidean Lipschitz constant of $\mathbf{f}$, and let μ denote the distance ratio constant of the rotation parameterization. The following equation holds:*

$$L_r \leq \mu L_e. \tag{5.5}$$

Proof. Using the definition of the rotational Lipschitz constant in (5.3), noting that $\mathbf{f}(\mathbf{x}_1) = \mathbf{p}_1$ and $\mathbf{f}(\mathbf{x}_2) = \mathbf{p}_2$, and applying the definitions of L_e and μ from (5.2) and (5.4), we have

$$\begin{aligned} L_r &= \sup_{\mathbf{x}_1 \neq \mathbf{x}_2} \frac{\operatorname{dist}(\mathbf{p}_1, \mathbf{p}_2)}{\|\mathbf{x}_2 - \mathbf{x}_1\|} \\ &\leq \sup_{\mathbf{x}_1 \neq \mathbf{x}_2} \frac{\|\mathbf{p}_2 - \mathbf{p}_1\|}{\|\mathbf{x}_2 - \mathbf{x}_1\|} \sup_{\substack{\mathbf{p}_1, \mathbf{p}_2 \in \mathcal{P} \\ \mathbf{p}_1 \neq \mathbf{p}_2}} \frac{\operatorname{dist}(\mathbf{p}_1, \mathbf{p}_2)}{\|\mathbf{p}_2 - \mathbf{p}_1\|} \\ &= L_e \mu. \end{aligned}$$

□

Theorem 5.1 can be used in practice to determine rotational bounds on the output of a function based on Euclidean bounds on the input. This result is summarized in the following proposition.

Proposition 5.1. *Consider a function $\mathbf{f} : \mathbb{R}^n \to \mathcal{P}$ that outputs a rotation parameter. Let L_e denote the Euclidean Lipschitz constant of the function, and let μ denote the distance ratio constant of the rotation parameterization. Let $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^n$ denote two inputs to the*

network, and let $\mathbf{p}_1, \mathbf{p}_2 \in \mathcal{P}$ *denote the corresponding outputs. If* $\|\mathbf{x}_2 - \mathbf{x}_1\| \leq \epsilon$ *for some* $\epsilon \geq 0$*, then the rotational distance between the outputs is bounded as follows:*

$$\text{dist}(\mathbf{p}_1, \mathbf{p}_2) \leq \epsilon \mu L_e. \tag{5.6}$$

Proof. Note that (5.5) implies

$$\begin{aligned} \frac{\text{dist}(\mathbf{p}_1, \mathbf{p}_2)}{\|\mathbf{x}_2 - \mathbf{x}_1\|} &\leq L_e \mu, && \forall \mathbf{x}_1 \neq \mathbf{x}_2 \\ \text{dist}(\mathbf{p}_1, \mathbf{p}_2) &\leq \|\mathbf{x}_2 - \mathbf{x}_1\| L_e \mu, && \forall \mathbf{x}_1 \neq \mathbf{x}_2. \end{aligned}$$

Combining $\|\mathbf{x}_2 - \mathbf{x}_1\| \leq \epsilon$ with the equation above yields (5.6). □

Note that this bound will only be useful if the right hand side of (5.6) is less than the maximum possible rotational distance of π.

5.3.5 Application to pose estimation neural networks

The function $\mathbf{f}$ that we have considered in this section will correspond to a feedforward pose estimation neural network. In order to apply the results from this section to such a network, there are several considerations we must first make.

First, note that for networks which take RGB images as their inputs, the input $\mathbf{x}$ will be a three-dimensional array. However, it is mathematically equivalent to consider such inputs to be vectors in $\mathbb{R}^n$. Also note that we have considered a function $\mathbf{f}$ which outputs rotation parameters, but a pose estimation neural network will output pose, which consists of both position and rotation. However, pose is often represented as a concatenation of position and rotation, so in these cases we can split a network into position and rotation sub-networks, and consider each sub-network separately. Finally, as we noted before, the exact Euclidean Lipschitz constant L_e is intractable to compute for all but very small toy networks, but an

upper bound can be computed instead by taking the product of the individual bounds of the layers of the network.

In summary, we would like to compute a bound on the rotational Lipschitz constant from Theorem 5.1 for a pose estimation neural network. This bound is a function of two values: the Euclidean Lipschitz constant L_e and the distance ratio constant μ. While we know how to compute an upper bound on L_e, we are not aware of any work in which μ is derived, for any rotation parameterization. Accordingly, we will devote the next three sections of the chapter to the derivation of μ for three types of rotation parameterizations: exponential coordinates, quaternions, and 2D projected points.

5.4 Exponential coordinates

5.4.1 Rotation parameterization

The first rotation parameterization we will consider is exponential coordinates. Although not commonly used in pose estimation neural networks, exponential coordinates are a fundamental and intuitive description of rotation used frequently in robotics applications. Additionally, as we will show later on, they have several properties which make them advantageous for developing pose estimation networks with provable sensitivity bounds.

Any 3D rotation can be described by a rotation of angle $\theta \in \mathbb{R}$ about a unit-length axis $\mathbf{e} \in \mathbb{R}^3$. Exponential coordinates, which we will denote as $\mathbf{s} \in \mathbb{R}^3$, are simply the axis multiplied by the angle: $\mathbf{s} = \theta\mathbf{e} \in \mathbb{R}^3$ [27, Ch. 3]. The term "exponential" comes from the fact that SO(3) is a Lie group, and exponential coordinates are a description of the Lie algebra of the Lie group, which can be mapped back to the group using the exponential map.

In general, exponential coordinates can be defined using any angle $\theta \in [0, \infty)$. We refer to the set of exponential coordinates defined with $\theta \in [0, \infty)$ as "unconstrained" exponential coordinates. However, exponential coordinates are typically defined with $\theta \in [0, \pi]$, which

is the minimum set of angles that can describe all 3D rotations. This set of exponential coordinates corresponds to the three-dimensional ball of radius π, which we denote as $\mathcal{B}_\pi$:

$$\mathcal{B}_\pi = \{\mathbf{x} \in \mathbb{R}^3 \quad | \quad \|\mathbf{x}\| \leq \pi\}.$$

There is a 1-to-1 mapping between points inside $\mathcal{B}_\pi$ and 3D rotations, and a 2-to-1 mapping between points on the surface $\mathcal{B}_\pi$ to 3D rotations. More specifically, antipodal points on the surface of the ball correspond to the same rotation (i.e., $\pi\mathbf{e}$ and $-\pi\mathbf{e}$ correspond to the same rotation) [27, Ch. 3]. Note that while we could reduce the set $\mathcal{B}_\pi$ so that it is a 1-to-1 mapping, it would not change the results in this section.

We now consider how exponential coordinates can be composed. Let $\mathbf{s}_1 = \theta_1\mathbf{e}_1$ and $\mathbf{s}_2 = \theta_2\mathbf{e}_2$ denote two sets of exponential coordinates, and let $\mathbf{s}_3 = \theta_3\mathbf{e}_3$ denote the coordinates of the composite rotation. The angle θ_3 can be determined using the following formula [2]:

$$\cos\tfrac{\theta_3}{2} = \cos\tfrac{\theta_1}{2}\cos\tfrac{\theta_2}{2} - \mathbf{e}_1 \cdot \mathbf{e}_2 \sin\tfrac{\theta_1}{2}\sin\tfrac{\theta_2}{2}. \tag{5.7}$$

5.4.2 Rotational distance

Using (5.7), we can derive the rotational distance between two pairs of exponential coordinates.

Proposition 5.2. *The rotational distance between two pairs of exponential coordinates,* $\mathbf{s}_1 = \theta_1\mathbf{e}_1$ *and* $\mathbf{s}_2 = \theta_2\mathbf{e}_2$*, is given by the following equation:*

$$\text{dist}(\mathbf{s}_1, \mathbf{s}_2) = 2\cos^{-1}(|\cos\tfrac{\theta_1}{2}\cos\tfrac{\theta_2}{2} + \mathbf{e}_1 \cdot \mathbf{e}_2 \sin\tfrac{\theta_1}{2}\sin\tfrac{\theta_2}{2}|). \tag{5.8}$$

Proof. The rotational distance can be determined by finding the angle corresponding to the composite axis-angle rotation between the first rotation $\mathbf{s}_1$ and the *inverse* of the second

rotation. The inverse of the second rotation is found by simply negating the angle of rotation θ_2, and is therefore described by the exponential coordinates $-\mathbf{s}_2 = -\theta_2\mathbf{e}_2$. To find the angle of the composite rotation, we can use (5.7):

$$\cos\tfrac{\theta_3}{2} = \cos\tfrac{\theta_1}{2}\cos\tfrac{-\theta_2}{2} - \mathbf{e}_1 \cdot \mathbf{e}_2 \sin\tfrac{\theta_1}{2}\sin\tfrac{-\theta_2}{2} \tag{5.9}$$

$$= \cos\tfrac{\theta_1}{2}\cos\tfrac{\theta_2}{2} + \mathbf{e}_1 \cdot \mathbf{e}_2 \sin\tfrac{\theta_1}{2}\sin\tfrac{\theta_2}{2} \tag{5.10}$$

where we have used the trigonometric identities $\sin(-x) = -\sin(x)$ and $\cos(-x) = \cos(x)$. Next, for convenience, we define

$$a \coloneqq \cos\tfrac{\theta_1}{2}\cos\tfrac{\theta_2}{2} + \mathbf{e}_1 \cdot \mathbf{e}_2 \sin\tfrac{\theta_1}{2}\sin\tfrac{\theta_2}{2} \tag{5.11}$$

and note that $a \in [-1, 1]$. Solving for θ_3 in (5.10) and using (5.11), we have

$$\theta_3 = 2\cos^{-1}(a). \tag{5.12}$$

Note that arccosine will map to the interval $[0, \pi]$, so θ_3 in (5.12) will be mapped to the interval $[0, 2\pi]$. However, we would like to express θ_3 as the rotational distance from zero, which is on the interval $[0, \pi]$. So, we can write

$$\operatorname{dist}(\mathbf{s}_1, \mathbf{s}_2) = \begin{cases} \theta_3, & \theta_3 \in [0, \pi] \\ 2\pi - \theta_3, & \theta_3 \in [\pi, 2\pi]. \end{cases} \tag{5.13}$$

Now note that θ_3 will be greater than π when the argument to the arccosine (i.e., a) is

negative, so using (5.12) we can rewrite (5.13) as

$$\text{dist}(\mathbf{s}_1, \mathbf{s}_2) = \begin{cases} 2\cos^{-1}(a), & a \in [0, 1] \\ 2\pi - 2\cos^{-1}(a), & a \in [-1, 0]. \end{cases} \tag{5.14}$$

Now, note that we can write $\cos^{-1}(|x|)$ in the following way, based on the sign of the argument:

$$\cos^{-1}(|x|) = \begin{cases} \pi - \cos^{-1}(x), & x \in [-1, 0] \\ \cos^{-1}(x), & x \in [0, 1]. \end{cases} \tag{5.15}$$

Using (5.15) we can write (5.14) as (5.8), which completes the proof. □

5.4.3 2D interpretation

To determine the distance ratio constant of exponential coordinates from Definition 5.4, we would need to optimize over all possible pairs of exponential coordinates, where each pair consists of the six degrees of freedom in $\mathbf{s}_1$ and $\mathbf{s}_2$, and the two constraints $\|\mathbf{s}_1\| \leq \pi$ and $\|\mathbf{s}_2\| \leq \pi$. The number of dimensions and constraints in this analysis makes it challenging, but we can interpret this problem in 2D which reduces the number of variables and simplifies the analysis.

Lemma 5.1. *Consider the exponential coordinates* $\mathbf{s}_1 = \theta_1\mathbf{e}_1$ *and* $\mathbf{s}_2 = \theta_2\mathbf{e}_2$ *where* $\theta_1, \theta_2 \in [0, \infty)$. *Define the variable* t *as follows:*

$$t \coloneqq \tfrac{1}{2}(1 - \mathbf{e}_1 \cdot \mathbf{e}_2), \qquad t \in [0, 1]. \tag{5.16}$$

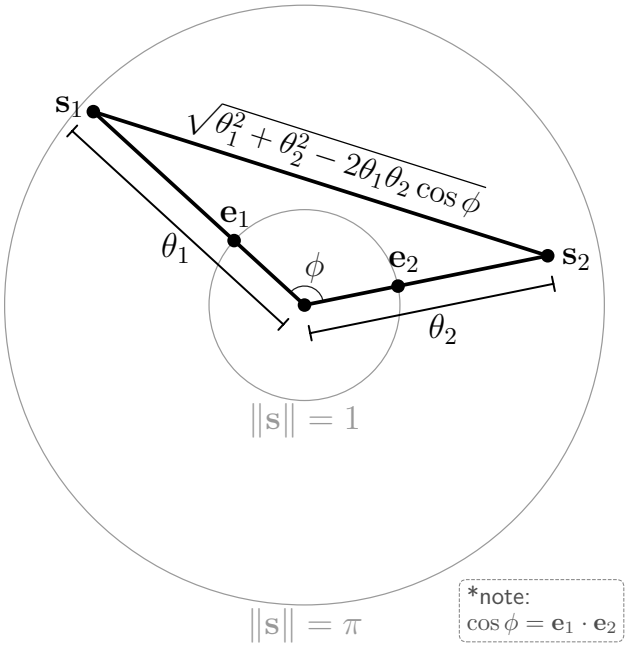


Figure 5.2: Exponential coordinates $\mathbf{s}_1$ and $\mathbf{s}_2$ shown in 2D. In 2D, two exponential coordinate vectors can be analyzed in terms of the angle ϕ between the vectors, and the magnitudes θ_1 and θ_2 of the vectors. The Euclidean distance distance between the exponential coordinates, $\|\mathbf{s}_2 - \mathbf{s}_1\| = (\theta_1^2 + \theta_2^2 - 2\theta_1\theta_2 \cos\phi)^{1/2}$, is computed using the law of cosines.

The rotational and Euclidean distances between $\mathbf{s}_1$ *and* $\mathbf{s}_2$ *are given as follows:*

$$\|\mathbf{s}_2 - \mathbf{s}_1\| = \sqrt{(\theta_1 - \theta_2)^2 + 4\theta_1\theta_2 t} \tag{5.17}$$

$$\text{dist}(\mathbf{s}_1, \mathbf{s}_2) = 2\cos^{-1}(|t\cos(\tfrac{\theta_1}{2}+\tfrac{\theta_2}{2}) + (1-t)\cos(\tfrac{\theta_1}{2}-\tfrac{\theta_2}{2})|). \tag{5.18}$$

Proof. We note that any two exponential coordinates $\mathbf{s}_1$ and $\mathbf{s}_2$ lie in a 2D plane, which allows this problem to be analyzed in 2D. A diagram is shown in Fig. 5.2.

First, note the following equation which relates the dot product and angle between two vectors:

$$\cos\phi = \mathbf{e}_1 \cdot \mathbf{e}_2. \tag{5.19}$$

To show (5.17), note that we can write the Euclidean distance between two points using the law of cosines (see Fig. 5.2), which for this problem is

$$\|\mathbf{s}_2 - \mathbf{s}_1\| = \sqrt{\theta_1^2 + \theta_2^2 - 2\theta_1\theta_2 \cos\phi}.$$

Using (5.19) and (5.16), the equation above can be manipulated to yield (5.17).

To show (5.18), we can substitute (5.19) into (5.8), apply the product-to-sum formula, and then manipulate the resulting equation algebraically which yields (5.18).

□

5.4.4 Distance ratio constants

We now approach the problem of analytically determining the distance ratio constant of exponential coordinates. We will start by deriving the distance ratio constant of exponential coordinates with angles on the interval $[0, 2\pi]$. We will then use this result to determine the distance ratio constant of unconstrained exponential coordinates and exponential coordinates in Theorems and 5.2 and 5.3.

Lemma 5.2. *The distance ratio constant of exponential coordinates* $\mathbf{s}_1 = \theta_1\mathbf{e}_1$ *and* $\mathbf{s}_2 = \theta_2\mathbf{e}_2$ *for which* $\theta_1, \theta_2 \in [0, 2\pi]$ *is 1:*

$$\mu = \sup_{\substack{\mathbf{s}_1, \mathbf{s}_2 \in \mathcal{B}_{2\pi} \\ \mathbf{s}_1 \neq \mathbf{s}_2}} \frac{\text{dist}(\mathbf{s}_1, \mathbf{s}_2)}{\|\mathbf{s}_2 - \mathbf{s}_1\|} = 1.$$

Proof. Recall from Lemma 5.1 that the Euclidean and rotational distances between any pair of exponential coordinates can be written as (5.17) and (5.18) where $t \in [0, 1]$. Using these

equations, we can write the argument of the distance ratio constant as

$$\frac{\text{dist}(\mathbf{s}_1, \mathbf{s}_2)}{\|\mathbf{s}_2 - \mathbf{s}_1\|} = \frac{2\cos^{-1}\left(|t\cos(\frac{\theta_1}{2} + \frac{\theta_2}{2}) + (1-t)\cos(\frac{\theta_1}{2} - \frac{\theta_2}{2})|\right)}{\sqrt{(\theta_1 - \theta_2)^2 + 4\theta_1\theta_2 t}}. \tag{5.20}$$

Using convexity, we will first show that the equation above is less than or equal to one, for the case in which $\theta_1, \theta_2 \in [0, 2\pi]$ and $\theta_1 + \theta_2 \in [0, 2\pi]$.

Consider the function $\cos(\sqrt{x})$. We note that this function is convex on the interval $x \in [0, \pi^2]$, which can be verified by determining that the function's second derivative is non-negative on this interval.

So for $t \in [0, 1]$, using the definition of convexity, we have

$$t\cos(\sqrt{x_1}) + (1-t)\cos(\sqrt{x_2}) \geq \cos(\sqrt{tx_1 + (1-t)x_2}). \tag{5.21}$$

Considering θ_1 and θ_2 such that $\theta_1, \theta_2 \in [0, 2\pi]$ and $\theta_1 + \theta_2 \in [0, 2\pi]$, we set $x_1 = (\frac{\theta_1}{2} + \frac{\theta_2}{2})^2$ and $x_2 = (\frac{\theta_1}{2} - \frac{\theta_2}{2})^2$. Note that $(\frac{\theta_1}{2} + \frac{\theta_2}{2})^2 \in [0, \pi^2]$ and $(\frac{\theta_1}{2} - \frac{\theta_2}{2})^2 \in [0, \pi^2]$. Substituting these values into (5.21) and manipulating the equation algebraically yields

$$t\cos(\tfrac{\theta_1}{2} - \tfrac{\theta_2}{2}) + (1-t)\cos(\tfrac{\theta_1}{2} + \tfrac{\theta_2}{2}) \geq \cos(\tfrac{1}{2}\sqrt{(\theta_1 - \theta_2)^2 + 4\theta_1\theta_2 t}) \tag{5.22}$$

where we have used the fact that $\cos(\sqrt{x^2}) = \cos(|x|) = \cos(x)$.

Next, we will apply the arccosine function to both sides of (5.22). Note that the argument of the cosine function on the RHS of (5.22) is on the interval $[0, \pi]$ and therefore applying the arccosine function to the RHS will return the original argument of the cosine function. Also, note that the LHS is on the interval $[-1, 1]$ because it is a convex combination of cosine functions. Since arccosine is a decreasing function, we can apply it to both sides of (5.22)

and flip the inequality:

$$\cos^{-1}\left(t\cos(\tfrac{\theta_1}{2}-\tfrac{\theta_2}{2})+(1-t)\cos(\tfrac{\theta_1}{2}+\tfrac{\theta_2}{2})\right)\leq\frac{1}{2}\sqrt{(\theta_1-\theta_2)^2+4\theta_1\theta_2 t}.$$

Furthermore, since arccosine is a decreasing function, we have $\cos^{-1}(|x|)\leq\cos^{-1}(x)$ and we can write the equation above as

$$2\cos^{-1}\left(\left|t\cos\left(\tfrac{\theta_1}{2}-\tfrac{\theta_2}{2}\right)+(1-t)\cos\left(\tfrac{\theta_1}{2}+\tfrac{\theta_2}{2}\right)\right|\right)\leq\sqrt{(\theta_1-\theta_2)^2+4\theta_1\theta_2 t}.$$

Dividing by the RHS yields

$$\frac{2\cos^{-1}\left(\left|t\cos(\frac{\theta_1}{2}+\frac{\theta_2}{2})+(1-t)\cos(\frac{\theta_1}{2}-\frac{\theta_2}{2})\right|\right)}{\sqrt{(\theta_1-\theta_2)^2+4\theta_1\theta_2 t}}\leq 1.$$

We can see that the expression on the LHS is the same as the RHS of (5.20). So, we have shown that the distance ratio constant is less than or equal to one for cases in which $\theta_1,\theta_2\in[0,2\pi]$ and $\theta_1+\theta_2\in[0,2\pi]$. We can complete the Lemma by showing the distance ratio constant is less than or equal to one in cases for which $\theta_1,\theta_2\in[0,2\pi]$ and $\theta_1+\theta_2\in[2\pi,4\pi]$.

Note that any points such that $\theta_1,\theta_2\in[0,2\pi]$ and $\theta_1+\theta_2\in[2\pi,4\pi]$ can be written as

$$\theta_1=2\pi-\theta_1'$$
$$\theta_2=2\pi-\theta_2'$$

where $\theta_1'+\theta_2'\in[0,2\pi]$. Plugging (θ_1,θ_2,t) and (θ_1',θ_2',t) into (5.18) shows that their rotational distances are the same. However, the Euclidean distance of (θ_1,θ_2,t) is larger than that of (θ_1',θ_2',t), which can be seen by referencing (5.17) and showing that the square of the Euclidean distance of (θ_1,θ_2,t) is larger than that of (θ_1',θ_2',t). Taking the difference in the

square of the Euclidean distances yields

$$((\theta_1 - \theta_2)^2 + 4\theta_1\theta_2 t) - ((\theta_1' - \theta_2')^2 + 4\theta_1'\theta_2' t) = 8\pi t(2\pi - (\theta_1' + \theta_2')).$$

Since t is non-negative, and $\theta_1' + \theta_2' \in [0, 2\pi]$, then the RHS of the expression above is non-negative, which implies the Euclidean distance of (θ_1, θ_2, t) is larger than that of $(\theta_1', \theta_2', t)$. Since the Euclidean distance of (θ_1, θ_2, t) is larger than that of $(\theta_1', \theta_2', t)$, but the rotational distance is the same for both, it follows that (5.20) for (θ_1, θ_2, t) is less than or equal to that for $(\theta_1', \theta_2', t)$, which we have showed is bounded by one.

The final step is to show that (5.20) achieves the upper bound of one. There are a few cases in which it is achieved, one of which being when $\theta_1 \in (0, \pi]$ and $\theta_2 = 0$, in which case both $\text{dist}(\mathbf{s}_1, \mathbf{s}_2)$ and $\|\mathbf{s}_2 - \mathbf{s}_1\|$ equal θ_1.

□

Next, we can use Lemma 5.2 to derive the distance ratio constant for unconstrained exponential coordinates. Note that the set of unconstrained exponential coordinates is all of $\mathbb{R}^3$.

Theorem 5.2. *The distance ratio constant of unconstrained exponential coordinates is 1:*

$$\mu = \sup_{\substack{\mathbf{s}_1, \mathbf{s}_2 \in \mathbb{R}^3 \\ \mathbf{s}_1 \neq \mathbf{s}_2}} \frac{\text{dist}(\mathbf{s}_1, \mathbf{s}_2)}{\|\mathbf{s}_2 - \mathbf{s}_1\|} = 1.$$

Proof. Recall from Lemma 5.1 that the Euclidean and rotational distances between any pair of exponential coordinates can be written as

$$\|\mathbf{s}_2 - \mathbf{s}_1\| = \sqrt{(\theta_1 - \theta_2)^2 + 4\theta_1\theta_2 t} \tag{5.23}$$

$$\text{dist}(\mathbf{s}_1, \mathbf{s}_2) = 2\cos^{-1}(|t\cos(\tfrac{\theta_1}{2} + \tfrac{\theta_2}{2}) + (1-t)\cos(\tfrac{\theta_1}{2} - \tfrac{\theta_2}{2})|) \tag{5.24}$$

respectively where $t \in [0, 1]$. We will start by considering the two following cases: $|\theta_1 - \theta_2| \leq \pi$ and $|\theta_1 - \theta_2| > \pi$.

For the case in which $|\theta_1 - \theta_2| > \pi$, upon inspection of (5.23), we can see that the "$(\theta_1 - \theta_2)^2$" term will be larger than π^2, and since θ_1, θ_2, and t are non-negative, the $4\theta_1\theta_2 t$ term will be non-negative. Therefore, the Euclidean distance will always be larger than π which is the maximum possible rotational distance, which implies that $\text{dist}(\mathbf{s}_1, \mathbf{s}_2)/\|\mathbf{s}_2 - \mathbf{s}_1\| < 1$ for all points such that $|\theta_1 - \theta_2| > \pi$.

For the case in which $|\theta_1 - \theta_2| \leq \pi$, we can shift θ_1 and θ_2 by a multiple of 2π as follows:

$$\theta_1' = \theta_1 - 2n\pi$$
$$\theta_2' = \theta_2 - 2n\pi$$

where $\theta_1', \theta_2' \in [-\pi, 2\pi]$ and $n \in \{0, 1, 2, ...\}$, and the larger of θ_1' and θ_2' is non-negative. Without loss of generality, we can assume that $\theta_1' > \theta_2'$, which results in two scenarios: $\theta_1' \geq 0$ and $\theta_2' < 0$, or $\theta_1' \geq 0$ and $\theta_2' \geq 0$. For both scenarios, we will show that $\text{dist}(\mathbf{s}_1, \mathbf{s}_2)/\|\mathbf{s}_2 - \mathbf{s}_1\|$ is smaller for (θ_1, θ_2, t) than $(\theta_1', \theta_2', t)$ which is bounded by one.

Case 1: $\theta_1' \geq 0$ and $\theta_2' \geq 0$. Both (θ_1, θ_2, t) and $(\theta_1', \theta_2', t)$ correspond to the same 3D rotation, so they have the same rotational distance. However, the Euclidean distance will be larger for (θ_1, θ_2, t), which is apparent by inspecting (5.23), where we can see that the "$(\theta_1 - \theta_2)^2$" term will be the same for both sets of points, but the "$4\theta_1\theta_2 t$" term will be larger for (θ_1, θ_2, t) than for $(\theta_1', \theta_2', t)$ since $0 \leq \theta_1' \leq \theta_1$ and $0 \leq \theta_2' \leq \theta_2$. Since (θ_1, θ_2, t) has the same rotational distance but a larger Euclidean distance than $(\theta_1', \theta_2', t)$, its value of $\text{dist}(\mathbf{s}_1, \mathbf{s}_2)/\|\mathbf{s}_2 - \mathbf{s}_1\|$ is smaller than that of $(\theta_1', \theta_2', t)$ which by Lemma 5.2 we know is less than or equal to one.

Case 2: $\theta_1' \geq 0$ and $\theta_2' < 0$. Both (θ_1, θ_2, t) and $(\theta_1', \theta_2', t)$ correspond to the same 3D rotation, so they have the same rotational distance. However, the Euclidean distance will

be larger for (θ_1, θ_2, t), which is apparent by inspecting (5.23), where we can see that the "$(\theta_1 - \theta_2)^2$" term will be the same for both sets of points, but the "$4\theta_1\theta_2 t$" term for $(\theta_1', \theta_2', t)$ will be non-positive and therefore smaller than that of (θ_1, θ_2, t) which is non-negative. Unlike Case 1, we cannot apply Lemma 5.2 to $(\theta_1', \theta_2', t)$ because $\theta_2' < 0$. However, we can note that since the axis-angle $(\theta, \mathbf{e})$ is equivalent to $(-\theta, -\mathbf{e})$, the pair $(\theta_1', \theta_2', t)$ is identical to the pair $(-\theta_1', \theta_2', 1-t)$, where $-\theta_1' \in [0, \pi]$ and $1-t \in [0, 1]$. So since (θ_1, θ_2, t) has the same rotational distance but a larger Euclidean distance than $(\theta_1', \theta_2', t)$ and $(-\theta_1', \theta_2', 1-t)$, it has a smaller value of $\text{dist}(\mathbf{s}_1, \mathbf{s}_2)/\|\mathbf{s}_2 - \mathbf{s}_1\|$ than $(\theta_1', \theta_2', t)$ and $(-\theta_1', \theta_2', 1-t)$ which by Lemma 5.2 we know is less than or equal to one.

□

Finally, we show that the distance ratio constant of exponential coordinates with angles $\theta \in [0, \pi]$ also equals one.

Theorem 5.3. *The distance ratio constant of exponential coordinates* $\mathbf{s}_1 = \theta_1\mathbf{e}_1$ *and* $\mathbf{s}_2 = \theta_2\mathbf{e}_2$ *for which* $\theta_1, \theta_2 \in [0, \pi]$ *is 1:*

$$\mu = \sup_{\substack{\mathbf{s}_1, \mathbf{s}_2 \in \mathcal{B}_\pi \\ \mathbf{s}_1 \neq \mathbf{s}_2}} \frac{\text{dist}(\mathbf{s}_1, \mathbf{s}_2)}{\|\mathbf{s}_2 - \mathbf{s}_1\|} = 1.$$

Proof. Lemma 5.2 states that the distance ratio constant of exponential coordinates with angles $\theta \in [0, 2\pi]$ is one. Exponential coordinates with angles $\theta \in [0, \pi]$ are a subset of those exponential coordinates, and therefore the distance ratio constant of exponential coordinates with angles $\theta \in [0, \pi]$ is upper bounded by one. Furthermore, this upper bound is achieved in a few ways, one of which being when $\theta_1 \in (0, \pi]$ and $\theta_2 = 0$, in which case both $\text{dist}(\mathbf{s}_1, \mathbf{s}_2)$ and $\|\mathbf{s}_2 - \mathbf{s}_1\|$ equal θ_1. □

5.5 Quaternions

5.5.1 Rotation parameterization

Quaternions are a common representation of rotation which can be derived from an axis-angle representation. Considering an angle $\theta \in \mathbb{R}$ and unit-length axis $\mathbf{e} \in \mathbb{R}^3$, a quaternion can be written as [27, App. A]

$$\mathbf{q} = \begin{bmatrix} \cos\frac{\theta}{2} \\ \mathbf{e}\sin\frac{\theta}{2} \end{bmatrix} \in \mathbb{R}^4. \tag{5.25}$$

Note that the set of all possible quaternions corresponds to all 4D vectors with unit norm, which is equivalent to all points on the 3-sphere $\mathcal{S}^3 \subset \mathbb{R}^4$:

$$\mathcal{S}^3 = \{\mathbf{x} \in \mathbb{R}^4 \quad | \quad \|\mathbf{x}\| = 1\}. \tag{5.26}$$

Note that quaternions are a double-covering of all possible 3D rotations. In other words, there are two quaternions that correspond to every 3D rotation. More specifically, every pair of antipodal quaternions represent the same rotation, which can be seen mathematically by taking quaternion $\mathbf{q}$ with angle θ, and replacing θ with the equivalent angle $\theta + 2\pi$. Note that while we could reduce the set $\mathcal{S}_3$ so that it is a 1-to-1 mapping, it would not change the results in this section.

5.5.2 Rotational distance

We can derive the rotational distance between two quaternions using the equation for the rotational distance of exponential coordinates in Proposition 5.2, which is written as a function of axis $\mathbf{e}$ and angle θ.

Proposition 5.3. *The rotational distance between quaternions $\mathbf{q}_1, \mathbf{q}_2 \in \mathcal{S}^3$ is given by the*

following equation:

$$\text{dist}(\mathbf{q}_1, \mathbf{q}_2) = 2\cos^{-1}(|\mathbf{q}_1 \cdot \mathbf{q}_2|). \tag{5.27}$$

Proof. Using (5.25), we can write the quaternion vectors as

$$\mathbf{q}_1 = \begin{bmatrix} \cos\frac{\theta_1}{2} \\ \mathbf{e}_1 \sin\frac{\theta_1}{2} \end{bmatrix}, \qquad \mathbf{q}_2 = \begin{bmatrix} \cos\frac{\theta_2}{2} \\ \mathbf{e}_2 \sin\frac{\theta_2}{2} \end{bmatrix}.$$

Taking the dot product of $\mathbf{q}_1$ and $\mathbf{q}_2$ yields

$$\mathbf{q}_1 \cdot \mathbf{q}_2 = \cos\tfrac{\theta_1}{2}\cos\tfrac{\theta_2}{2} + \mathbf{e}_1 \cdot \mathbf{e}_2 \sin\tfrac{\theta_1}{2}\sin\tfrac{\theta_2}{2}. \tag{5.28}$$

The RHS of (5.28) is exactly the expression in the absolute value in (5.8), which is the equation for the rotational distance between axis-angle pairs (θ_1,$\mathbf{e}_1$) and (θ_2,$\mathbf{e}_2$). Since $\mathbf{q}_1$ and $\mathbf{q}_2$ correspond to the same 3D rotations as (θ_1,$\mathbf{e}_1$) and (θ_2,$\mathbf{e}_2$), respectively, plugging (5.28) into (5.8) yields (5.27). □

We can also express the rotational distance between two quaternions as a function of the angle between their 4D quaternion vectors. This result will be used in Section 5.5.3.

Proposition 5.4. *Consider two quaternions $\mathbf{q}_1, \mathbf{q}_2 \in \mathcal{S}^3$, and let ϕ denote the angle between the 4D quaternion vectors. The rotational distance between the quaternions can be expressed in terms of ϕ by the following equation:*

$$\text{dist}(\mathbf{q}_1, \mathbf{q}_2) = \begin{cases} 2\phi, & \phi \in [0, \pi/2] \\ 2\pi - 2\phi, & \phi \in [\pi/2, \pi]. \end{cases} \tag{5.29}$$

Proof. Applying the formula which relates the dot product and angle between two vectors,

and noting that $\mathbf{q}_1$ and $\mathbf{q}_2$ are unit length, we have

$$\cos\phi = \frac{\mathbf{q}_1 \cdot \mathbf{q}_2}{\|\mathbf{q}_1\|\|\mathbf{q}_2\|} = \mathbf{q}_1 \cdot \mathbf{q}_2. \tag{5.30}$$

Plugging (5.30) into (5.27) yields

$$\text{dist}(\mathbf{q}_1, \mathbf{q}_2) = 2\cos^{-1}(|\cos\phi|). \tag{5.31}$$

Now note the following property

$$\cos^{-1}(|\cos\phi|) = \begin{cases} \phi, & \phi \in [0, \pi/2] \\ \pi - \phi, & \phi \in [\pi/2, \pi]. \end{cases} \tag{5.32}$$

Plugging (5.32) into (5.31) yields (5.29). □

5.5.3 2D interpretation

To determine the distance ratio constant of quaternions, we need to optimize over all possible pairs of quaternions, where each pair consists of the eight degrees of freedom in $\mathbf{q}_1$ and $\mathbf{q}_2$, and the two constraints $\|\mathbf{q}_1\| = 1$ and $\|\mathbf{q}_2\| = 1$. The number of dimensions and constraints in this analysis makes it challenging, but we can interpret this problem in 2D which reduces the number of variables and simplifies the analysis.

Lemma 5.3. *Consider two quaternions $\mathbf{q}_1, \mathbf{q}_2 \in \mathcal{S}^3$, and let c denote their Euclidean distance. The ratio of the rotational distance and Euclidean distance between the two quater-*

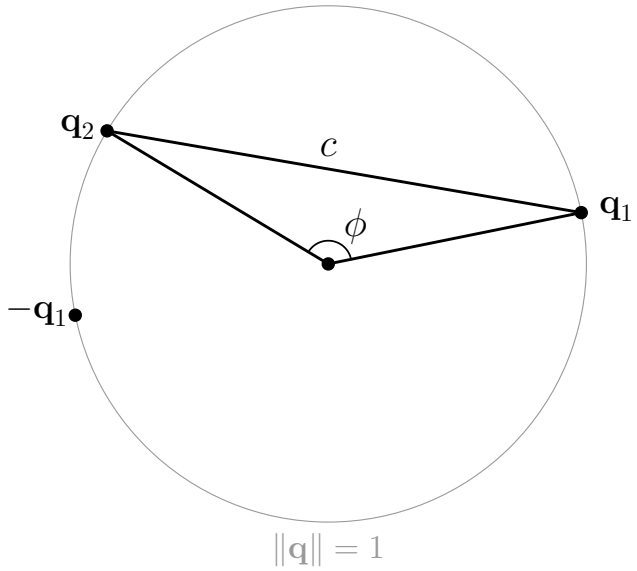


Figure 5.3: Quaternions $\mathbf{q}_1$ and $\mathbf{q}_2$ shown in 2D. In 2D, two quaternions can be analyzed in terms of the angle ϕ between the vectors, and the chord length c, which is equivalent to their Euclidean distance $\|\mathbf{q}_2 - \mathbf{q}_1\|$. Note that when the angle ϕ is larger than $\pi/2$, then $\mathbf{q}_2$ will be closer to $-\mathbf{q}_1$ than $\mathbf{q}_1$.

nions is given by the following equation:

$$\frac{\text{dist}(\mathbf{q}_1, \mathbf{q}_2)}{\|\mathbf{q}_2 - \mathbf{q}_1\|} = \begin{cases} 4\sin^{-1}(\frac{c}{2})/c, & c \in (0, \sqrt{2}] \\ (2\pi - 4\sin^{-1}(\frac{c}{2}))/c, & c \in [\sqrt{2}, 2]. \end{cases} \tag{5.33}$$

Proof. We start by noting that any two quaternions lie in a 2D plane, and hence we can analyze their relationship in 2D. A diagram is shown in Fig. 5.3. Since $\mathbf{q}_1$ and $\mathbf{q}_2$ lie on the unit circle, the distance between them can be interpreted as a chord length. Denoting this chord length as c, we have

$$c = \|\mathbf{q}_2 - \mathbf{q}_1\|. \tag{5.34}$$

Let ϕ denote the angle between the $\mathbf{q}_1$ vector and the $\mathbf{q}_2$ vector. We can relate the chord length c, angle ϕ, and radius ρ using the trigonometric formula $\phi = 2\sin^{-1}(\frac{c}{2\rho})$. Since

quaternions are unit-length, in this case $\rho = 1$, and we can write the relationship between c and ϕ as

$$\phi = 2\sin^{-1}(\tfrac{c}{2}). \tag{5.35}$$

Plugging (5.35) into (5.29), and converting the intervals accordingly, yields

$$\text{dist}(\mathbf{q}_1, \mathbf{q}_2) = \begin{cases} 4\sin^{-1}(\tfrac{c}{2}), & c \in (0, \sqrt{2}] \\ 2\pi - 4\sin^{-1}(\tfrac{c}{2}), & c \in [\sqrt{2}, 2]. \end{cases} \tag{5.36}$$

Plugging (5.36) and (5.34) into the LHS of (5.33) yields the RHS of (5.33). Note that in (5.33) we do not consider $c = 0$ as it corresponds to $\mathbf{q}_1 = \mathbf{q}_2$, which is not considered in the distance ratio constant from Definition 5.4. □

5.5.4 Distance ratio constant

Next, we will derive the distance ratio constant. To make our analysis easier, we will use the 2D interpretation from Section 5.5.3.

Theorem 5.4. *The distance ratio constant of quaternions is* $\pi/\sqrt{2}$*:*

$$\mu = \sup_{\substack{\mathbf{q}_1, \mathbf{q}_2 \in \mathcal{S}^3 \\ \mathbf{q}_1 \neq \mathbf{q}_2}} \frac{\text{dist}(\mathbf{q}_1, \mathbf{q}_2)}{\|\mathbf{q}_2 - \mathbf{q}_1\|} = \frac{\pi}{\sqrt{2}}.$$

Proof. In Lemma 5.3 we showed that we can write the sensitivity of any two quaternions in terms of the single variable c, which represents their Euclidean distance and chord length in

2D. Taking the derivative of (5.33) with respect to c yields

$$\frac{d}{dc}\frac{\text{dist}(\mathbf{q}_1,\mathbf{q}_2)}{\|\mathbf{q}_2-\mathbf{q}_1\|}=\begin{cases}\frac{4}{c^2}\left(\frac{c}{\sqrt{4-c^2}}-\sin^{-1}(\frac{c}{2})\right), & c\in(0,\sqrt{2}]\\ \frac{-4}{c^2}\left(\frac{c}{\sqrt{4-c^2}}+\cos^{-1}(\frac{c}{2})\right), & c\in[\sqrt{2},2].\end{cases} \tag{5.37}$$

First consider the $c\in(0,\sqrt{2}]$ case of (5.37). The second term can be rearranged as

$$\begin{aligned}\frac{c}{\sqrt{4-c^2}}-\sin^{-1}(\tfrac{c}{2}) &= (\tfrac{c}{2})/\sqrt{(1-(\tfrac{c}{2}))^2}-\sin^{-1}(\tfrac{c}{2})\\ &= \tan(\sin^{-1}(\tfrac{c}{2}))-\sin^{-1}(\tfrac{c}{2})\end{aligned} \tag{5.38}$$

where in the last step we used the trig identity $\tan(\sin^{-1}x)=x/\sqrt{1-x^2}$. Note that $c/2$ is positive, so $\sin^{-1}(\frac{c}{2})$ is on the interval $(0,\pi/2]$. Since $\tan x > x$ for $x\in(0,\pi/2]$, the bottom equation in (5.38) is positive, which since $c>0$, implies the $c\in(0,\sqrt{2}]$ case of (5.37) is positive.

Next, consider the $c\in[\sqrt{2},2]$ case of (5.37). Restricting the interval to $[\sqrt{2},2)$, we can see that the second term consists of the sum of two non-negative terms, which is non-negative. Since $-4/c^2$ is negative, the $c\in[\sqrt{2},2)$ case of (5.37) is non-positive.

In summary, the derivative of (5.33) with respect to c is positive for $c\in(0,\sqrt{2}]$ and non-positive for $c\in[\sqrt{2},2)$, so (5.33) is maximized at $c=\sqrt{2}$ on the interval $c\in(0,2)$. Plugging $c=\sqrt{2}$ into (5.33) yields $\pi/\sqrt{2}$. As (5.33) equals zero when $c=2$, it has a maximum value of $\pi/\sqrt{2}$ on the interval $(0,2]$. □

5.5.5 Unconstrained quaternions

We now consider unconstrained quaternions, which are constructed by eliminating the unit-length constraint of quaternions, thereby allowing any vector in $\mathbb{R}^4$ to be associated with a 3D rotation after normalization into a unit quaternion. Unconstrained quaternions are advan-

tageous as they allow a neural network to output any 4D vector as a valid parameterization of rotation. We now derive the distance ratio constant of unconstrained quaternions.

Theorem 5.5. *The distance ratio constant of unconstrained quaternions is* ∞*:*

$$\mu = \sup_{\substack{\mathbf{q}_1, \mathbf{q}_2 \in \mathbb{R}^4 \\ \mathbf{q}_1 \neq \mathbf{q}_2}} \frac{\text{dist}(\mathbf{q}_1, \mathbf{q}_2)}{\|\mathbf{q}_2 - \mathbf{q}_1\|} = \infty.$$

Proof. Consider any two unconstrained quaternions $\mathbf{q}_1, \mathbf{q}_2 \in \mathbb{R}^4$ which do not correspond to the same the rotation. Let $\alpha > 0$ denote their rotational distance. Now consider the scalar ϵ and scalar multiples of $\mathbf{q}_1$ and $\mathbf{q}_2$: $\epsilon\mathbf{q}_1$ and $\epsilon\mathbf{q}_2$. Since scaling an unconstrained quaternion does not change its associated rotation, we have

$$\alpha = \text{dist}(\mathbf{q}_1, \mathbf{q}_2) = \text{dist}(\epsilon\mathbf{q}_1, \epsilon\mathbf{q}_2) > 0.$$

Taking the limit as ϵ goes to zero, we have

$$\lim_{\epsilon \to 0} \frac{\text{dist}(\epsilon\mathbf{q}_1, \epsilon\mathbf{q}_2)}{\|\epsilon\mathbf{q}_2 - \epsilon\mathbf{q}_1\|} = \frac{\alpha}{0} = \infty.$$

□

Mapping an unconstrained quaternion to a unit quaternion is performed using a unit-normalization function. We now calculate the Euclidean Lipschitz constant of a unit-normalization function.

Proposition 5.5. *The unit-normalization function has a Euclidean Lipschitz constant of* ∞*:*

$$\sup_{\mathbf{x}_1 \neq \mathbf{x}_2} \frac{\|\mathbf{f}(\mathbf{x}_2) - \mathbf{f}(\mathbf{x}_1)\|}{\|\mathbf{x}_2 - \mathbf{x}_1\|} = \infty. \tag{5.39}$$

Proof. To prove this proposition, we can show that the fraction in (5.39) approaches infinity for some vectors $\mathbf{x}_1$ and $\mathbf{x}_2$. Consider two unit vectors $\mathbf{u}_1 \neq \mathbf{u}_2$. Let $\epsilon\mathbf{u}_1$ and $\epsilon\mathbf{u}_2$ denote scalings of these vectors by $\epsilon > 0$. Since $\mathbf{f}$ is a normalization function, we have $\mathbf{f}(\epsilon\mathbf{u}_1) = \mathbf{u}_1$ and $\mathbf{f}(\epsilon\mathbf{u}_2) = \mathbf{u}_2$. Let b denote the Euclidean distance between $\mathbf{f}(\mathbf{x}_1)$ and $\mathbf{f}(\mathbf{x}_2)$:

$$b = \|\mathbf{f}(\epsilon\mathbf{u}_2) - \mathbf{f}(\epsilon\mathbf{u}_1)\| = \|\mathbf{u}_2 - \mathbf{u}_1\| > 0.$$

The fraction in (5.39) for vectors $\epsilon\mathbf{u}_1$ and $\epsilon\mathbf{u}_2$ as ϵ approaches 0 is

$$\lim_{\epsilon \to 0} \frac{\|\mathbf{f}(\epsilon\mathbf{u}_2) - \mathbf{f}(\epsilon\mathbf{u}_1)\|}{\|\epsilon\mathbf{u}_2 - \epsilon\mathbf{u}_1\|} = \frac{b}{0} = \infty.$$

□

5.6 2D projection parameterizations

5.6.1 Rotation parameterization

Many pose estimation neural networks parameterize pose via a set of 2D points, which represent 3D points on the object which have been projected into the 2D image plane [36, 49, 46]. Given a set of 2D points, and knowledge of the true 3D locations of the points on the object, the pose can be determined by solving the Perspective-n-Point (PnP) problem. The PnP problem is usually solved via an algorithm, and therefore the mapping from 2D points to pose usually cannot be expressed in closed-form.

Note that if the set of 2D points are exact projections of the 3D points, then the pose of the object can be determined exactly using geometric analysis. In these cases, it may be possible to derive the distance ratio constant analytically, as we have done for quaternions and exponential coordinates. However, in estimation applications, the 2D points will be inexact and have some error, which means the true pose cannot be determined exactly, and

instead must be estimated using a PnP algorithm. PnP algorithms usually rely on a nonlinear optimization, and are also often used in conjunction with other algorithms such as Random Sample Consensus (RANSAC). Unfortunately, the high complexity of these algorithms makes it intractable to analyze and derive closed-form sensitivity bounds for PnP methods, so we will not attempt to do so in this chapter.

5.6.2 Monte Carlo simulation

Although it is too difficult to mathematically analyze 2D projection methods, we will perform a Monte Carlo simulation which shows the rotational sensitivity for an example pose estimation scenario. In this simulation we consider a cube, and we consider the points of interest to be the eight vertices of the cube. We generate sample data by placing the cube at random poses, and project the points of interest into 2D. We then consider two cases: the case in which we know the 2D points exactly, and the case in which the 2D points have some error, in which case the pose must be estimated using a PnP algorithm. In both cases, we can determine the rotational sensitivity for each sample.

The parameters of a 2D projection parameterization can be described as a vector $\mathbf{v} \in \mathbb{R}^{2n}$ where n denotes the number of points of interest on the object. For parameters $\mathbf{v}_1, \mathbf{v}_2 \in \mathbb{R}^{2n}$, we can write the argument of the distance ratio constant in (5.4) as $\text{dist}(\mathbf{v}_1, \mathbf{v}_2)/\|\mathbf{v}_2 - \mathbf{v}_1\|$.

The results are shown in Fig. 5.4. We can observe that there is greater variation in sensitivity when the 2D points have noise.

5.7 Summary of rotational Lipschitz constants for various rotation parameterizations

In summary, we have derived the distance ratio constant for exponential coordinates and quaternions, as well as unconstrained versions of both of these parameterizations. The results are summarized in Fig. 5.5. We also verified these results to be accurate by randomly

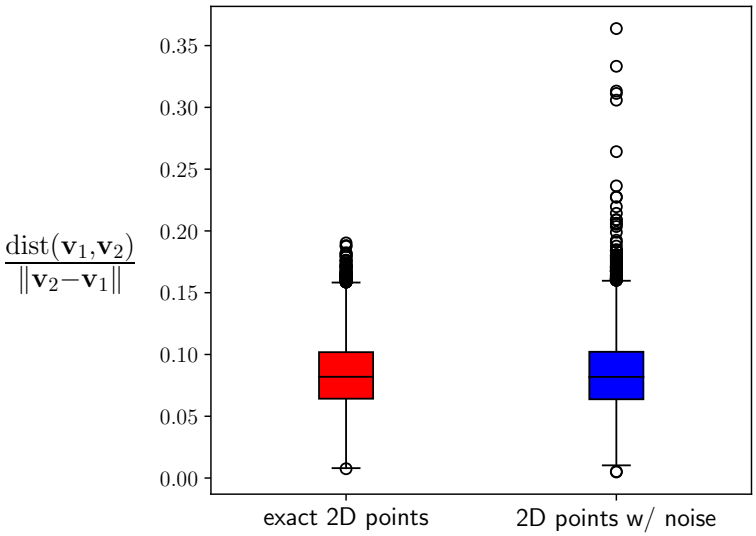


Figure 5.4: Monte Carlo simulation showing the rotational sensitivity when pose is parameterized by 2D projected points. In this simulation, the object of interest is a cube, and the points are the eight vertices of the cube. We consider 10,000 randomly sampled poses in this simulation. The red box plot shows the case in which the 2D points have no error, and the rotation can be exactly determined. The blue box plot shows the case in which the 2D points have some noise, and the rotation must be estimated using a PnP algorithm.

sampling 10^7 pairs of 3D rotations and calculating the argument of the distance ratio constant for each pair.

We now consider the task of constructing a pose estimation network using each of these parameterizations, and calculating the rotational Lipschitz constant for each network. Unfortunately, this task is difficult for most of these parameterizations. Using exponential coordinates would require that the neural network only output vectors with magnitude less than or equal to π, which is not straightforward to accomplish. Similarly, using quaternions would require the neural network to only output vectors with unit magnitude. This unit magnitude constraint could be enforced using a unit-normalization function, but as we showed in Proposition 5.5, this function has an infinite Lipschitz constant. Similarly, unconstrained quaternions have an unbounded distance ratio constant. Additionally, we do

parameterization	**symbol** ($\mathbf{p}$)	**set** ($\mathcal{P}$)	μ
exponential coordinates	$\mathbf{s}$	$\mathcal{B}_\pi$	1
exponential coordinates, unconstrained	$\mathbf{s}$	$\mathbb{R}^3$	1
quaternions	$\mathbf{q}$	$\mathcal{S}^3$	$\pi/\sqrt{2}$
quaternions, unconstrained	$\mathbf{q}$	$\mathbb{R}^4$	∞

Figure 5.5: Parameterizations, symbols, set of parameters, and distance ratio constant μ for various rotation parameterizations that we have derived in this chapter (see Theorems 5.3, 5.2, 5.4, & 5.5). The distance ratio constant μ is defined in (5.4).

not know of a way to calculate a closed-form rotational Lipschitz constant for 2D projection parameterizations.

However, one rotation parameterization we have considered, unconstrained exponential coordinates, do not have any of these challenges, as they associate any vector in $\mathbb{R}^3$ with a valid 3D rotation, and have a bounded distance ratio constant. As a result, it is possible to build a pose estimation neural network which outputs rotation in the form of an unconstrained three-dimensional vector, and then determine a bound on the rotational Lipschitz constant using Theorem 5.1.

5.8 Simulation

As we noted in the previous section, it is possible to calculate a bound on the rotational Lipschitz constant of a neural network which outputs unconstrained exponential coordinates. In this section, we will design and train such a network, and then calculate a bound on its rotational Lipschitz constant. Our network will output six-dimensional vectors, in which the first three states represent position and last three states represent unconstrained exponential coordinates.

We constructed a feedforward network with rectified linear unit (ReLU) activation functions, which has a similar architecture to the original AlexNet [23]. The network architecture

is shown in Fig. 5.6.

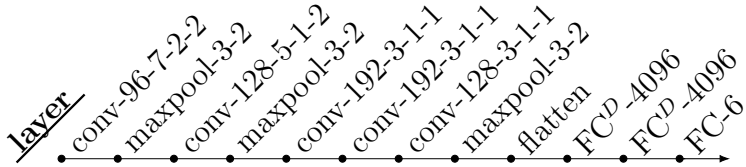


Figure 5.6: Sequence of layers (applied left-to-right) in our neural network. The layers are denoted as follows: conv-o-k-s-p denotes a convolution layer with o output channels, kernel size k, stride s and padding p; maxpool-k-s denotes a max pooling function with a kernel size of k and stride s; and FC-o denotes a fully-connected function with o output channels. The superscript D denotes that dropout is applied after the layer. All convolution and fully-connected layers, except for the final layer, have a ReLU activation.

We consider our object of interest to be the "soup can" object from the YCB data set. We generated each image in the training and test data by generating renderings of the soup can at random poses using the computer graphics program Blender, and then superimposing the rendering on a random background image. Our training data consisted of 200,000 renderings superimposed on random background images from the SUN397 dataset. Our test data consisted of 10,000 renderings superimposed on background images from the 2017 COCO dataset. Each image was of size $3\times256\times256$, and sample images are shown in Fig. 5.7.

We created the cost function for the network as follows. Let $\mathbf{z}_i \in \mathbb{R}^3$ and $\hat{\mathbf{z}}_i \in \mathbb{R}^3$ denote the true and estimated position for the i^{th} data point, respectively. Let $\mathbf{s}_i \in \mathcal{B}_\pi$ denote the true exponential coordinates, and $\hat{\mathbf{s}}_i \in \mathbb{R}^3$ denote the estimated unconstrained coordinates for the i^{th} data point. Letting m denote the number of data points, we used the following cost function:

$$\text{cost} = \sum_{i=1}^{m} \|\mathbf{z}_i - \hat{\mathbf{z}}_i\|^2 + \text{dist}(\mathbf{s}_i, \hat{\mathbf{s}}_i)$$

where $\text{dist}(\cdot, \cdot)$ is the rotational distance equation from (5.8).

Figure 5.7: Sample images used to train the pose estimation neural network used in the simulation. The object of interest is the "soup can" from the YCB dataset. The images were generated synthetically by rendering the object at a random pose using Blender, and then overlaying the render onto a random background image.

We trained the network using the Adadelta optimization algorithm with an initial learning rate of 10^{-1}, and reduced the learning rate to 10^{-2} when the error plateaued. We trained the network for 235 epochs, which resulted in average test sample rotation and position errors of 13° and 7cm, respectively. Note that about 1% of the network's exponential coordinate estimates had a norm larger than π (i.e., were outside of the set $\mathcal{B}_\pi$) for both training and test data.

The network outputs a six-dimensional vector which represents position and exponential coordinates. In order to analyze the position and rotation portions of the network independently, we split the network into two sub-networks by splitting the final fully-connected layer into two parts. We determined upper bounds on the Euclidean Lipschitz constants of the position and rotation sub-networks to be 13×10^9 and 84×10^9, respectively. Using the latter value along with Theorems 5.1 and 5.2, we determined the following upper bound on

the rotational Lipschitz constant of the network:

$$L_r \leq 84 \times 10^9.$$

Note that all of our computed Lipschitz bounds are very large, but are of similar magnitude to classification networks with comparable architectures.

Finally, we can use Proposition 5.1 to determine a bound on the rotational change of the output, given a bound on input perturbations. Using (5.6), we can determine that if the Euclidean distance of two inputs is less than or equal to $\epsilon = 1.1 \times 10^{-11}$, then the rotational distance between the outputs will be less than or equal to 1 radian. We could also determine this bound for different values of ϵ.

5.9 Summary

In this chapter, we approached the task of deriving sensitivity bounds for pose estimation neural networks. We created a sensitivity measure that is a type of Lipschitz constant which measures the maximum rotational distance between two network estimates with respect to the Euclidean distance between the corresponding inputs. We then showed how a bound on this measure can be calculated from the Euclidean Lipschitz constant of the network and the distance ratio constant of the rotation parameterization. We derived the distance ratio constant for various rotation parameterizations, and discussed why most of these parameterizations, except for unconstrained exponential coordinates, make it difficult to compute a network-wide bound. We then constructed a neural network which outputs pose as a concatenation of position and exponential coordinates, and calculated a bound on the rotational Lipschitz constant of the network.

There are several useful directions of future work we can consider. One is to further analyze 2D projection parameterizations. These parameterizations have shown to be very

effective when used in pose estimation neural networks, and are also versatile due to the fact that these networks can be applied to images with a variety of camera intrinsics.

Finally, we again note that our sensitivity bounds were very large, but are of comparable magnitude to sensitivity bounds for classification networks. The high sensitivity of these bounds is a direct result of the Euclidean Lipschitz constant bound of the network being large. Therefore, in order to decrease our bounds and make them more practical, it is important to develop a way to calculate tighter bounds on the Euclidean Lipschitz constant of a network, or to constrain the Euclidean Lipschitz constant of the network during training. Note that both of these areas are actively being researched and are applicable to a wide variety of neural networks beyond pose estimation networks.

We have now discussed four different topics of research, which correspond to Chapters 2, 3, 4, and 5 of this dissertation. In the next and final chapter, we will provide some additional insights, tie some of these topics together, and discuss some important areas of future work.

Chapter 6

CONCLUSION

In this dissertation, we have analyzed the areas of object pose estimation and pose estimation neural networks. We were motivated by the fact that deep learning techniques have shown to be successful for object classification and pose estimation, but are not theoretically well-understood, and lack rigorous tools for mathematical analysis. We developed several provable and mathematically rigorous methods which allow us to better understand, verify, and implement pose estimation neural networks, and neural networks in general. Our work employed tools from the areas of deep learning, control theory, mechanics, 3D rotations, estimation theory, and mathematical analysis.

We note that although a main focus of this work is deep learning, all of our chapters include contributions that are independent of the area of deep learning. In Chapter 2, we provided a connection between the task of visual pose estimation and observability in control theory, and the results are independent of which method of pose estimation is being used. In Chapter 3, we developed an unscented filter which parameterizes pose using exponential coordinates. This filter could be adapted to scenarios in which the pose estimates are provided by methods other than neural networks. In Chapter 4, we developed a technique to analyze the local Lipschitz constants of neural networks, but our overall approach of determining the local Lipschitz bound could be applied to composite functions other than neural networks. Finally, in Chapter 5, we defined and derived the distance ratio constant of several rotation parameterizations, which is a fundamental and new result in the area of 3D rotations.

Note that while we explored several areas, the most common thread that ties all of

our work together is the sensitivity of pose estimation neural networks and pose estimation in general. Therefore, continuing our sensitivity analysis is an important direction of future work, and further connects the work in this dissertation. There are several areas of sensitivity analysis we are particularly interested in, which we will discuss in detail: exploration of the mathematical functions which relate the 3D world to 2D images, further analysis of sensitivity bounds (as in Chapter 4), and constraining networks to limit their sensitivity.

The first area of future research we are interested in comes from the observation that image-based methods, such as object classification and pose estimation, are ultimately methods to discern properties of the 3D world from 2D images. Therefore, an interesting area of future research involves analyzing of the mathematical functions which describe how the 3D world is distilled into a 2D image. These mathematical functions are very complex, and depend on many physical properties of the 3D world, such as geometry, material properties, and lighting. These functions are often studied in the field of image rendering, which has produced rendering methods such as rasterization and ray tracing. However, rendering analysis is usually approached from the standpoint of computational efficiency, rather than the standpoint of rigorous mathematical analysis. Therefore, we believe that rigorous mathematical analysis of these functions is an open and interesting area of research. Note that this area of research is also related to differentiable rendering, which is currently actively being researched in connection with deep learning.

The next area of future research we are interested in involves further Lipschitz sensitivity analysis of neural networks, similar to our work in Chapter 4. One advantage of Lipschitz bounds is that they give a fundamental and general characterization of sensitivity, and can therefore be applied to classification networks, pose estimation networks (as in Chapter 5), and other networks. Also, as we have previously mentioned, the only method to calculate global Lipschitz bounds for most neural networks involves taking the product of the bounds

of each of the layers, which results in a very loose bound. A method to improve upon this bound would be of huge significance in the area deep learning. Furthermore, the framework of Lipschitz analysis is often general and can be applied to a variety of mathematical functions, so the impact of this work potentially extends beyond neural networks.

The final area of future research we are interested in involves constraining a network to limit its sensitivity. This area is dependent upon sensitivity analysis, as constraining a network requires a way of measuring a network's sensitivity. Constraining a network is currently a popular area of research, and constraints are often applied during the training of a network. Development of an effective method to constrain a network's sensitivity would have wide-ranging applicability in the field of deep learning. For example, such a method could be applied to our work by constructing the pose estimation network from Chapter 5 with lower sensitivity, which would result in a lower rotational sensitivity bound.

BIBLIOGRAPHY

[1] A. Alaeddini and K. A. Morgansen. Optimal disease outbreak detection in a community using network observability. In *American Control Conference (ACC)*, 2016.

[2] S. L. Altmann. Hamilton, Rodrigues, and the quaternion scandal. *Mathematics Magazine*, 62(5):291–308, 1989.

[3] P. L. Bartlett, D. J. Foster, and M. J. Telgarsky. Spectrally-normalized margin bounds for neural networks. In *Advances in Neural Information Processing Systems (NIPS)*. 2017.

[4] M. Brossard, S. Bonnabel, and J. Condomines. Unscented Kalman filtering on Lie groups. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[5] H. Coskun, F. Achilles, R. DiPietro, N. Navab, and F. Tombari. Long short-term memory Kalman filters: Recurrent neural estimators for pose regularization. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[6] J. L. Crassidis and J. L. Junkins. *Optimal Estimation of Dynamic Systems*. CRC Press, 2nd edition, 2011.

[7] S. Dittmer, E. J. King, and P. Maass. Singular values for ReLU layers. *IEEE Transactions on Neural Networks and Learning Systems*, 31(9):3594–3605, 2020.

[8] E. Eade. Lie groups for 2D and 3D transformations, 2017. Available at http://ethaneade.com/lie.pdf.

[9] M. Fazlyab, A. Robey, H. Hassani, M. Morari, and G. Pappas. Efficient and accurate estimation of Lipschitz constants for deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.

[10] Four lamps, 2012. Available at https://www.blendswap.com/blends/view/46257.

[11] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. arXiv, 2014.

[12] H. Gouk, E. Frank, B. Pfahringer, and M. J. Cree. Regularisation of neural networks by enforcing Lipschitz continuity. *Machine Learning*, 110(2):393–416, 2021.

[13] T. Haarnoja, A. Ajay, S. Levine, and P. Abbeel. Backprop KF: Learning discriminative deterministic state estimators. In *Advances in Neural Information Processing Systems (NIPS)*. 2016.

[14] S. Hauberg, F. Lauze, and K. S. Pedersen. Unscented Kalman filtering on Riemannian manifolds. *Journal of mathematical imaging and vision*, 46(1):103–120, 2013.

[15] B. T. Hinson and K. A. Morgansen. Observability-based optimal sensor placement for flapping airfoil wake estimation. *Journal of Guidance, Control, and Dynamics*, 37(5):1477–1486, 2014.

[16] M. Jordan and A. G. Dimakis. Exactly computing the local Lipschitz constant of ReLU networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[17] S. J. Julier and J. K. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004.

[18] W. Kang and L. Xu. Partial observability for some distributed parameter systems. *International Journal of Dynamics and Control*, 2(4):587–596, 2014.

[19] J. L. Kelley. *General Topology*. D. Van Nostrand Company, Inc., 1st edition, 1955.

[20] A. Kendall, M. Grimes, and R. Cipolla. PoseNet: A convolutional network for real-time 6-DOF camera relocalization. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[21] Kitchen chair, 2012. Available at https://www.blendswap.com/blends/view/40140.

[22] A. J. Krener and K. Ide. Measures of unobservability. In *48th IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, 2009.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *Conference on Neural Information Processing Systems (NIPS)*, 2012.

[24] F. Latorre, P. Rolland, and V. Cevher. Lipschitz constant estimation of neural networks via sparse polynomial optimization. In *International Conference on Learning Representations (ICLR)*, 2020.

[25] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[26] V. Lepetit and P. Fua. Monocular model-based 3D tracking of rigid objects: A survey. *Foundations and Trends in Computer Graphics and Vision*, 1(1):1–89, 2005.

[27] K. M. Lynch and F. C. Park. *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, 1st edition, 2017.

[28] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[29] R. M. Murray, S. S. Sastry, and Z. Li. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Inc., 1st edition, 1994.

[30] P. Müller and H. Weber. Analysis and optimization of certain qualities of controllability and observability for linear dynamical systems. *Automatica*, 8(3):237–246, 1972.

[31] H. Nijmeijer and A. van der Schaft. *Nonlinear Dynamical Control Systems*. Springer-Verlag, 1st edition, 1990.

[32] R. Novak, Y. Bahri, D. A. Abolafia, J. Pennington, and J. Sohl-Dickstein. Sensitivity and generalization in neural networks: an empirical study. In *International Conference on Learning Representations (ICLR)*, 2018.

[33] J. Peck, J. Roels, B. Goossens, and Y. Saeys. Lower bounds on the robustness to adversarial perturbations. In *Advances in Neural Information Processing Systems (NIPS)*. 2017.

[34] P. Poirson, P. Ammirato, C. Fu, W. Liu, J. Kosecka, and A. C. Berg. Fast single shot detection and pose estimation. In *International Conference on 3D Vision (3DV)*, 2016.

[35] J. Qi, K. Sun, and W. Kang. Optimal PMU placement for power system dynamic state estimation by using empirical observability Gramian. *IEEE Transactions on Power Systems*, 30(4):2041–2054, 2015.

[36] M. Rad and V. Lepetit. BB8: A scalable, accurate, robust to partial occlusion method for predicting the 3D poses of challenging objects without using depth. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[37] R. Rojas. *Neural Networks: A Systematic Introduction*. Springer, 1st edition, 1996.

[38] K. Scaman and A. Virmaux. Lipschitz regularity of deep neural networks: analysis and efficient estimation. In *Advances in Neural Information Processing Systems (NeurIPS)*. 2018.

[39] M. Ó. Searcòid. *Metric Spaces*. Springer, 1st edition, 2007.

[40] Seat 600 L '60, 2013. Available at https://www.blendswap.com/blends/view/69075.

[41] D. Simon. *Optimal State Estimation: Kalman, H-Infinity, and Nonlinear Approaches*. John Wiley & Sons, Inc., 1st edition, 2006.

[42] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015.

[43] A. Singh, J. Sha, K. S. Narayan, T. Achim, and P. Abbeel. BigBIRD: A large-scale 3D database of object instances. In *International Conference on Robotics and Automation (ICRA)*, 2014.

[44] J. Sokolić, R. Giryes, G. Sapiro, and M. R. D. Rodrigues. Robust large margin deep neural networks. *IEEE Transactions on Signal Processing*, 65(16):4265–4280, 2017.

[45] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. arXiv, 2013.

[46] B. Tekin, S. N. Sinha, and P. Fua. Real-time seamless single shot 6D object pose prediction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[47] D. Terjék. Adversarial Lipschitz regularization. In *International Conference on Learning Representations (ICLR)*, 2020.

[48] V. Tjeng, K. Y. Xiao, and R. Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations (ICLR)*, 2019.

[49] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. Birchfield. Deep object pose estimation for semantic robotic grasping of household objects. In *Conference on Robot Learning (CoRL)*, 2018.

[50] Y. Tsuzuku, I. Sato, and M. Sugiyama. Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*. 2018.

[51] T.-W. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, L. Daniel, D. S. Boning, and I. S. Dhillon. Towards fast computation of certified robustness for ReLU networks. In *International Conference on Machine Learning (ICML)*, 2018.

[52] T.-W. Weng, H. Zhang, P.-Y. Chen, J. Yi, D. Su, Y. Gao, C.-J. Hsieh, and L. Daniel. Evaluating the robustness of neural networks: An extreme value theory approach. In *International Conference on Learning Representations (ICLR)*, 2018.

[53] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox. PoseCNN: A convolutional neural network for 6D object pose estimation in cluttered scenes. In *Robotics: Science and Systems (RSS)*, 2018.

[54] D. Zou, R. Balan, and M. Singh. On Lipschitz bounds of general convolutional neural networks. *IEEE Transactions on Information Theory*, 66(3):1738–1759, 2020.