Assured Autonomy for Safety-Critical and Learning-Enabled Systems

Vicenc Rubies Royo

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2020-184
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-184.html

November 19, 2020

Assured Autonomy for Safety-Critical and Learning-Enabled Systems

by

Vicenc Rubies Royo

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering - Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Claire J. Tomlin, Chair
Professor S. Shankar Sastry
Professor Laurent El Ghaoui
Professor Dusan Stipanovic

Fall 2020

Assured Autonomy for Safety-Critical and Learning-Enabled Systems

Abstract

Assured Autonomy for Safety-Critical and Learning-Enabled Systems

by

Vicenc Rubies Royo

Doctor of Philosophy in Engineering - Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Claire J. Tomlin, Chair

Autonomous systems are becoming ever more complex. This growth in complexity stems primarily from continual improvements in computational power, which have enabled, among many things, the use of more sophisticated high-dimensional dynamical models or the use of deep neural networks for perception and decision-making. Unfortunately, this increase in complexity is coupled with an increase in uncertainty on how these systems might behave in safety-critical settings where guarantees of performance are needed.

In this dissertation, we will first address the challenges involved in the computation of safety certificates for high-dimensional safety-critical systems and how machine learning, and in particular artificial neural networks, can provide scalable approximate solutions which work well in practice. However, reliance on neural networks for autonomy poses itself a challenge, since these function approximators can sometimes produce erroneous behaviors when exposed to noise or adversarial attacks, for example. With this in mind, in the second half of the dissertation we will address the challenges involved in the verification of neural networks, and in particular, how to assess whether deep feedforward neural networks adhere to safety specifications.

To my parents, Joan and Teresita, and my sisters, Immaculada and Maria.

Sense vosaltres res d'això hauria estat possible.

Contents

List of Figures

List of Tables

Acknowledgments

First and foremost I would like to thank my advisor, Claire, for her continual support and encouragement during this journey. Thank you for all the times you gave me confidence when I couldn't find it in me. I could not have made it without you.

Special thanks to Professor Shankar Sastry for being part of my dissertation committee, and Professors Laurent El Ghaoui and Dusan Stipanovic for having been part of my qualifying exam committee, as well as my dissertation committee.

I also want to thank David Fridovich-Keil, Jaime Fisac, Sylvia Herbert, Somil Bansal and all of my friends and collaborators in the Hybrid Systems Lab for having made this PhD adventure the most intellectually rewarding period in my life. It has also been a pleasure to have had the opportunity to collaborate with Dusan Stipanovic, Roberto Calandra, Roy Dong, Eric Mazumdar and Dexter Scobee these past few years. While curiosity and the pursuit of knowledge were the primary factors that set off many of these collaborations, it was ultimately the human dimension, having met each one of them at a personal level, that made the whole experience truly worthwhile.

Life in the West Coast has felt in many ways similar to my life back at home in Barcelona. For that, I have to thank the (in)famous "Berkeley Gang" and its members, which include Alejandro Castillejo, Carlos Florensa, Ignasi Clavera, Julia Gómez, Eduard Ansaldo Giné, Miquel Crusells-Girona, Marc Dordal, and Batman. Thank you for all the joyful moments we have spent together.

Above all, I want to thank my family: my parents, Joan and Teresita, my sisters, Immaculada and Maria, and those who never truly left. Without their unconditional love and support none of this would have been possible.

Chapter 1

Introduction

Technological advancements of the past few decades have led historians to term this current period in history the Fourth Industrial Revolution [83, 75]. This period of technological progress is characterized by an unprecedented increase in automation arising from the fields of control, robotics and artificial intelligence. These advancements in automation are largely propelled by an outstanding increase in computational power not available in decades past. The systems that are currently being developed are more sophisticated, with a larger number of degrees of freedom and greater number of sensors. In consequence, models used to describe these systems are also becoming more complex and harder to analyze.

With increased complexity, however, a new problem arises: how can we provide assurances in the form of safety guarantees? When a system is governed by a simple set of rules it is often possible to accurately predict its behavior and correct it before it becomes unsafe. If instead the set of rules describing the system's behavior is too large and intricate, providing safety guarantees becomes a challenging task.

The aim for this dissertation will be to tackle the problem of safety assurance by providing a set of new methodologies for verification of complex safety-critical systems and learning-enabled systems. The overarching theme will be to use sound theoretical principles from control theory, optimization and machine learning, to deliver a set of algorithms capable of providing guarantees of performance and safety.

1.1 Autonomous Systems and Assured Autonomy

The term 'autonomous system' is a broad term meant to describe systems which can operate autonomously, that is, without explicit human inputs. For the purposes of this dissertation we will focus our attention on two sub-types of systems: safety-critical systems and learning-enabled systems.

Safety-Critical Systems

As the name suggests, safety-critical systems refer to autonomous systems for which safety is of critical importance. In particular, these are systems whose failure can result irreversible or catastrophic damage. Examples include most vehicles, such as cars, quadrotors and airplanes, but also civil infrastructures such as the power grid or water management structures. Models used to describe these systems are usually derived from first principles and exhibit a range of accuracies which allow to capture a wide range of possible future behaviors of the system and plan for future contingencies.

Learning-Enabled Systems

In this dissertation we will take the liberty of defining learning-enabled systems as those autonomous systems whose behavior relies on the outputs of an artificial (feedforward) neural network. Neural networks are a type of function approximator widely used in machine learning whose parameters are usually determined through an iterative optimization procedure (termed *training* or *learning*) such as stochastic gradient descent. Unlike safety-critical systems, the behavior of learning-enabled systems relies on the quality of the data provided for learning, as well as the learning algorithm itself. This dependency on data and learning algorithm can make learning-enabled systems unpredictable at run-time. Therefore, neural network models employed in safety-critical applications should always be verified before deployment.

Challenges in Assured Autonomy

The main goal of assured autonomy is to endow safety-critical and learning-enabled systems with guarantees of safety. As we shall see, providing these types of guarantees is a hard task. For safety-critical systems, the main challenge will be to scale verification tools to more complex, higher-dimensional models. For learning-enabled systems, the challenge will be to build those tools and also make them scalable.

1.2 Thesis Overview and Contributions

The main contribution of this thesis will be to provide safety analysis tools for large dynamical systems, and algorithms for the verification of feedforward neural networks. These tools are built on top of concepts drawn from reachability theory, optimization and machine learning, which ensures their robustness and reliability.

The thesis is organized in two parts. In the first half we will be concerned with safety-critical systems. In Ch. 2 we provide an introduction to dynamical models, which are the main mathematical framework for modeling physical systems. We then provide an overview on the topic of optimal control, how it ties to the concept of safety, and why previous approaches to safety analysis have poor scalability. In Ch. 3 we introduce a classification-based algorithm based on approximate dynamic programming which is able to provide safety guarantees for high-dimensional dynamical systems.

Using this algorithm we are able to accomplish a safe trajectory tracking task on a quadrotor. In Ch. 4 we develop a different algorithm for safety analysis based on concepts from reinforcement learning which can also scale to high-dimensional systems.

In the second half of the thesis we concern ourselves with learning-enabled systems. In Ch. 5 we provide an introduction to feedforward neural networks and the problem of input-output verification. We then draw a few connections between reachability theory and neural network verification. In Ch. 6 we introduce a method for efficiently over-approximating the mapping of a box-shaped input set through a neural network, and use it provide an iterative verification algorithm. The algorithm exploits so-called shadow price information in order to simplify the problem at every step. Our experiments show improvements in speed and memory required to solve verification problems in comparison to other state-of-the-art heuristics. Finally, in Ch. 7, we conclude the thesis with some closing remarks and pointers for future work.

Part I

Assured Autonomy for Safety-Critical Systems

Chapter 2

Background and Preliminaries

In this chapter we provide a brief introduction to dynamical systems, optimal control and safety analysis. As we shall see, safety problems can be cast as (robust) optimal control problems which can be solved numerically by approximating the solution to a partial differential equation or a variational inequality.

2.1 Optimal Control Overview

Dynamical Systems

A key aspect in guaranteeing the safety of autonomous systems is our ability to predict their behavior over time. This can be accomplished through the mathematical framework of *dynamical systems*. In short, dynamical systems describe the evolution of a set of relevant quantities, or *states*, denoted as $x \in \mathcal{X} \subseteq \mathbb{R}^n$, over time through a differential equation:

$$\dot{x} = f(x, u, d, t), \tag{2.1}$$

where $u \in \mathcal{U} \subseteq \mathbb{R}^{n_u}$ is the *control input* and $d \in \mathcal{D} \subseteq \mathbb{R}^{n_d}$ is the *disturbance* to the system, and where the set $\mathcal{U}$ and $\mathcal{D}$ are compact. The control set $\mathcal{U}$ can be interpreted as the set of actions available to the controller to influence the evolution of the state, whereas the disturbance set $\mathcal{D}$ represents a set of actions available to some external entity which much like the controller can also influence the evolution of the state. For example, as we shall see in Sec. 3.3, if the state x represents the spatial configuration of a quadrotor in space, then the dynamics f describe the continuous time evolution of this configuration, the input u describes the instantaneous thrust applied by each propeller and d represents some external physical disturbance such as wind.

The dynamical system (2.1) describes the continuous-time evolution of the state. When the *initial state* of the system is specified to be x for some *initial time* t, then, under some technical conditions[1],

[1]Trajectories of (2.1) will be uniquely defined provided that the dynamical system f is uniformly continuous and bounded, and Lipschitz in $\mathcal{X}$ for all time $\tau \in [t, \infty)$.

the *control signal* $u(\cdot) : [t, \infty) \to \mathcal{U}$ together with the *disturbance signal* $d(\cdot) : [t, \infty) \to \mathcal{D}$ define the unique *trajectory of the system* for $T \geq t$ as

$$\xi(T; t, x, u(\cdot), d(\cdot)) = \int_t^T f(\xi(\tau; t, x, u(\cdot), d(\cdot)), u(\tau), d(\tau))d\tau + x, \tag{2.2}$$

or in differential form ($\dot{\xi} = d\xi/dT$) as

$$\dot{\xi}(T; t, x, u(\cdot), d(\cdot)) = f(\xi(T; t, x, u(\cdot), d(\cdot)), u(\tau), d(\tau)) \tag{2.3}$$

$$\xi(t; t, x, u(\cdot), d(\cdot)) = x. \tag{2.4}$$

For notational convenience, we will write $\xi_{x,t}^{\mathbf{u},\mathbf{d}}(\tau) = \xi(\tau; t, x, u(\cdot), d(\cdot))$ to denote trajectories when necessary. Finally, while dynamical systems can have an explicit dependence on time, in this work we will consider only time-invariant systems, so we will henceforth drop the dependence on t, meaning that $\dot{x} = f(x, u, d)$.

Feedback Control

The trajectory defined in (2.2) requires specifying the control signal $u(\cdot)$ (as well as the disturbance signal $d(\cdot)$) which determines which controls to apply at every instant in time. When the control signal is specified as an explicit function of time, rather than implicitly through the state of the system, this control scheme is known as *open-loop control*. In other words, the sequence of actions has been stipulated before the system has started to evolve. While intuitive, this form of control is problematic because models can be slightly imprecise and noise is often present, making an accurate prediction of the state impossible. In an ideal world the sequence of open-loop actions would suffice to produce desirable trajectories, in practice, however, errors compound over time and the realized trajectories can be far from optimal. To address this, a different type of control scheme known as *feedback control* uses the current state of the system to inform which actions to take. Unlike open-loop control, feedback allows the system to correct for inaccuracies during the execution of a trajectory based on the instantaneous value of the state. This formulation introduces the concept of a *feedback policy*, a function $\pi : \mathbb{R}^n \times \mathbb{R} \to \mathcal{U}$ which dictates which control u to input into the system at time t through the map $u = \pi(x, t)$.

Optimal Control

In the previous section we introduced the concept of a feedback policy, a function which produces an adaptive control signal based on the instantaneous value of the state. To accomplish this, however, some metric of optimality or 'goodness' must be defined for our problem in order for the controller to provide appropriate corrective actions. Assuming that no disturbance signal $d(\cdot)$ is present, this can be accomplished through the definition of a *cost functional*

$$\mathcal{V}^{\mathbf{u}}(x, t) = \int_t^T c(\xi_{x,t}^{\mathbf{u}}(\tau), u(\tau))d\tau + l(\xi_{x,t}^{\mathbf{u}}(T)), \tag{2.5}$$

a map from the initial conditions (x, t) and control signal $\mathbf{u} = u(\cdot)$, to a scalar value. The function $c : \mathbb{R}^n \times \mathbb{R}^{n_u} \to \mathbb{R}$ is the *running cost* and $l : \mathbb{R}^n \to \mathbb{R}$ is the *terminal cost*. The value of $\mathcal{V}^{\mathbf{u}}(x, t)$ represents a *cost* and, therefore, the goal of the controller will be to produce a control signal that minimizes it. The optimal control problem can thus be written as follows:

$$V(x,t) = \inf_{u(\cdot)\in\mathbb{U}_t^T} \mathcal{V}^{\mathbf{u}}(x,t), \tag{2.6}$$

where $\mathbb{U}_t^T$ represents the collection of measurable functions $u(\cdot) : [t, T] \to \mathcal{U}$, and $V(x, t)$ is known as the *value function*, which provides the value for the initial conditions (x, t).

In order to tackle the minimization problem in (2.6), it is possible to use Bellman's principle of optimality, which states:

> *"An optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision."[9]*

Starting from (2.5) we have that

$$V(x,t) = \inf_{u(\cdot)\in\mathbb{U}_t^T} \int_t^T c(\xi_{x,t}^{\mathbf{u}}(\tau), u(\tau))d\tau + l(\xi_{x,t}^{\mathbf{u}}(T)) \tag{2.7}$$

$$= \inf_{u_1(\cdot)\in\mathbb{U}_t^{t+\delta}} \Big\{ \int_t^{t+\delta} c(\xi_{x,t}^{\mathbf{u}_1}(\tau), u_1(\tau))d\tau \tag{2.8}$$

$$+ \inf_{u_2(\cdot)\in\mathbb{U}_{t+\delta}^T} \int_{t+\delta}^T c(\xi_{x',t'}^{\mathbf{u}_2}(\tau), u_2(\tau))d\tau + l(\xi_{x',t'}^{\mathbf{u}_2}(T)) \Big\} \tag{2.9}$$

$$= \inf_{u_1(\cdot)\in\mathbb{U}_t^{t+\delta}} \Big\{ \int_t^{t+\delta} c(\xi_{x,t}^{\mathbf{u}_1}(\tau), u(\tau))d\tau + V(\xi_{x,t}^{\mathbf{u}_1}(t+\delta), t+\delta) \Big\}, \tag{2.10}$$

for some $\delta \in [0, T - t]$, where $t' = t + \delta$ and $x' = \xi_{x,t}^{\mathbf{u}_1}(t')$. We obtain that the optimal value at (x, t) is the same as optimizing over the interval $[t, t + \delta]$ and adding the contribution of $V(\xi_{x,t}^{\mathbf{u}_1}(t + \delta), t + \delta)$, that is the value of acting optimally thereafter. Assume that V is everywhere differentiable. Multiplying (2.10) by $1/\delta$ and taking the limit $\delta \to 0$, we then obtain the partial differential equation

$$0 = \partial_t V(x,t) + \inf_{u\in\mathcal{U}} \{c(x,u) + \nabla_x V \cdot f(x,u)\} \tag{2.11}$$

$$V(x,T) = l(x), \tag{2.12}$$

where ∂_t represents the partial derivative with respect to time and ∇_x is the gradient with respect to the state. This partial differential equation is known as the Hamilton-Jacobi-Bellman (HJB)

equation. $H(x, p, u) := c(x, u) + p \cdot f(x, u)$ is known as the *Hamiltonian*, and the optimal control is given by

$$u^*(x, t) = \arg\inf_{u \in \mathcal{U}} H(x, \nabla_x V, u). \tag{2.13}$$

While the differentiablity assumptions we have made can be quite strong, these can be relaxed, which leads to the topic of *viscosity solutions*. For a detailed overview on viscosity solutions for the HJB equation we refer the reader to [28].

Robust Optimal Control

In the beginning of this chapter we introduced dynamic systems which depended on the control input u and disturbance d. So far, however, we have only seen how optimal controls can be found by solving the HJB equation when no disturbance signal is present. It is therefore natural to ponder how one would compute optimal controls when a disturbance *is* present, and in particular, when that disturbance acts adversarially. This *worse-case* disturbance framework gives rise to robust optimal control. Given the cost functional

$$\mathcal{V}^{\mathbf{u},\mathbf{d}}(x, t) = \int_t^T c(\xi_{x,t}^{\mathbf{u},\mathbf{d}}(\tau), u(\tau), d(\tau))d\tau + l(\xi_{x,t}^{\mathbf{u},\mathbf{d}}(T)), \tag{2.14}$$

with $\mathbf{d} = d(\cdot)$, the controller seeks to minimize the cost, whereas the disturbance seeks to maximize it. This can be posed as a *zero-sum differential game* as follows:

$$V(x, t) = \inf_{u(\cdot) \in \mathbb{U}_t^T} \sup_{d(\cdot) \in \mathbb{D}_t^T} \mathcal{V}^{\mathbf{u},\mathbf{d}}(x, t). \tag{2.15}$$

Unfortunately, this formulation uses an *open loop information structure*, which implies that the controller must commit to a control signal first, followed by the disturbance picking its own disturbance signal after. This formulation is too stringent in that it gives too much information to the disturbance and doesn't allow the controller to adapt. To change this, it is necessary to define a new strategy information pattern, which in essence establishes what information each player (the controller vs. disturbance) is given at every point in time. *Non-anticipative strategies* are a type of strategy information pattern represented by the set Γ_t^T which contains the collection of maps $\delta : \mathbb{U}_t^T \to \mathbb{D}_t^T$ such that the dependence of $\delta[u]$ on u is causal:

$$\Gamma_t^T := \{\delta : \mathbb{U}_t^T \to \mathbb{D}_t^T \mid u(\tau) = \hat{u}(\tau) \;\; a.e. \;\; \tau \in [t, T] \tag{2.16}$$

$$\Rightarrow \;\; \delta[u](\tau) = \delta[\hat{u}](\tau) \;\; a.e. \;\; \tau \in [t, T]\} \tag{2.17}$$

The predicate in set Γ_t^T enforces that the disturbance always acts in the same way for control signals which are the identical up to some time $\tau \in [t, T]$. This effectively prevents the disturbance from exploiting future information of the controller's signal. Under this strategy information pattern, we can now write in terms of Γ_t^T the following, more suitable zero-sum differential game

$$V(x, t) = \sup_{\delta \in \Gamma_t^T} \inf_{u(\cdot) \in \mathbb{U}_t^T} \mathcal{V}^{\mathbf{u},\delta[\mathbf{u}]}(x, t). \tag{2.18}$$

It may seem that the inversion of the supremum and infimum imply that the controller now has the advantage over the disturbance. However, we note that now the disturbance is picking a causal mapping (not a signal $d(\cdot)$ defined at the onset), which will adaptively change the disturbance signal according to $u(\cdot)$. Moreover, the disturbance still has the advantage of knowing what the instantaneous input of the controller is, whereas the controller does not know what the instantaneous disturbance action will be[2]. Using analogous steps to the ones shown in (2.7), we arrive at the Hamilton-Jacobi-Isaacs (HJI) equation[3]

$$0 = \partial_t V(x,t) + \inf_{u\in\mathcal{U}} \sup_{d\in\mathcal{D}} \{c(x,u,d) + \nabla_x V \cdot f(x,u,d)\} \tag{2.19}$$

$$V(x,T) = l(x), \tag{2.20}$$

where the Hamiltonian is $H(x,p,u,d) := c(x,u,d) + p \cdot f(x,u,d)$ and similarly to (2.13), the optimal control and disturbance are given by

$$u^*(x,t), d^*(x,t) = \arg\inf_{u\in\mathcal{U}} \sup_{d\in\mathcal{D}} H(x, \nabla_x V, u, d). \tag{2.21}$$

Variational Inequalities

Before ending this overview on optimal control, it is important to introduce a particular cost functional which will be relevant in the next chapters. The cost functional we will be concerned about will be

$$\mathcal{V}^{\mathbf{u}}(x,t) = \min_{\tau\in[t,T]} \; l(\xi^{\mathbf{u}}_{x,t}(\tau)), \tag{2.22}$$

with corresponding value function

$$V(x,t) = \inf_{u(\cdot)\in\mathbb{U}_t^T} \mathcal{V}^{\mathbf{u}}(x,t). \tag{2.23}$$

In what follows we show how to use Bellman's principle of optimality to obtain a specific type of equation known as a variational inequality. Starting with

$$V(x,t) = \inf_{u(\cdot)\in\mathbb{U}_t^T} \; \min_{\tau\in[t,T]} \; l(\xi^{\mathbf{u}}_{x,t}(\tau)) \tag{2.24}$$

$$= \inf_{u_1(\cdot)\in\mathbb{U}_t^{t+\delta}} \; \inf_{u_2(\cdot)\in\mathbb{U}_{t+\delta}^T} \; \min\{ \min_{\tau\in[t,t+\delta]} \; l(\xi^{\mathbf{u}}_{x,t}(\tau)), \; \min_{\tau\in[t+\delta,T]} \; l(\xi^{\mathbf{u}}_{x,t}(\tau))\} \tag{2.25}$$

$$= \inf_{u_1(\cdot)\in\mathbb{U}_t^{t+\delta}} \; \min\{ \min_{\tau\in[t,t+\delta]} \; l(\xi^{\mathbf{u_1}}_{x,t}(\tau)), \; \inf_{u_2(\cdot)\in\mathbb{U}_{t+\delta}^T} \; \min_{\tau\in[t+\delta,T]} \; l(\xi^{\mathbf{u_2}}_{x',t'}(\tau))\} \tag{2.26}$$

$$= \inf_{u_1(\cdot)\in\mathbb{U}_t^{t+\delta}} \; \min\{ \min_{\tau\in[t,t+\delta]} \; l(\xi^{\mathbf{u_1}}_{x,t}(\tau)), \; V(\xi^{\mathbf{u_1}}_{x,t}(t+\delta), t+\delta)\} \tag{2.27}$$

[2]This fact reverts the order of the supremum and infimum in the differential equation formulation to the more intuitive $\inf_{u\in\mathcal{U}} \sup_{d\in\mathcal{D}}$ where control is the first player and disturbance the second.

[3]Isaacs worked extensively in the field of differential games. Hence the change of nomenclature from Bellman, whose work was primarily on optimal control.

At this point note that the infimum of the point-wise minimum between two functions is the same as the minimum over infima ($\inf_x \min\{a(x), b(x)\} = \min\{\inf_x a(x), \inf_{x'} b(x')\}$),

$$V(x,t) = \min\{ \inf_{u_1(\cdot)\in\mathbb{U}_t^{t+\delta}} \min_{\tau\in[t,t+\delta]} l(\xi_{x,t}^{\mathbf{u_1}}(\tau)), \inf_{u_1'(\cdot)\in\mathbb{U}_t^{t+\delta}} V(\xi_{x,t}^{\mathbf{u_1'}}(t+\delta), t+\delta)\} \tag{2.28}$$

Note that the above equality holds for *all* $\delta \in [0, t-T]$. If we set $\delta = 0$, then

$$V(x,t) = \min\{l(x), V(x,t)\} \implies V(x,t) \leq l(x) \tag{2.29}$$

This conveys the intuitive notion that the value of every state and any time is always upper-bounded by $l(x)$. Assuming that V is everywhere differentiable, if we multiply (2.28) by $1/\delta$ ($\delta > 0$) and take the limit $\delta \to 0$, we have that

$$0 = \min\{\lim_{\delta\to 0} \frac{l(x) - V(x,t)}{\delta},\ \partial_t V(x,t) + \inf_{u\in\mathcal{U}} \Big(\nabla_x V \cdot f(x,u)\Big)\} \tag{2.30}$$

From (2.29) the first term in the minimum either evaluates to infinity (if $l(x) - V(x) > 0$) or to zero (if $l(x) - V(x) = 0$). Noting the property that ($0 = \min\{0, a(x)\} \implies a(x) \geq 0$), we have that one of the following conditions must be met in order for (2.30) to hold[4]:

$$l(x) - V(x,t) > 0 \quad \wedge \quad \partial_t V(x,t) + \inf_{u\in\mathcal{U}} \Big(\nabla_x V \cdot f(x,u)\Big) = 0 \tag{2.31}$$

$$l(x) - V(x,t) = 0 \quad \wedge \quad \partial_t V(x,t) + \inf_{u\in\mathcal{U}} \Big(\nabla_x V \cdot f(x,u)\Big) \geq 0. \tag{2.32}$$

The last line can be broken further into two conditions given the following logical equivalence ($a = 0 \wedge b \geq 0 \iff [a = 0 \wedge b > 0] \oplus [a = 0 \wedge b = 0]$), so now *one of three conditions* must be met:

$$l(x) - V(x,t) > 0 \quad \wedge \quad \partial_t V(x,t) + \inf_{u\in\mathcal{U}} \Big(\nabla_x V \cdot f(x,u)\Big) = 0 \tag{2.33}$$

$$l(x) - V(x,t) = 0 \quad \wedge \quad \partial_t V(x,t) + \inf_{u\in\mathcal{U}} \Big(\nabla_x V \cdot f(x,u)\Big) > 0 \tag{2.34}$$

$$l(x) - V(x,t) = 0 \quad \wedge \quad \partial_t V(x,t) + \inf_{u\in\mathcal{U}} \Big(\nabla_x V \cdot f(x,u)\Big) = 0. \tag{2.35}$$

Given the following equivalence,

$$(a > 0 \wedge b = 0) \oplus (a = 0 \wedge b > 0) \oplus (a = 0 \wedge b = 0) \iff 0 = \min\{a, b\} \tag{2.36}$$

we finally arrive to what is known as the Hamilton-Jacobi-Bellman variational inequality (HJB-VI):

$$0 = \min\{\ l(x) - V(x,t)\ ,\ \partial_t V(x,t) + \inf_{u\in\mathcal{U}} \Big(\nabla_x V \cdot f(x,u)\Big)\ \} \tag{2.37}$$

$$V(x,T) = l(x). \tag{2.38}$$

[4]Given two statements A (2.31) and B (2.32), this is equivalent to requiring $A \oplus B \equiv 1$, where $\oplus$ denotes logical exclusive OR (XOR).

Note that the conditions in (2.31) and (2.32) are automatically encoded in the HJB-VI. Similarly, following the same steps used to obtain the HJB-VI, we can equivalently obtain the Hamilton-Jacobi-Isaacs variational inequality (HJI-VI):

$$0 = \min\{\ l(x) - V(x,t)\ ,\ \partial_t V(x,t) + \inf_{u\in\mathcal{U}} \sup_{d\in\mathcal{D}} \Big(\nabla_x V \cdot f(x,u,d)\Big)\ \} \tag{2.39}$$

$$V(x,T) = l(x). \tag{2.40}$$

2.2 Safety Analysis

Set Definitions

In this first half of the dissertation we are concerned with ensuring safety for safety-critical systems. One must first, however, concretize what these safety properties are mathematically. Given that our dynamical models determine the evolution of the state of the system, our safety specifications will be in the form of *sets* of states. To that end, let us define the *target set* $\mathcal{T} \subseteq \mathbb{R}^n$ to be a generic collection of states, and the *backward-reachable set* (BRS) $\mathcal{R}(\mathcal{T})$ as follows:

Definition 1. *(Backward-reachable set). The backward-reachable set of an autonomous dynamical system $\dot{x} = f(x)$ is the collection of initial states $x \in \mathbb{R}^n$ such that trajectories eventually enter the set $\mathcal{T}$.*

$$\mathcal{R}(\mathcal{T}) := \{x \in \mathbb{R}^n \mid \tau \in [t,T],\ \xi_{x,t}(\tau) \in \mathcal{T}\} \tag{2.41}$$

The above definition of the backward-reachable set assumes that the dynamics are only a function of the state. In the presence of an input signal (and no disturbance signal), the definition of backward-reachable set must be expanded to include the additional definitions of a *maximal backward-reachable set* $\overline{\mathcal{R}}(\mathcal{T})$ and *minimal backward-reachable set* $\underline{\mathcal{R}}(\mathcal{T})$:

Definition 2. *(Maximal backward-reachable set). The maximal backward-reachable set of a dynamical system $\dot{x} = f(x,u)$ is the collection of initial states $x \in \mathbb{R}^n$ such that there exists some control signal $u(\cdot) \in \mathbb{U}$ which is capable of steering the trajectory into the set $\mathcal{T}$.*

$$\overline{\mathcal{R}}(\mathcal{T}) := \{x \in \mathbb{R}^n \mid \exists u(\cdot) \in \mathbb{U},\ \tau \in [t,T],\ \xi^{\mathbf{u}}_{x,t}(\tau) \in \mathcal{T}\} \tag{2.42}$$

Definition 3. *(Minimal backward-reachable set). The minimal backward-reachable set of a dynamical system $\dot{x} = f(x,u)$ is the collection of initial states $x \in \mathbb{R}^n$ such that for every control signal $u(\cdot) \in \mathbb{U}$ trajectories can't stay clear of the set $\mathcal{T}$.*

$$\underline{\mathcal{R}}(\mathcal{T}) := \{x \in \mathbb{R}^n \mid \forall u(\cdot) \in \mathbb{U},\ \tau \in [t,T],\ \xi^{\mathbf{u}}_{x,t}(\tau) \in \mathcal{T}\} \tag{2.43}$$

The maximal backward-reachable set (2.42) is of particular interest for *liveness* problems, where we want to guarantee whether for some states there exist controls which steer trajectories into $\mathcal{T}$. In

contrast, the minimal backward-reachable set (2.43) is important for *safety* because it defines the set of states for which no control action exists which avoids $\mathcal{T}$.

Finally, when both a control signal and a disturbance signal are present, the maximal and minimal naming convention changes to *enforceable* and *inevitable* [28] respectively, in order to highlight the fact that when two control inputs are act against each other the geometrical interpretation of maximal vs. minimal becomes ambiguous.

Definition 4. *(Enforceable backward-reachable set). The enforceable backward-reachable set of a dynamical system $\dot{x} = f(x, u, d)$ is the collection of initial states $x \in \mathbb{R}^n$ such that for every non-anticipative disturbance strategy $\delta : \mathbb{U} \to \mathbb{D}$, there exists some control signal $u(\cdot) \in \mathbb{U}$ which is capable of steering the trajectory into the set $\mathcal{T}$.*

$$\overline{\mathcal{R}}(\mathcal{T}) := \{x \in \mathbb{R}^n \mid \forall \delta \in \Gamma,\ \exists u(\cdot) \in \mathbb{U},\ \tau \in [t, T],\ \xi_{x,t}^{\mathbf{u},\mathbf{d}}(\tau) \in \mathcal{T}\} \tag{2.44}$$

Definition 5. *(Inevitable backward-reachable set). The inevitable backward-reachable set of a dynamical system $\dot{x} = f(x, u, d)$ is the collection of initial states $x \in \mathbb{R}^n$ such that there exists some non-anticipative disturbance strategy $\delta : \mathbb{U} \to \mathbb{D}$, such that for every control signal $u(\cdot) \in \mathbb{U}$ trajectories can't stay clear of the set $\mathcal{T}$.*

$$\underline{\mathcal{R}}(\mathcal{T}) := \{x \in \mathbb{R}^n \mid \exists \delta \in \Gamma,\ \forall u(\cdot) \in \mathbb{U},\ \tau \in [t, T],\ \xi_{x,t}^{\mathbf{u},\mathbf{d}}(\tau) \in \mathcal{T}\} \tag{2.45}$$

Similar to previous definitions, enforceable backward-reachable sets (2.44) are useful for robust liveness problems. In terms of safety, inevitable backward-reachable sets (2.45) provide the collection of states for which some disturbance signal will steer trajectories into $\mathcal{T}$ despite the controller's best effort. If $\mathcal{T}$ is an undesirable set of states for example, $\underline{\mathcal{R}}(\mathcal{T})$ provides the states for which our system is doomed to fail.

In line with the safety interpretation of inevitable and minimal backward-reachable sets, we finalize this section by concertizing the two sets which will be relevant for safety analysis. These sets will be the *failure set* $\mathcal{F}$ and the *constraint set* $\mathcal{K}$. The set $\mathcal{F}$ represents the collection of configurations deemed unacceptable for the system. On the other hand, the constraint set $\mathcal{K}$ represents the collection of acceptable configurations. Hence, the constraint set and failure set are by definition complements of each other $\mathcal{F} = \mathcal{K}^c$.

It is important to point out that states inside the constraint set $\mathcal{K}$ are not necessarily safe. Rather, the *set of safe states* will be $\Omega = \mathcal{K} \cap \underline{\mathcal{R}}(\mathcal{F})^c$, whereas the set of unsafe states will be the complement $\Omega^c = \mathcal{F} \cup \underline{\mathcal{R}}(\mathcal{F})$.

Set Definitions as (Robust) Optimal Control Problems

While the sets introduced thus far are conceptually useful, it is necessary to find a way of actually computing them. This can be accomplished by defining an *implicit surface function*, namely a Lipschitz function of the form $l(\cdot) : \mathbb{R}^n \to \mathbb{R}$ mapping states to a scalar value. This scalar value is used to delineate sets. For instance, let l be such that

$$x \in \mathcal{T} \iff l(x) \leq 0 \tag{2.46}$$

then, the following cost functionals $\mathcal{V}$ can be defined for our trajectories:

$$\mathcal{V}^{\mathbf{u}}(x) := \min_{\tau\in[t,T]} l(\xi^{\mathbf{u}}_{x,t}(\tau)) \qquad \mathcal{V}^{\mathbf{u},\mathbf{d}}(x) := \min_{\tau\in[t,T]} l(\xi^{\mathbf{u},\mathbf{d}}_{x,t}(\tau)). \tag{2.47}$$

The predicates from definitions (2.42)-(2.45) can now be re-written in terms of (robust) optimal control problems

(*Maximal BRS*) $$\exists u(\cdot),\ \tau\in[t,T],\ \xi^{\mathbf{u}}_{x,t}(\tau)\in\mathcal{T} \iff \inf_{u(\cdot)}\ \mathcal{V}^{\mathbf{u}}(x)\le 0 \tag{2.48}$$

(*Minimal BRS*) $$\forall u(\cdot),\ \tau\in[t,T],\ \xi^{\mathbf{u}}_{x,t}(\tau)\in\mathcal{T} \iff \sup_{u(\cdot)}\ \mathcal{V}^{\mathbf{u}}(x)\le 0 \tag{2.49}$$

(*Enforceable BRS*) $$\forall\delta,\ \exists u(\cdot),\ \tau\in[t,T],\ \xi^{\mathbf{u},\mathbf{d}}_{x,t}(\tau)\in\mathcal{T} \iff \sup_{\delta}\inf_{u(\cdot)}\ \mathcal{V}^{\mathbf{u},\mathbf{d}}(x)\le 0 \tag{2.50}$$

(*Inevitable BRS*) $$\exists\delta,\ \forall u(\cdot),\ \tau\in[t,T],\ \xi^{\mathbf{u},\mathbf{d}}_{x,t}(\tau)\in\mathcal{T} \iff \inf_{\delta}\sup_{u(\cdot)}\ \mathcal{V}^{\mathbf{u},\mathbf{d}}(x)\le 0. \tag{2.51}$$

Recalling the definition of the value (2.6) and (2.15), the right-hand sides of (2.57)-(2.58) can be written in terms of the value function $V(x,t)$. More concretely, once $V(x,t)$ is obtained by solving the HJB-VI or HJI-VI (2.37) with boundary condition $V(x,T) = l(x)$, the following property holds:

$$\tau\in[t,T],\ \xi_{x,t}(\tau)\in\mathcal{T} \iff V(x,t)\le 0. \tag{2.52}$$

Accordingly, for safety analysis, if we define another implicit surface function $g(x)$ such that $x\in\mathcal{F} \iff g(x)\le 0$ and $V(x,T) = g(x)$, then following relation also holds:

$$x\in\Omega^c \text{ (unsafe states)} \iff V(x,t)\le 0. \tag{2.53}$$

That is, the state x at time $t\le T$ is unsafe if it belongs to the zero sublevel set of $V(x,t)$.

2.3 Safety and Liveness

Thus far we have been mainly concerned about safety. However, safety alone is not particularly interesting unless it is also coupled to an underlying task. For example, guaranteeing that a quadrotor can reach a room from some starting location within a building while remaining safe is a more useful problem than guaranteeing safety alone.

Problems that involve reaching a target set while avoiding obstacles are known as *reach-avoid* problems, and the aim is to compute the *reach-avoid set* (RAS) $\mathcal{RA}(\mathcal{T};\mathcal{K})$, which is defined as follows:

Definition 6. *(Enforceable reach-avoid set). The enforceable reach-avoid set of a dynamical system $\dot{x} = f(x,u,d)$ is the collection of initial states $x\in\mathbb{R}^n$ such that for all non-anticipative disturbance strategies $\delta:\mathbb{U}\to\mathbb{D}$, there exists some control signal $u(\cdot)\in\mathbb{U}$ that is able to drive trajectories into $\mathcal{T}$ while remaining in $\mathcal{K}$ at all preceeding times.*

$$\mathcal{RA}(\mathcal{T};\mathcal{K}) := \{x\in\mathbb{R}^n \mid \forall\delta\in\Gamma,\ \exists u(\cdot)\in\mathbb{U},\ \tau\in[t,T],\ \xi^{\mathbf{u},\mathbf{d}}_{x,t}(\tau)\in\mathcal{T}\wedge\forall s\in[t,\tau]\ \xi^{\mathbf{u},\mathbf{d}}_{x,t}(s)\in\mathcal{K}\}. \tag{2.54}$$

The predicate in (2.54) ensures that states which are part of the reach-avoid set are those initial states such that for all disturbances there exists some control sequence which is able to drive the subsequent trajectory into $\mathcal{T}$ while remaining within the constraint set $\mathcal{K}$. Note that this last requirement of remaining within $\mathcal{K}$ is only enforced before reaching $\mathcal{T}$, not thereafter. Also note that we have omitted the definition of the *maximal reach-avoid set* for simplicity given that it is practically equivalent to (2.54) but without a disturbance.

As before, the reach-avoid problem can be formulated as a (robust) optimal control problem by defining two implicit surface functions, one for the target set and one for the failure set

$$x \in \mathcal{T} \iff l(x) \leq 0 \qquad\qquad x \in \mathcal{F} \iff g(x) > 0. \tag{2.55}$$

In this case, the cost functional is given by

$$\mathcal{V}^{\mathbf{u}}_{\mathcal{RA}}(x) := \min_{\tau \in [t,T]} \ \max\{\, l(\xi^{\mathbf{u}}_{x,t}(\tau)) \,,\ \max_{s \in [t,\tau]} g(\xi^{\mathbf{u}}_{x,t}(s)) \,\}. \tag{2.56}$$

The maximum in the above expression acts as an "overwriting" mechanism. If a trajectory always remains inside $\mathcal{K}$, the second maximization term is always negative and $\mathcal{V}^{\mathbf{u}}(x)$ is negative only if the trajectory ever enters into $\mathcal{T}$. On the other hand, if at some point the constraints are violated the maximum will be positive, which ensures that trajectories that eventually reach $\mathcal{T}$ while also violating constraints yield a positive cost.

With this cost functional defined, we can now also define the maximal and enforceable reach-avoid sets $\mathcal{RA}(\mathcal{T}; \mathcal{K})$ in terms of the value function of the optimal (robust) control problem,

(*Maximal RAS*) $$\exists u(\cdot),\ \tau \in [t,T],\ \xi^{\mathbf{u}}_{x,t}(\tau) \in \mathcal{T} \iff \inf_{u(\cdot)}\ \mathcal{V}^{\mathbf{u}}_{\mathcal{RA}}(x) \leq 0 \tag{2.57}$$

(*Enforceable RAS*) $$\forall \delta,\ \exists u(\cdot),\ \tau \in [t,T],\ \xi^{\mathbf{u},\mathbf{d}}_{x,t}(\tau) \in \mathcal{T} \iff \sup_{\delta} \inf_{u(\cdot)}\ \mathcal{V}^{\mathbf{u},\mathbf{d}}_{\mathcal{RA}}(x) \leq 0. \tag{2.58}$$

Again, similar to (2.52), we can solve for the value function $V(x,t)$ using slight variations[5] of the the HJB-VI or HJI-VI from (2.37) with boundary condition $V(x,T) = \max\{l(x), g(x)\}$. In this case, once $V(x,t)$ has been computed the following property holds:

$$\tau \in [t,T],\ \xi_{x,t}(\tau) \in \mathcal{T}\ \wedge\ \forall s \in [t,\tau]\ \xi_{x,t}(s) \in \mathcal{K} \iff V(x,t) \leq 0. \tag{2.59}$$

That is, the state x at time $t \leq T$ can reach $\mathcal{T}$ while remaining inside $\mathcal{K}$ if it belongs to the zero sublevel set of $V(x,t)$.

2.4 Computational Techniques and Limitations

So far we have seen that the safety problem can be cast as a (robust) optimal control problem, whose solution $V(x,t)$ can be obtained by solving the associated the HJB(-VI) or HJI(-VI) equation. The goal of this chapter will be to provide a high-level overview of how these solutions can be computed numerically and the limitations that lie therein.

[5]Obtaining the variational inequality for the cost functional $\mathcal{V}^{\mathbf{u}}_{\mathcal{RA}}(x)$ follows a similar process as (2.23), which we omit for brevity.

Finite Difference Methods

Finite difference methods, also known as grid-based methods, are a type of approach for numerically approximating the solutions of partial differential equations (PDEs). The main idea is to discretize the space of continuous variables on a grid and use the discretized version of the partial differential equation to update every grid point. For illustrative purposes, assuming the $x \in \mathbb{R}$ so that the state is one-dimensional, attempting to approximate the solution of (2.11) can be done by re-writing the PDE in discrete-time and using the discretization as an update rule:

$$V(x, t-\Delta t) \leftarrow V(x,t) + \Delta t \inf_{u \in \mathcal{U}} \{c(x,u) + \frac{V(t, x+\Delta x/2) - V(t, x-\Delta x/2)}{\Delta x} \cdot f(x,u)\} \quad (2.60)$$

with $V(x,T) = l(x)$, where Δt and Δx represent the size of the discretization in the time and state variables respectively. Once this update rule has been established, one can cycle through each point in the grid and update it accordingly. In practice, an approach like the one shown here is prone to instabilities leading to erroneous approximations, given that there are many factors which influence how well finite difference methods will converge to the correct solution. Fortunately, for Hamilton-Jacobi type equations extensive work has been done [63, 71, 72] and tools have been developed which are able to effectively compute accurate approximations. Figure 2.1 shows the typical progression of the value function when numerically computing reachable sets for a reach-avoid problems using [63].

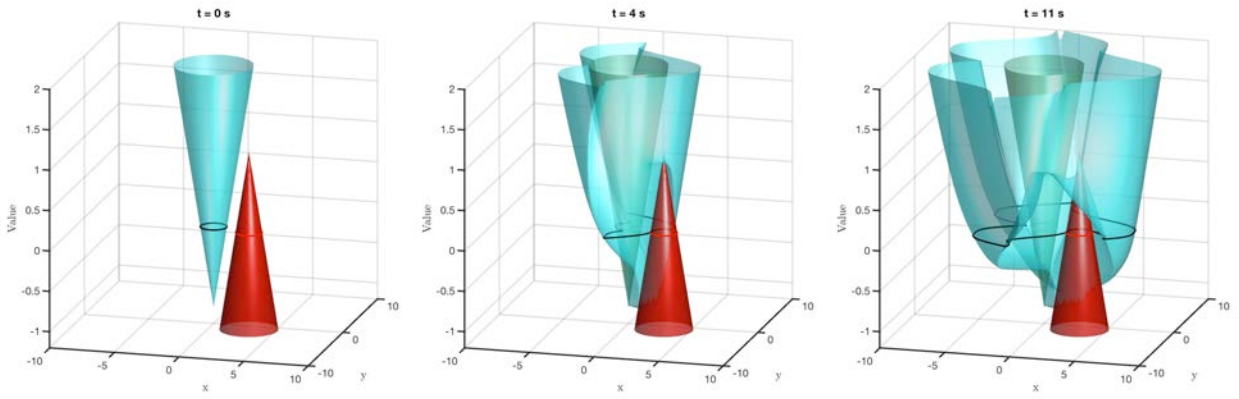


Figure 2.1: Visual progression of the value function when solving Hamilton-Jacobi equations on a grid (figure courtesy of Syliva Herbert).

Curse of Dimensionality

One of the downsides of using finite difference methods for optimal control problems is that the discretization happens in both the time and state variables. While time is one dimensional, the state variables are often multi-dimensional. In essence, this makes the time and space complexity of

solving optimal control problems on a grid exponential. In practical terms, this means that problems with more that 5 state variables are considered intractable. This computational limitation due to exponential growth is known as the *curse of dimensionality*.

An interesting aspect of the curse of dimensionality is that it is pervasive within many engineering problems and is not unique to finite difference methods for optimal control. In fact, other reachability approaches which do not employ optimal control to compute reachable sets also suffer from similar problems. For instance, methods that use ellipsoids or polytopes [48, 49, 59, 45] to (over-)approximate reachable sets suffer from the curse of dimensionality through the dependency of volume and state dimension[6]. However, when the target sets are convex or the dynamics are linear these effects only start to become noticeable at slightly higher dimensions than in finite difference case. In general, a trade-off exists between the accuracy of the approximation of the reachable sets and scalability. As we will see, this fact will reappear again in the second half of this dissertation.

2.5 Chapter Summary

In this chapter we first presented an overview of dynamical systems and optimal control. We showed that the solution of (robust) optimal control problems can be equivalently expressed as the solution to the HJB/I equation or variational inequality. Secondly, we introduced the topic of safety and how safety specifications can be expressed in terms of sets of states, and, in particular, in terms of the constraint set $\mathcal{K} = \mathcal{F}^c$. Once these safety specifications have been laid out, we can then define a variety of reachable sets $\mathcal{R}$ which can be obtained by solving (robust) optimal control problems. By solving the associated HJB/I equation or variational inequality, we obtain the value function $V(x,t)$ whose zero sublevel set is the reachable set for the problem. Finally, we introduce the concept of the curse of dimensionality and why grid-based methods cannot scale.

[6]Consider the extreme case of fully enclosing an n-dimensional sphere inside the smallest possible n-dimensional cube. As n grows the ratio between the volumes of the cube and the sphere goes to zero.

Chapter 3

High Dimensional Reachability Analysis

This chapter is based on the paper "A Classification-based Approach for Approximate Reachability" [79], written in collaboration with David Fridovich-Keil, Sylvia Herbert, and Claire Tomlin.

As we have seen in Ch. 2, Hamilton-Jacobi reachability analysis solves an important class of optimal control problems and differential games. These tools are typically used *offline* to perform theoretical safety analysis and provide goal satisfaction guarantees for nonlinear systems. Applications include collision avoidance [62, 18], vehicle platooning [19], administering anesthesia [44], and others [8, 39, 24]. We can characterize any reachability method (including Hamilton-Jacobi reachability) according to the following criteria: (a) generality of system dynamics, (b) computation of control and/or disturbance policies, (c) flexibility in representation of sets, and (d) computational scalability. The traditional grid-based Hamilton-Jacobi reachability methods presented in Sec. 2.4 perform well for the first three criteria, but suffer from poor computational scalability due to the curse of dimensionality. Recent work has investigated decomposing high-dimensional systems for reachability [15, 16]; nevertheless, grid-based Hamilton-Jacobi reachability is often intractable for analyzing coupled high-dimensional and/or multi-agent systems.

Other reachability methods are more scalable but require linear or affine system dynamics. Such methods may require representing sets using approximative shapes (e.g. polytopes, hyperplanes) [32, 35, 48, 49, 59], or not account for control and disturbance inputs [69]. More complex dynamics can be handled by the methods in [5, 20, 26, 32, 60], but may be less scalable or unable to represent complex set geometries.

Traditional Hamilton-Jacobi reachability methods represent the value function directly over a grid, which implicitly specifies the reachable (or avoid, reach-avoid) set, the optimal controller, and if needed, the optimal disturbance. By contrast, in this work, we will compute an approximation of the optimal controller and disturbance directly. Equipped with these approximations, we can compute estimates of the value function and the reachable sets by simulating the known system dynamics with the learned control and disturbance policies. If a set representation is also required (e.g. for visualization), a grid may be populated using simulated data.

Neural Networks Applied to Control Systems

Feedforward neural networks are a type of parametric function approximator constructed as a composition of nonlinear functions. In Ch. 5 we provide an in-depth introduction to feedforward neural networks. Recently, these types of function approximators have become popular for high-dimensional control tasks. In deep reinforcement learning, for example, neural networks have been employed to learn controllers for complex robotic manipulation tasks, e.g. unscrewing a bottle cap and inserting a peg in a slot [53, 42, 68, 31]. The control theory literature also includes examples in which neural networks have been successfully employed to find approximate solutions optimal control problems or to learn dynamical system models [6, 21, 76, 98, 23].

Neural networks have also been used for approximate reachability analysis [41, 25]. Though conceptually related to these approaches, our method differs in that it exploits the structure of control-affine systems, which we will define shortly, to cast the optimal control problem into a (repeated) classification problem. These neural net classifiers can then be used, under some conditions, for verification—i.e. they can be used to provide safety and/or goal satisfaction *guarantees*.

In this section we will introduce a class of dynamical systems whose optimal controllers display so called "bang-bang" behavior. That is, for any given state, when the constraints on the controls are bounded, closed intervals, and each component is independent of each other, the optimal control will in general belong to one of the extremal values of the set[1] (i.e. one of the vertices of the hyperbox representing the set of controls). This property hints at the possibility of *casting the reachability problem as a classification problem.*

In this section we introduce our classification-based method for approximating the optimal control of Hamilton-Jacobi reachability when the dynamics are control-affine. Even though we will use feedforward neural networks to build the classifiers, it is possible to use other methods (e.g. SVM, decision trees). Ultimately, the choice of the classifier determines how conservative the results of the procedure will be. We leave a full investigation of classifier performance for future work.

3.1 Control-Affine Systems

A control/disturbance-affine system is a special type of dynamical system of the form

$$\dot{x} = \alpha(x) + \sum_{i=1}^{N_u} \beta_i(x) u_i + \sum_{j=1}^{N_d} \gamma_j(x) d_j \ , \tag{3.1}$$

where $\alpha, \beta_i, \gamma_j : \mathbb{R}^n \to \mathbb{R}^n$. We will assume that both control and disturbance are bounded by interval constraints along each dimension, i.e. $u_i \in [u^i_{min}, u^i_{max}]$ for $i = 1, \ldots, N_u$, and $d_j \in [d^j_{min}, d^j_{max}]$ for $j = 1, \ldots, N_d$. Observe that when dynamics f are of the form (3.1), the objective in (2.21) is affine in the instantaneous control u and disturbance d at every time t. The optimal solution, therefore, lies at one of the 2^{N_u} (or 2^{N_d}) corners of the hyperbox containing

[1] Assuming that the gradient of the value function exists and is non-zero.

u (or d). That is, the optimal control and disturbance policies are "bang-bang"[2] (we refer the reader to chapter 4 of [55]). Furthermore, the optimal values for any u_i or d_j at a certain state and time are mutually independent; therefore, for control/disturbance-affine systems, we can frame the Hamilton-Jacobi reachability problem (2.18) as a series of $N_u + N_d$ *binary* classification problems at each time.

3.2 Classification-based Reachability

Algorithm 1 describes the process of learning these classifiers in detail. We begin by discretizing the time-horizon T into small (evenly spaced) intervals of size $\Delta t > 0$ in line 3, and proceed to use the dynamic programming principle backwards in time to build a sequence of approximately optimal control and disturbance policies. In total, the number of classifiers will be $\frac{T}{\Delta t}(N_u + N_d)$.

Algorithm 1: Learning policies and disturbances

1 **Input:** $\dot{x} = f(x, u, d), \mathcal{X}, \mathcal{U}, \mathcal{D}, T, \Delta t, C, N, Train(\cdot, \cdot)$
2 **Initialize** $\Pi_u, \Pi_d \leftarrow \{\}$
3 **For** $k = 0, \ldots, \lfloor T/\Delta t \rfloor$
4 **Initialize** $P, U^*, D^* \leftarrow \{\}$ ▷Initialize empty arrays for training data.
5 **For** $q = 1, \ldots, N$
6 **Sample** $x \sim \text{Unif}\{\mathcal{X}\}$ ▷ Sample states unifromly at random.
7 **Initialize** $u^*, d^* \leftarrow u_{min}, d_{min}$
8 $\hat{x} \leftarrow \xi(-k\Delta t; x, -(k+1)\Delta t, u_{min}, d_{min})$
9 $\hat{c} \leftarrow C(\hat{x}, \Pi_u, \Pi_d)$ ▷ Obtain value for baseline trajectory.
10 **For** $i = 1, \ldots, N_u$ ▷ Find best control.
11 $\hat{u} \leftarrow u_{min}$; $\hat{u}^i \leftarrow u^i_{max}$
12 $x' \leftarrow \xi(-k\Delta t; x, -(k+1)\Delta t, \hat{u}, d_{min})$
13 **If** $(C(x', \Pi_u, \Pi_d) < \hat{c})$: $u^*_i \leftarrow u^i_{max}$ ▷ Rollout remainder of trajectory.
14 **For** $j = 1, \ldots, N_d$ ▷Find best disturbance.
15 $\hat{d} \leftarrow d_{min}$; $\hat{d}^i \leftarrow d^i_{max}$
16 $x' \leftarrow \xi(-k\Delta t; x, -(k+1)\Delta t, u_{min}, \hat{d})$
17 **If** $(C(x', \Pi_u, \Pi_d) > \hat{c})$: $d^*_i \leftarrow d^i_{max}$ ▷ Rollout remainder of trajectory.
18 $U^* \leftarrow \{U^*, u^*\}$ ▷Append control.
19 $D^* \leftarrow \{D^*, d^*\}$ ▷Append disturbance.
20 $P \leftarrow \{P, x\}$ ▷Append state.
21 $\Pi^u_{-(k+1)\Delta t} \leftarrow Train(P, U^*), \Pi_u \leftarrow \{\Pi_u, \Pi^u_{-(k+1)\Delta t}\}$ ▷Train classifier and store it.
22 $\Pi^d_{-(k+1)\Delta t} \leftarrow Train(P, D^*), \Pi_d \leftarrow \{\Pi_d, \Pi^d_{-(k+1)\Delta t}\}$ ▷Train classifier and store it.
23 **Return** Π_u, Π_d

[2]For many physical systems, it is preferable to apply a smooth control signal. We note that the bang-bang control resulting from (2.21) need only be applied at the boundary of the reach-avoid set.

The inputs to the algorithm are the dynamical system f, the set of states $\mathcal{X} \subset \mathbb{R}^n$, set of controls $\mathcal{U} := [u_{min}, u_{max}]^{N_u}$, set of disturbances $\mathcal{D} := [d_{min}, d_{max}]^{N_u}$, time horizon T, time discretization Δt, value evaluation function C and classification procedure $Train$.

At an intermediate time $t < 0$, we will have already obtained the binary classifiers for the control and disturbance policies from $t + \Delta t$ to 0: $\Pi^u_{(t+\Delta t):0}$ and $\Pi^d_{(t+\Delta t):0}$. Here, Π^u_τ and Π^d_τ each denote a *set of classifiers* for the discrete time step τ (i.e. $|\Pi^u_\tau| = N_u$ and $|\Pi^d_\tau| = N_d$). We now define the function C, which computes the cost (2.18) if control and disturbance acted according to these pre-trained policies:

$$C(x, \Pi^u_{(t+\Delta t):0}, \Pi^d_{(t+\Delta t):0}) := \mathcal{V}^{\mathbf{u},\mathbf{d}}(x, t), \tag{3.2}$$

where, due to our discretization, the control signal $\mathbf{u}$ and disturbance signal $\mathbf{d}$ are piecewise constant over time, i.e. $u(x, t) = \Pi^u_\tau(x)$ and $d(x, t) = \Pi^d_\tau(x)$ for $t \in [\tau, \tau + \Delta t)$, for all discrete time steps τ.

At time t, we can determine for some arbitrary state x the optimal control and disturbance as follows. First, compute the cost of applying $u_{min} = (u^0_{min}, \ldots, u^{N_u}_{min})$ and $d_{min} = (d^0_{min}, \ldots, d^{N_d}_{min})$ from t to $t + \Delta t$; that is, let $\hat{c} = C\big(\xi(t + \Delta t; x, t, u_{min}, d_{min}), \Pi^u_{(t+\Delta t):0}, \Pi^d_{(t+\Delta t):0}\big)$. Now, separately for each component i of the control vector u (and likewise for d), set $u^i(t) = u_{max}$ and compute the cost. If the cost is less than (greater than, for disturbance) $\hat{c}$, then this is the optimal control (disturbance) in dimension i at time t. This corresponds to lines 7-17.

Equipped with this procedure for computing *approximately optimal*[3] control and disturbance actions, we record the computed state-action pairs (lines 18-20) for N states sampled uniformly over $\mathcal{X}$[4] (lines 5-6). We then train separate binary classifiers for *each component* of u and d, and add them to their current set Π^u_τ or Π^d_τ. These are finally appended to the time-indexed control and disturbance policy sets $\Pi_u (= \{\{\Pi^{u_1}_0, ..., \Pi^{u_{N_u}}_0\}, ..., \{\Pi^{u_0}_\tau, ..., \Pi^{u_{N_u}}_\tau\}\})$ and Π_d (lines 21-22). $Train(\cdot, \cdot)$ denotes a training procedure given state-action pairs. The end of this chapter contains further details pertaining to how the classifiers were trained.

Two of the main benefits of performing approximate reachability analysis using binary classifiers rather than grids are memory usage and time complexity. The memory footprint of medium-sized neural networks of the sort used in this work can be on the order of 10^3 parameters or ~ 10 Kb, as opposed to ~ 10 Gb for dense grids of 4D systems. In our experience, Algorithm 1 typically terminates after an hour for the 6D and 7D systems presented in the end of Sec. 3.3, whereas grid-based methods are completely intractable for coupled systems of that size.

Special Case: Value Function Convergence

For some instances of problem (2.18) the value function $V(x, t)$ converges: $\lim_{t\to-\infty} V(x, t) = V^*(x)$. From (2.21), the corresponding optimal control and disturbance policies also converge.

[3] Approximately optimal, since we compute policies at time t based on previously trained control and disturbance policies for $\tau > t$.

[4] While other distributions could be used, in this work we focus solely on uniform sampling. Different sampling strategies may result in different algorithm performance.

While in this work we make no claims regarding convergence of the classifiers to the true optimal policies, our empirical results do suggest convergence in practice (see Fig. 3.6). When this happens, we denote $\Pi^u_{-T} = \Pi^u_{-\infty}$ (resp. $\Pi^d_{-T} = \Pi^d_{-\infty}$), for T large enough. In practice, the horizon can be progressively increased as needed. A benefit of converged policies is that when estimating $V^*(x)$ we only require the last set of binary classifiers $\Pi^u_{-\infty}$ and $\Pi^d_{-\infty}$, allowing us to store only $N_u + N_d$ classifiers.

Summary of Guarantees

Algorithm 1 returns a set of approximately optimal policies for the control and the disturbance for a finite number of time steps. Recalling (3.2), in order to obtain an estimate of the value at a certain state x and time t, it suffices to simulate an entire trajectory from that state and time using the learned policies. The value $V^{\Pi_u,\Pi_d}(x,t)$ is the cost of the associated trajectory, measured according to (2.56).

A benefit of working with policy approximators rather than value function approximators is that in the case of no disturbance, the value function induced by the learned control policy will always upper-bound the true value. This means that a reach-avoid set computed via Algorithm 1 will be a subset of the true reach-avoid set.

For reachability problems involving a disturbance, if the optimal disturbance policy is known *a priori*, the same guarantee still applies. However, if the optimal disturbance is unknown and must also be learned, no guarantees can be made because the learned disturbance policy will not generally be optimal. We formalize this result with the following proposition.

Proposition 1. *If we assume (a) no disturbance, or (b) access to a worst-case optimal disturbance policy, then the computed reach-avoid set is a subset of the true set.*

Proof. First assume no disturbance. Due to the use of function approximators, the control policy Π_u will be suboptimal relative to the optimal controller $u^*(\cdot)$, meaning it is less effective at minimizing the cost functional (2.56). Therefore, $V^{\Pi_u}(x,t) \geq V(x,t)$. Denoting the neural network reach-avoid sets as $\mathcal{RA}^{\Pi_u}_t := \{x : V^{\Pi_u}(x,t) \leq 0\}$, this inequality implies that $\mathcal{RA}^{\Pi_u}_t \subseteq \mathcal{RA}_t$. □

Note that this applies to *all* states x and times t, not just those that were sampled in Algorithm 1. When optimizing over both control and disturbance this guarantee does not hold because the disturbance will generally be suboptimal and therefore not worst-case. However, when provided with an optimal disturbance policy at the onset, we recover the case of optimizing over only control.

3.3 Examples

In this section, we will present two reachability problems *without* disturbances, and compare the results of our proposed method with those obtained from a full grid-based approach [61]. In each case, we observe that our method agrees with the ground truth, with a small but expected degree of

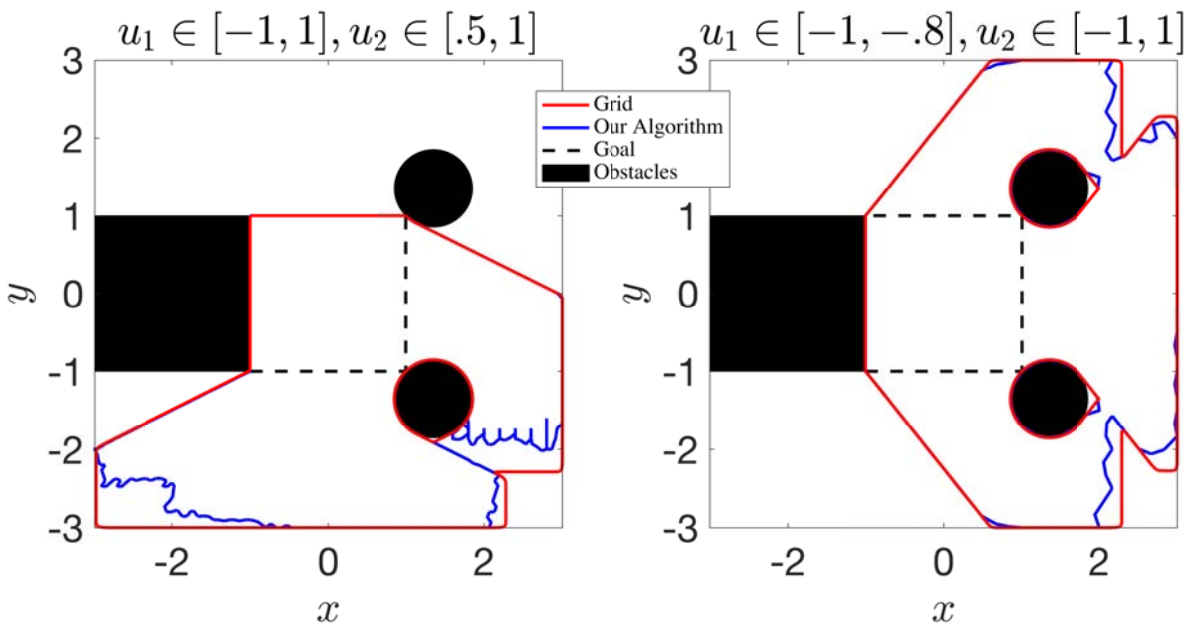


Figure 3.1: Reach-avoid set computation for 2D dynamics in (3.3) for two different control bounds. Sets computed using our method are subsets of the true sets.

conservatism. For these examples, the set $\mathcal{L} := \{x \mid l(x) \leq 0\}$ is a box of side-length 2 centered at $(x, y) = (0, 0)$, and $\mathcal{G} := \{x \mid g(x) > 0\}$ consists of the outer boundaries (i.e. $\max\{|x|, |y|\} \leq 3$) and the shaded obstacles (Fig. 3.1 and Fig. 3.2).

2D point

Consider a 2D dynamical system with inputs $u_1 \in [\underline{u}_1, \overline{u}_1]$ and $u_2 \in [\underline{u}_2, \overline{u}_2]$ which evolves as follows:

$$\dot{x} = u_1, \ \dot{y} = u_2 \tag{3.3}$$

Fig. 3.1 shows the reach-avoid sets for two different control bounds. We overlay the sets computed by our method on top of that computed using a dense 121×121 grid [61]. The red set was computed using standard Hamilton-Jacobi reachability and the blue set was computed using our classification-based method. Points inside the reach-avoid sets represent states from which there exists a control sequence which reaches the target while avoiding all obstacles. As guaranteed in Proposition 1, the set computed via Algorithm 1 is always a subset of the ground truth, meaning that every state marked in Fig. 3.1 as safe is also safe using the optimal controller. The computation time for the grid-based approach was 20 seconds, while for the classification-based it was 10 minutes.

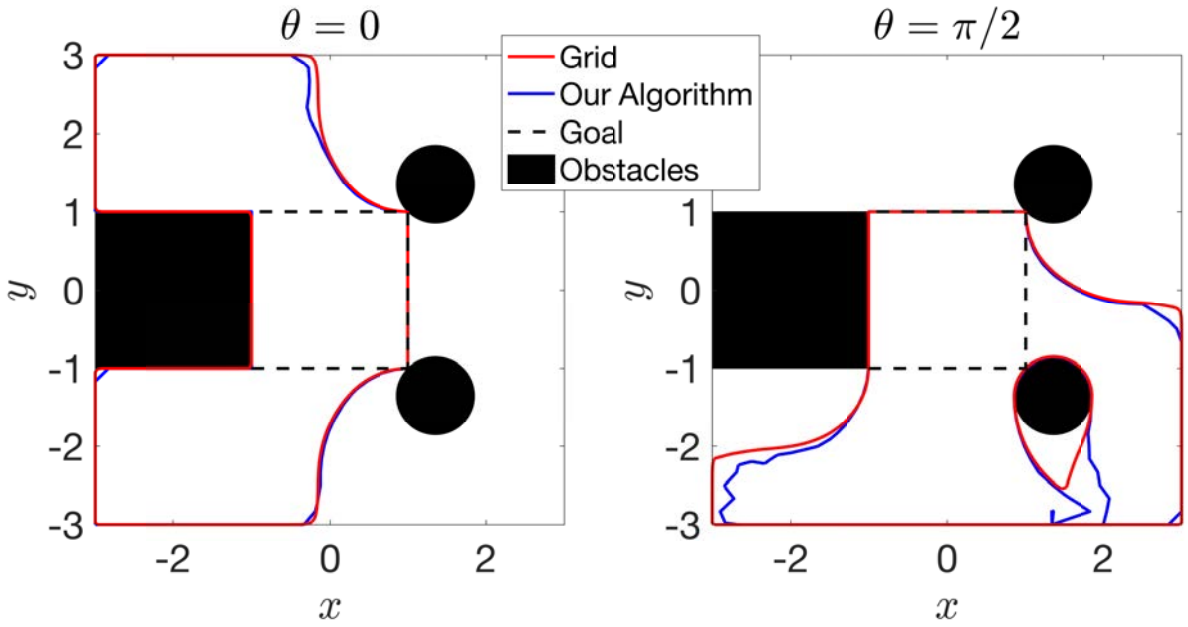


Figure 3.2: Reach-avoid set computation for the 4D dynamics in (3.4) for two different 2D slices and tangential speed $v = 1$.

4D unicycle

Next, we consider a higher-dimensional system representing a 4D unicycle model:

$$\dot{s} = \begin{bmatrix} \dot{x} & \text{(x-position)} \\ \dot{y} & \text{(y-position)} \\ \dot{\theta} & \text{(yaw angle)} \\ \dot{v} & \text{(tangential speed)} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ u_\omega \\ u_a \end{bmatrix} \tag{3.4}$$

in which controls are tangential acceleration $u_a \in [0, 1]$ and yaw rate $u_\omega \in [-1, 1]$. Fig. 3.2 shows a computed a reach-avoid set for this system for different 2D slices of the 4D state space on a 121^4 grid. As expected, our approach yields a conservative subset of the true reach-avoid set. In this case, the computation time for the grid-based approach was 3 days, while for the classification-based it was 30 minutes.

FaSTrack overview

FaSTrack (Fast and Safe Tracking) is a recent method for safe real-time motion planning [38]. FaSTrack breaks down an autonomous system into two agents: a simple *planning model* used for real-time motion planning, and a more complicated *tracking model* used to track the generated plan. To ensure safe tracking, FaSTrack computes the largest *relative* distance between the two models (tracking error), and the planning algorithm uses this result to enlarge obstacles for collision-

checking. The computation also provides an optimal feedback controller to ensure that the tracker remains within this bound during planning.

To solve for the largest tracking error in FaSTrack, we set the cost $l(r,t)$ in (2.22) as the distance to the origin in relative position space. We denote relative states by $r \in \mathcal{R} \subset \mathbb{R}^{N_r}$, and solve a modified form of (2.22):

$$\mathcal{V}^{\mathbf{u},\mathbf{d}}(t,r) := \max_{\tau \in [t,0]} l(\xi_{r,t}^{\mathbf{u},\mathbf{d}}(\tau),\tau) \tag{3.5}$$

Note that there is no constraint function $g(r,t)$. Also, we now take the *maximum* value over time because we want to find the maximum relative distance that could occur between the two models. Finally, observe that in this formulation, the disturbance actually encompasses two separate quantities: the original notion of disturbance (e.g. wind), *and* the planning model's control input, which directly affects the relative state dynamics. Henceforth, policy $\Pi_{-\infty}^{d}$ will represent the concatenated disturbance and planning algorithm policies.

Following Sec. 3.2, when the optimal converged disturbance policy $\Pi_{-\infty}^{d}$ is known analytically, the policies learned in Algorithm 1 will (by Prop. 1) yield a value function which *over-approximates* the optimal value function, i.e. $V^{\Pi_u,d^*}(r,t) \geq V(r,t)$. Thus, the maximum relative distance ever achieved between tracking mode and planning model, from any initial relative state, will always be *greater* when using the binary classifier policies than the optimal policy. For safe trajectory tracking, this translates into enlarging obstacles by a larger amount, meaning we still preserve safety.

FaSTrack Reachability Precomputation

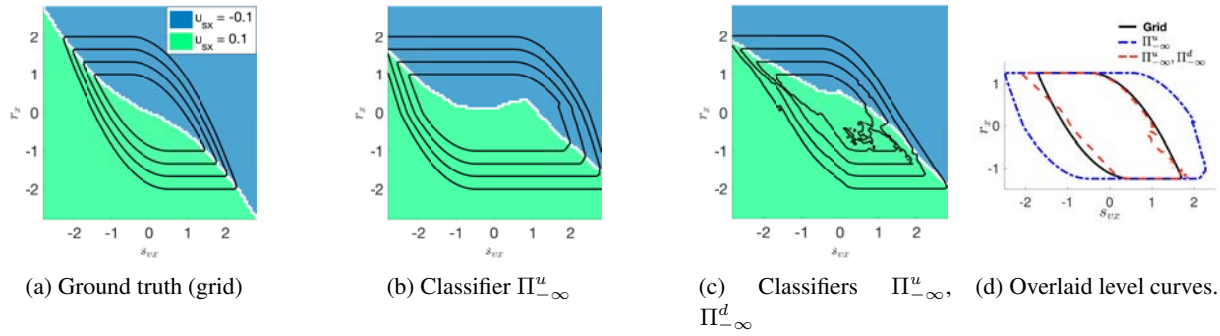
We employ Algorithm 1 to find the largest tracking error for two nonlinear models of the *tracking model*, which become control-affine under small angle assumptions. First, we consider a 6D near-hover model which decouples into three 2D subsystems and thus admits a comparison to grid-based methods. Then, we present results for a *fully-coupled* 7D model that cannot be solved exactly using grid-based techniques and use it for quadrotor control.

6D Decoupled

We first consider a 6D quadrotor tracking model and 3D geometric planning model. Here, the quadrotor control consists of pitch (θ) and roll (ϕ) angles, and thrust acceleration (T), while the planning model's maximum speeds are b_x, b_y, and b_z in each dimension. All of our results assume $\phi, \theta \in [-0.1, 0.1]$ rad, $T - g \in [-2.0, 2.0]$ m/s^2, and $b_x = b_y = b_z = 0.25$ m/s. We assume a maximum velocity disturbance of 0.25 m/s in each dimension. The relative position states (r_x, r_y, r_z) and the tracker's velocity states (x_{vx}, x_{vy}, x_{vz}) adhere to the following *relative dynamics*:

$$\begin{bmatrix} \dot{r}_x \\ \dot{r}_y \\ \dot{r}_z \end{bmatrix} = \begin{bmatrix} x_{vx} - d_{vx} - b_x \\ x_{vy} - d_{vy} - b_y \\ x_{vz} - d_{vz} - b_z \end{bmatrix}, \begin{bmatrix} \dot{x}_{vx} \\ \dot{x}_{vy} \\ \dot{x}_{vz} \end{bmatrix} = \begin{bmatrix} g \tan\theta \\ -g \tan\phi \\ T - g \end{bmatrix} \tag{3.6}$$

Without yaw, these dynamics decouple into three 2D subsystems, $(r_x, x_{vx}), (r_y, x_{vy})$, and (r_z, x_{vz}), and we use the technique in [17] to solve for the value function using (3.5) independently



(a) Ground truth (grid) (b) Classifier $\Pi^u_{-\infty}$ (c) Classifiers $\Pi^u_{-\infty}$, $\Pi^d_{-\infty}$ (d) Overlaid level curves.

Figure 3.3: Level sets of V^* in the (r_x, v_x) states (setting other states to zero) for (a) ground truth grid-based representation, (b) neural network $\Pi^u_{-\infty}$ trained on optimal disturbance policy $d^*(\cdot)$, and (c) neural networks $\Pi^u_{-\infty}$ and $\Pi^d_{-\infty}$ trained jointly. We encode the optimal (learned) control at each state as a different color. (d) Overlay of level sets from (a-c). Our method only yields a conservative result (a superset of the ground truth; see Sec. 3.3) when $\Pi^u_{-\infty}$ is trained against $d^*(\cdot)$.

for each 2D subsystem using grid-based techniques. Fig. 3.3 shows the level sets of the value function V^* and corresponding optimal tracker control policies. Fig. 3.3a is the grid-based ground truth, while Fig. 3.3b shows the induced value function for the neural network classifier policy $\Pi^u_{-\infty}$ trained against the optimal disturbance policy $d^*(\cdot)$, and Fig. 3.3c shows the induced value function when $\Pi^u_{-\infty}$ and $\Pi^d_{-\infty}$ were trained jointly. Note that the classification-based results shown here did *not* take advantage of system decoupling. Corroborating our theoretical results, the level sets of the value function induced by our learned classifiers over-approximate the true level sets when the disturbance plays optimally (Fig. 3.3d). Also, observe that using a learned (and hence, generally suboptimal) $\Pi^d_{-\infty}$, the resulting level sets in Fig. 3.3c still well-approximate (though they do not include) those in 3.3a. For each level curve, the maximum tracking error x is the largest value of the level curve along the r_x axis. Observe in Fig. 3.3d that the maximum tracking error is similar in all three cases. Finally, the line that separates the colored areas in the background of each figure in Fig. 3.3 denotes the decision boundary for the controller in each case.

7D Coupled

In this example, we introduce yaw (ψ) into the model as an extra state in (3.7) and introduce yaw rate control $\dot{\psi} \in [-1.0, 1.0]$ rad/s. The relative position dynamics in (r_x, r_y, r_z) are identical to (3.6). The remaining states evolve as:

$$\begin{bmatrix} \dot{x}_{vx} \\ \dot{x}_{vy} \\ \dot{x}_{vz} \\ \dot{x}_{\psi} \end{bmatrix} = \begin{bmatrix} g(\sin\theta\cos x_{\psi} + \sin\phi\sin x_{\psi}) \\ g(-\sin\phi\cos x_{\psi} + \sin\theta\sin x_{\psi}) \\ T\cos\phi\cos\theta - g \\ \dot{\psi} \end{bmatrix} \tag{3.7}$$

This dynamical model is now *7D*. It is too high-dimensional and coupled in the controls for current grid-based Hamilton-Jacobi reachability schemes, yet our proposed method is still able to compute a safety controller and the associated largest tracking error.

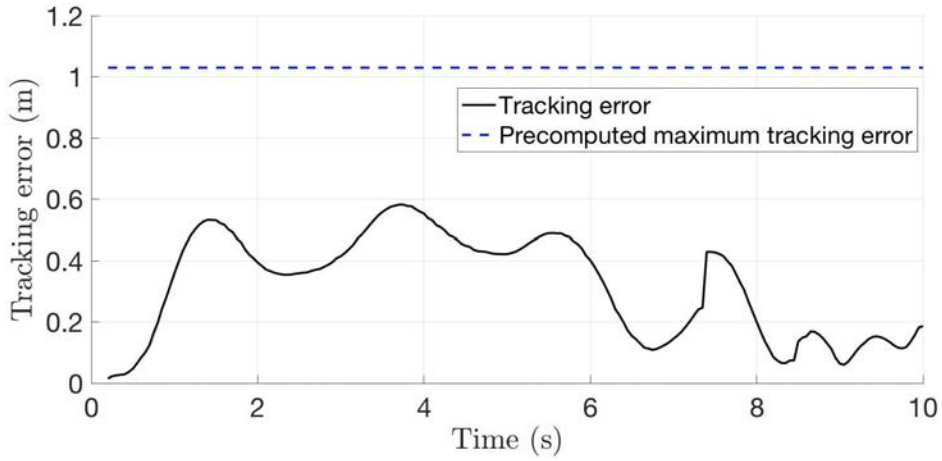


Figure 3.4: Relative distance between the quadrotor (tracking model) and planned trajectory (planning model) over time during a hardware test wherein a Crazyflie 2.0 must navigate through a motion capture arena around spherical obstacles. The quadrotor stays well within the computed tracking error bound throughout the flight. Note that the tracking error is large because our controller accounts for adversarial disturbances, unlike many common controllers.

Hardware Demonstration

We tested our learned controller on a Crazyflie 2.0 quadrotor in a motion capture arena. Fig. 3.4 displays results for (3.7). As shown, the quadrotor stays well within the computed error bound. For this experiment Π_u was trained using a sub-optimal disturbance policy. Even though we do not have a rigorous safety guarantee in this general case because we computed the disturbance, these results corroborate our intuition from Fig. 3.3 where the computed error bound remains essentially unchanged when using a learned disturbance instead of the optimum. However, by Prop. 1, with the optimal disturbance we could compute a strict guarantee. The hardware demonstration can be seen in our video: `https://youtu.be/_thXAaEJYGM`. Figure 3.5 shows our experimental setup with the quadrotor tracking the trajectory.

Figure 3.5: Quadrotor flying using a neural network classifier.

3.4 Implementation Details

In this work we train each binary classifier by minimizing the cross-entropy loss between inputs and labels via stochastic gradient descent. We run the classification problem for a pre-specified number of gradient steps between each new set of policies. Since we expect policies to vary slowly over time, we initialize the weights for each new network with those from its predecessor. This serves two purposes. First, it serves as a "warm start" leading to faster stochastic gradient descent convergence. Second, it provides a practical indicator of policy convergence—i.e. if the initial classification accuracy of a new policy is almost equal to that of its predecessor, the policy has most likely converged. Fig. 3.6 shows a typical learning curve when running Algorithm 1. The figure shows the progression of the validation error (against unseen state-action pairs) in each iteration.

All feedforward neural network classifiers had two hidden layers of 20 neurons each, with rectified linear units (ReLUs) as the activation functions, and a final softmax output. The gradient descent algorithm employed was RMSprop with learning rate $\alpha = 0.001$ and momentum constant $\beta = 0.95$. When using function approximators, it is in general unclear how many samples should be taken as a function of the state dimension. In our case, the number of points N sampled at each iteration was $1k$ for the 2D example, and $200k$ for the 4D, 6D and 7D system. All initial weights and biases were drawn from a uniform probability distribution between $[-0.1, 0.1]$. All computations were performed on a 12 core, 64-bit machine with Intel® Core™ i7-5820K CPUs @ 3.30GHz. In our implementation we did not employ any form of parallelization. All code for the project can be found at `https://github.com/HJReachability/Classification_Based_Reachability`.

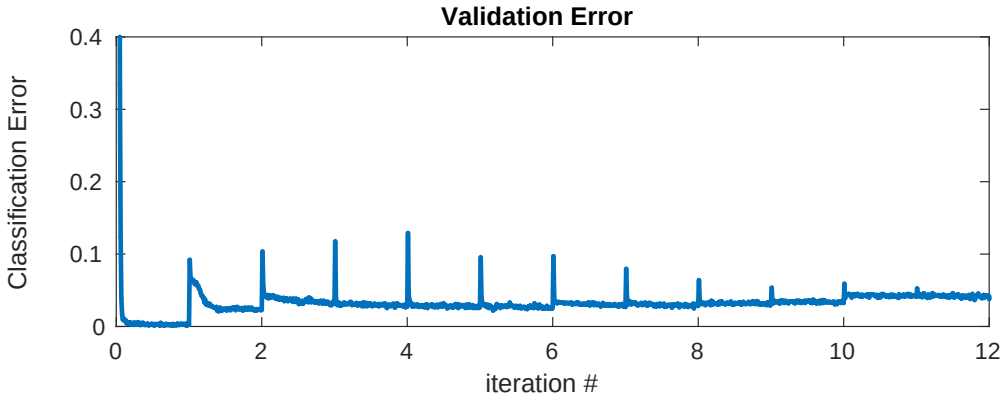


Figure 3.6: Learning curves for a single classifier of the 6D decoupled system. Classification error decreases between spikes, which mark each new k in Algorithm 1. Spikes shrinking hints that classifiers eventually converge.

3.5 Chapter Summary

In this chapter we introduced a classification-based approach for approximate reachability. We showed that for control-affine systems, which are very general, yet realistic types of dynamical models, the optimal policy for reachability problems can be readily encoded as a set of neural network classifiers. These classifiers can be trained sequentially via approximate dynamic programming by evaluating how each component of the control (disturbance) vector influences the subsequent trajectory of the system and the associated value. Once the value is computed for each possible "bang-bang" control for every sampled state, we gather the approximately optimal controls and states into pairs, which are then used as training data for classification. We show that this procedure yields good policies which induce good approximations of the value function and associated reachable sets. Lastly, we demonstrate the efficacy of the trained classifiers on a tracking task using a quadrotor.

Chapter 4

Reachability Analysis and Reinforcement Learning

This chapter is based on the paper "Bridging Hamilton-Jacobi Safety Analysis and Reinforcement Learning" [29], written in collaboration with Jaime F. Fisac, Neil Lugovoy, Shormona Ghosh, and Claire Tomlin.

In Ch. 3 we introduced a classification-based technique which is able to learn an approximately optimal controller via dynamic programming for reachability problems. In this chapter we will introduce a different approach that directly approximates the value function (2.23) which draws its inspiration from reinforcement learning literature.

4.1 Reinforcement Learning and Safety

In recent years, reinforcement learning techniques [88] have proven their usefulness in computing data-driven approximate solutions to optimal control problems seeking the maximization of a discounted additive payoff in complex and high-dimensional systems [64, 37, 82, 52]. Unfortunately, functions representing a sum of rewards over time are not well suited to capture the safety objective, since safety is not determined by how much a system fails on average, but by whether it fails *at all*. Partly for this reason, reinforcement learning techniques have not seen widespread use for safety analysis.

Another consequence of this disconnect between formulations is that controllers computed through reinforcement learning are typically not inherently safety-preserving, a limitation that has hindered their applicability to physical autonomous systems. In recent years, there has been a growing interest around "safe learning" schemes. Some approaches have proposed formalizing safety as stability [10] or near-constraint satisfaction [66, 1]. Others have built on the Hamilton-Jacobi reachability literature to provide constraint satisfaction guarantees by computing a safety-preserving control policy and overriding the learning controller when it attempts to violate computed safety constraints [33, 3, 30]. Unfortunately, this family of methods inherits the difficulty in scaling

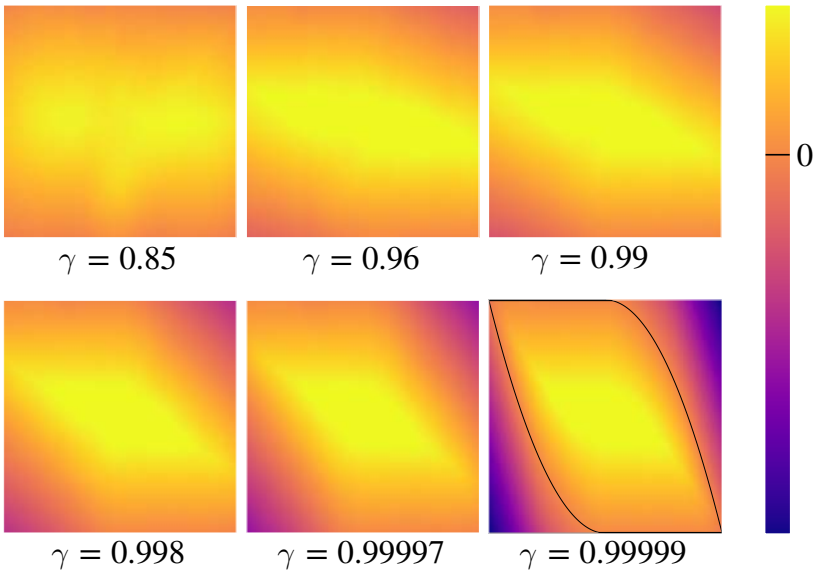


Figure 4.1: Multiple snapshots of the neural network output of our Safety Q-learning algorithm for a double-integrator system. As we anneal the discount factor $\gamma \to 1$ during Q-learning, our learned discounted safety value function asymptotically approaches the undiscounted value, allowing us to recover the safe set and optimal safety policy with very high accuracy.

up computations beyond low-dimensional systems.

In this chapter we present work that seeks to unlock a new family of tools for safety analysis by rendering a wide range of state-of-the-art methods in the reinforcement learning literature readily usable for safety analysis in high-dimensional systems. Building on the initial work in [2], which introduced a time discount into the minimum-payoff optimal control problem, we propose a similar, tighter discounted formulation of Hamilton-Jacobi safety analysis, obtaining a contraction Bellman operator that lends itself to the use of temporal difference learning techniques. We prove the key properties of this new discounted *Safety Bellman Equation* and show that our resulting *Safety Q-learning* algorithm converges to the safety state-action value function in finite Markov decision processes.

The Safety Q-learning scheme allows us to recover the globally optimal solution to the corresponding Hamilton-Jacobi analysis (to resolution completeness [7, 74]) in low-dimensional problems where dense computation is viable: we validate our results using tabular Q-learning against a double integrator system, achieving high accuracy relative to the analytic solution. We further observe comparable performance when replacing the state-space grid with a neural network function approximator. Crucially, annealing the discount factor during learning allows asymptotic recovery of the solution to the *undiscounted* safety problem (Fig. 4.1).

We evaluate deep Safety Q-learning on a variety of simulated robotics tasks, and observe consistently accurate results against numerical dynamic programming solutions. In high-dimensional systems beyond the reach of traditional numerical methods, predicted safety accurately matches the

empirical performance of the learned safety controller.

We finally implement policy optimization through an adaptation of the basic REINFORCE algorithm [95] to our discounted safety formulation, and explore the potential of using sate-of-the-art methods by similarly adapting the soft actor-critic (SAC) scheme [36]. The promising results on an 18-dimensional problem suggest the usability of this family of reinforcement learning methods for learning policies with the ability to preserve safety in high-dimensional systems.

It is important to clarify that our formulation yields a promising new tool for safety *analysis*: it is not in itself a safe learning framework, since it requires experiencing failure states in order to learn about safety. Our approach is primarily meant to be used as a computational tool in conjunction with a model (simulation) of the system dynamics; its *truly* model-free application, learning constraint satisfaction online directly on the real system, should be limited to training conditions that are not safety-critical (for example, a vehicle test track with only virtual obstacles). Once the safety analysis has been computed (learned), the resulting control policy can be applied to the physical system in similar conditions to other safety controllers, including safe learning of performance objectives [30]. While our analysis here is presented for deterministic dynamics, robust and stochastic extensions are possible (and have been explored to some extent in [2]). We expect that such extensions will be important for implementation of our formulation on physical systems, which is of course its ultimate intended application.

4.2 Parallels between Reachability Analysis and Reinforcement Learning

In Ch. 2 the cost functional (2.5) was introduced. Reinforcement learning seeks to solve an optimization problem with a very similar cost functional. In discrete time, it is written as follows:

$$\mathcal{V}^{\mathbf{u}}(x,t) := \sum_{i=0}^{(T-t)/\Delta t} \gamma^{i-t} r(\xi^{\mathbf{u}}_{x,t}(i\Delta t + t), u(i\Delta t + t)) \tag{4.1}$$

where $r(x,u) : \mathcal{X} \times \mathcal{U} \to \mathbb{R} < \infty$ is known as the reward function, and $\gamma \in [0,1)$ is known as the *discount factor*, which is a parameter that regulates how far into the future states and controls in a trajectory contribute to the value. For infinite time problems where $T = \infty$ the discount factor ensures that V is finite. Using Bellman's principle of optimality it can be shown that the infinite time optimal value function must satisfy the condition

$$V(x) = \max_{u \in \mathcal{U}} \; r(x,u) + \gamma V\big(x + f(x,u)\Delta t\big) \;\; . \tag{4.2}$$

Crucially, this condition, when interpreted as an update rule, induces a contraction mapping in the space of value functions (under the supremum norm), which implies that its successive application to any initial V will ultimately converge to the unique solution of (4.2). This enables key convergence results in reinforcement learning schemes, most notably temporal-difference learning methods such as Q-learning [93].

In contrast, if we examine the discrete time cost functional for reachability analysis, where again $l(x)$ is the implicit surface function defined in (2.46),

$$\mathcal{V}^{\mathbf{u}}(x,t) := \min_{\tau\in\{t,t+\Delta t,\dots,T\}} l(\xi^{\mathbf{u}}_{x,t}(\tau)) \tag{4.3}$$

we obtain, by following analogous steps to the ones shown in (2.24), that the infinite time optimal value function must satisfy

$$V(x) = \min\left\{l(x), \max_{u\in\mathcal{U}} V\big(x + f(x,u)\Delta t\big)\right\} \quad . \tag{4.4}$$

An important observation about (4.4) is that, unlike (4.2), it does *not* induce a contraction mapping on V and therefore it is not generally possible to converge to the fixed point by application of value iteration or temporal difference learning.

4.3 The Search for a Contraction Mapping

Our key observation stems from an intuitive interpretation of time-discounting in the problem of cumulative rewards: at every instant, there is a small probability $1-\gamma$ of transitioning to an absorbing state from which no more rewards will be accrued. Thus in (4.2) the discount factor $\gamma \in [0,1)$ can be seen as the probability of the episode continuing, with $1-\gamma$ conversely representing the probability of transitioning to a terminal state. The following, equivalent definition of the optimal value function clearly conveys this idea:

$$V(x) = \max_{u\in\mathcal{U}} \ (1-\gamma)r(x,u) + \gamma\Big(r(x,u) + V\big(x + f(x,u)\Delta t\big)\Big) \quad . \tag{4.5}$$

The first term in the sum represents the remaining reward that can be obtained if the episode terminates immediately after one step, whereas the second term shows the rewards that can be accrued if the episode were to continue. An analogous interpretation in the problem of minimum payoff over time can be achieved by modifying (4.4) to account for such a transition. Here if, with probability $1-\gamma$, an episode were to end after the current time step, the minimum future $l(\cdot)$ would be equal to the current $l(x)$. This induces the discrete-time discounted dynamic programming equation

$$V(x) = (1-\gamma)l(x) + \gamma\min\left\{l(x), \max_{u\in\mathcal{U}} V\big(x + f(x,u)\Delta t\big)\right\}. \tag{4.6}$$

This equation yields a strictly tighter contraction mapping than the recent analysis in [2]. By discounting locally towards the current $l(x)$, rather than towards a global upper bound L on l, we significantly reduce the amount of information loss due to discounting. This is shown in at the end of this chapter.

Letting l_i be the value of l achieved by a discrete-time state trajectory $\xi_x^{\mathbf{u}}$ at the i-th time step, the explicit form of the objective maximized in (4.6) is a "time-discounted" minimum:

$$J(\xi_x^{\mathbf{u}}) = (1-\gamma)l_0 + \gamma\Big[\min\big\{l_0, (1-\gamma)l_1 + \gamma(\min\{l_1, (1-\gamma)l_2 + \gamma\ldots)\}\big\}\Big] \ . \tag{4.7}$$

We prove two key properties of our proposed equation.

Theorem 1. *(Contraction mapping) The discounted Safety Bellman Equation* (4.6) *induces a contraction mapping under the supremum norm. That is, let $V, \tilde{V} : \mathcal{X} \to \mathbb{R}$, then there exists a constant $\kappa \in [0,1)$ such that $\|B[V] - B[\tilde{V}]\|_\infty \leq \kappa \|V - \tilde{V}\|_\infty$.*

Proof. It will suffice to show that for *all* states $x \in \mathcal{X}$, $|B[V](x) - B[\tilde{V}](x)| < \kappa\|V - \tilde{V}\|_\infty$. We have:

$$\begin{aligned}
&|B[V](x) - B[\tilde{V}](x)| \\
&\quad = \gamma|\min\{l(x), \max_{u\in\mathcal{U}} V\big(x + f(x,u)\Delta t\big)\} \\
&\qquad - \min\{l(x), \max_{\tilde{u}\in\mathcal{U}} \tilde{V}\big(x + f(x,\tilde{u})\Delta t\big)\}| \\
&\quad \leq \gamma|\max_{u\in\mathcal{U}} V\big(x + f(x,u)\Delta t\big) - \max_{\tilde{u}\in\mathcal{U}} \tilde{V}\big(x + f(x,\tilde{u})\Delta t\big)| \ .
\end{aligned}$$

Now, without loss of generality suppose the first maximum is the larger one, and let $u^* \in \mathcal{U}$ achieve it. We continue:

$$\begin{aligned}
&|B[V](x) - B[\tilde{V}](x)| \\
&\quad \leq \gamma|V\big(x + f(x,u^*)\Delta t\big) - \tilde{V}\big(x + f(x,u^*)\Delta t\big)| \\
&\quad \leq \gamma \max_{u\in\mathcal{U}} |V\big(x + f(x,u)\Delta t\big) - \tilde{V}\big(x + f(x,u)\Delta t\big)| \\
&\quad \leq \gamma \sup_{\tilde{x}} |V(\tilde{x}) - \tilde{V}(\tilde{x})| = \gamma\|V - \tilde{V}\|_\infty \ .
\end{aligned}$$

Thus the sought contraction constant is in fact $\gamma \in [0,1)$. □

Proposition 2. *(Value approximation) In the limit of no discounting, the fixed-point solution to the Safety Bellman Equation* (4.6) *converges to the undiscounted safety value function.*

Proof. Taking the limit of the optimization of (4.7) as γ goes to 1 we recover:

$$\lim_{\gamma\to 1} V(x) = \max_{\mathbf{u}^{0:T}} \min\big\{l_0, l_1, l_2, \ldots\big\} \ ,$$

which solves (4.4) and is equivalent to the discrete-time approximation of (2.47). □

The above two theoretical results enable the use of reinforcement learning techniques for safety analysis. We end this section with an important consequence of Theorem 1.

Theorem 2. *(Convergence of Safety Q-learning) Let $\mathbf{X} \subseteq \mathcal{X}$ and $\mathbf{U} \subseteq \mathcal{U}$ be finite discretizations of the state and action spaces, and let $\mathbf{f} : \mathbf{X} \times \mathbf{U} \to \mathbf{X}$ be a discrete transition function approximating the system dynamics. The Q-learning scheme applied to the discounted safety problem and executed on the above discretization converges, with probability 1, to the optimal state-action safety value function*

$$Q(\mathbf{x}, \mathbf{u}) := (1-\gamma)l(\mathbf{x}) + \gamma \min\left\{l(\mathbf{x}), \max_{\mathbf{u}' \in \mathbf{U}} Q\big(\mathbf{f}(\mathbf{x}, \mathbf{u}), \mathbf{u}'\big)\right\} \ ,$$

in the limit of infinite exploration time and given partly-random episode initialization and learning policy with full support over $\mathbf{X}$ and $\mathbf{U}$ respectively. Concretely, learning is carried out by the update rule:

$$\begin{aligned} Q_{k+1}(\mathbf{x}, \mathbf{u}) \leftarrow Q_k(\mathbf{x}, \mathbf{u}) + \alpha_k \Big[&(1-\gamma)l(\mathbf{x}) + \\ &\gamma \min\left\{l(\mathbf{x}), \max_{\mathbf{u}' \in \mathbf{U}} Q\big(\mathbf{f}(\mathbf{x}, \mathbf{u}), \mathbf{u}'\big)\right\} - Q_k(\mathbf{x}, \mathbf{u})\Big] \ , \end{aligned}$$

for learning rate $\alpha_k(\mathbf{x}, \mathbf{u})$ satisfying

$$\sum_k \alpha_k(\mathbf{x}, \mathbf{u}) = \infty \quad \sum_k \alpha_k^2(\mathbf{x}, \mathbf{u}) < \infty \ ,$$

for all $\mathbf{x} \in \mathbf{X}, \mathbf{u} \in \mathbf{U}$.

Proof. Our proof follows from the general proof of Q-learning convergence for finite-state, finite-action Markov decision processes presented in [91]. Our transition dynamics $\mathbf{f}$, initialization and policy randomization, and learning rate α_k satisfy Assumptions 1, 2, and 3 in [91] in the standard way. The only critical difference in the proof is the contraction mapping, which we obtain under the supremum norm by Theorem 1: with this, Assumption 5 in [91] is met, granting convergence of Q-learning by Theorem 3 in [91]. □

We stress that, beyond Q-learning, the contraction-mapping property of our discounted safety backup opens the door to straightforward application of a wide variety of reinforcement learning schemes to safety analysis. We dedicate the following section to a first demonstration in which we explore the application of canonical reinforcement learning algorithms in the two main families: value learning and policy optimization.

4.4 Results

We present the results of implementing our proposed discounted Safety Bellman Equation in multiple reinforcement learning schemes: tabular Q-learning [93], deep Q-learning (DQN) [64],

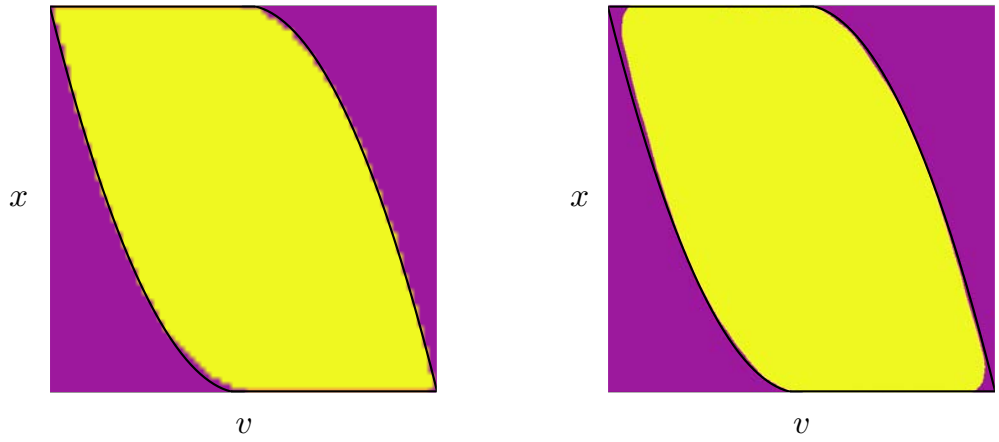


Figure 4.2: Safe sets learned by tabular (left) and deep Q-learning (right) with the discounted Safety Bellman Equation compared to the analytic set (black).

REINFORCE [95], and soft actor-critic (SAC) [36], and four different dynamical systems. We first validate the computed safety value function and safe set against analytically and numerically obtained ground-truth references in traditionally tractable systems. We consider two dynamical systems commonly used as benchmarks in control theory, namely a 2-D double-integrator system and a 4-D cart-pole system. We then demonstrate the scalability and usefulness of our formulation in higher-dimensional nonlinear systems, for which exact safety analysis is generally considered intractable. We use simulation environments common in reinforcement learning [13], namely a 6-D lunar lander system and an 18-D "half-cheetah" system.

Validation: comparison to ground truth

Analytic validation: double integrator

The double integrator is a classic reachability example where the control policy seeks to keep the system in the set $\{[x, v] \in \mathbb{R}^2 : x \in [x_{\text{min}}, x_{\text{max}}]\}$ with the dynamics characterized by:

$$\dot{x} = v \ , \quad \dot{v} = u \ , \tag{4.8}$$

with $|u| \leq u_{\text{max}}$, where x can be seen as position, v as velocity, and u as an acceleration input. Analytically, the safe set is characterized by the interior of the boundary defined by the parabolic segments

$$\begin{cases} x_{\text{low}} + \frac{v^2}{2u_{\text{max}}} & v \leq 0 \\ x_{\text{high}} - \frac{v^2}{2u_{\text{max}}} & v \geq 0 \end{cases} \tag{4.9}$$

and the boundaries $x = x_{\text{low}}$, $x = x_{\text{high}}$. Although simple, this example proves a useful context for visualizing the effect of γ, since the entire value function can be represented in two dimensions.

It can be seen in Fig. 4.1 how as γ is annealed the time horizon of safety is effectively extended: for lower values the value function resembles $l(\cdot)$, and for higher values it approaches the undiscounted value function. Final accuracy and in-training performance are shown in Fig. 4.3 and Fig. 4.4.

Using tabular Q-learning with $l(\cdot)$ as the signed Euclidean distance to the boundary of the constraint set and annealing γ to 1 similar to [70], we observe convergence to the safe set up to the resolution of the grid. Independently training 100 deep Q-networks [64] with fully-connected layers using our discounted Safety Bellman Equation we find near-convergence to the safe set with an average 2.26×10^{-5} (minimum 0, maximum 1.27×10^{-4}) fraction of points incorrectly characterized as safe and an average 1.76×10^{-4} (minimum 3.26×10^{-5}, maximum 3.31×10^{-4}) of points falsely characterized as unsafe. Classification is visualized in Fig. 4.2.

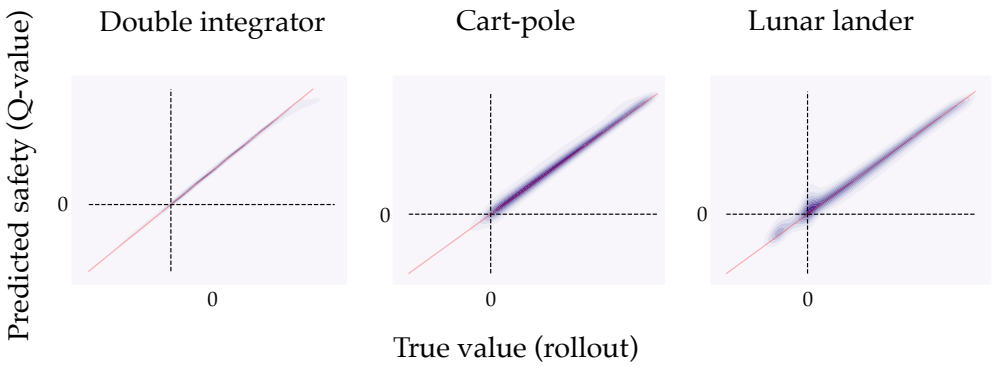


Figure 4.3: Predicted vs. achieved minimum signed distance to violations for 10^6 simulated rollouts with 100 trained networks. Red line indicates identity.

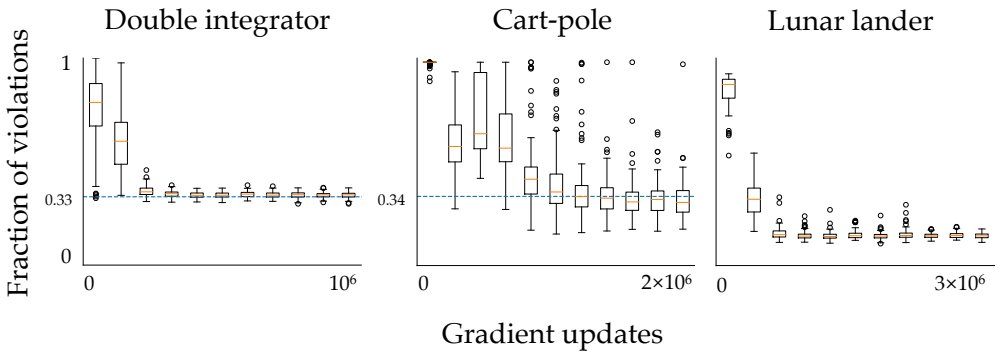


Figure 4.4: Fraction of initial conditions resulting in violations as training proceeds. Each data point is a sample average from 1000 episodes; statistics are taken over 100 independent training runs. As learning progresses, the fraction of violations reliably decreases, approaching the ground-truth fraction of unsafe states (from which violation is inevitable) for the double integrator and cart-pole. Lunar lander ground truth is unknown.

Numerical validation: cart-pole

The cart-pole system (inverted pendulum) is a classic control problem and one ripe for safety analysis. A cart moving on a one-dimensional track is attached by a pivot to a pole. The control policy seeks to keep the pole from falling and to keep the cart from the edge of the track by applying accelerations to the cart. For this system, the ground truth safe set must be computed numerically on a grid using dynamic programming [62]. Over 100 random seeds we find that an average 5.16×10^{-5} (minimum 4.90×10^{-6}, maximum 2.56×10^{-4}) fraction of points are misclassified as safe and an average fraction 5.80×10^{-4} (minimum 4.24×10^{-4}, maximum 8.4×10^{-4}) fraction of points are misclassified as unsafe, relative to the numerically approximated ground truth. In reality, the precision of the numerical ground truth is limited by the grid resolution; thus, if we consider any points less than one full grid cell away from a safe grid point to be safe, we find that only an average 1.47×10^{-6} (minimum 0, maximum 4.54×10^{-5}) fraction of points are misclassified as safe by our method (Figs. 4.3 and 4.4).

Scalability: safety for high dimensional systems

The two examples we have shown thus far help us validate our approach against well-established safety analysis tools. However, a motivating factor of this work is to enable safety analysis for systems that are too high-dimensional for traditional approaches. In this section we will explore how our method fares in two high-dimensional systems from the OpenAI Gym environment collection [13].

Temporal difference: lunar lander

We first consider a lunar lander system with 6 states $s = [x, y, \theta, \dot{x}, \dot{y}, \dot{\theta}]$ (vehicle pose and velocities). The signed distance safety function is defined as $l(s) = \max\{l_{\text{fly}}(s), l_{\text{land}}(s)\}$, with $l_{\text{fly}}(s) = \min\{x - x^w_{\text{min}}, x^w_{\text{max}} - x, y - y^w_{\text{min}}, y^w_{\text{max}} - y\}$, and $l_{\text{land}}(s) = \min\{x - x^p_{\text{min}}, x^p_{\text{max}} - x, \theta - \theta_{\text{min}}, \theta_{\text{max}} - \theta, \dot{y} - \dot{y}_{\text{min}}\}$. Terms marked with superscript w indicate viewing window limits, and terms marked with superscript p indicate landing pad limits. The margin $l(\cdot)$ is thus constructed to allow either flying in free space or landing on the pad; this example illustrates the ability to encode arbitrary state constraints through a signed distance function.

We train 100 Safety DQNs with different random seeds and compare learned values against the observed safety by performing on-policy rollouts in simulation (Fig. 4.3). Since computing the safety value function through dynamic programming is intractable on 6-dimensional systems, there is no known ground truth to compare against (Fig. 4.4). While the learned Q-value function may be suboptimal, it does give accurate safety predictions for its induced best-effort policy. We present x-y slices of a sample trained value function in Fig. 4.5, where the learned safety structure can be seen.

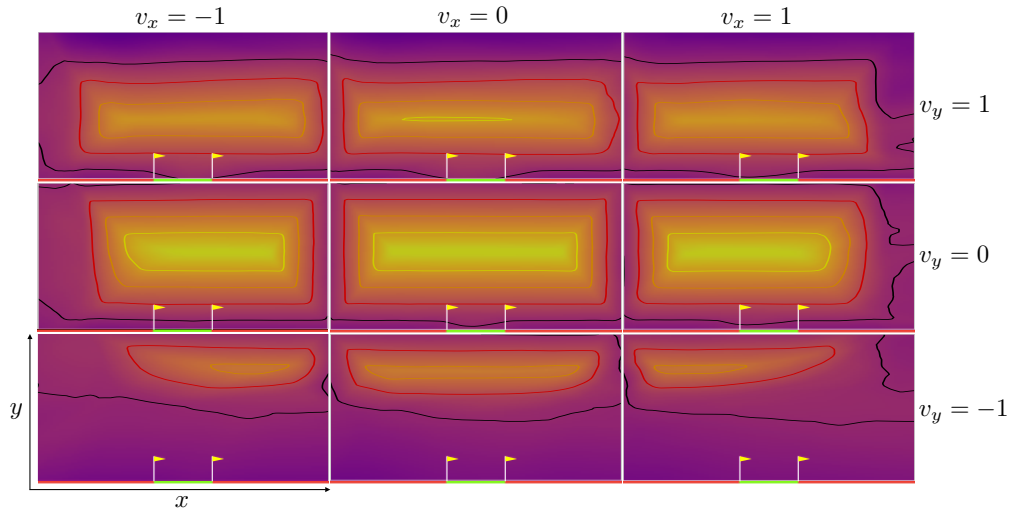


Figure 4.5: Slices of the learned lunar lander value function overlaid on the image of the viewing window for $\theta = 0$ and $\dot{\theta} = 0$. Computed safe set boundary in black. At low speeds, the values near the ground are higher close to the landing pad, revealing the effect of l_{land}. For large downward velocities, ground collision is inevitable from the lower half of the screen.

Policy optimization: half-cheetah

Many successful modern reinforcement learning methods use neural networks to directly represent control policies and search for efficient strategies. A number of policy gradient algorithms derive their policy update from the REINFORCE rule [95]:

$$\nabla_\theta \mathbb{E}_{\xi\sim\pi_\theta}[J(\xi)] = \mathbb{E}_{\xi\sim\pi_\theta}[\nabla_\theta \log(p_{\pi_\theta}(\xi))J(\xi)] \ , \tag{4.10}$$

with $p_{\pi_\theta}(\cdot)$ denoting the probability of taking a trajectory ξ under the stochastic policy π_θ parametrized by θ, and $J(\cdot)$ denoting the outcome of ξ. Taking $J(\cdot)$ to represent the time-discounted minimum payoff $l(\cdot)$ of the trajectory as in (4.7), we can directly optimize a policy for discounted safety.

We consider an 18-dimensional half-cheetah system within the MuJoCo physics simulator [90], and define $l(\cdot)$ to be the minimum height of the head and the front leg, so that a failure occurs if either touches the ground (Fig. 4.6). Note that we must (at least in part) initialize trajectories at configurations from which the system could in principle maintain safety. Running policy gradient using REINFORCE, all policies trained for discounted safety attempt to balance, though not always successfully, and some learn to sit. In contrast, policies trained with the standard reinforcement learning formulation using $l(\cdot)$ as an additive reward tend to raise the front leg and sometimes jump, and invariably fall over. Defining an alternative reward that purely penalizes forbidden contacts similarly failed to yield safe learned behaviors.

Using the more sophisticated soft actor-critic (SAC) algorithm [36] we find that after hyperparameter optimization, all policies trained across 20 random seeds using a discounted sum of $l(\cdot)$ launch the cheetah into the air and always fall over. Using a discounted sum of contact penalties,

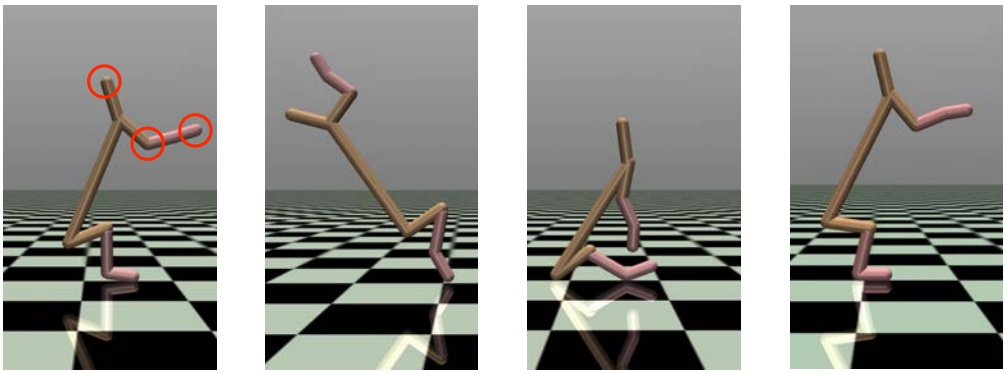


Figure 4.6: Learned half-cheetah safety policies aimed to keep the head and front leg off the ground. *Left to right:* typical starting configuration; an unsafe jumping policy learned using a sum of discounted heights; a safe sitting policy learned using discounted safety or (less reliably) discounted sum of contact penalties; a safe standing policy learned using discounted safety.

65% of policies do learn to sit; however, the remaining 35% produce unsafe jumping behavior. We speculate that the sparsity of the reward signal makes learning challenging. Across the 20 random seeds, all policies trained with discounted safety visibly attempt to stand: 80% of them succeed in doing so reliably, with an additional 5% reliably sitting if standing fails. The different emergent policies are depicted in Fig. 4.6.

4.5 Chapter Summary

In this chapter we have presented a learning based approach to reachability problems. We showed that the optimality condition for the infinite time value function can be modified to incorporate a discount factor and yield an update rule which is a contraction mapping. Using this fact, we are able to modify several reinforcement learning algorithms, such as tabular Q-learning, DQN, REINFORCE and SAC, in order to learn the optimal value function or control policy for the reachability problem. We test these modified algorithms in a variety of low-dimensional and high-dimensional robotic tasks. Our results show promise in scaling reachability tools to higher dimensional systems.

Part II

Assured Autonomy for Learning-Enabled Systems

Chapter 5

Background and Preliminaries

In this chapter we provide an introduction to feedforward rectifier linear unit (ReLU) neural networks, or (feedforward) ReLU networks for short, which are a type of function approximator widely used in machine learning. We highlight some of the challenges associated with the deployment of these approximators in the real world, and formalize some of the problems in guaranteeing their behavior.

5.1 Neural Networks, Performance Guarantees and Safety

Neural networks have become ubiquitous in many engineering applications, particularly in domains relating to vision [50], speech recognition [34, 46], reinforcement learning [85, 65] and control [94]. While many of these applications are used in contexts where safety is of little concern, practitioners are increasingly employing them in safety-critical domains such as self-driving cars [11], healthcare [73] or the power-grid [87], where performance guarantees are a requirement. This can be particularly concerning when taking into account adversarial attacks on neural networks [4, 58], which show how brittle some of these systems can be in practice.

Providing guarantees of performance for neural networks is computationally hard. It can been shown that the verification problem (which will be described in the next sections) is NP-Complete [43, 86]. Despite this fact, it is possible to find approximate solutions to certain problems in polynomial time [99, 100]. In general, the stronger a desired guarantee is, the more computationally challenging the problem becomes.

It is important to point out that the use of neural networks happens in several stages: first, the neural network must be trained using an appropriate learning algorithm and data, and, then, once training is completed, it can be deployed in the real world. Paradigms for ensuring performance guarantees exist during training [81, 51, 54], while the parameters of the network are being updated, and during deployment, once the parameters of the network have been *fixed*[1]. In this dissertation we will be concerned with the latter type of performance guarantees and will forego delving into the

[1]The dichotomy between training and deployment is also known as *training* vs. *inference*.

details of how (robust) training is accomplished for neural networks. For further information on neural network training we direct the reader to [22].

5.2 Feedforward ReLU Networks

A feedforward ReLU network is a parameterized function $f_\theta(x)$ defined as

$$\begin{aligned} \hat{z}_{i+1} &= W_i z_i + b_i \,, \text{ for } i = 1, ..., K-1 \\ z_j &= \max\{\hat{z}_j, \mathbf{0}\} \,, \text{ for } j = 2, ..., K-1 \ , \end{aligned} \tag{5.1}$$

with $W_i \in \mathbb{R}^{n_{i+1} \times n_i}$, $b \in \mathbb{R}^{n_{i+1}}$, $z_1 = x \in \mathcal{B}$ and $f_\theta(x) = \hat{z}_K$. We define the *input set* $\mathcal{B}$ to be a bounded convex set in $\mathbb{R}^{n_1}$, and, similarly, we also define the *output set* $\mathcal{S}$, to be a convex set or a union of convex sets in $\mathbb{R}^{n_K}$. The set $\theta = \{W_i, b_i\}_{i=1,...,K-1}$ represents the set of parameters of the network.

From the definition of Equation (5.1), it is clear that ReLU networks are piece-wise affine functions. This follows from the fact that any composition of an affine function (i.e. linear function plus a bias term) and a piece-wise affine function results in a piece-wise affine function. An important aspect that will come into play in later sections, is that the domain is partitioned into polytopic regions $\mathcal{P}_i$ such that $\bigcup_i \mathcal{P}_i = \mathbb{R}^{n_1}$. For a closer look at the properties of ReLU networks, including the partition of the domain into polytopic regions, we direct the reader to [67, 84, 77].

5.3 The Verification Problem

In this section we introduce the verification problem for neural networks. This problem is very general in its formulation: it takes in an input set $\mathcal{B}$ and a (ReLU) neural network $f_\theta(x)$, and tries to predict how these interact with a set $\mathcal{S}$ in the output space. The ambiguity of what the input set $\mathcal{B}$ and output set $\mathcal{S}$ represent is deliberate since neural networks can be used in a variety of contexts where the inputs and outputs can also vary. For example, for advisory-type neural networks in collision avoidance, the input space can represent the set of relative states between two vehicles, and the output space can represent the probability of collision. In this context, $\mathcal{B}$ could represent a set of configurations between vehicles and $\mathcal{S}$ some interval of probabilities. If the neural network is used for digit recognition in images (i.e. $[0, 1, ..., 9]$), the input space would be the set of pixel images, and the output would correspond to a 10-dimensional vector of probabilities which sum up to one. In this case, $\mathcal{B}$ could represent a "volume" of noisy images around some nominal image, and $\mathcal{S}$ a half-space denoting the outputs whose inputs are more likely to resemble 0 than 1.

Verification Problem

Given a feedforward neural network $f_\theta(x)$, an input set $\mathcal{B}$ and an output set $\mathcal{S}$, the solution to the *verification* problem returns whether the proposition

$$\forall x \in \mathcal{B}, f_\theta(x) \notin \mathcal{S} \tag{5.2}$$

is true or false. In other words, it seeks an answer to the question: does the mapping of the input set $\mathcal{B}$ through the neural network intersect with the output set $\mathcal{S}$? In most settings, the input set is taken to be a *fixed* hyperrectangle, norm ball or polytope in the input space, while the output set is a polytope or union of polytopes.

5.4 Connections between Reachability and Neural Network Verification

In Ch. 2 we introduced the topic of reachability theory for dynamical systems. Reachability is also intimately tied to verification. If we view $\mathcal{B}$ and $\mathcal{S}$, as the constraint set $\mathcal{K}$ and failure set $\mathcal{F}$ introduced in Sec. 2.2 respectively, (5.2) can be viewed as a discrete time reachability problem with time horizon $T = 1$, where the dynamics evolve autonomously as follows:

$$x_{t+1} = f_\theta(x_t). \tag{5.3}$$

Let us define the avoid set as

$$\mathcal{A} := \{x_t \in \mathbb{R}^n \mid x_t \in \mathcal{K} \ \wedge \ f_\theta(x_t) \notin \mathcal{F}\} \ , \tag{5.4}$$

which is the set of states starting at $\mathcal{K}$ such that in one step they do not enter into $\mathcal{F}$. We would like to know whether $\mathcal{A} = \mathcal{K}$, that is, whether all states in the constraint set remain "safe". Note that f_θ is a piece-wise affine function as per Sec. 5.2, which means that the domain of f_θ is partitioned into convex polytopic regions $\mathcal{P}_i$:

$$x_{t+1} = f_\theta(x_t) = F_i x_t + f_i \quad \text{if } A_i x_t \leq b_i \ , \tag{5.5}$$

where $F_i, A_i \in \mathbb{R}^{n_1 \times n_1}$, $f_i, b_i \in \mathbb{R}^{n_1}$, and the index i enumerates each polytopic region. In light of this, solving the reachability (or verification) problem can be addressed by (a) subdividing the constraint set $\mathcal{K}$ ($\mathcal{B}$) into its polytopic subcomponents[2], (b) propagating each polytopic subcomponent with its corresponding affine transform (5.5) and (c) checking that each mapped subcomponent does not intersect with $\mathcal{F}$ ($\mathcal{S}$) . Checking for intersection is a convex problem which can be efficiently solved. Note that the same geometrical reasoning also holds when $y = f_\theta(x)$, and x and y do not belong in the same vector space. Figure 5.1 shows what the exact computation of the reachable set looks like.

Unfortunately, even for single layer ReLU networks, this approach cannot scale for large input spaces. The underlying reason being the polytopic subdivision of the domain of f_θ, which is caused by the ReLU nonlinearity $\phi(z) = \max\{z, \mathbf{0}\}$ which can be equivalently expressed as a matrix vector product

$$\phi(z) = Dz \ , \tag{5.6}$$

where $D_{ij} = 1$ if $(z_i \geq 0 \ \wedge \ i = j)$ is true, and 0 otherwise. If we let $x_{t+1} = \phi(x_t)$ and pick $\mathcal{B} = \mathcal{K} = \{x_t \in \mathbb{R}^{n_1} \mid ||x_t||_\infty \leq \gamma\}$ for any $\gamma > 0$, the number of polytopic subcomponents of $\mathcal{K}$

[2]Here we assume that the constraint set is convex.

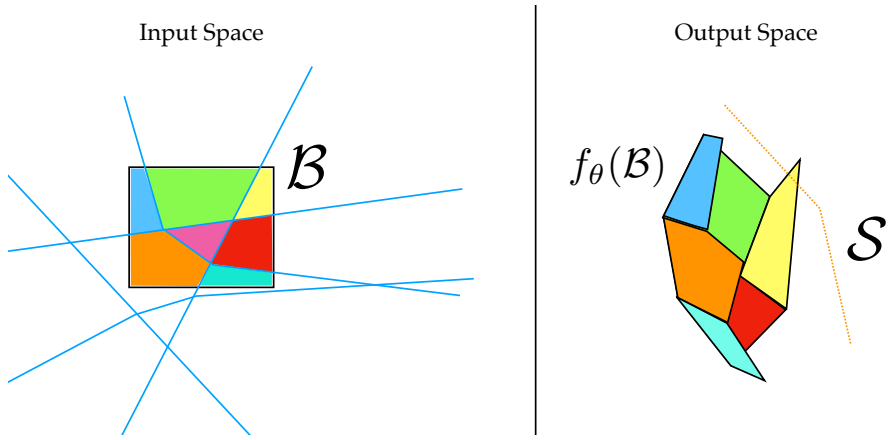


Figure 5.1: Exact computation of the reachable set of the ReLU network. Different colors correspond to different polytopic regions and their corresponding mapping into the output space. Each polytopic region is governed by a different affine transform.

that need to be propagated will be 2^{n_1}. In other words, *the number of polytopic regions grows at least exponentially* with input dimension, which hinders the scalability of this type of approach. Furthermore, since the number of regions also depends on the number of layers of the neural network, this problem can also arise in small dimensional input spaces.

In the same spirit as approximate reachability for non-linear systems, which foregoes exactness of the reachable sets with over- or under-approximations in the form of polytopes or ellipsoids, in the next chapter we present over-approximation methodologies for neural network verification with better scaling properties.

5.5 Related Work

Starting with the works from [40, 43], the authors use Satisfiability Modulo Theory (SMT) solvers to answer Equation (5.2) for ReLU networks and, more generally, networks containing piece-wise linear activations. In their approaches, an answer is reached by leveraging the finite set of possible activations induced by the network's non-linearities. A similar reasoning is found in [56, 89] using Mixed Integer Linear Programming (MILP) solvers. Other approaches inspired by reachability include the work from [97], where the structure of the domain induced by the ReLU non-linearities is exploited to compute the exact set of possible outputs in a similar way to the approach described in Sec. 5.4. In contrast, in [96] the set of possible outputs is approximated by gridding the input set.

Of particular interest for the next chapter are the works of [27] and [47]. In [27], they introduce a novel convex relaxation of the ReLU non-linearity in order to render the problem easier to solve for the SMT solver. In [47], the authors used this relaxation and duality to compute rapid over-approximations of the set of possible outputs for the network. In [78] they expand on this duality approach. A new interesting direction which does not rely on SMT or MILP solvers was presented

by [92]. In this work, a divide and conquer approach was used to repeatedly partition the input set into smaller sub-domains and check the property individually for each partition. Other works which exploit similar over-approximation techniques can be found in [14, 57, 99, 100].

In the next chapter, we build upon the input-splitting technique from [92]. We experimentally show that splits based on input-output gradient metrics are generally inefficient. We provide a new methodology that substantially reduces the number of splits and the runtime required to verify a network for a given input set.

5.6 Chapter Summary

In this chapter we have provided an overview of feedforward ReLU networks and introduced the verification problem. When then discussed one approach for computing the exact mapping of $\mathcal{B}$ through the network, and saw that it also suffers from the curse of dimensionality. Finally, we presented some of the related work in the field of neural network verification.

Chapter 6

Shadow Price Verification

This chapter is based on the paper "Fast Neural Network Verification via Shadow Prices" [80], written in collaboration with Roberto Calandra, Dusan Stipanovic, and Claire Tomlin.

In Ch. 5 we introduced the verification problem for ReLU networks and saw that computing the *exact* mapping of the input set $\mathcal{B}$ through the network in order to verify its intersection with the output set $\mathcal{S}$ was too computationally expensive. In this chapter we first present an approach which *over-approximates* the image of $\mathcal{B}$ through the ReLU network (which we denote as $f_\theta(\mathcal{B})$). Then, using this over-approximation technique we present a splitting heuristic and an iterative algorithm which are able to solve the verification problem efficiently. Finally, we introduce a variant of the algorithm which is substantially faster.

6.1 Over-Approximating the Image

In general, the image of input set $\mathcal{B}$ through a feedforward neural network is not convex (see Fig. 5.1). The non-convexity arises from the non-linearities introduced by the ReLU activation function $\phi(z) := \max\{z, \mathbf{0}\}$ at every layer. In order to over-approximate the image of $\mathcal{B}$, $f_\theta(\mathcal{B})$, it is convenient to rewrite (5.1) as a recursive set definition:

$$\hat{\mathcal{Z}}_{i+1} := \{W_i z_i + b_i \mid z_i \in \text{proj}_{n_i}(\mathcal{Z}_i)\} \qquad \text{for } i = 1, ..., K-1 \tag{6.1}$$

$$\mathcal{Z}_j := \{[\hat{z}_j, z_j]^T \in \mathbb{R}^{2n_j} \mid z_j = \phi(\hat{z}_j), \hat{z}_j \in \hat{\mathcal{Z}}_j\} \quad \text{for } j = 2, ..., K-1 \tag{6.2}$$

$$\mathcal{Z}_1 := \{[\mathbf{0}, z'] \in \mathbb{R}^{2n_1} \mid z' \in \mathcal{B}\}, \tag{6.3}$$

where $\text{proj}_n(\cdot)$ is the projection operator of its argument onto its last n dimensions. For an appropriate selection of a predicate $g(\cdot, \cdot)$ (discussed in the next section), we can instead replace (6.2) with

$$\tilde{\mathcal{Z}}_j := \{[z_j, \hat{z}_j]^T \in \mathbb{R}^{2n_j} \mid g(z_j, \hat{z}_j), \hat{z}_j \in \hat{\mathcal{Z}}_j\} \quad \text{for } j = 2, ..., K-1, \tag{6.4}$$

in a way which ensures that we have $\tilde{\mathcal{Z}}_j \supseteq \mathcal{Z}_j$, and consequently, $\hat{\mathcal{Z}}_K \supseteq f_\theta(\mathcal{B})$. That is, if we choose the predicate appropriately, we obtain a convex over-approximation of the image $f_\theta(\mathcal{B})$.

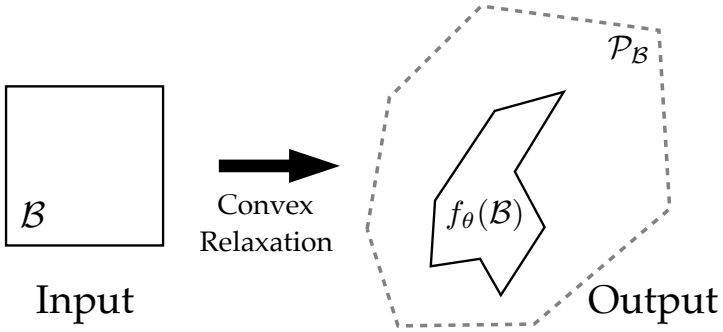


Figure 6.1: Example of a convex over-approximation of the image of $\mathcal{B}$.

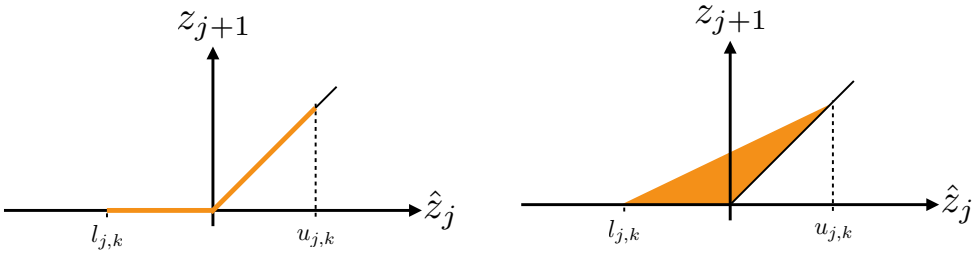


Figure 6.2: *(Left)* Bounded output of a ReLU node. *(Right)* Convex envelope for the triangle relaxation.

Figure 6.1 shows a generic convex over-approximation of the image. Since $\hat{\mathcal{Z}}_K$ is generally a polytope, we will instead denote the over-approximation as $\mathcal{P}_\mathcal{B}$, that is, $\mathcal{P}_\mathcal{B} \equiv \hat{\mathcal{Z}}_K$.

Triangle Relaxation

The first type of predicate we will introduce produces the tightest over-approximation of the image, which is known as the *triangle relaxation* as shown in Fig. 6.2. The predicate for this type of relaxation is the following:

$$g(z_j, \hat{z}_j) := \bigwedge_k (z_{j[k]} \geq 0 \quad \wedge \quad z_{j[k]} \geq \hat{z}_{j[k]} \quad \wedge \quad z_{j[k]} \leq d_{j[k]}(\hat{z}_{j[k]} - l_{j[k]})). \tag{6.5}$$

where $(\cdot)_{[k]}$ denotes the k-th element of a vector, $d_{j[k]} := \frac{u_{j[k]}}{u_{j[k]} - l_{j[k]}}$, where $l_{j[k]}$ and $u_{j[k]}$ are lower and upper bounds for $\hat{z}_{j[k]}$ respectively (i.e. $l_{j[k]} \leq \hat{z}_{j[k]} \leq u_{j[k]}$). To ease the notation, we simply state that any positive lower bound or negative upper bound is set to zero. From (5.1), the recursive

relation given by this predicate can be equivalently written as

$$\begin{aligned} &\hat{z}_{i+1} = W_i z_i + b_i \,, \text{ for } i = 1, ..., K-1 \\ &z_j \succeq 0 \,, \quad \text{for } j = 2, ..., K-1 \\ &z_j \succeq \hat{z}_j \,, \\ &z_j \preceq D_j(\hat{z}_j - l_j) \,. \end{aligned} \tag{6.6}$$

Here, the matrix D_j is diagonal, so that $D_j = diag(d_j)$.

Computing the lower and upper bounds $l_{j[k]}$ and $u_{j[k]}$ for this relaxation can be accomplished in a layer-by-layer fashion by solving linear programs (LPs) from $j = 1$ to $K-2$ of the form

$$\begin{aligned} l_{j[k]} &= \min_{\underline{z}} \ (W_{j,[k,:]}\text{proj}_{n_j}(\underline{z}) + b_{j[k]}) \quad \text{s.t. } \tilde{A}_j \underline{z} \preceq \tilde{b}_j \,, \\ u_{j[k]} &= \max_{\bar{z}} \ (W_{j,[k,:]}\text{proj}_{n_j}(\bar{z}) + b_{j[k]}) \quad \text{s.t. } \tilde{A}_j \bar{z} \preceq \tilde{b}_j \,, \end{aligned} \tag{6.7}$$

where $W_{j[k,:]}$ denotes the k-th row of W_j. The constraints $\tilde{A}_j$ and $\tilde{b}_j$ represent a polytope in a $\tilde{n}_j$-dimensional space, where $\tilde{n}_j = \sum_{l=1}^{j} n_l$. This polytope grows in dimension as a result of taking into consideration more and more upper and lower bounds from earlier layers.

The constraint set given by $\tilde{A}_1$ and $\tilde{b}_1$ is provided as part of the verification problem. Together, they represent the input set $\mathcal{B} := \{x \mid \tilde{A}_1 x \leq \tilde{b}_1\}$ which we assume to be a box in $\mathbb{R}^{n_1}$. The expressions for $\tilde{A}_j$ and $\tilde{b}_j$ for the j-th ($j \geq 2$) layer are given by the following recursion,

$$\begin{aligned} &\tilde{A}_j = \begin{bmatrix} \tilde{A}_{j-1} & 0 \\ O_{j-2} & R_{j-1} \end{bmatrix}, R_{j-1} = \begin{bmatrix} 0 & -I \\ W_{j-1} & -I \\ -D_{j-1}W_{j-1} & I \end{bmatrix}, \\ &\tilde{b}_j = \begin{bmatrix} \tilde{b}_{j-1} \\ r_{j-1} \end{bmatrix}, r_{j-1}^T = [0^T - b_{j-1}^T \ \ (D_{j-1}(b_{j-1} - l_{j-1}))^T] \end{aligned} \tag{6.8}$$

where O_{j-2} is a matrix of zeros whose number of columns is equal to $\sum_{k=1}^{j-2} n_k$. Note how the dimensionality of the constraints given by $\tilde{A}_j$ and $\tilde{b}_j$ increases as we try to compute upper and lower bounds for deeper layers. These expressions can be derived from the inequalities provided in (6.6) in conjunction with (6.7). For clarity, we provide the expressions for $\tilde{A}_{2:3}$ and $\tilde{b}_{2:3}$

$$\tilde{A}_2 = \begin{bmatrix} \tilde{A}_1 & 0 \\ 0 & -I \\ W_1 & -I \\ -D_1 W_1 & I \end{bmatrix}, \tilde{b}_2 = \begin{bmatrix} \tilde{b}_1 \\ 0 \\ -b_1 \\ D_1(b_1 - l_1) \end{bmatrix},$$
$$\tilde{A}_3 = \begin{bmatrix} \tilde{A}_1 & 0 & 0 \\ 0 & -I & 0 \\ W_1 & -I & 0 \\ -D_1 W_1 & I & 0 \\ 0 & 0 & -I \\ 0 & W_2 & -I \\ 0 & -D_2 W_2 & I \end{bmatrix}, \tilde{b}_2 = \begin{bmatrix} \tilde{b}_1 \\ 0 \\ -b_1 \\ D_1(b_1 - l_1) \\ 0 \\ -b_2 \\ D_2(b_2 - l_2) \end{bmatrix}. \tag{6.9}$$

Note that to generate $\tilde{A}_3$ and $\tilde{b}_3$, first we have to solve a series of LPs using the constraints given by $\tilde{A}_2$ and $\tilde{b}_2$. This approach builds an increasingly large dimensional polytope whose projection onto the output space over-approximates the image of $f_\theta(\mathcal{B})$. Namely,

$$f_\theta(\mathcal{B}) \subseteq \mathcal{P}_\mathcal{B} := \{W_{K-1}\text{proj}_{n_{K-1}}(z) + b_{K-1} \mid \tilde{A}_{K-2} z \leq \tilde{b}_{K-2}\}. \tag{6.10}$$

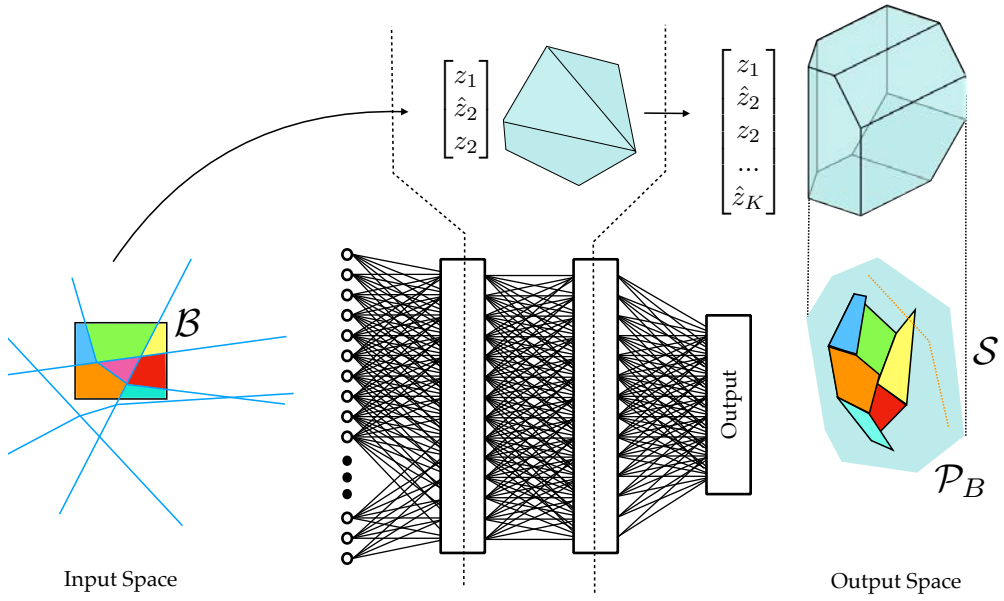


Figure 6.3: The growth in the number of constraints across the layers of a neural network encodes a high-dimensional polytope whose projection over-approximates the image.

In Fig. 6.3 we provide a depiction of how the growing number of constraints represents a high-dimensional polytope whose projection over-approximates the image.

6.2 Overview of Verification via Input-Splits

One of the useful features of this bounded convex over-approximation is that it provides sufficient conditions to check for intersection with the output set $\mathcal{S}$. From Fig. 6.1, if the intersection of the over-approximation with $\mathcal{S}$ is empty, then so is the intersection of the image and $\mathcal{S}$:

$$\mathcal{P}_{\mathcal{B}} \cap \mathcal{S} = \emptyset \implies f_\theta(\mathcal{B}) \cap \mathcal{S} = \emptyset \tag{6.11}$$

On the other hand, the inverse of this statement is generally *not true*. However, if the intersection is non-empty, we can subdivide the set $\mathcal{B}$ into smaller partitions and use (6.11) on each partition separately. The next section describes this idea in detail.

Recursively Splitting Sets

In Sec. 6.1 we explained how to build a convex over-approximation of the image $f_\theta(\mathcal{B})$ in a layer-by-layer basis. Note, that for any split of the input set into two subsets $\mathcal{B}_1$ and $\mathcal{B}_2$ such that $\mathcal{B}_1 \cup \mathcal{B}_2 = \mathcal{B}$, the associated over-approximations $\mathcal{P}_{\mathcal{B}_1}$ and $\mathcal{P}_{\mathcal{B}_2}$ will have that

$$\mathcal{P}_{\mathcal{B}_1} \cup \mathcal{P}_{\mathcal{B}_2} \subseteq \mathcal{P}_{\mathcal{B}}\,. \tag{6.12}$$

This follows from the fact that splitting the input set reduces all the feasible regions for the LPs in (6.7), which results in *greater lower bounds* or *smaller upper bounds* in the computation of $\mathcal{P}_{\mathcal{B}_1}$ and $\mathcal{P}_{\mathcal{B}_2}$.

Splitting the input set $\mathcal{B}$ is particularly useful for verification since it breaks the problem into two sub-problems. Specifically, and without loss of generality, one of three things may happen:

1. The intersection is empty for both $\mathcal{P}_{\mathcal{B}_1}$ and $\mathcal{P}_{\mathcal{B}_2}$, and thus the image $f_\theta(\mathcal{B})$ does not intersect with $\mathcal{S}$.

2. The intersection is non-empty for $\mathcal{P}_{\mathcal{B}_1}$ but empty for $\mathcal{P}_{\mathcal{B}_2}$, in which case $\mathcal{B}_2$ can be discarded from the verification problem, but $\mathcal{B}_1$ must be kept for further analysis.

3. Finally, both $\mathcal{P}_{\mathcal{B}_1}$ and $\mathcal{P}_{\mathcal{B}_2}$ intersect with $\mathcal{S}$. $\mathcal{B}_1$ and $\mathcal{B}_2$ must be kept for further analysis.

Given these outcomes, a natural algorithm arises for the verification of the property of interest; starting with $\mathcal{B}$, we compute the convex over-approximation $\mathcal{P}_{\mathcal{B}}$. If the intersection with $\mathcal{S}$ is empty for $\mathcal{P}_{\mathcal{B}}$, the intersection is empty for the image as well and we are done. Otherwise, we can split $\mathcal{B}$ in two halves $\mathcal{B}_1$ and $\mathcal{B}_2$, and compute $\mathcal{P}_{\mathcal{B}_1}$ and $\mathcal{P}_{\mathcal{B}_2}$. Given the list of possible outcomes, the algorithm will either: end with the property being false, be able to discard one of the halves, or, in the worst case, keep both $\mathcal{B}_1$ and $\mathcal{B}_2$ for further analysis. This procedure induces a growing *binary tree* whose nodes represent smaller and smaller bisections of the input set $\mathcal{B}$.

Algorithm 2 provides the aforementioned procedure for verifying whether $f_\theta(\mathcal{B})$ intersects with $\mathcal{S}$, which as previously stated we assume to be a union of a finite number convex sets. This assumption is required so that in Line 3 the intersection check can be done efficiently via convex

Algorithm 2: Recursive Splitting (Depth First Search)

1 **Verification**$(\mathcal{S}, \mathcal{B}, \theta)$
2 $\quad \mathcal{P}_{\mathcal{B}} \leftarrow \text{ConvexOverApprox}(\mathcal{B}, \theta)$
3 $\quad$ **If** $\mathcal{P}_{\mathcal{B}} \cap \mathcal{S} = \emptyset$ **then** ▷Intersection is empty.
4 $\quad\quad$ **Return** False
5 $\quad$ **Else** ▷Intersection is not empty.
6 $\quad\quad$ **If** $\text{IsExact}(\mathcal{P}_{\mathcal{B}})$ **then** ▷Check over-approximation is tight.
7 $\quad\quad\quad$ **Return** True
8 $\quad\quad \mathcal{B}_1, \mathcal{B}_2 \leftarrow \text{Split}(\mathcal{B})$ ▷Bisect the set.
9 $\quad\quad$ **Return** $\text{Verification}(\mathcal{S}, \mathcal{B}_1, \theta) \vee \text{Verification}(\mathcal{S}, \mathcal{B}_2, \theta)$

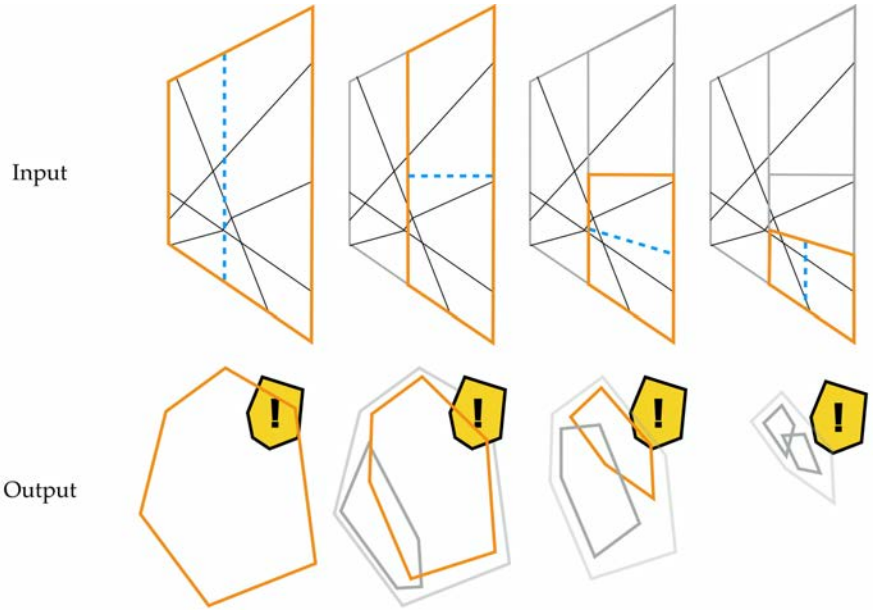


Figure 6.4: Visual progression (from left to right) of Alg. 2. The yellow set with the exclamation symbol represents the set $\mathcal{S}$. In this instance, the intersection of the image with output set is empty. In the bottom row, over-approximations with empty intersections are grayed out.

optimization packages. When the intersection is non-empty, Line 6 checks whether the over-approximation is tight/exact or not. If it is exact, it must be true that $f_\theta(\mathcal{B})$ intersects with $\mathcal{S}$. If it is not, in line 8 we split the set into two halves and attempt to solve the verification problem for each half separately. The next sections describe the auxiliary functions "IsExact" and "Split" in lines 6 and 8. Figure 6.4 shows pictorially what the progression of the algorithm looks like.

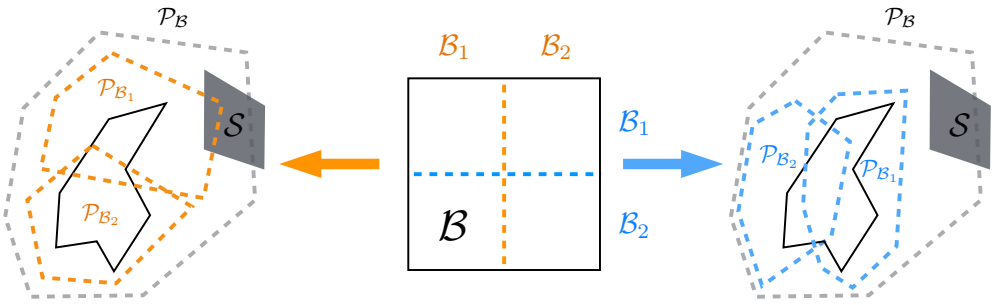


Figure 6.5: The choice of axis along which the set $\mathcal{B}$ is split may affect the verification time. In this example, a vertical split (orange) results in two over-approximations, one of which still intersects with $\mathcal{S}$, whereas the horizontal split (blue) results in tighter over-approximations.

Over-approximation and Image Equivalence

In the previous section we hinted at the fact that some instances of $\mathcal{P}_\mathcal{B}$ can be tight or exact, that is $\mathcal{P}_\mathcal{B} \equiv f_\theta(\mathcal{B})$. From Sec. 5.4, we know that a ReLU network sub-divides the domain into convex polytopes $\mathcal{P}_i$, and that within each polytope the input-output relation is affine, i.e. it is a piece-wise affine function. Then the following must hold:

$$\begin{aligned} &1.)\ \mathcal{B} \subseteq \mathcal{P}_i \iff u_{j[k]} \leq 0 \ \text{ or } \ 0 \leq l_{j[k]}\ \forall j, k \\ &2.)\ \mathcal{B} \subseteq \mathcal{P}_i \implies \mathcal{P}_\mathcal{B} \equiv f_\theta(\mathcal{B}) \end{aligned} \tag{6.13}$$

The first line in (6.13) follows from the fact that the boundaries of the polytopes $\mathcal{P}_i$ arise from the discontinuity of the non-linear activations $\max\{z, 0\}$. The second line follows from the first one: if $l_{j,k} \leq u_{j,k} \leq 0$ or $0 \leq l_{j,k} \leq u_{j,k}$ for all j and k, then the relaxations of the ReLU activations will be exact and $f_\theta(\mathcal{B})$ is just an affine transformation of $\mathcal{B}$.

Splitting Criterion

From Algorithm 2, a natural question arises: what is considered to be a "good" split of the input set? While simply stated, this question is far from trivial. Picking appropriate splits has important consequences regarding the time and memory efficiency of input-splitting verification algorithms. In Fig. 6.5, an example is provided showcasing this phenomenon. The horizontal split results in an over-approximation that guarantees that the image of $\mathcal{B}$ does not intersect with $\mathcal{S}$. In contrast, the vertical split results in an over-approximation that still intersects $\mathcal{S}$.

To the best of our knowledge, current input-splitting methods [92] use gradient information between inputs and outputs to decide which axis to split. We will denote this heuristic as the *input-output gradient* heuristic. These methods leverage the structure of the Jacobian of the ReLU network to compute bounds on the gradient between inputs and outputs. In particular, note that for

any point in the domain, the Jacobian for a ReLU network can always be written as

$$J = W_{K-1} \prod_{i=1}^{K-2} \Sigma_i W_i \tag{6.14}$$

for $\Sigma_i = diag(v_i)$ and the Boolean vector $v_i \in \{0,1\}^{n_{i+1}}$. Starting from $i = K-1$ going backwards to $i = 1$, these methods select appropriate values of v_i to compute upper bounds for $||\frac{df_\theta(x)}{dx_k}||_\infty \leq U_k$ for $k = 1, ..., n_1$. Using the side lengths Δx_k of the box, $\mathcal{B}$ is split in half across the axis with greatest smear value $s_k = U_k \Delta x_k$. Geometrically, this mechanism tries to reduce in half the box along the axis that causes most "stretching" for any of the outputs. This reasoning, however, can be counterproductive when considering which splits to make. In particular, the upper bound might be very loose in practice, and even in cases where U_k can be achieved, it may be the case that the polytopic region that achieves this upper bound is very small.

Fundamentally, the over-approximation is caused by the convex relaxations of the ReLU non-linearities, therefore, we posit that an effective splitting procedure should leverage the information of the relaxed nodes (i.e. nodes for which the convex relaxation is not exact) rather than using bounds on the gradients between input and outputs. In the next sections we investigate how to estimate changes in the upper and lower bounds of any given node given a specific split in $\mathcal{B}$.

6.3 Shadow Prices and Bound Rates

We now study the sensitivity of the lower and upper bounds for a relaxed ReLU node with respect to changes in the shape of the input box. To that end, we introduce some useful properties of linear programs and relate them to our problem at hand.

Measuring Constraint Sensitivity

For a linear program with non-empty feasible region the following holds

$$\underline{p}^* = \min_{s.t.\ A\underline{z} \preceq b} w\underline{z} \quad = \quad w\underline{z}^* + (A\underline{z}^* - b)^T \underline{\lambda}^* \tag{6.15}$$

$$\bar{p}^* = \max_{s.t.\ A\bar{z} \preceq b} w\bar{z} \quad = \quad w\bar{z}^* + (b - A\bar{z}^*)^T \bar{\lambda}^* \,, \tag{6.16}$$

where $\bar{z}^*$ and $\underline{z}^*$ correspond to the primal's maximizer/minimizer and $\bar{\lambda}^* \succeq 0$ and $\underline{\lambda}^* \succeq 0$ correspond to the dual's minimizer/maximizer. This result follows from strong duality and complementary slackness [12]. A useful feature of the right-hand sides of (6.15) is that they link how small perturbations of the constraint parameters (A and b) affect the optimal values $\bar{p}^*$ and $\underline{p}^*$. In the field of economics, the rate of increase/decrease of the optimal value with respect to a certain constraint is known as the *shadow price*. In some instances the shadow prices are the optimal dual variables.

From (6.15), we can readily study how small changes in the size of the input box affect the upper and lower bounds of all nodes in the first layer of the network. In particular, it follows that for

the k-th node in the *first layer* and the i-th bias of our constraint set,

$$\frac{dl_{1[k]}}{d\tilde{b}_{1[i]}} = -\underline{\lambda}^*_{k[i]} \qquad \frac{du_{1[k]}}{d\tilde{b}_{1[i]}} = \bar{\lambda}^*_{k[i]}. \tag{6.17}$$

This result is expected, since the growth of bias terms results in a bigger box and, thus, in smaller lower bounds and bigger upper bounds. A nice feature of most modern LP solvers is that they provide the associated optimal dual variables when solving the primal problem. These rates can therefore be computed without any noticeable computational overhead when generating the convex over-approximations.

Forward Computation of the Bound Rates

With the definitions for $\tilde{A}_j$ and $\tilde{b}_j$ available from the end of Sec. 6.1 and (6.15), we can proceed to compute the rates of the upper and lower bounds with respect to the bias terms of our input box. Denoting $(\underline{\lambda}^*_{j,k}, \bar{\lambda}^*_{j,k})$ and $(\underline{z}^*_{j,k}, \bar{z}^*_{j,k})$ as the set of primal and dual optimal variables for the maximization and minimization problems from (6.7), we have that

$$\begin{aligned}
\frac{dl_{j[k]}}{d\tilde{b}_{1[i]}} &= \frac{d}{d\tilde{b}_{1[i]}}\left(\tilde{b}_j^T \underline{\lambda}^*_{j[k]}\right) - \frac{d}{d\tilde{b}_{1[i]}}\left((\tilde{A}_j \underline{z}^*_{j[k]})^T \underline{\lambda}^*_{j[k]}\right) \\
\frac{du_{j[k]}}{d\tilde{b}_{1[i]}} &= -\frac{d}{d\tilde{b}_{1[i]}}\left(\tilde{b}_j^T \bar{\lambda}^*_{j[k]}\right) + \frac{d}{d\tilde{b}_{1[i]}}\left((\tilde{A}_j \bar{z}^*_{j[k]})^T \bar{\lambda}^*_{j[k]}\right).
\end{aligned} \tag{6.18}$$

Before proceeding to drive the analytic expression of (6.18), it is worth noting that changes in the bias terms of the input box $\tilde{b}_{1,:}$ only affect (through the computation of the upper and lower bounds) a subset of the entries of $\tilde{A}_j$ and $\tilde{b}_j$. In particular, given $R_{1:j-1}$ and $r_{1:j-1}$ defined in (6.8), only entries containing dependencies with $D_{1:j-1}$ and $l_{1:j-1}$ will contribute to the gradient expression in (6.18). Hence, focusing our attention to the first term, for any vector λ

$$\begin{aligned}
\frac{d}{d\tilde{b}_{1[i]}}(\tilde{b}_j^T\lambda) &= \frac{d}{d\tilde{b}_{1[i]}}\left(\tilde{b}_1^T\lambda_0 + \sum_{l=1}^{j-1} r_l^T \lambda_l\right) \\
&= \lambda_{0[i]} + \sum_{l=1}^{j-1} \frac{d}{d\tilde{b}_{1[i]}}(D_l(b_l - l_l))^T \lambda_{\hat{l}} \\
&= \lambda_{0[i]} + \sum_{l=1}^{j-1}\sum_{t=0}^{n_l} \frac{d}{d\tilde{b}_{1[i]}} \frac{u_{l[t]}b_{l[t]} - u_{l[t]}l_{l[t]}}{u_{l[t]} - l_{l[t]}} \lambda_{\hat{l}[t]} \\
&= \lambda_{0[i]} + \sum_{l=1}^{j-1}\sum_{t=0}^{n_l} \left(c_1 \frac{du_{l[t]}}{d\tilde{b}_{1[i]}} + c_2 \frac{dl_{l[t]}}{d\tilde{b}_{1[i]}}\right) \lambda_{\hat{l}[t]}
\end{aligned} \tag{6.19}$$

where $c_1 = \frac{l_{l[t]}(l_{l[t]} - b_{l[t]})}{(u_{l[t]} - l_{l[t]})^2}$ and $c_2 = \frac{u_{l[t]}(b_{l[t]} - u_{l[t]})}{(u_{l[t]} - l_{l[t]})^2}$ and $\lambda_{\hat{l}}, \lambda_l$ correspond to a subset of the entries in λ_l and λ respectively. The term $b_{l[t]}$ denotes entry t of the bias term in layer l. From (6.19) it is clear that the rates of the upper and lower bounds for layer j depend on all the rates from previous layers.

Algorithm 3: Bound Estimation-based Splitting

Split$(\mathcal{B}, (l,u), (\underline{z}^*, \underline{\lambda}^*), (\bar{z}^*, \bar{\lambda}^*))$
  For $j = 1, ..., K-1$ **do**
    For $k = 1, ..., n_j$ **do**
      $\frac{dl_{j[k]}}{d\tilde{b}_{1[i]}} \leftarrow$ LowerBoundRates$(\mathcal{B}, (l,u), (\underline{z}^*, \underline{\lambda}^*), (\bar{z}^*, \bar{\lambda}^*))$ (as per Sec. 6.3)
      $\frac{du_{j[k]}}{d\tilde{b}_{1[i]}} \leftarrow$ UpperBoundRates$(\mathcal{B}, (l,u), (\underline{z}^*, \underline{\lambda}^*), (\bar{z}^*, \bar{\lambda}^*))$ (as per Sec. 6.3)
  $i = 1, c = L(1)$
  For $k = 2, ..., n_1$ **do**
    If $L(k) < c$ **then**
      $c \leftarrow L(k)$
      $i \leftarrow k$
  $\mathcal{B}_1 \leftarrow \mathcal{B}_i$ ▷Split in half through i-th axis.
  $\mathcal{B}_2 \leftarrow \mathcal{B} \backslash \mathcal{B}_i$
  Return $\mathcal{B}_1, \mathcal{B}_2$

The rates for the second term can be derived in a similar manner. For any vector z and λ,

$$\begin{aligned}
\frac{d}{d\tilde{b}_{1[i]}}(\tilde{A}_j z)^T \lambda &= \frac{d}{d\tilde{b}_{1[i]}}((\tilde{A}_1 z_0)^T \lambda_0 + \sum_{l=1}^{j-1}(R_l \begin{bmatrix} z_{l-1} \\ z_l \end{bmatrix})^T \lambda_l) \\
&= \sum_{l=1}^{j-1} \frac{d}{d\tilde{b}_{1[i]}} \left(-D_l W_l z_{l-1}\right)^T \lambda_{\hat{l}} \\
&= -\sum_{l=1}^{j-1} \sum_{t=0}^{n_l} \frac{d}{d\tilde{b}_{1[i]}} \frac{(W_{l[t,:]} z_{l-1}) u_{l[t]}}{u_{l[t]} - l_{l[t]}} \lambda_{\hat{l}[t]} \\
&= \sum_{l=1}^{j-1} \sum_{t=0}^{n_l} \left(\hat{c}_1 \frac{dl_{l[t]}}{d\tilde{b}_{1[i]}} - \hat{c}_2 \frac{du_{l[t]}}{d\tilde{b}_{1[i]}} \right) \lambda_{\hat{l}[t]}
\end{aligned} \tag{6.20}$$

where $\hat{c}_1 = \frac{u_{l[t]}(w_{l[t]} z_{l-1})}{(u_{l[t]} - l_{l[t]})^2}$ and $\hat{c}_2 = \frac{l_{l[t]}(w_{l[t]} z_{l-1})}{(u_{l[t]} - l_{l[t]})^2}$.

Using (6.19) together with (6.20) we can compute expressions for (6.18) in a forward manner using the optimal primal and dual variables. That is, we have the rate of change of *all* the upper bounds and lower bounds of each node in the network with respect to the bias terms of our input box $\mathcal{B}$.

6.4 Bound Estimation and Splitting

With the information provided in Sec. 6.3, we have means to estimate how the lower and upper bounds of our convex relaxation change as a function of the biases of the input set. In particular,

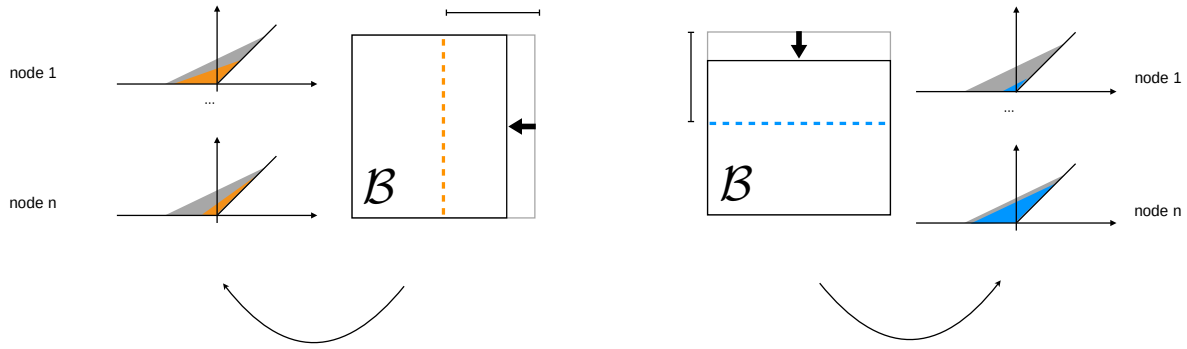


Figure 6.6: Choosing split based on predicted upper and lower bounds for each node.

splitting the box $\mathcal{B}$ in half can be viewed as *translating one of its facets to the center*. Since the translation of any facet is achieved by reducing the associated bias term by a certain amount $\Delta\tilde{b}_{1,i}$, the estimated new lower and upper bounds for the resulting set $\mathcal{B}_i \subset \mathcal{B}$ will be

$$\begin{aligned} l_{j[k]}^{\mathcal{B}_i} &\approx l_{j[k]} + \frac{dl_{j[k]}}{db_{1[i]}} \Delta\tilde{b}_{1[i]} \\ u_{j[k]}^{\mathcal{B}_i} &\approx u_{j[k]} + \frac{du_{j[k]}}{db_{1[i]}} \Delta\tilde{b}_{1[i]} \,. \end{aligned} \tag{6.21}$$

Using these estimates, we propose the following metric for determining which axis to split along:

$$L(i) = -\sum_{j=1}^{K-2} \sum_{k=1}^{n_j} \max\{0, u_{j[k]}^{\mathcal{B}_i}\} \min\{0, l_{j[k]}^{\mathcal{B}_i}\}. \tag{6.22}$$

Whichever axis minimizes $L(i)$ is chosen for splitting. The max/min terms in the sum ensure that upper and lower bounds that start close to zero have less contribution in the overall splitting decision. Note how the cost metric is only zero whenever all relaxations are tight. We summarize this splitting procedure in Algorithm 3. Figure 6.6 depicts what the splitting decision looks like based on the predicted upper and lower bounds.

6.5 Limitations and Convex Relaxation Alternatives

Parallel Relaxation

In Sec. 6.1 we presented the triangle relaxation. As we saw, this type of relaxation yields the tightest over-approximation possible. However, in order to generate it, it is necessary to solve LPs for *every single* node in the neural network. While LP solvers are generally fast, neural networks can have thousands of nodes. This limits the types of networks whose over-approximation can be computed

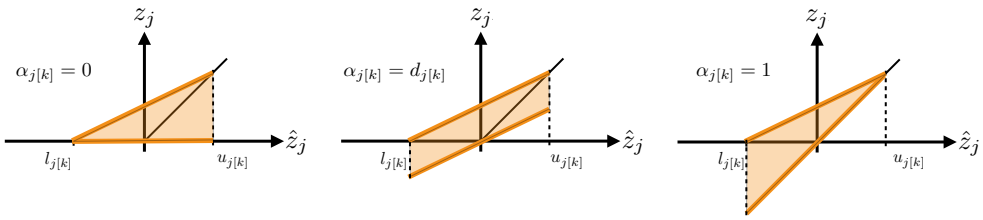


Figure 6.7: Alternate convex relaxations of the ReLU non-linearity.

using the triangle relaxation. Consider instead a relaxation like the one shown in Fig. 6.7, whose predicate is given by

$$g(z_j, \hat{z}_j) := \bigwedge_k \big(\; z_{j[k]} \geq \alpha_{j[k]} \hat{z}_{j[k]} \quad \wedge \quad z_{j[k]} \leq d_{j[k]} (\hat{z}_{j[k]} - l_{j[k]}) \; \big) \; . \tag{6.23}$$

Unlike the triangle relaxation which is specified with two lower bounds and a single upper bound, this type of relaxation is given by a single upper bound and a single lower bound. The term $\alpha_{j[k]} \in [0, 1]$ determines the slope of the lower bound. Again from (5.1), the recursive relation given by this predicate can be written as

$$\begin{aligned} &\hat{z}_{i+1} = W_i z_i + b_i \,, \text{ for } i = 1, ..., K-1 \\ &A_j \hat{z}_j \preceq z_j \preceq D_j (\hat{z}_j - l_j), \quad \text{for } j = 2, ..., K-1. \end{aligned} \tag{6.24}$$

The matrices D_j, A_j are diagonal, so $D_j = diag(d_j)$ and $A_j = diag(\alpha_j)$.

Unlike the triangle relaxation, the fact that we use only *one upper bound* and *one lower bound* allows us to compute an over-approximation of $f_\theta(\mathcal{B})$ without using LP solvers, which is much faster in practice, but at the expense of an approximation that is less tight. For a given input set $\mathcal{B}$, using predicate (6.23), it can be shown that for any $\hat{z}_{j+1}, j = 1, ..., K-1$ in (5.1) and $z_1 \in \mathcal{B}$, there exist two affine functions $f^L_{j+1}(z_1)$ and $f^U_{j+1}(z_1)$ such that

$$M^L_j z_1 + h^L_j + t^L_j = f^L_{j+1}(z_1) \;\preceq\; \hat{z}_{j+1} \;\preceq\; f^U_{j+1}(z_1) = M^U_j z_1 + h^U_j + t^U_j, \tag{6.25}$$

with $M^L_j, M^U_j \in \mathbb{R}^{n_j \times n_1}$ and $h^L_j, h^U_j, t^L_j, t^U_j \in \mathbb{R}^{n_j}$. Denote for any real matrix M,

$$\begin{aligned} M\big|_+ &= (M > 0) \odot M \\ M\big|_- &= (M < 0) \odot M \end{aligned}$$

to be the operators that return the positive and negative entries of their argument, where $\odot$ is the Hadamard (element-wise) product. For $c = \{L, U\}$, the terms in (6.25) have the relationship

$$\begin{array}{lll} M^c_j = S^c_{j,1} & h^c_j = b_j + \sum_{k=1}^{j-1} K^c_{j,k} b_k & t^c_j = \sum_{k=1}^{j-1} L^c_{j,k} l_{k+1} \\ S^c_{j,k} = K^c_{j,k} W_k & K^U_{j,k} = (S^U_{j,k+1}\big|_+ D_{k+1} + S^U_{j,k+1}\big|_- A_{k+1}) & L^U_{j,k} = S^U_{j,k+1}\big|_+ D_{k+1} \\ S^c_{j,j} = W_j & K^L_{j,k} = (S^L_{j,k+1}\big|_+ A_{k+1} + S^L_{j,k+1}\big|_- D_{k+1}) & L^L_{j,k} = S^L_{j,k+1}\big|_- D_{k+1}. \end{array} \tag{6.26}$$

To see why relationships (6.25) and (6.26) hold, assume that for some layer we have all preceding lower and upper bounds l_j and u_j, and preceding diagonal matrices D_j and A_j. Starting from $\hat{z}_{j+1} = W_j z_j + b_j$, we can derive an upper bound in terms of z_{j-1} by following these steps:

$$\hat{z}_{j+1} = W_j z_j + b_j \tag{6.27}$$

$$\hat{z}_{j+1} = W_j\Big|_+ z_j + W_j\Big|_- z_j + b_j \tag{6.28}$$

$$A_j\hat{z}_j \preceq z_j \preceq D_j(\hat{z}_j - l_j) \;\rightarrow\; \hat{z}_{j+1} \preceq W_j\Big|_+ D_j(\hat{z}_j - l_j) + W_j\Big|_- A_j\hat{z}_j + b_j \tag{6.29}$$

$$\hat{z}_{j+1} \preceq (W_j\Big|_+ D_j + W_j\Big|_- A_j)\hat{z}_j + b_j - W_j\Big|_+ D_j l_j \tag{6.30}$$

$$\hat{z}_{j+1} \preceq K_{j-1}\hat{z}_j + b_j - L_{j-1}l_j \tag{6.31}$$

$$\hat{z}_j = W_{j-1}z_{j-1} + b_{j-1} \;\rightarrow\; \hat{z}_{j+1} \preceq K_{j-1}W_{j-1}z_{j-1} + K_{j-1}b_{j-1} + b_j - L_{j-1}l_j\ . \tag{6.32}$$

In (6.28) we separate the weight matrix into its positive and negative components. In (6.29) using the convex relaxation from (6.24), we can find an upper bound for $\hat{z}_{j+1}$ by multiplying the upper bound with the positive component of the weight matrix, and the lower bound with the negative component of the weight matrix. In (6.30) and (6.31) we rearrange and relabel some terms. Finally, in (6.32) we substitute $\hat{z}_j$ in terms of z_{j-1}. Noting that the right-hand side of the last inequality follows the same pattern as the right-hand side of (6.27) (i.e. we have $K_{j-1}W_{j-1}$ instead of W_j, and $K_{j-1}b_{j-1} + b_j - L_{j-1}l_j$ instead of b_j), we can repeat this entire procedure until we have $\hat{z}_{j+1}$ bounded in terms of z_1, obtaining (6.25) as a result. Lastly, deriving the lower bound for $\hat{z}_{j+1}$ in terms of z_{j-1} follows the same procedure, with the exception that the terms multiplying $W_j\big|_+$ and $W_j\big|_-$ are reversed.

Computing the bound rates in (6.18) follows the same progression as in Sec. 6.3. However, the rates must be computed using the relations shown in (6.26). For our experiments that we show in the next section we assumed the lower bound's slope to be the same as the upper bound's, that is $\alpha_{j[k]} = d_{j[k]}$ for all j, k. This type of relaxation we will denote as *parallel relaxation*, which corresponds to the middle plot in Fig. 6.7.

6.6 Experiments

In this section, we present the experimental results obtained on a set of benchmark verification tasks. As a baseline, we compare against the input-output gradient-based method discussed at the end of Sec. 6.2. The first half of the experiments are based on the triangle relaxation. The second half are based on the parallel relaxation. In the last section we discuss the impact of the choice of relaxation.

Airborne Collision Avoidance System Verification

The ACAS benchmark verification task comprises a set of ten properties $\phi_{1:10}$ to be checked on a subset of 45 feedforward ReLU networks. All of the neural networks have the same architecture, with 5 inputs, 5 outputs and 6 hidden layers with 50 neurons each. The five inputs represent a

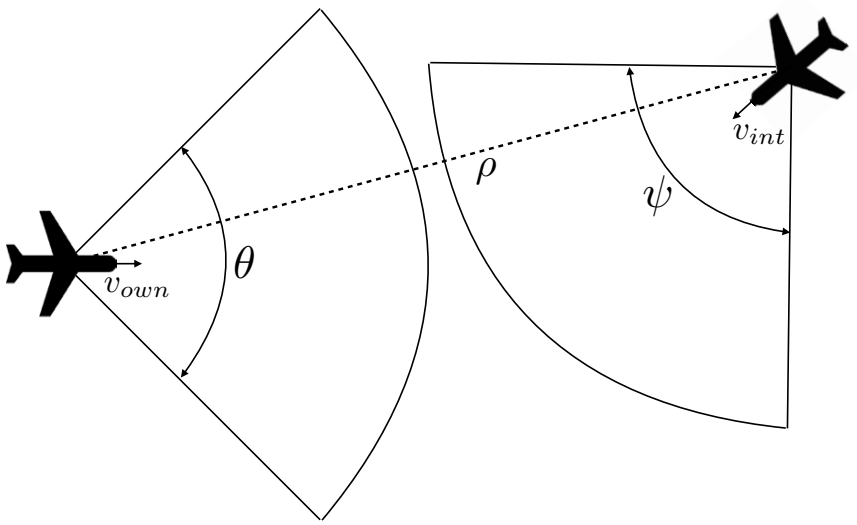


Figure 6.8: Visualization of what an input set $\mathcal{B}$ might look like for the ACAS benchmark. Here the set is two dimensional, and captures a range of possible headings for both the ownship and the intruder.

specific configuration between two aircraft, one which is denoted as the *ownship*, and the other as the *intruder*. The inputs are ρ (distance between aircraft), θ (heading angle of ownship), ψ (heading angle of intruder), v_{own} (speed of ownship) and v_{int} (speed of intruder). The output corresponds to five scalars: COC (Clear of Conflict), *weak left*, *weak right*, *strong left* and *strong right*. The output with the greatest value is the advice action for the ownship. The properties $\phi_{1:10}$ specify a box-shaped subset $\mathcal{B}$ in the input space and the set $\mathcal{S}$ in the output space. A property is said to be *unsatisfied* for a given network if (5.2) is true, otherwise the network *satisfies* the property. For convenience we make it explicit in the following statement,

$$\phi \text{ is unsatisfied } \iff f_\theta(\mathcal{B}) \cap \mathcal{S} = \emptyset. \tag{6.33}$$

Details for each property are given in Appx. A. For further details on this benchmark we direct the reader to [43]. Figure 6.8 depicts an instance of the input set $\mathcal{B}$ for this type of verification problem.

The utility of verification in this setting is clear: there exist certain relative configurations between aircraft which should never (or always) violate certain specifications. In Fig. 6.8 for example, if the range of headings is small it should never be the case that the advisory be *COC*, since both aircraft are headed for collision. The ACAS benchmark defines a set of relative configurations for which the appropriate (or inappropriate) advisory is known *a priori*. Verification then ensures that the network behaves as expected under those known circumstances.

Experimental Details

There are a few technical aspects that need to be clarified before proceeding to the results. In our implementation, we did *not* use any form of parallelism, even though this approach can be readily

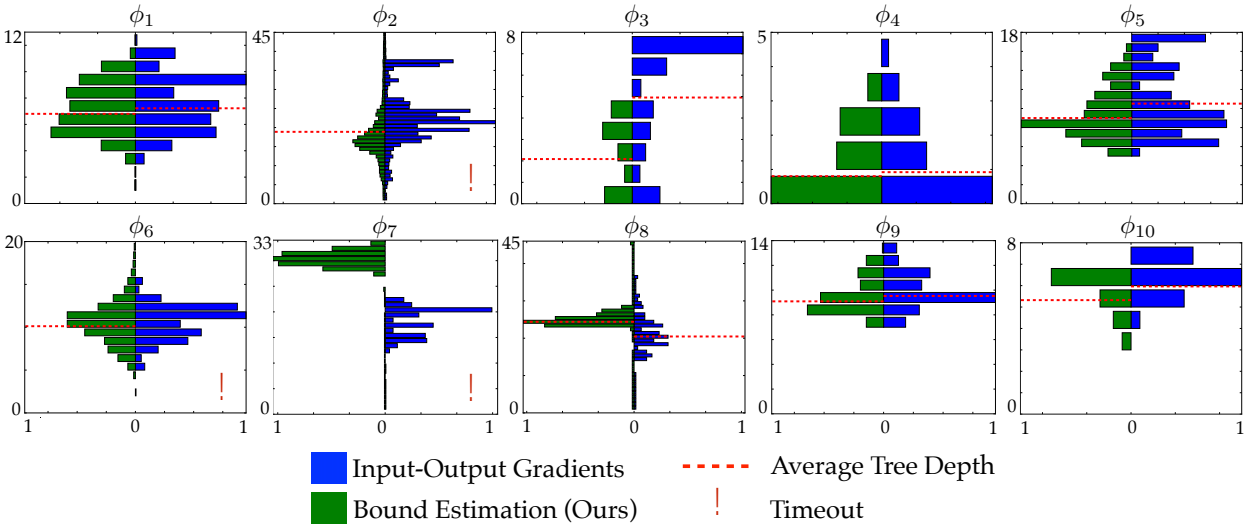


Figure 6.9: Horizontal histograms displaying the number of branches of each length generated by each type of splitting procedure. Each pair of histograms is normalized with the maximum branch length reached for that specific property. For histograms where timeouts occur we do not report the average tree depth, since the associated tree has not finished growing. For property 2 we do report the average tree depth only for networks that could be verified by both IOG and BE procedures.

extended to a parallel implementation similar to [92]. Each individual network to be verified was given a maximum execution time of 3 hours, at which point the verification function halts and a timeout flag is returned. All verification tasks were performed on a 12 core, 64-bit machine with Intel Core i7-5820K CPUs @ 3.30GHz. All code was written in Python using the Gurobi optimization package.

Comparison of Splitting Procedures for the Triangle Relaxation

In this section, we provide a side-by-side comparison between input-output gradient-based splits (IOG) and splits generated by our approach based on shadow prices and bound estimation (BE). In this section we employ the triangle relaxation to compute over-approximations.

Table 6.1 shows the verification results for each of the 10 properties. The first column enumerates the properties and the second column shows the number of networks the property was tested against. The third and fifth columns show the verification results, which can either be unsatisfied, satisfied or timeout, for the IOG and BE-based splits, respectively. Columns four and six show the average depth for the binary trees generated during the verification procedures. When timeouts occurred, we did not report the average depth and standard deviations, as they are misleading metrics representing partially built binary trees, the only exception being property 2[1], for which 30 networks were able

[1]Similar to [92], we omit networks $\{N_{4,2}, N_{5,3}\}$.

ϕ	# NNs	IOG		BE (Ours)	
		U/S/T	Search depth	U/S/T	Search depth
1	45	45/0/0	7.22 ± 2.11	45/0/0	6.79 ± 1.90
2	34	0/30/4	$17.87 \pm 9.00^*$	0/32/2	$14.29 \pm 7.94^*$
3	42	42/0/0	4.96 ± 2.50	42/0/0	2.03 ± 1.49
4	42	42/0/0	0.92 ± 1.13	42/0/0	0.85 ± 1.03
5	1	1/0/0	10.5 ± 3.65	1/0/0	9.25 ± 2.79
6	1	0/0/1	N/A	1/0/0	10.1 ± 2.48
7	1	0/0/1	N/A	0/0/1	N/A
8	1	0/1/0	24.1 ± 13.40	0/1/0	18.33 ± 10.05
9	1	1/0/0	9.53 ± 1.47	1/0/0	9.09 ± 1.47
10	1	1/0/0	5.96 ± 0.80	1/0/0	5.34 ± 0.89

Table 6.1: Verification results (U: unsatisfied. S: satisfied. T: timeout) and search depth (mean $\pm$ standard deviation) for all properties of the ACAS benchmark. These results show that our approach validates the properties as well as, or better than, IOG – even with a reduced search depth. We use * in property 2 to denote that the comparison is *only* for networks where neither IOG nor BE procedures had a timeout.

to be verified by both IOG and BE procedures[2].

The horizontal histograms in Fig. 6.9 depict the number of branches (horizontal axis) of a specific depth (vertical axis) for the binary trees generated by the verification tasks. For each histogram, the distribution on the left (green) corresponds to the distribution generated by BE-based splitting. The distributions on the right (blue) correspond to IOG-based splits. The dashed red line in each plot shows the average depth, which is reported in Table 6.1. In cases where timeouts occurred for either type of procedure we include an exclamation mark.

Table 6.3 shows a few other metrics from the verification tasks. The first two columns are the same as Tbl. 6.1. Columns three and four show the exact number of nodes that were generated during verification of each task. The fifth column shows the ratio of times between BE-based splits and IOG-based splits for a given verification task. Values above 1.0 imply a strict improvement of BE-splits over IOG-splits. In cases where timeouts occurred we include an asterisk symbol.

[2]IOG timeouts: networks $\{N_{3,3}, N_{3,4}, N_{3,8}, N_{4,9}\}$. BE timeouts: networks $\{N_{3,3}, N_{4,9}\}$.

ϕ	# NNs	# Nodes		$t_{\text{IOG}}/t_{\text{BE}}$
		IOG	BE	
1	45	5241	4541	**1.145**
2	34*	799†	553†	**1.146**†
3	43	464	164	**2.770**
4	43	84	82	**1.085**
5	1	491	369	**1.140**
6	1*	>1808*	1210	>**1.466***
7	1*	>1269*	>983*	1.0*
8	1	95	245	0.823
9	1	947	737	**1.264**
10	1	105	63	**1.297**

Table 6.2: Number of nodes for the two split mechanisms (smaller is better) and the ratio of computational time for the two methods. We can see that for 8 out of 10 properties our approach (BE) reduces the number of nodes generated, as well as the computational time. * indicates that some verification did not complete within the allocated time. We use † in property 2 to denote that the comparison is *only* for networks where neither IOG nor BE procedures had a timeout.

Property	(U, S, T)	IOG		BE (Ours)		t_{IOG}/t_{BE}
		avg. depth	t_{IOG} [sec]	avg. depth	t_{BE} [sec]	
ϕ_1	(45,0,0)	10.41 ± 2.73	3781	9.47 ± 2.27	2824	**x1.34**
ϕ_2	(0,33,1)	8.31 ± 2.96	153	5.82 ± 2.5	68	**x2.25**
ϕ_3	(42,0,0)	7.87 ± 2.48	482	5.27 ± 1.87	178	**x2.7**
ϕ_4	(42,0,0)	4.64 ± 2.33	88	3.14 ± 1.41	59	**x1.5**
ϕ_5	(1,0,0)	11.01 ± 1.62	773	10.61 ± 1.64	576	**x1.34**
$\phi_{6.1}$	(1,0,0)	12.84 ± 2.46	1694	12.56 ± 2.30	1606	**x1.05**
$\phi_{6.2}$	(1,0,0)	17.67 ± 4.01	6953	15.62 ± 3.36	4470	**x1.5**
ϕ_8	(0,1,0)	12.21 ± 0.77	1802	12.97 ± 1.00	3242	x0.56
ϕ_9	(1,0,0)	11.59 ± 1.41	1335	10.90 ± 1.25	870	**x1.53**
ϕ_{10}	(1,0,0)	10.33 ± 1.73	328	9.84 ± 1.52	282	**x1.16**

Table 6.3: Verification results (U: unsatisfied. S: satisfied. T: timeout) and search depth (mean $\pm$ standard deviation) for all properties of the ACAS benchmark. Except for property 8, BE-based splits outperform IOG-based splits in all tasks. Property 7 is omitted since both heuristics timed out.

Comparison of Splitting Procedures for the Parallel Relaxation

In the previous section we compared IOG-based splits vs BE-based splits under the triangle relaxation. In this section we repeat the same experiments but using the parallel relaxation. In Fig. 6.10 we show the average over all the trees generated for each of the verified ACAS properties. We omit property 7 since it timed-out for both heuristics.

In Table 6.3 we show the average depth of the binary trees generated during verification, and the corresponding computational time. Except for property 8, BE-based splits outperforms IOG-based

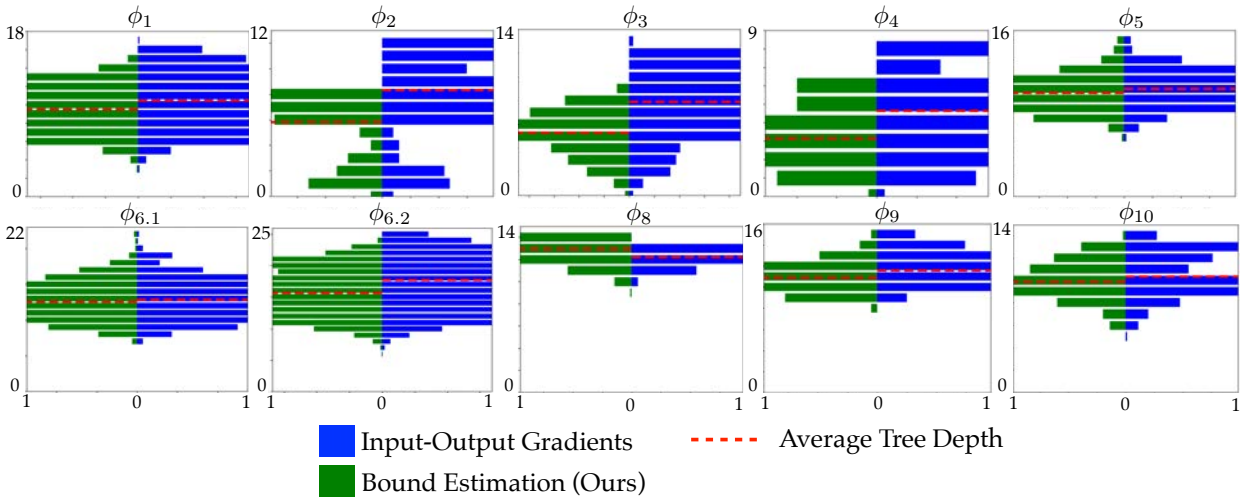


Figure 6.10: Experiments on ACAS: horizontal histograms displaying the number of branches of each length generated by each type of splitting procedure. Each pair of histograms is normalized with the maximum branch length reached for that specific property. ϕ_7 timed out for both splitting criteria.

splits in all verification tasks were neither timed out. From these results, we see that splitting based on shadow price information can produce speed-up gains of up to $\times 2.7$, as seen in property ϕ_3, and only for one property (ϕ_8) we observe a decrease in computational performance.

Discussion on the Choice of Relaxation

In the previous sections we explored how BE-based splits affect verification performance in comparison to IOG-based splits, under different convex relaxations. This section will be devoted to briefly analyzing how the choice of relaxation affected the verification results.

Property	IOG (avg. depth)		BE (avg. depth)	
	triangle	parallel	triangle	parallel
ϕ_1	7.22 ± 2.11	10.41 ± 2.73	6.79 ± 1.90	9.47 ± 2.27
ϕ_3	4.96 ± 2.50	7.87 ± 2.48	2.03 ± 1.49	5.27 ± 1.87
ϕ_4	0.92 ± 1.13	4.64 ± 2.33	0.85 ± 1.03	3.14 ± 1.41
ϕ_5	10.5 ± 3.65	11.01 ± 1.62	9.25 ± 2.79	10.61 ± 1.64
ϕ_9	9.53 ± 1.47	11.59 ± 1.41	9.09 ± 1.47	10.90 ± 1.25
ϕ_{10}	5.96 ± 0.80	10.33 ± 1.73	5.34 ± 0.89	9.84 ± 1.52

Table 6.4: Side-by-side comparison of the average three depth depending on the type of relaxation used. The comparison is made only for a subset of the properties for which it is known that the intersection of the image and the output set is empty.

Table 6.4 corroborates our intuition that the triangle relaxation should always result in verification

trees that are shallower than the parallel relaxation. Yet, despite this result, verification on the ACAS benchmark is much faster using parallel relaxations than triangle relaxations. The reason for that is that parallel relaxations do not require computing LPs for every node. While the results for the triangle relaxation in the experimental section only show relative speed between BE- and IOG-based splits, the overall time for verifying a *single* network is in the order of 2 hours. In contrast, verifying a network using the parallel relaxation as shown in Tbl. 6.3 can take less than 10 minutes in many cases.

6.7 Chapter Summary

In this chapter we introduced the concept of convex relaxations of the ReLU network in order to over-approximate $f_\theta(\mathcal{B})$. We then introduced an algorithm capable of verifying ReLU networks by repeatedly splitting the input set into smaller and smaller regions. However, deciding how to split the input set is non-trivial. Therefore, we introduced a heuristic based on shadow prices which estimates how different splits will affect all intermediate upper and lower bounds in the network. In the experimental section we compare how this heuristic compares to alternate heuristics on the ACAS verification benchmark. Our results show that using shadow prices to determine splitting decisions works best in practice. Finally, we provide a comparison between the triangle and parallel relaxation for verification.

Chapter 7

Conclusion

The overarching goal of this dissertation has been to provide a new set of tools for assured autonomy of safety-critical and learning-enabled systems. In the first half of the thesis we provided two approaches for guaranteeing safety in high-dimensional dynamical models using neural networks to approximate the optimal policy or value function for the reachability problem. Our experiments indicated that the resulting reachable sets are accurate in practice despite the limitations posed by the use of function approximators. In the second part of the thesis we introduced the verification problem for ReLU networks and provided an algorithm capable of solving it by using over-approximations and iterative refinements. We saw that the refinements require the choice of a splitting heuristic which we based on shadow price metrics. Our experimental results show that the use of shadow prices enables faster verification than other state-of-the-art heuristics for refinement-based verification.

7.1 Next Steps in Assured Autonomy of Safety-Critical Systems

In Ch. 4 we were able to connect the concepts of reachability theory and reinforcement learning into a safety learning framework. However, current safety learning tools, unlike some existing reinforcement learning techniques, are not able to incorporate high-dimensional observations such as images into their analysis. One of the main challenges will be how to define implicit surface functions for problems with high-dimensional observations. The space of images, for instance, is too large and complex for implicit surface functions to be defined over it. There exist approaches in reinforcement learning which use projections into low-dimensional latent spaces to make problems more tractable. Following this idea, and interesting direction would be to learn implicit surface functions in latent spaces.

Another interesting research direction would consist in finding function approximators for the safety learning problem which posses some form of convergence or error guarantees. The value function computed using Q-networks presented in Ch. 4 is neither an over-approximation or an under-approximation of the true value function. While simulation can be used to get a conservative

approximation of the true value, it would be convenient to use function approximators which can directly provide (conservative) information about the value of states.

7.2 Next Steps in Assured Autonomy of Learning-Enabled Systems

While Ch. 6 provided a useful framework for verifying neural networks, verification is ultimately a very challenging problem due to its NP-Complete computational complexity. As we saw by comparing the triangle and the parallel relaxations side-by-side, there is often a worthwhile trade-off between approximation accuracy and speed when dealing with verification problems. Trade-offs of this form might be useful outside of verification, however. In particular, when training neural networks, adding some structure to the weights and biases to ease the cost of verification at the expense of a small reduction in performance might be well worth it.

Along these same lines, it might also prove fruitful to investigate probabilistic or statistical guarantees rather than absolute ones of the type provided in this thesis. Sampling large quantities of inputs can be done very fast, which might allow for cheap-to-compute probabilistic over-approximations. While the types of guarantees would be weaker due to their inherent probabilistic nature, for many applications high-confidence guarantees might be more than enough.

Bibliography

[1] Joshua Achiam et al. “Constrained Policy Optimization”. In: *International Conference on Machine Learning*. 2017, pp. 22–31.

[2] Anayo K. Akametalu et al. “A Minimum Discounted Reward Hamilton-Jacobi Formulation for Computing Reachable Sets”. In: *arXiv preprint* (2018). URL: https://arxiv.org/abs/1809.00706.

[3] Anayo Kenechuku Akametalu et al. “Reachability-based safe learning with Gaussian processes”. In: *Conference on Decision and Control (CDC)* (2014), pp. 1424–1431. DOI: 10.1109/CDC.2014.7039601. URL: http://www.ece.ubc.ca/%7B~%7Dkaynama/papers/CDC2014%7B%5C_%7Dsafelearning.pdf%20http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7039601.

[4] Naveed Akhtar and Ajmal Mian. “Threat of adversarial attacks on deep learning in computer vision: A survey”. In: *IEEE Access* 6 (2018), pp. 14410–14430.

[5] Matthias Althoff. “An Introduction to CORA 2015.” In: *ARCH@ CPSWeek*. 2015, pp. 120–151.

[6] Somil Bansal et al. “Learning quadrotor dynamics using neural network for flight control”. In: *Decision and Control (CDC), 2016 IEEE 55th Conference on*. IEEE. 2016, pp. 4653–4660.

[7] Jérôme Barraquand and Jean -Claude Latombe. “Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles”. In: *Algorithmica* 10.2 (Oct. 1993), p. 121. ISSN: 1432-0541. DOI: 10.1007/BF01891837. URL: https://doi.org/10.1007/BF01891837.

[8] Alexandre M. Bayen et al. “Aircraft Autolander Safety Analysis Through Optimal Control-Based Reach Set Computation”. In: *AIAA J. Guidance, Control, and Dynamics* 30.1 (2007).

[9] Richard Bellman. *Dynamic Programming*. 1st ed. Princeton, NJ, USA: Princeton University Press, 1957. URL: http://books.google.com/books?id=fyVtp3EMxasC%7B%5C&%7Dpg=PR5%7B%5C&%7Ddq=dynamic+programming+richard+e+bellman%7B%5C&%7Dclient=firefox-a%7B%5C#%7Dv=onepage%7B%5C&%7Dq=dynamic%20programming%20richard%20e%20bellman%7B%5C&%7Df=false.

[10] Felix Berkenkamp et al. "Safe learning of regions of attraction for uncertain, nonlinear systems with Gaussian processes". In: *Conference on Decision and Control (CDC)* (2016), pp. 4661–4666. DOI: 10.1109/CDC.2016.7798979. arXiv: 1603.04915.

[11] Mariusz Bojarski et al. "End to end learning for self-driving cars". In: *arXiv preprint arXiv:1604.07316* (2016).

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

[13] Greg Brockman et al. "OpenAI Gym". In: *arXiv preprint* (2016). URL: https://arxiv.org/abs/1606.01540.

[14] Rudy Bunel et al. "Piecewise Linear Neural Network verification: A comparative study". In: *CoRR* abs/1711.00455 (2017). arXiv: 1711.00455. URL: http://arxiv.org/abs/1711.00455.

[15] Mo Chen, Sylvia Herbert, and Claire J Tomlin. "Exact and efficient Hamilton-Jacobi-based guaranteed safety analysis via system decomposition". In: *Proc. IEEE Int. Conf. Robotics and Automation*. 2016.

[16] Mo Chen, Sylvia Herbert, and Claire J Tomlin. "Fast reachable set approximations via state decoupling disturbances". In: *Proc. IEEE Conf. Decision and Control*. 2016.

[17] Mo Chen et al. "Decomposition of Reachable Sets and Tubes for a Class of Nonlinear Systems". In: *IEEE Trans. Autom. Control (to appear)* (2016).

[18] Mo Chen et al. "Robust Sequential Path Planning Under Disturbances and Adversarial Intruder". In: *The American Institute of Aeronautics and Astronautics* (2016).

[19] Mo Chen et al. "Safe Platooning of Unmanned Aerial Vehicles via Reachability". In: *Proc. IEEE Conf. Decision and Control*. 2015.

[20] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. "Flow*: An analyzer for non-linear hybrid systems". In: *International Conference on Computer Aided Verification*. Springer. 2013, pp. 258–263.

[21] Zheng Chen and Sarangapani Jagannathan. "Generalized Hamilton–Jacobi–Bellman formulation-based neural network control of affine nonlinear discrete-time systems". In: *IEEE Transactions on Neural Networks* 19.1 (2008), pp. 90–106.

[22] Ivan Nunes Da Silva et al. "Artificial neural network architectures and training processes". In: *Artificial neural networks*. Springer, 2017, pp. 21–28.

[23] Shi-Lu Dai, Cong Wang, and Min Wang. "Dynamic learning from adaptive neural network control of a class of nonaffine nonlinear systems". In: *IEEE transactions on neural networks and learning systems* 25.1 (2014), pp. 111–123.

[24] Jerry Ding et al. "Reachability Calculations for Automated Aerial Refueling". In: *Proc. IEEE Conf. Decision and Control*. 2008.

[25] Badis Djeridane and John Lygeros. "Neural approximation of PDE solutions: An application to reachability computations". In: *IEEE Conference on Decision and Control*. IEEE. 2006, pp. 3034–3039.

[26] Tommaso Dreossi, Thao Dang, and Carla Piazza. "Parallelotope bundles for polynomial reachability". In: *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*. ACM. 2016, pp. 297–306.

[27] Rüdiger Ehlers. "Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks". In: *CoRR* abs/1705.01320 (2017). arXiv: 1705.01320. URL: http://arxiv.org/abs/1705.01320.

[28] Jaime Fernandez Fisac. *Game-Theoretic Safety Assurance for Human-Centered Robotic Systems*. University of California, Berkeley, 2019.

[29] Jaime F Fisac et al. "Bridging hamilton-jacobi safety analysis and reinforcement learning". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 8550–8556.

[30] Jaime F. Fisac* et al. "A general safety framework for learning-based control in uncertain robotic systems". In: *IEEE Transactions on Automatic Control (in press)* (2018).

[31] Carlos Florensa et al. "Reverse curriculum generation for reinforcement learning". In: *arXiv preprint arXiv:1707.05300* (2017).

[32] Goran Frehse et al. "SpaceEx: Scalable verification of hybrid systems". In: *International Conference on Computer Aided Verification*. Springer. 2011, pp. 379–395.

[33] Jeremy H. Gillula and Claire J. Tomlin. "Guaranteed Safe Online Learning via Reachability: tracking a ground target using a quadrotor". In: *International Conference on Robotics and Automation (ICRA)* (May 2012), pp. 2723–2730. DOI: 10.1109/ICRA.2012.6225136. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6225136.

[34] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. "Speech recognition with deep recurrent neural networks". In: *IEEE international conference on Acoustics, speech and signal processing (ICASSP)*. 2013, pp. 6645–6649.

[35] Mark R Greenstreet and Ian Mitchell. "Integrating projections". In: *International Workshop on Hybrid Systems: Computation and Control*. Springer. 1998, pp. 159–174.

[36] Tuomas Haarnoja et al. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *CoRR* abs/1801.01290 (2018). arXiv: 1801.01290. URL: http://arxiv.org/abs/1801.01290.

[37] Nicolas Heess et al. "Emergence of Locomotion Behaviours in Rich Environments". In: *CoRR* abs/1707.02286 (2017). arXiv: 1707.02286. URL: http://arxiv.org/abs/1707.02286.

[38] Sylvia L. Herbert* et al. "FaSTrack: a Modular Framework for Fast and Guaranteed Safe Motion Planning". In: *IEEE Conference on Decision and Control* (2017).

[39] Haomiao Huang et al. “A differential game approach to planning in adversarial scenarios: A case study on capture-the-flag”. In: *Proc. IEEE Int. Conf. Robotics and Automation*. 2011.

[40] Xiaowei Huang et al. “Safety Verification of Deep Neural Networks”. In: *CoRR* abs/1610.06940 (2016). arXiv: 1610.06940. URL: http://arxiv.org/abs/1610.06940.

[41] Frank Jiang et al. “Using neural networks to compute approximate and guaranteed feasible Hamilton-Jacobi-Bellman PDE solutions”. In: *arXiv preprint arXiv:1611.03158* (2016).

[42] Gregory Kahn et al. “Plato: Policy learning using adaptive trajectory optimization”. In: *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE. 2017, pp. 3342–3349.

[43] Guy Katz et al. “Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks”. In: *CoRR* abs/1702.01135 (2017). arXiv: 1702.01135. URL: http://arxiv.org/abs/1702.01135.

[44] Shahab Kaynama et al. “Computing the viability kernel using maximal reachable sets”. In: *Proceedings of the 15th ACM international conference on Hybrid Systems: Computation and Control*. ACM. 2012, pp. 55–64.

[45] Shahab Kaynama et al. “Scalable safety-preserving robust control synthesis for continuous-time linear systems”. In: *IEEE Transactions on Automatic Control* 60.11 (2015), pp. 3065–3070. ISSN: 00189286. DOI: 10.1109/TAC.2015.2411872. arXiv: 1312.3399.

[46] Yoon Kim. “Convolutional neural networks for sentence classification”. In: *arXiv preprint arXiv:1408.5882* (2014).

[47] J. Zico Kolter and Eric Wong. “Provable defenses against adversarial examples via the convex outer adversarial polytope”. In: *CoRR* abs/1711.00851 (2017). arXiv: 1711.00851. URL: http://arxiv.org/abs/1711.00851.

[48] Alexander B Kurzhanski and Pravin Varaiya. “Ellipsoidal techniques for reachability analysis: internal approximation”. In: *Systems & control letters* 41.3 (2000), pp. 201–211.

[49] Alexander B Kurzhanski and Pravin Varaiya. “On ellipsoidal techniques for reachability analysis. part ii: Internal approximations box-valued constraints”. In: *Optimization methods and software* 17.2 (2002), pp. 207–237.

[50] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *nature* 521.7553 (2015), p. 436.

[51] Mathias Lecuyer et al. *Certified Robustness to Adversarial Examples with Differential Privacy*. 2019. arXiv: 1802.03471 [stat.ML].

[52] Sergey Levine et al. “End-to-End Training of Deep Visuomotor Policies”. In: *CoRR* abs/1504.00702 (2015). arXiv: 1504.00702. URL: http://arxiv.org/abs/1504.00702.

[53] Sergey Levine et al. “End-to-end training of deep visuomotor policies”. In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1334–1373.

[54] Bai Li et al. “Second-order adversarial attack and certifiable robustness”. In: (2018).

[55] Daniel Liberzon. *Calculus of variations and optimal control theory: a concise introduction*. Princeton University Press, 2011.

[56] Alessio Lomuscio and Lalit Maganti. “An approach to reachability analysis for feed-forward ReLU neural networks”. In: *CoRR* abs/1706.07351 (2017). arXiv: 1706.07351. URL: http://arxiv.org/abs/1706.07351.

[57] Jingyue Lu and M Pawan Kumar. “Neural Network Branching for Neural Network Verification”. In: *arXiv preprint arXiv:1912.01329* (2019).

[58] Aleksander Madry et al. “Towards deep learning models resistant to adversarial attacks”. In: *arXiv preprint arXiv:1706.06083* (2017).

[59] John N Maidens et al. “Lagrangian methods for approximating the viability kernel in high-dimensional systems”. In: *Automatica* 49.7 (2013), pp. 2017–2029.

[60] Anirudha Majumdar et al. “Convex optimization of nonlinear feedback controllers via occupation measures”. In: *The International Journal of Robotics Research* 33.9 (2014), pp. 1209–1230.

[61] Ian Mitchell. *A Toolbox of Level Set Methods*. http://people.cs.ubc.ca/~mitchell/ToolboxLS/index.html. 2009.

[62] Ian M. Mitchell, A. M. Bayen, and C. J. Tomlin. “A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games”. In: *IEEE Transactions on Automatic Control* 50.7 (July 2005), pp. 947–957. ISSN: 0018-9286. DOI: 10.1109/TAC.2005.851439. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1463302.

[63] Ian Mitchell and Jeremy Templeton. “A Toolbox of Hamilton-Jacobi Solvers for Analysis of Nondeterministic Continuous and Hybrid Systems”. In: vol. 3414. Mar. 2005, pp. 480–494. DOI: 10.1007/978-3-540-31954-2_31.

[64] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), pp. 529–533. ISSN: 0028-0836. DOI: 10.1038/nature14236. URL: http://dx.doi.org/10.1038/nature14236.

[65] Volodymyr Mnih et al. “Playing atari with deep reinforcement learning”. In: *arXiv preprint arXiv:1312.5602* (2013).

[66] TM M Moldovan and P Abbeel. “Safe exploration in Markov decision processes”. In: *International Conference on Machine Learning (ICML)* (2012). arXiv: arXiv:1205.4810v3. URL: http://arxiv.org/abs/1205.4810.

[67] Guido F Montufar et al. “On the number of linear regions of deep neural networks”. In: *Advances in neural information processing systems*. 2014, pp. 2924–2932.

[68] Anusha Nagabandi et al. “Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning”. In: *arXiv preprint arXiv:1708.02596* (2017).

[69] Petter Nilsson and Necmiye Ozay. "Synthesis of separable controlled invariant sets for modular local control design". In: *American Control Conference (ACC), 2016*. IEEE. 2016, pp. 5656–5663.

[70] OpenAI. *OpenAI Five*. https://blog.openai.com/openai-five/. [Online; accessed 18 September 2018]. 2018. URL: https://blog.openai.com/openai-five/.

[71] Stanley Osher, Ronald Fedkiw, and K Piechor. "Level set methods and dynamic implicit surfaces". In: *Appl. Mech. Rev.* 57.3 (2004), B15–B15.

[72] Stanley Osher and Chi-Wang Shu. "High-Order Essentially Nonoscillatory Schemes for Hamilton–Jacobi Equations". In: *Siam Journal on Numerical Analysis - SIAM J NUMER ANAL* 28 (Aug. 1991). DOI: 10.1137/0728049.

[73] Scott M Pappada et al. "Neural network-based real-time prediction of glucose in patients with insulin-dependent diabetes". In: *Diabetes technology & therapeutics* 13.2 (2011), pp. 135–141.

[74] Peng Cheng and S.M. LaValle. "Resolution complete rapidly-exploring random trees". In: 2003, pp. 267–272. DOI: 10.1109/robot.2002.1013372.

[75] Elena G Popkova, Yulia V Ragulina, and Aleksei V Bogoviz. "Fundamental differences of transition to industry 4.0 from previous industrial revolutions". In: *Industry 4.0: Industrial Revolution of the 21st Century*. Springer, 2019, pp. 21–29.

[76] Boštjan Potočnik et al. "Model-based predictive control of hybrid systems: a probabilistic neural-network approach to real-time control". In: *Journal of Intelligent and Robotic Systems* 51.1 (2008), pp. 45–63.

[77] Maithra Raghu et al. "On the expressive power of deep neural networks". In: *Proceedings of the 34th International Conference on Machine Learning*. Vol. 70. JMLR. org. 2017, pp. 2847–2854.

[78] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. "Certified Defenses against Adversarial Examples". In: *CoRR* abs/1801.09344 (2018). arXiv: 1801.09344. URL: http://arxiv.org/abs/1801.09344.

[79] Vicenc Rubies-Royo et al. "A Classification-based Approach for Approximate Reachability". In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 7697–7704.

[80] Vicenc Rubies-Royo et al. "Fast neural network verification via shadow prices". In: *arXiv preprint arXiv:1902.07247* (2019).

[81] Hadi Salman et al. "Provably robust deep learning via adversarially trained smoothed classifiers". In: *Advances in Neural Information Processing Systems*. 2019, pp. 11292–11303.

[82] John Schulman et al. "Trust Region Policy Optimization". In: *CoRR* abs/1502.05477 (2015). arXiv: 1502.05477. URL: http://arxiv.org/abs/1502.05477.

[83] Klaus Schwab. *The fourth industrial revolution*. Currency, 2017.

[84] Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. "Bounding and Counting Linear Regions of Deep Neural Networks". In: *CoRR* abs/1711.02114 (2017). arXiv: 1711.02114. URL: http://arxiv.org/abs/1711.02114.

[85] David Silver et al. "Mastering the game of Go without human knowledge". In: *Nature* 550.7676 (2017), p. 354.

[86] Aman Sinha, Hongseok Namkoong, and John Duchi. "Certifying some distributional robustness with principled adversarial training". In: *arXiv preprint arXiv:1710.10571* (2017).

[87] Jian Sun et al. "An integrated critic-actor neural network for reinforcement learning with application of DERs control in grid frequency regulation". In: *International Journal of Electrical Power & Energy Systems* 111 (2019), pp. 286–299.

[88] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*. MIT press, 1998.

[89] Vincent Tjeng and Russ Tedrake. "Verifying Neural Networks with Mixed Integer Programming". In: *CoRR* abs/1711.07356 (2017). arXiv: 1711.07356. URL: http://arxiv.org/abs/1711.07356.

[90] Emanuel Todorov, Tom Erez, and Yuval Tassa. "Mujoco: A physics engine for model-based control". In: *International Conference on Intelligent Robots and Systems (IROS)* (2012), pp. 5026–5033. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.296.6848&rep=rep1&type=pdf.

[91] John N. Tsitsiklis. "Asynchronous Stochastic Approximation and Q-Learning". In: *Machine Learning* 16.3 (1994), pp. 185–202.

[92] Shiqi Wang et al. "Formal Security Analysis of Neural Networks using Symbolic Intervals". In: *CoRR* abs/1804.10829 (2018). arXiv: 1804.10829. URL: http://arxiv.org/abs/1804.10829.

[93] Christopher J. C. H. Watkins and Peter Dayan. "Q-learning". In: *Machine Learning* 8.3-4 (May 1992), pp. 279–292. ISSN: 0885-6125. DOI: 10.1007/BF00992698. URL: http://link.springer.com/10.1007/BF00992698.

[94] Tyler Westenbroek et al. "Feedback Linearization for Unknown Systems via Reinforcement Learning". In: *arXiv preprint arXiv:1910.13272* (2019).

[95] Ronald J. Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine Learning* 8.3 (May 1992), pp. 229–256. ISSN: 1573-0565. DOI: 10.1007/BF00992696. URL: https://doi.org/10.1007/BF00992696.

[96] Weiming Xiang, Hoang-Dung Tran, and Taylor T. Johnson. "Output Reachable Set Estimation and Verification for Multi-Layer Neural Networks". In: *CoRR* abs/1708.03322 (2017). arXiv: 1708.03322. URL: http://arxiv.org/abs/1708.03322.

[97] Weiming Xiang, Hoang-Dung Tran, and Taylor T. Johnson. "Reachable Set Computation and Safety Verification for Neural Networks with ReLU Activations". In: *CoRR* abs/1712.08163 (2017). arXiv: 1712.08163. URL: http://arxiv.org/abs/1712.08163.

[98] Huaguang Zhang, Yanhong Luo, and Derong Liu. "Neural-network-based near-optimal control for a class of discrete-time affine nonlinear systems with control constraints". In: *IEEE Transactions on Neural Networks* 20.9 (2009), pp. 1490–1503.

[99] Huan Zhang et al. "Efficient neural network robustness certification with general activation functions". In: *Advances in neural information processing systems*. 2018, pp. 4939–4948.

[100] Huan Zhang et al. "Efficient neural network robustness certification with general activation functions". In: *Advances in neural information processing systems*. 2018, pp. 4939–4948.

Appendix A

ACAS Properties

Here we list the set of properties to be verified in the ACAS benchmark.

Property ϕ_1:
- **Description:** If the intruder is distant and is significantly slower than the ownship, the score of a COC advisory will always be below a certain fixed threshold.
- **Tested on:** all 45 networks.
- **Input constraints:** $\rho \geq 55947.691$, $v_{\text{own}} \geq 1145$, $v_{\text{int}} \leq 60$.
- **Desired output property:** the score for COC is at most 1500.

Property ϕ_2:
- **Description:** If the intruder is distant and is significantly slower than the ownship, the score of a COC advisory will never be maximal.
- **Tested on:** $N_{x,y}$ for all $x \geq 2$ and for all y, except $N_{4,2}$ and $N_{5,3}$.
- **Input constraints:** $\rho \geq 55947.691$, $v_{\text{own}} \geq 1145$, $v_{\text{int}} \leq 60$.
- **Desired output property:** the score for COC is not the maximal score.

Property ϕ_3:
- **Description:** If the intruder is directly ahead and is moving towards the ownship, the score for COC will not be minimal.
- **Tested on:** all networks except $N_{1,7}$, $N_{1,8}$, and $N_{1,9}$.
- **Input constraints:** $1500 \leq \rho \leq 1800$, $-0.06 \leq \theta \leq 0.06$, $\psi \geq 3.10$, $v_{\text{own}} \geq 980$, $v_{\text{int}} \geq 960$.
- **Desired output property:** the score for COC is not the minimal score.

Property ϕ_4:
- **Description:** If the intruder is directly ahead and is moving away from the ownship but at a lower speed than that of the ownship, the score for COC will not be minimal.
- **Tested on:** all networks except $N_{1,7}$, $N_{1,8}$, and $N_{1,9}$.

- **Input constraints:** $1500 \leq \rho \leq 1800$, $-0.06 \leq \theta \leq 0.06$, $\psi = 0$, $v_{\text{own}} \geq 1000$, $700 \leq v_{\text{int}} \leq 800$.
- **Desired output property:** the score for COC is not the minimal score.

Property ϕ_5:
- **Description:** If the intruder is near and approaching from the left, the network advises "strong right".
- **Tested on:** $N_{1,1}$.
- **Input constraints:** $250 \leq \rho \leq 400$, $0.2 \leq \theta \leq 0.4$, $-3.141592 \leq \psi \leq -3.141592 + 0.005$, $100 \leq v_{\text{own}} \leq 400$, $0 \leq v_{\text{int}} \leq 400$.
- **Desired output property:** the score for "strong right" is the minimal score.

Property ϕ_6:
- **Description:** If the intruder is sufficiently far away, the network advises COC.
- **Tested on:** $N_{1,1}$.
- **Input constraints:** $12000 \leq \rho \leq 62000$, $(0.7 \leq \theta \leq 3.141592) \vee (-3.141592 \leq \theta \leq -0.7)$, $-3.141592 \leq \psi \leq -3.141592 + 0.005$, $100 \leq v_{\text{own}} \leq 1200$, $0 \leq v_{\text{int}} \leq 1200$.
- **Desired output property:** the score for COC is the minimal score.

Property ϕ_7:
- **Description:** If vertical separation is large, the network will never advise a strong turn.
- **Tested on:** $N_{1,9}$.
- **Input constraints:** $0 \leq \rho \leq 60760$, $-3.141592 \leq \theta \leq 3.141592$, $-3.141592 \leq \psi \leq 3.141592$, $100 \leq v_{\text{own}} \leq 1200$, $0 \leq v_{\text{int}} \leq 1200$.
- **Desired output property:** the scores for "strong right" and "strong left" are never the minimal scores.

Property ϕ_8:
- **Description:** For a large vertical separation and a previous "weak left" advisory, the network will either output COC or continue advising "weak left".
- **Tested on:** $N_{2,9}$.
- **Input constraints:** $0 \leq \rho \leq 60760$, $-3.141592 \leq \theta \leq -0.75 \cdot 3.141592$, $-0.1 \leq \psi \leq 0.1$, $600 \leq v_{\text{own}} \leq 1200$, $600 \leq v_{\text{int}} \leq 1200$.
- Desired output property: the score for "weak left" is minimal or the score for COC is minimal.

Property ϕ_9:
- **Description:** Even if the previous advisory was "weak right", the presence of a nearby intruder will cause the network to output a "strong left" advisory instead.
- **Tested on:** $N_{3,3}$.
- **Input constraints:** $2000 \leq \rho \leq 7000$, $-0.4 \leq \theta \leq -0.14$, $-3.141592 \leq \psi \leq -3.141592 + 0.01$, $100 \leq v_{\text{own}} \leq 150$, $0 \leq v_{\text{int}} \leq 150$.

- **Desired output property:** the score for "strong left" is minimal.

Property ϕ_{10}:
- **Description:** For a far away intruder, the network advises COC.
- **Tested on:** $N_{4,5}$.
- **Input constraints:** $36000 \leq \rho \leq 60760$, $0.7 \leq \theta \leq 3.141592$, $-3.141592 \leq \psi \leq -3.141592 + 0.01$, $900 \leq v_{\text{own}} \leq 1200$, $600 \leq v_{\text{int}} \leq 1200$.
- **Desired output property:** the score for COC is minimal.