Synthesis and Verification of Neural Control Barrier Functions for Safe Reinforcement Learning with Guarantees

Master Thesis

Xinyu Wang

Delft University of Technology

TU Delft

Synthesis and Verification of Neural Control Barrier Functions for Safe Reinforcement Learning with Guarantees

by

Xinyu Wang

Student Name	Student Number
Xinyu Wang	5454522

Instructor:	Javier Alonso-Mora
Teaching Assistant:	Luzia Knoedler
Project Duration:	12, 2022 - 11, 2023
Faculty:	ME, Delft

TU Delft

Abstract

While learning-based control techniques often outperform classical controller designs, safety requirements limit the acceptance of such methods in many applications. Recent developments address this issue through Certified Learning (CL), which combines a learning-based controller with formal methods to provide safety guarantees. This thesis focuses on the CL based on Control Barrier Functions (CBFs), as CBFs have been widely used for safety-critical systems. However, it is non-trivial to design a CBF. Utilizing neural networks as CBFs has shown great success, but it necessitates their certification as CBFs. In this work, we leverage bound propagation techniques and the Branch-and-Bound scheme to efficiently verify that a neural network satisfies the conditions to be a CBF over the continuous state space. To accelerate training, we further present a framework that embeds the verification scheme into the training loop to synthesize and verify a neural CBF simultaneously. In particular, we employ the verification scheme to identify partitions of the state space that are not guaranteed to satisfy the CBF conditions and expand the training dataset by incorporating additional data from these partitions. The neural network is then optimized using the augmented dataset to meet the CBF conditions. We show that for a non-linear control-affine system, our framework can efficiently certify a neural network as a CBF and render a larger safe set than state-of-the-art neural CBF works. We further employ our learned neural CBF to derive a safe controller to illustrate the practical use of our framework.

Contents

List of Figures

List of Tables

Acronyms

BBS Branch-and-Bound scheme.
BBV Branch-and-Bound Verification scheme.
BBVT Branch-and-Bound Verification-in-the-loop Training.
BF Barrier Function.
BRT Backward Reachable Tube.
BVF Backward Value Function.

CBC Control Barrier Condition.
CBF Control Barrier Function.
CBVF Control Barrier-Value Function.
CBVF-VI Control Barrier-Value Function Variational Inequality.
CE Counterexample.
CL Certified Learning.
CMDP Constrained Markov Decision Process.

DL Deep Learning.

FCNN Fully-Connected Neural Network.

GP Gaussian Process.

HJI-RA Hamilton-Jacobian-Issac Reachability Analysis.
HJI-VI Hamilton-Jacobian-Issac Variational Inequality.

LP Linear Programming.

MDP Markov Decision Process.
MIP Mixed Integer Programming.
MPC Model Predictive Control.

nCBF Neural Control Barrier Function.
NLP Nonlinear Program.
NN Neural Network.

QP Quadratic Program.

RA Reachability Analysis.
RL Reinforcement Learning.

SAT Boolean Satisfiability Problem.
SGD Stochastic Gradient Descent.

SMT Satisfiability Modulo Theory.

SOS sum-of-squares.

TD Temporal-Difference.

1 Introduction

1.1. Motivation

Safety is a critical element of autonomous systems, including self-driving cars and manipulators interacting with humans. As these systems become increasingly complex, ensuring their safe operation poses significant challenges. Various techniques have been developed to address this challenge, such as Model Predictive Control (MPC), Hamilton-Jacobian-Issac Reachability Analysis (HJI-RA), Control Barrier Function (CBF), and Constrained Markov Decision Process (CMDP). We will discuss the advantages and limitations of these techniques and propose pertinent research questions in Section 1.2. The contributions of this thesis are detailed in Section 1.3. For readers unfamiliar with the aforementioned techniques, a comprehensive background chapter is available in the Appendix. A.

While traditional control schemes like the MPC offer safety guarantees, they face challenges in scaling for high-dimensional systems and often rely on specific assumptions, such as the existence of a terminal set and solver feasibility. Additionally, these methods struggle to capture complex patterns within extensive datasets. In contrast, data-driven methods such as Reinforcement Learning (RL) and Deep Learning (DL) have demonstrated remarkable success in handling large, complex systems by learning from raw data, facilitating more accurate predictions and adaptation to unstructured environments. However, the absence of safety guarantees and interpretability impedes the practical deployment of these data-driven methods. Therefore, numerous efforts have been made to improve the safety of the RL. Different safe RL approaches have emerged, including RL with Safety Encouragement, CMDP, and Certified Learning (CL). The RL with Safety Encouragement aims to optimize the reward while reducing constraint violations by embedding risk signals into objective functions, which may lead to unsafe actions during the early training stages. Solving a CMDP means deriving a feasible policy that consistently satisfies constraints. Although a finite discrete CMDP can be solved efficiently by Linear Programming (LP), solving a CMDP in a continuous state space remains an open question. Given these limitations, this thesis focuses on the third category of safe RL, namely CL. Unlike the other approaches that simultaneously optimize reward and minimize constraint violations, the CL comprises two subsystems: a neural controller responsible for performance enhancement through interacting with the environment and a certificate that ensures unsafe actions from the neural controller are projected into a safe action space, thus maintaining system safety at all times. The general framework of the CL is shown in Fig. 1.1.

The HJI-RA and the CBFs are two common formal methods that synthesize certificates and provide safety guarantees. Both certificates address safety through set invariance and safe controllers can be derived easily. The certificate (i.e., safe value function) from the HJI-RA is generated by solving the Hamilton-Jacobian-Issac Variational Inequality (HJI-VI) over a grid map but is prone to numerical errors and may result in undesired jerk behaviors. On the other hand, the CBFs ensure safety with a smooth controller and exhibit asymptotical stability even if the system violates the constraints [1]. Consequently, the utilization of the CBFs for deriving a forward invariant set has received significant attention in the control and learning community [2].

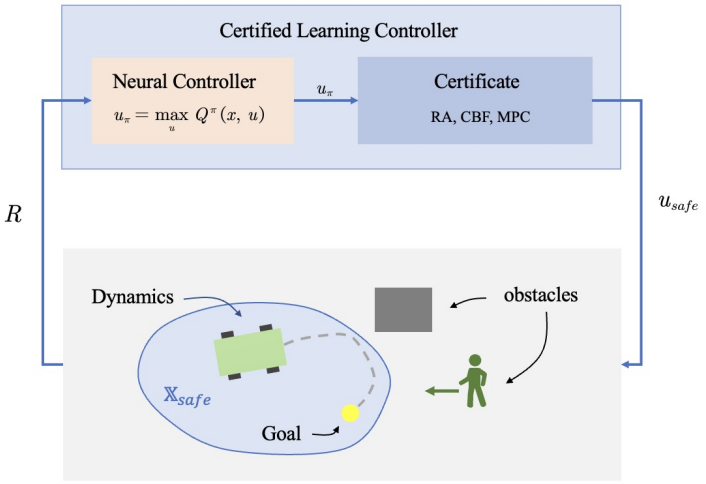


Figure 1.1: Framework of Certified Learning. The environment contains robots and obstacles. The robot learns a control policy from data (e.g., RL) while a certificate (e.g., CBF, Reachability Analysis (RA), MPC) filters out unsafe actions and only applies safe actions to the system.

However, no general and scalable technique exists for designing CBFs. Recent works [3, 4] synthesize CBFs using Neural Networks (NNs) as function templates, which are referred to as Neural Control Barrier Functions (nCBFs). Yet, these works rely on an initial guess of the forward invariant set or the function structure of the CBF. An improper initial guess usually results in a suboptimal nCBF and constructing an optimal CBF that renders a maximum forward invariant set is challenging.

Although utilizing NNs as CBFs offers universal approximation capabilities, it necessitates their certification as CBFs to provide safety guarantees. Verifying the NN as an nCBF in the continuous state space presents a significant challenge. Specifically, since the NN is trained using a finite set of data points, it will only be verified on those points. Outside the certified points, safety is no longer guaranteed. There are works [5, 6] that use the Satisfiability Modulo Theory (SMT) to verify their NNs. However, they are restricted to very simple NNs due to expensive computation.

1.2. Research Questions

Considering the challenges outlined, the research questions of this thesis are:

1. How can we find an nCBF that renders the maximum safe area?
2. How can an nCBF be verified efficiently?

1.3. Contribution

A recent work [7] introduced the Control Barrier-Value Function (CBVF) which is a safe value function and renders the maximum forward invariant set for a chosen time span. Inspired by that, in this work, we synthesize a continuous nCBF that approximates the infinite-horizon CBVF and renders a safe set that is close to the maximum forward invariant set. Then, we leverage bound propagation techniques [8] and the Branch-and-Bound scheme (BBS) to efficiently verify nCBFs. In particular, we partition the state space and utilize linear bound propagation techniques to provide lower and upper bounds of the NN and its Jacobian. These bounds are used to verify if the NN satisfies the conditions to be a CBF. The BBS is applied to refine the partition to improve scalability and achieve less conservative bounds. We refer to the above verification scheme as the Branch-and-Bound Verification scheme (BBV). This approach is similar to [9], however, we verify CBFs instead of barrier functions. To accelerate training, we embed the BBV into the training loop to synthesize and verify an nCBF simultaneously, which we refer to as the Branch-and-Bound Verification-in-the-loop Training (BBVT), see Fig. 1.2.

The contributions of this thesis are listed as follows:

1. The design of a novel loss function to synthesize an nCBF that renders a safe set close to the

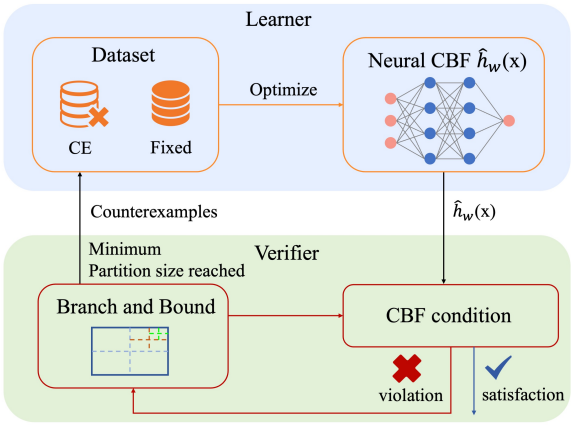


Figure 1.2: A schematic overview of the presented Branch-and-Bound Verification-in-the-loop Training. The framework comprises two key components: the learner and the verifier, which operate sequentially. The learner optimizes the nCBF using a fixed dataset and a Counterexample dataset. The verifier leverages bound propagation techniques and the Branch-and-Bound scheme to refine a partition of the state space until the CBF conditions are satisfied or counterexamples are generated.

maximum forward invariant set.

2. Introduction of an efficient method for verifying an nCBF.
3. Validation on an inverted pendulum and 2D navigation task.

The rest of this thesis is laid out as follows: A comprehensive literature review is presented in Chapter 2, followed by Chapter 3 which explains the fundamental theory and properties of a CBF, alongside the precise definition of the system and problem statement. Chapter 4 elaborates on our method, including the loss function, the BBV, and the training framework. Finally, Chapter 5 describes the validation of our method on an inverted pendulum and demonstrates the practical use of our learned nCBF on a 2D navigation task. This thesis resulted in a conference paper that will be presented at ECC 2024. The paper is available in the Appendix B.

2

Related Work

Before addressing the outlined research questions, it is essential to establish a comprehensive understanding of the current state-of-the-art techniques in the safe RL. We start with the safe RL without guarantees in Section 2.1 and focus more on the CL based on CBFs and the HJI-RA in Section 2.2. Since the NN Verification is a necessary tool to verify our nCBFs, the relevant literature is discussed in Section 2.3.

2.1. Safe Reinforcement Learning without Guarantees

The RL employs a data-driven methodology, offering adaptability to unstructured contexts and scalability to complex systems at the expense of providing formal guarantees. In this section, we provide a brief introduction to two mainstream approaches, namely the RL with safety encouragement and the CMDP. A comprehensive review of these two categories of safe RL can be found in [10].

2.1.1. Reinforcement Learning with Safety Encouragement

To enhance safety in RL, researchers frequently modify the reward function [11, 12] or integrate risk signals into the Temporal-Difference (TD) error [13] so that the value function or Q-function reflects the risk associated with future trajectories. However, the efficacy of these methods is limited when the risk scenario constitutes only a small fraction of the training dataset. Therefore, some works propose the utilization of a Safety Critic [14, 15], which predicts the risk associated with the current state and action. Nevertheless, training a Safety Critic necessitates a dataset containing unsafe states and actions, the collection of which involves potentially hazardous attempts. To augment the training dataset, domain randomization is extensively employed to train agents across a spectrum of randomized and perturbed environments and scenarios. This approach has demonstrated empirical success in facilitating sim-to-real transfer in safety-critic tasks [16, 17].

In contrast to the aforementioned works where agents require a large number of trials to achieve a safe policy, certain advanced strategies enable agents to adapt to unknown environments much more rapidly during the learning process. To mitigate noise and better capture system dynamics, recent works in [18, 19] employ model ensembles to train a collision model online, predicting the probability of collision given state and a sequence of actions. However, these approaches encounter challenges during the early stages of training. To ensure safety throughout the learning process, the works in [20, 21, 22] confine exploration to safe state space exclusively. Nevertheless, a major limitation of these methods arises from the intractable computation of generating a valid safe space, particularly for complex continuous systems. An alternative and more tractable strategy involves constraining the control policy instead of exploration space. By directing policy updates toward safer directions, these methodologies [23, 24, 25] gradually diminish constraint violations.

2.1.2. Constrained Markov Decision Process

The CMDP is frequently utilized in the safe RL due to its capability to incorporate constraints that express various safety notions. For readers unfamiliar with CMDPs, please refer to Section A.2. Given a CMDP problem, an intuitive approach involves transforming the CMDP into an unconstrained Markov Decision Process (MDP) using Lagrangian methods [26, 27]. Subsequently, standard RL algorithms can be applied to solve the resulting MDP.

To retain hard constraints, researchers solve the discrete CMDP directly [28] by converting it into a constrained LP. However, such an approach is limited to discrete CMDPs. For continuous CMDPs, the Backward Value Function (BVF) [29] is utilized to predict the accumulated cost along the trajectory and improve the control policy iteratively, which shares a similar idea that projects policy updates in a safe direction. Other works [30, 31] assume the existence of a safe baseline policy and bootstrap the control policy to solve the CMDP. However, in practice, a safe baseline policy is not always available.

2.2. Certified Learning

The CL usually consists of a controller and a certificate. The controller is designed to maximize the reward and accomplish the task, while the certificate is tasked with filtering out unsafe actions and forcing the satisfaction of the constraints. Two common certificates are CBFs and safe value functions from HJI-RA, which are widely used in safety-critical applications. Section 2.2.1 and 2.2.2 explore the relevant literature of these two certificate functions as well as the works that combine RL with CBFs or the HJI-RA to achieve safe learning.

2.2.1. Reachability Analysis

The HJI-RA [32] is extensively utilized to generate the Backward Reachable Tube (BRT) (i.e., safe value function) for the reach-avoid tasks. To compute BRTs, the HJI-RA explicitly solves the HJI-VI on a discretized state space [33], resulting in an exponential scaling of computational and spatial complexity with respect to system dimensionality. This is often referred to as the *curse of dimensionality* [32]. Although BRTs from the HJI-RA provide a maximum control invariant set, the curse of dimensionality limits its practical use to only up to 5D systems [32]. There are several works [34, 35, 36] addressing the curse of dimensionality by using sets of pre-specified shapes, such as polytopes, hyperplanes, and zonotopes. These specific shapes enable the approximation of continuous reachable sets, thus avoiding discrete state space. However, such a method results in conservative BRTs, and some works [37] rely on the linearization of nonlinear dynamics. Another strategy to mitigate the curse of dimensionality is decomposition. The work in [38] decomposes the complex nonlinear dynamics into multiple simple subsystems. The BRTs of these low-dimensional subsystems are computed first, and then the full-dimensional BRT of the original high-dimensional system is reconstructed from the low-dimensional BRTs without additional approximation error.

Several learning-based methods have also been explored for computing approximate safe value functions and using them as the safety filter. Recent works [39, 40] train NNs with large quantities of data to approximate safe value functions and BRTs. Other researchers use the TD Learning [41] or Q-learning [42, 43] to simultaneously learn a safe value function and control policy.

2.2.2. Control Barrier Function

Many works use CBFs to ensure the safety of a system [44, 45, 46]. However, it is non-trivial to construct CBFs. In recent years, new techniques emerged to automatically synthesize CBFs. For a system with polynomial dynamics, a CBF can be obtained by solving a sum-of-squares (SOS) optimization problem [47]. Unfortunately, the SOS scales poorly to higher dimensional systems [48]. To address this shortcoming, NNs have been employed to approximate CBFs. They are trained by supervised learning [3, 4, 49] or RL with the Actor-Critic framework [50, 51]. The loss function designed to encourage the NN to satisfy the Control Barrier Condition (CBC) will be defined later in Section 3. However, the quality of the nCBF in those works depends on an initial guess of the forward invariant set, the CBF candidate, or the exploration strategy. An improper initial guess results in a conservative nCBF with a small forward invariant set. To address the conservativity, in this work, we learn an nCBF that renders a safe set close to the maximum forward invariant set. Furthermore, the training does not require an initial guess.

The combination of a learning-based system with a CBF can be seen in [52, 53, 54]. They usually solve a

Quadratic Program (QP) to get the least modified safe control input.

2.3. Neural Network Verification

As NNs increasingly serve as core techniques in various applications, including safety-critical tasks, certifying the property of an NN has become an important research area in machine learning [55, 56]. Four categories of verification methods, which are dedicated to verifying properties and bounding the output of NNs, are briefly introduced in Section 2.3.1. In addition to the output of NNs, Lipschitz constants play a significant role in characterizing many properties of NNs, including robustness [57] and generalization [58]. Since the Lipschitz constant can be computed by upper bounding the norm of Jacobian [59], the related works about computing Lipschitz constants and the Jacobian of NNs are presented in Section 2.3.2, followed by a discussion about the nCBF verification in Section 2.3.3.

2.3.1. Bound the Neural Network

Exact Verification [60, 61, 62, 63, 64] encodes the linear transform layer and activation layer of an NN as a sequence of constraints. The properties of the NN are then verified by solving Mixed Integer Programming (MIP) or using Boolean Satisfiability Problem (SAT) [65, 66]. However, solving MIP and SAT has been shown to be NP-hard. As a result, *Exact Verification* can not scale to complex systems. In contrast, *Constraints Relaxation* methods overcome the extensive computational burden by over-approximating the effect of each activation layer with linear constraints at the expense of accuracy. The representative works include Fast-Lin [67], CROWN [8], and Domain Abstraction [68]. Instead of using linear constraints, *Reachability-based* methods [69, 70, 71] propagate the reachable set represented by a rectangle or ellipsoid through a neural network and obtain the bound of the output layer. For a detailed review, please refer to [2, 72].

2.3.2. Compute Lipschitz Constant and Bound Jacobian

Several works [64, 73] have been proposed to compute global Lipschitz constants. However, the local Lipschitz constants are much tighter than the global Lipschitz constants and they characterize the local behavior of NNs. Therefore, LipMIP [74] used MIP to compute exact local Lipschitz constants, while LipBaB [75] combined relatively loose interval-bound propagation with branch-and-bound to compute exact results. However, these methods suffer from scalability issues when applied to larger models due to their computational cost. To address efficiency concerns, FastLip [67] and its improved version RecurJac [76] used recursive procedures to bound the Jacobian. While these methods are much more efficient, their bounds are relatively loose due to the use of strictly looser relaxations. Recent advancements in [59, 77] provide novel approaches to computing the Jacobian of NNs with tight bound via bound propagation, offering promising avenues for efficient computation of local Lipschitz constants.

2.3.3. Neural Control Barrier Function Verification

Commonly, NNs are trained through backpropagation of the empirical loss on a finite set of data points. Therefore, it is important to note that even an empirical loss of zero does not guarantee that the certificate is valid everywhere in the state space. Only a few works have verified their NNs, such as [5, 6, 78], which leverage the SMT to provide Counterexamples (CEs) and guarantee the correctness of the synthesis procedure. However, the SMT is limited to simple NNs with around 20 neurons in one or two hidden layers due to the need for expensive computation. In contrast to using the SMT for exact verification, several efficient NN verification methods using linear bound propagation techniques have been developed [8, 67]. These bounding methods provide a new direction to verify neural certificates. The work in [9] partitions the state space with a BBS and verifies the property of the discrete-time stochastic barrier function for each partition leveraging the method in [8]. Our work extends the BBS of [9] to CBFs for continuous-time deterministic control-affine systems where the control input constraints must be considered and use the BBV scheme to verify the learned continuous nCBF.

3

Problem Formulation

3.1. Preliminaries

Given the following continuous-time control-affine system

$$\dot{x} = f(x) + g(x)u, \quad x(0) = x_0, \tag{3.1}$$

where $x \in \mathbb{X} \subset \mathbb{R}^n$, $u \in \mathbb{U} \subset \mathbb{R}^m$, $f : \mathbb{R}^n \to \mathbb{R}^n$ denotes the autonomous dynamics, and $g : \mathbb{R}^n \to \mathbb{R}^{n \times m}$ denotes the input dynamics. We assume that f, g are Lipschitz continuous and $\mathbb{X}$, $\mathbb{U}$ are compact sets.

The safety requirement for the system in (3.1) is encoded via a state admissible set $\mathbb{X}_a \subseteq \mathbb{X}$ and a convex input admissible set $\mathbb{U}_a \subseteq \mathbb{U}$. A safe system stays in the state admissible set for all time. To formally define safety, we use $x_\pi(t; x_0)$ to refer to a trajectory of the system in (3.1) at time t with initial condition x_0 and control policy $u = \pi(x)$. Safety is then defined as:

Definition 1 (Safety)**.** The system in (3.1) is *safe* if $x_\pi(t; x_0) \in \mathbb{X}_a$ and $u = \pi(x_\pi(t; x_0)) \in \mathbb{U}_a$, $\forall t \in [0, \infty]$.

However, it should be noted that $\mathbb{X}_a$ is not safe everywhere as there may not exist a control input that transitions a state close to the boundary towards the interior of $\mathbb{X}_a$. A safe set should have the property that if the system starts in the safe set, it stays inside for all time. Towards formally defining this property, let a set C be defined as the *0-superlevel set* of a continuously differentiable function $h : \mathbb{R}^n \to \mathbb{R}$, *i.e.*,

$$C = \{x \in \mathbb{X} : h(x) \geq 0\},$$
$$\partial C = \{x \in \mathbb{X} : h(x) = 0\}.$$

Then forward invariance and a safe set are defined as follows.

Definition 2 (Foward invariance)**.** The set C is *forward invariant* if for every $x_0 \in C$, there exists a control policy $u = \pi(x) \in \mathbb{U}_a$ such that the trajectory of system in (3.1) $x_\pi(t; x_0) \in C$, $\forall t \in [0, \infty]$.

Definition 3 (Safe set)**.** The set C is a *Safe Set* if C is *forward invariant* and $C \subseteq \mathbb{X}_a$.

A CBF renders a safe set and can be used to derive safe control inputs. Before defining CBFs, we must introduce extended class $\mathcal{K}_\infty$ functions. An extended class $\mathcal{K}_\infty$ function is a mapping $\alpha : \mathbb{R} \to \mathbb{R}$ that is strictly increasing and for which $\alpha(0) = 0$ holds. We define a continuous CBF as:

Definition 4 (Control Barrier Function)**.** Let $C \subseteq \mathbb{X}_a$ be the 0-superlevel set of a continuously differentiable function $h : \mathbb{R}^n \to \mathbb{R}$, then h is a CBF in $\mathbb{X}_a$ for system in (3.1) if there exists an extended class $\mathcal{K}_\infty$ function α such that

$$\sup_{u \in \mathbb{U}_a} [L_f h(x) + L_g h(x) u] \geq -\alpha(h(x)), \tag{3.2}$$

for all $x \in \mathbb{X}_a$, where L_f, L_g represent Lie derivatives.

With the definition of a CBF, we may derive sufficient conditions for a safe system. According to the main result in [1], the following theorem holds:

Theorem 1 ([1, Theorem 2])**.** *If function h is a CBF for the system in* (3.1) *and $\frac{\partial h}{\partial x}(x) \neq 0$ for all $x \in \partial C$, then any Lipschitz continuous controller $\pi(x) \in K_{cbf}(x)$ with*

$$K_{cbf}(x) = \{u \in \mathbb{U}_a : L_f h(x) + L_g h(x)u + \alpha(h(x)) \geq 0\}, \tag{3.3}$$

renders the set C safe. Additionally, the set C is asymptotically stable in $\mathbb{X}_a$.

Suppose a feedback controller $u = \pi_{nominal}(x)$ is given, we can consider the following QP based safe controller that finds the minimum perturbation on u to guarantee safety for the system in (3.1):

$$\begin{aligned} u_{\text{safe}} = &\arg\min_{u \in \mathbb{U}_a} \|u - \pi_{nominal}(x)\|^2 \\ &s.t.\ L_f h(x) + L_g h(x)u + \alpha(h(x)) \geq 0. \end{aligned} \tag{3.4}$$

3.2. Neural Control Barrier Function

With Theorem 1, we can ensure the safety of the system in (3.1) as long as a CBF is found and its gradient does not vanish on ∂C. Then, the objective of this work is to automatically synthesize an nCBF and verify it for the continuous state space. The problem is defined as follows.

Problem 1. *Given the system in* (3.1)*, state admissible set $\mathbb{X}_a$, convex input admissible set $\mathbb{U}_a$ and $\alpha(x) = \gamma x$ where γ is a positive constant, synthesize an nCBF denoted by $\hat{h}_w(x)$, where w are the parameters of the NN, and renders set C safe for the system in* (3.1)*. This is equivalent to*

$$\hat{C} \subseteq \mathbb{X}_a, \tag{3.5a}$$

$$\textit{inequality } (3.2) \textit{ holds in } \mathbb{X}_a, \tag{3.5b}$$

where $\hat{C} = \{x \in \mathbb{X} : \hat{h}_w(x) \geq 0\}$ is the 0-superlevel set of the nCBF.

Remark 1. The condition $\frac{\partial h}{\partial x}(x) \neq 0$ for all $x \in \partial C$ is omitted since it generally holds in our setting as we only consider a Tanh-based Fully-Connected Neural Network (FCNN). More specifically, since $\frac{\partial tanh}{\partial x}(x) \in (0, 1]$ for all x, the condition is only violated if either $w = 0$ or catastrophic cancellation occurs in the linear layers, which will almost surely never happen.

4

Neural Control Barrier Function Training and Verification

In this work, we design a new empirical loss to synthesize an nCBF, which is introduced in Section 4.1. As the training set only contains a finite set of data points, the CBC may not hold in the continuous state space. Therefore, in Section 4.2, we present the BBV to verify nCBFs. Nevertheless, it is often necessary to iterate through multiple training and verification cycles before successfully learning an nCBF. Thus, we introduce the BBVT in Section 4.3, which embeds the BBV in the training loop to accelerate training for certifiability.

4.1. Learning a Neural Control Barrier Function

The primary goal of this work is to train an NN $\hat{h}_w(x)$ until it satisfies conditions (3.5a) and (3.5b) and render a large forward invariant set. Towards this end, we leverage the main result in [7, Theorem 3], where a CBVF is shown to recover the maximum safe set subject to safety constraints. Contrary to [7], we are interested in infinite-horizon properties. Thus we extend the time-dependent Control Barrier-Value Function Variational Inequality (CBVF-VI) to the infinite-horizon. Let $h(x)$ denote the infinite-horizon CBVF and $\rho(x) : \mathbb{X} \to \mathbb{R}$ denote the signed-distance function for the set $\mathbb{X}_a$, which is defined as $\rho(x) = \inf_{y \in \mathbb{X}/\mathbb{X}_a} \|y - x\|$ if $x \in \mathbb{X}_a$ and $\rho(x) = -\inf_{y \in \mathbb{X}_a} \|y - x\|$ if $x \in \mathbb{X}/\mathbb{X}_a$. The infinite-horizon CBVF-VI is defined as

$$\begin{aligned} 0 = \min\{&\rho(x) - h(x), \\ &\max_{u \in \mathbb{U}_a} L_f h(x) + L_g h(x) u + \gamma h(x)\}. \end{aligned} \tag{4.1}$$

We use an NN $\hat{h}_w(x)$ to approximate the infinite-horizon CBVF $h(x)$. Then, the empirical loss is defined as follows:

$$\begin{aligned} \mathcal{L} = &\frac{1}{N_1} \sum_{x \in \mathbb{X}_a} \|\min\{\rho(x) - \hat{h}_w(x), \\ &\sup_{u \in \mathbb{U}_a} L_f \hat{h}_w(x) + L_g \hat{h}_w(x) u + \gamma \hat{h}_w(x) - \lambda\}\| && \text{(4.2a)} \\ &+ \frac{1}{N_2} \sum_{x \in \mathbb{X}/\mathbb{X}_a} \max\{\hat{h}_w(x) + \lambda, 0\}. && \text{(4.2b)} \end{aligned}$$

where λ is a small positive constant to encourage the strict satisfaction of the conditions. The loss term (4.2a) shapes the NN to be the solution of the infinite-horizon CBVF-VI introduced in (4.1), which

encourages the satisfaction of condition (3.5b). The loss term (4.2b) ensures that the nCBF is negative in the inadmissible area $\mathbb{X}/\mathbb{X}_a$, which is equivalent to condition (3.5a). Since the system is control-affine, the optimal solution u^* for $\sup_{u\in\mathbb{U}_a}[L_f\hat{h}_w(x) + L_g\hat{h}_w(x)u]$ must be one of the vertices of $\mathbb{U}_a$. Let $\mathbb{U}_a^V$ denote the vertices of the input admissible set, we choose control input $u^\star = \arg\max_{u\in\mathbb{U}_a^V} L_g\hat{h}_w(x)u$. However, the Lie derivative of $\hat{h}_w(x)$ in the early training stage may not align with the Lie derivative of the true CBVF $h(x)$. This results in an undesirable optimization path and the NN can get stuck at deadlock. The occurrence of a deadlock situation signifies that improvements at certain data points cause constraint violations at other data points, as noted in [79]. To facilitate the training process and avoid deadlocks, we borrow ideas from [5, 4], which use a nominal controller to guide the training. Here, we train another neural network $\hat{h}_\phi$ with the same structure as $\hat{h}_w$ based on the loss of [42] and choose $u^* = \arg\max_{u\in\mathbb{U}_a^V} \hat{h}_\phi(x + (f(x) + g(x)u)\Delta t)$ by simulating one step ahead to guide the training of the nCBF $\hat{h}_w$.

4.2. Verifying the Learned Neural Control Barrier Function

Since the NN is trained on finite data points, one must note that the NN may not satisfy the CBF conditions everywhere in the state space, even if the empirical loss decreases to zero. In fact, condition (3.5b) may be violated almost everywhere, which means the NN may fail to render a forward invariant set and the safety guarantee no longer exists. In this section, we propose to use the BBV to verify the learned nCBF in the continuous state space. Specifically, our primary goal is to verify the satisfaction of conditions (3.5a) and (3.5b).

Before we explain our verification scheme in detail, we introduce some notations first. Let the partition of the state space be denoted as hyperrectangles $\mathbb{B}(x_i, \epsilon_i) = \{x : \mid x - x_i \mid \leq \epsilon_i\}$ centered at point $x_i \in \mathbb{X}$ with radius $\epsilon_i \in \mathbb{R}^n$, see Fig. 4.1. Initially, all hyperrectangles have the same radius $\epsilon_i = \epsilon_{init}$. Let $\mathcal{B} = \{\mathbb{B}(x_0, \epsilon_0), \ldots, \mathbb{B}(x_N, \epsilon_N)\}$ denote the set of all hyperrectangles, $\mathcal{B}_{\mathbb{X}/\mathbb{X}_a} \subset \mathcal{B}$ denote the set of hyperrectangles that covers the inadmissible area $\mathbb{X}/\mathbb{X}_a$, and $\mathcal{B}_{\mathbb{X}_a} \subset \mathcal{B}$ denote the set of hyperrectangles that covers the admissible area.

To verify condition (3.5a), which is equivalent to $\hat{h}_w(x) < 0, \forall x \in \mathbb{X}/\mathbb{X}_a$, we rely on the linear bounds of the NN computed using CROWN [8]. The linear bounds are defined as follows:

$$\hat{h}_{li} \leq \hat{h}_w(x) \leq \hat{h}_{ui},\ x \in \mathbb{B}(x_i, \epsilon_i). \tag{4.3}$$

We use these linear bounds to certify the satisfaction of condition (3.5a). In particular, the upper bound $\hat{h}_{ui}$ can be used to check for non-positivity

$$\hat{h}_w(x) \leq \hat{h}_{ui} \leq 0, x \in \mathbb{B}(x_i, \epsilon_i), \mathbb{B}(x_i, \epsilon_i) \in \mathcal{B}_{\mathbb{X}/\mathbb{X}_a}. \tag{4.4}$$

However, this upper bound tends to be conservative when $\mathbb{B}(x_i, \epsilon_i)$ covers a large area. Therefore, we leverage the BBS that starts from the coarse partition and refines each hyperrectangle when $\hat{h}_{ui} > 0$ until $\hat{h}_{ui} \leq 0$ or $\epsilon_i \leq t_{gap}$, where $t_{gap} > 0$ is the minimum partition size, see Fig. 4.1. The refinement is done by splitting the state space in half in each dimension, see Algorithm 1. If condition (4.4) holds for all hyperrectangles in $\mathcal{B}_{\mathbb{X}/\mathbb{X}_a}$, then the condition (3.5a) holds in the continuous state space.

Although verifying condition (3.5a) is simple, verifying condition (3.5b) $\sup_{u\in\mathbb{U}_a}[L_f\hat{h}_w(x) + L_g\hat{h}_w(x)u] \geq -\gamma\hat{h}_w(x),\ \forall x \in \mathbb{X}_a$ is challenging. For improved readability, we denote $q(x) = \sup_{u\in\mathbb{U}_a}[L_f\hat{h}_w(x) + L_g\hat{h}_w(x)u + \gamma\hat{h}_w(x)]$. Hence, verifying condition (3.5b) is equivalent to verifying $q(x) \geq 0, \forall x \in \mathbb{X}_a$. Let q_{li} define a lower bound of $q(x)$ for $x \in \mathbb{B}(x_i, \epsilon_i)$. Then the following condition has to hold:

$$q(x) \geq q_{li} \geq 0,\ x \in \mathbb{B}(x_i, \epsilon_i), \mathbb{B}(x_i, \epsilon_i) \in \mathcal{B}_{\mathbb{X}_a}. \tag{4.5}$$

Similarly to condition (3.5a), the BBS starts from a coarse partition and refines each hyperrectangle when $q_{li} < 0$ until $q_{li} \geq 0$ or $\epsilon_i \leq t_{gap}$. If condition (4.5) holds for all hyperrectangles in $\mathcal{B}_{\mathbb{X}_a}$, then condition (3.5b) holds in the continuous state space.

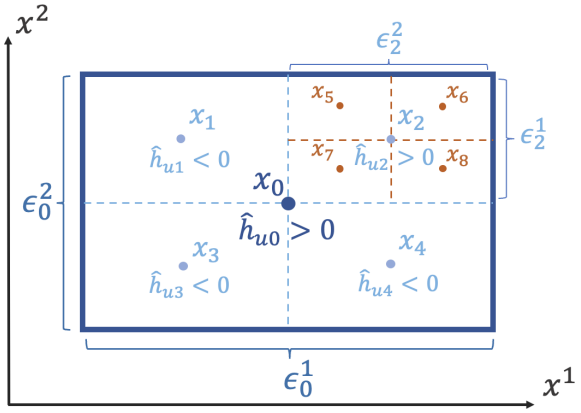


Figure 4.1: An example of the BBV in a 2D state space. The scheme starts with a coarse partition $\mathbb{B}(x_0, \epsilon_0)$ and refines it using the Branch-and-Bound scheme. For each hyperrectangle $\mathbb{B}(x_i, \epsilon_i)$, $i = 0, 1, 2, \ldots$, upper bounds for the neural network are computed. In this case, the hyperrectangles $\mathbb{B}(x_0, \epsilon_0)$ and $\mathbb{B}(x_2, \epsilon_2)$ are refined as $\hat{h}_{u0} > 0$, $\hat{h}_{u2} > 0$.

Algorithm 1 Branch and Bound Scheme (One Iteration)

Input:
System dimension n
Initial hyperrectangles $\mathcal{B} = \{\mathbb{B}(x_0, \epsilon_{init}), \ldots \mathbb{B}(x_N, \epsilon_{init})\}$
2^n offset directions: $offset\_dirs$
function BranchAndBound($\mathcal{B}$)
  for $\mathbb{B}(x_i, \epsilon_i)$ in $\mathcal{B}$ **do**
    if $\mathbb{B}(x_i, \epsilon_i) \in \mathcal{B}_{\mathbb{X}/\mathbb{X}_a}$ **then**
      Get $\hat{h}_{ui}$ from (4.3)
    else
      Get q'_{li} from (4.10a)
    end if
    if $\hat{h}_{ui} \geq 0$ or $q'_{li} < 0$ **then**
      for dir in $offset\_dirs$ **do**
        $x_{new} \leftarrow x_i + 0.25 * dir * \epsilon_i$
        $\epsilon_{new} \leftarrow 0.5 * \epsilon_i$
        $\mathcal{B}.add(\mathbb{B}(x_{new}, \epsilon_{new}))$
      end for
    end if
    $\mathcal{B}.discard(\mathbb{B}(x_i, \epsilon_i))$
  end for
end function

However, the challenge arises in the computation of q_{li}. The computation of q_{li} can be reframed as an optimization problem within the hyperrectangle $\mathbb{B}(x_i, \epsilon_i)$

$$q_{li} = \min_{x} \; q(x) \tag{4.6a}$$
$$\text{s.t. } x \in \mathbb{B}(x_i, \epsilon_i). \tag{4.6b}$$

The term $q(x)$ is a complex function containing nonlinear dynamic functions f, g, a neural network $\hat{h}_w$ as well as its Jacobian, which renders a constrained Nonlinear Program (NLP) in (4.6a). The state-of-the-art NLP solver [80] requires gradients of the objective function, which involves computation of the Hessian of the NN. The expensive computation makes it impractical to solve (4.6a) directly.

Although computing the lower bound of $q(x)$ is quite complex, computing the bound of the components of $q(x)$ separately is much simpler. We can compute the bound of the NN using CROWN [8] and its Jacobian leveraging a recent result in [59] or [77]:

$$\hat{h}_{li} \leq \hat{h}_w(x) \leq \hat{h}_{ui}, \forall x \in \mathbb{B}(x_i, \epsilon_i), \tag{4.7}$$
$$J_{li} \leq \nabla \hat{h}_w(x) \leq J_{ui}, \forall x \in \mathbb{B}(x_i, \epsilon_i). \tag{4.8}$$

Furthermore, we can approximate the nonlinear dynamic functions f and g using Taylor Models as done in [81] or sampling:

$$x_{li} \leq f(x) + g(x)u^\star \leq x_{ui}, \forall x \in \mathbb{B}(x_i, \epsilon_i). \tag{4.9}$$

In (4.6a), the objective function depends on the variable x and is constrained within the feasible region for x. We simplify (4.6a) by considering three independent variables subject to independent constraints. This results in

$$q'_{li} = \min_{h,J,x} q'(h, J, x) = \langle J, x \rangle + \gamma h \tag{4.10a}$$
$$\text{s.t. } \hat{h}_{li} \leq h \leq \hat{h}_{ui}, \tag{4.10b}$$
$$J_{li} \leq J \leq J_{ui}, \tag{4.10c}$$
$$x_{li} \leq x \leq x_{ui}, \tag{4.10d}$$

where x denotes the value of $f(x) + g(x)u$, h denotes the value of $\hat{h}_w(x)$ and J denotes the value of $\nabla \hat{h}_w(x)$. When (4.7), (4.8), and (4.9) are over-approximations of the true intervals, it is clear that the optimal solution q'_{li} from (4.10a) is an over-approximation of the optimal solution q_{li} from (4.6a), which means $q'_{li} \leq q_{li}$. To efficiently solve (4.10a), we may compute the optimal solution independently for each term, taking the minimum over the set of vertices.

Although the theoretical complexity of the BBV is still exponential in the dimension of the state space, it improves the scalability in practice. One must note that our method is a sound verification method instead of a complete one, which means the failure to obtain $\mathcal{B}_{\mathbb{X}/\mathbb{X}_a}$ and $\mathcal{B}_{\mathbb{X}_a}$ that satisfy condition (4.4), (4.5) does not imply the invalidation of the nCBF, as we over-approximate the conditions. We want to emphasize that the chosen over-approximation method, CROWN [59], has been the winning strategy at the Verification of Neural Networks Competition for multiple years [82].

4.3. Branch and Bound Verification-in-the-loop Training

Although the BBV provides a practical way to certify the NN as an nCBF, it requires several training and verification processes until an nCBF is obtained. Therefore, leveraging the information from the verification and ensuring the satisfaction of conditions (3.5a) and (3.5b) becomes the task of BBVT. This

Algorithm 2 Branch-and-Bound Verification-in-the-loop Training

Input:
Neural Network $\hat{h}_w$, Training Dataset $\mathcal{D}$, Maximum Iterations n_{max}
loss function $\mathcal{L}$ in (4.2) , learning rate α , minimum partition size t_{gap} , verify after k epochs
Initialize:
`satisfaction` $\leftarrow$ False
function MAIN
  while not `satisfaction` and n_{max} **do**
    for i in k **do**
      for x in $\mathcal{D}$ **do**
        $w \leftarrow w - \alpha \nabla_w \mathcal{L}(x, w)$
      end for
    end for
    $n_{max} \leftarrow n_{max} - 1$
    initialize hyperrectangles $\mathcal{B}$
    while not reach t_{gap} **do**
      run BanchAndBound($\mathcal{B}$) in Algorithm. 1
      if all $\hat{h}_{ui} < 0$ and $q'_{li} \geq 0$ **then**
        `satisfaction` $\leftarrow$ True
        break
      end if
    end while
    $\mathcal{D}.augment(\mathcal{B})$
  end while
  return $\hat{h}_w$
end function

type of method is also known as the CE guided inductive synthesis [83]. See Fig. 1.2 for an overview of the framework.

We start with the initial fixed training dataset $\mathcal{D}$ that contains a number of uniformly sampled points. During the training procedure, we optimize the NN to decrease the loss in (4.2) using $\mathcal{D}$. After k epochs, the verifier starts with a coarse partition of the state space. The upper bound $\hat{h}_{ui}$, $\forall\, \mathbb{B}(x_i, \epsilon_i) \in \mathcal{B}_{\mathbb{X}/\mathbb{X}_a}$ and lower bound q'_{li}, $\forall\, \mathbb{B}(x_i, \epsilon_i) \in \mathcal{B}_{\mathbb{X}_a}$ are computed. The hyperrectangles, whose $\hat{h}_{ui} \geq 0$ or $q'_{li} \leq 0$, are split until $\epsilon_i \leq t_{gap}$. After reaching the minimum partition size t_{gap}, the hyperrectangles whose $\hat{h}_{ui} \geq 0$ or $q'_{li} \leq 0$ are treated as the violation areas. The centre points are added to the CE dataset and the training procedure is repeated until the verifier returns `satisfaction` or the maximum number of iterations $n_{\max}$ is reached, see Algorithm 2.

Note that although the universal approximation theorem in [84] guarantees the existence of $\hat{h}_w(x)$ to be an nCBF that renders maximum forward invariant set, this is under the assumption that the NN has a sufficient number of neurons. Thus, in practice, the training procedure is not guaranteed to converge to an nCBF, but if the verifier returns `satisfaction`, the NN is an nCBF for the given system in the continuous state space.

5 Results

In this section, we evaluate our proposed framework on two systems: an inverted pendulum and a 2D navigation task. The experimental setup is introduced in Section 5.1. In Section 5.2 and Section 5.3, we provide a comprehensive assessment on the inverted pendulum, addressing the verification efficiency and the size of the safe set, respectively. In Section 5.4, we consider a 2D navigation task with nonconvex constraints to display the practical use of our framework and combine the learned nCBF with RL to achieve safe learning. Our code is available on GitHub `https://github.com/tud-amr/ncbf-simultaneous-synthesis-and-verification`.

We consider the following baseline methods:

- LST: The Level Set Toolbox (LST) [33] generates a safe value function by HJI-RA over a discrete grid.
- NeuralCLBF: Neural Control Lyapunov Barrier Function (NeuralCLBF) [4] parametrizes the CBF as an NN and optimizes it according to their empirical loss based on (3.2) and a nominal safe set.
- SMT: [5] trains a neural Lyapunov function with the SMT generating CEs and ensures the validation of the result. The constraints considered by the SMT are conditions (3.5a) and (3.5b). To have a fair comparison, the training loss is chosen to be the same as in (4.2).

5.1. Experimental Setup

5.1.1. Inverted Pendulum

Let $s = [\theta, \dot{\theta}] \in \mathbb{X} \subset \mathbb{R}^2$ be the state variable and $u \in \mathbb{U} \subset \mathbb{R}$ be the control input. We consider the state space $\mathbb{X} = \{s : \theta \in [-\pi, \pi], \dot{\theta} \in [-5, 5]\}$ and the input space $\mathbb{U} = \{u : u \in [-12, 12]\}$. The dynamics of

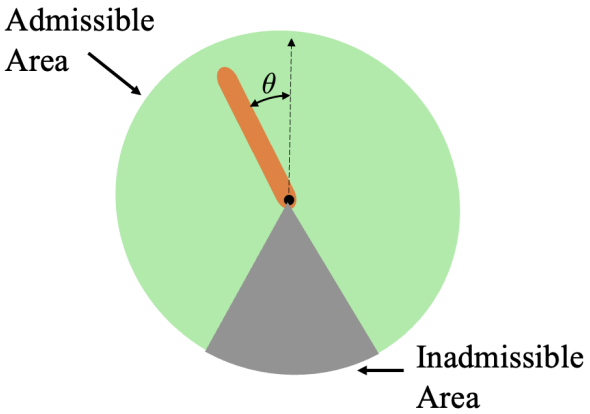


Figure 5.1: The workspace of the considered inverted pendulum.

Table 5.1: Hyper-parameter for the nCBF Training.

γ	0.5	λ	0.05
learning rate r	10^{-3}	learning rate decay β	0.995
verify after every k epochs	20	minimum partition gap t_{gap}	0.005
initial radius ϵ_{init} (inverted pendulum)	[0.2, 0.2]	initial radius ϵ_{init} (2D navigation)	[0.2, 0.2, 0.2, 0.2]
Num. fixed points (inverted pendulum)	10^5	Num. fixed points (2D navigation)	10^6
n_{max}	100		

the inverted pendulum are given by:

$$\begin{aligned} \dot{\theta} &= \dot{\theta}, \\ \ddot{\theta} &= \frac{3g}{2l} sin(\theta) - \frac{3\beta}{ml^2}\dot{\theta} + \frac{3}{ml^2}u, \end{aligned} \tag{5.1}$$

where $m = 1$, $b = 0.1$, $g = 9.81$, and $l = 1$. The state admissible set is $\mathbb{X}_a = \{s : \theta \in [-\frac{5\pi}{6}, \frac{5\pi}{6}], \dot{\theta} \in [-4, 4]\}$ and the input admissible set is $\mathbb{U}_a = \mathbb{U}$, see Fig. 5.1.

5.1.2. 2D Navigation Task

We consider a 2D navigation task in which a point robot should reach a goal position while avoiding obstacles, see Fig. 5.6a. Let $s = [x, y, \dot{x}, \dot{y}] \in \mathbb{X} \subset \mathbb{R}^4$ be the state variable and $u = [a_x, a_y] \in \mathbb{U} \subset \mathbb{R}^2$ be the control input representing the acceleration along the x-axis and y-axis. The dynamics of the point robot are:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \ddot{x} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} s^T + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a_x \\ a_y \end{bmatrix}. \tag{5.2}$$

We consider the admissible position set $X^1 = \{s : x \in [0, 4], y \in [0, 4]\}$ except the obstacle set $X^2 = \{s : x \in [1.5, 2.5], y \in [0, 2]\}$, together with velocity constraints $X^3 = \{s : \dot{x} \in [-1, 1], \dot{y} \in [-1, 1]\}$. Thus, the state admissible set is $\mathbb{X}_a = X^1 \cup (X^2)^C \cup X^3$, where $(\cdot)^C$ represents the complement of a set. See Fig. 5.6 for a pictorial representation of the set. The input admissible set is $U_a = \{u : a_x \in [-1, 1], a_y \in [-1, 1]\}$.

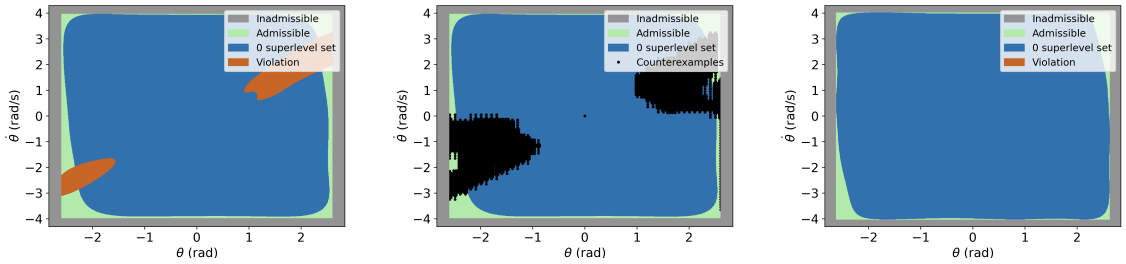
5.1.3. Training Configuration

For both systems, we train the nCBF using Pytorch on NVIDIA A40, and Stochastic Gradient Descent is used as the optimizer to avoid local minima. The used hyper-parameters can be found in Table 5.1, where γ and λ are parameters in loss function (4.2). For the inverted pendulum, we choose a Tanh-based FCNN with one hidden layer which consists of 36 neurons. For the 2D navigation task, a larger Tanh-based FCNN is required since the shape of the environment is more complex. Here we choose a Tanh-based FCNN with two hidden layers, each of which consists of 256 neurons.

5.2. Verification and Efficiency

In this section, we use the inverted pendulum to discuss the certification of the trained NN as an CBF. To showcase the disadvantage of training without verification, we train the nCBF with a fixed data set and stop training the nCBF after 200 epochs. We then examine the satisfaction of condition (3.5b) with a denser testing dataset. The 0-superlevel set of the trained NN is shown in blue in Fig. 5.2a. The orange area indicates the testing data points that violate condition (3.5b).

We resume the training with the same dataset and use the BBVT to augment the training dataset with CEs every k epochs until the verifier returns `satisfaction`. Figure. 5.2b shows the distribution of the CEs after the first verification loop. As we augment the dataset, the verifier returns `satisfaction` after 240 epochs, see Fig. 5.2c.



(a) Training without verification-in-the-loop **(b)** Distribution of counterexamples **(c)** Training with verification-in-the-loop

Figure 5.2: Shapes of 0-superlevel sets of the NNs trained with and without the BBVT for the inverted pendulum. In Fig. 5.2a the NN is trained with a fixed dataset and evaluated on a denser testing dataset to showcase that condition (3.5b) is not satisfied for the continuous state space. Figure. 5.2b shows the counterexamples added to the dataset according to the BBVT. Figure. 5.2c showcases that, after training the NN with the BBVT, no validations are detected since the NN is an nCBF.

Table 5.2: Verification and efficiency comparison for the inverted pendulum. The BBVT is compared against LST, NeuralCLBF, and the SMT to synthesize an nCBF. LST and NeuralCLBF do not verify their safe value function, which is represented by '-' in columns 3 and 4. To validate the verification process, we calculate the ratio of points that violate condition (3.5b) on a uniform grid with a size of $10^3 \times 10^3$ within the state space $\mathbb{X}$.

	Stop criteria	Total computation time (s)	Average verification time (s/epoch)	Average generation time (s/per counterexample)	Violation points/testing points (%)
LST(0.2)	value converges	5.34	-	-	1.9
LST(0.05)	value converges	104.48	-	-	0.0064
LST(0.02)	value converges	1075.38	-	-	0.0007
NeuralCLBF	loss converges	584.6	-	-	0.0013
SMT	max # iter reached	5311.68	14.73	1.34	0.7742
BBVT(ours)	verified	1214.15	16.20	0.004	0.0

To highlight the efficacy of BBVT, we evaluate the training time, verification time, and the ratio of violation areas for our framework and the baseline methods. The results are shown in Table 5.2. We first compare our method with LST [33]. The table shows the results of LST for three different grid gaps, which are 0.2 , 0.05 and 0.02 respectively. It is evident that an increased grid density leads to improved accuracy at the cost of longer computation time. However, a dense grid map is not always possible, since the memory space of LST grows exponentially, which is referred to as the *Curse of Dimensionality*. With a grid gap of 0.02, LST requires 153.39kB memory space, while we only need to store the parameters of the nCBF, which is 1.2kB. This is important for embedded devices such as the control unit on drones.

Then, we compare our method with NeuralCLBF. Due to the lack of a verification process and counterexample data set, the fixed data set for NeuralCLBF contains 10^6 data points in order to have a fair comparison with our method. Since NeuralCLBF learns an nCBF based on a nominal safe set, the training process is assisted by prior knowledge and results in less training time, see Table 5.2. However, there are sparse areas that violate the conditions as discussed in [4] and how these sparse areas grow with the complexity of the system has not been studied yet.

We also compare our method with the SMT. However, the SMT did not return `satisfaction` until the maximum number of iterations $n_{\max}$ was reached, see Table 5.2. Although there exist some works [5, 6] that use the SMT to verify a neural controller, they only use a very simple FCNN with around 9 neurons. In our case, the computation time of the SMT grows dramatically since the NN is more complex. Also, the SMT can only generate several counterexamples at each iteration, while BBVT generates all the counterexamples in state space $\mathbb{X}$, which is more efficient than SMT.

5.3. Size of Safe Set

We will compare the size of the forward invariant set derived using our framework and the baseline methods in this section. Since the SMT failed to verify the nCBF and LST with a grid gap of 0.2 has a large violation area, we compare our framework only against LST with a grid gap of 0.05 and NeuralCLBF with the nominal safe set being $\mathbb{X}_n = \{s : \|s\| < \frac{3\pi}{4}\}$).

The forward invariant sets derived by the different methods are illustrated in Fig. 5.3. We see that the size of the forward invariant set from NeuralCLBF is conservative, while our method approximates

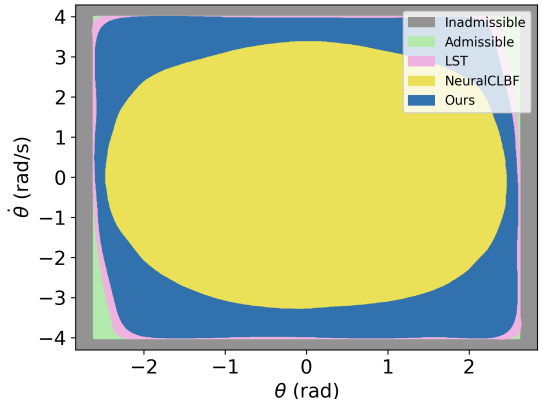


Figure 5.3: The forward invariant set of safe value functions obtained by different methods for the inverted pendulum.

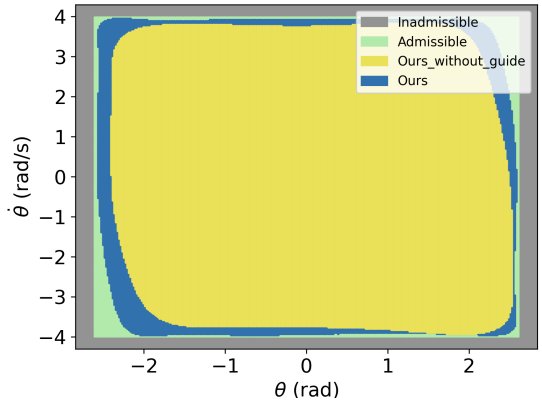


Figure 5.4: The 0-superlevel sets of nCBFs obtained by training with and without a nominal controller.

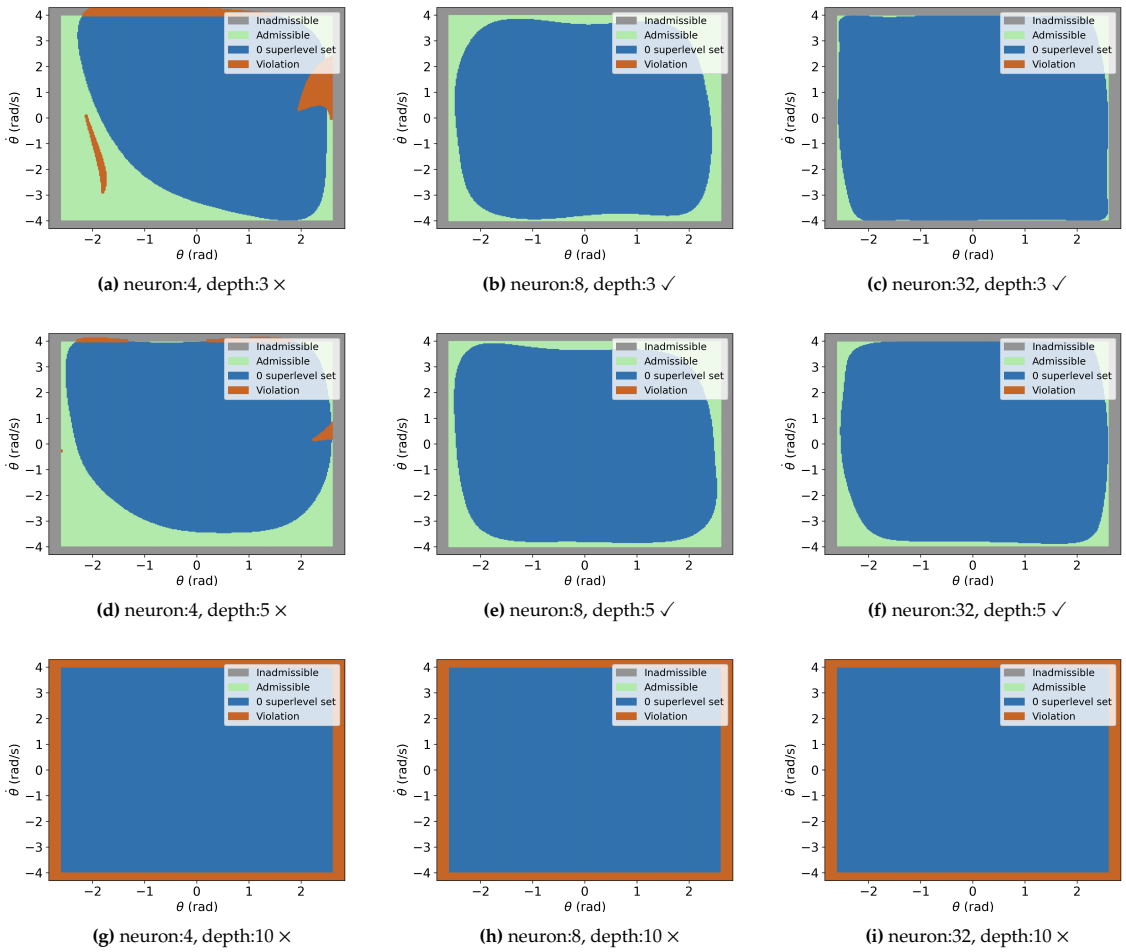
the CBVF and renders a safe set that is close to the maximum forward invariant set. Since the forward invariant set of the CBF is always a subset of that from HJI-RA, which is discussed in [7], it is not surprising that LST renders a larger safe set than ours. We note that $\lambda > 0$ in (4.2a) encourages the satisfaction of the CBF conditions at the expense of rendering a smaller safe set.

5.3.1. Impact of Nominal Controller

In Section 4.1, we mentioned that we borrow ideas from [4, 5], which use a nominal controller to facilitate the training process and avoid deadlocks. Here, we train another NN $\hat{h}_\phi$ based on the loss of [42] and choose $u^* = \arg\max_{u \in \mathbb{U}_a^V} \hat{h}_\phi(x + (f(x) + g(x)u)\Delta t)$ to be the nominal controller and guide the training of the nCBF $\hat{h}_w$. Figure. 5.4 shows the training results with and without a nominal controller. Without a nominal controller, it took a longer time to learn an nCBF, and the training process became trapped in a local minimum resulting in a smaller control invariant set. One must note that a badly chosen nominal controller can lead to an incorrect optimization direction, which makes training unstable, while deadlocks happen occasionally when no nominal controller guides the training. Therefore, there is a tradeoff between using a nominal controller or not.

5.3.2. Impact of Neural Network Architecture

Figure. 5.5 shows the training results for different NN architectures but using the same hyperparameters as stated in Table 5.1. Among the groups with the same depth, the NN with more neurons in each layer has more compacity to fit the shape of the nCBF and renders a larger safe set. However, increasing the depth of the NN harms the training. Stacking more layers does not help with training due to the



(a) neuron:4, depth:3 × **(b)** neuron:8, depth:3 ✓ **(c)** neuron:32, depth:3 ✓

(d) neuron:4, depth:5 × **(e)** neuron:8, depth:5 ✓ **(f)** neuron:32, depth:5 ✓

(g) neuron:4, depth:10 × **(h)** neuron:8, depth:10 × **(i)** neuron:32, depth:10 ×

Figure 5.5: Shapes of 0-superlevel sets of NNs with different depth and width. Number of the neurons indicates the width of each hidden layer. The depth of the NN refers to the number of layers it contains, including input, hidden, and output layers. × and ✓ represent whether the verifier returns `satisfaction` after reaching the maximum epochs.

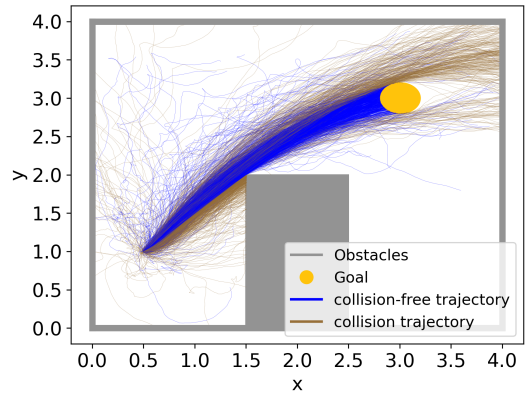
notorious problem of vanishing/exploding gradients [85]. We observe that the NN with 10 layers can not converge from the beginning. To address such a problem, normalized initialization [86] and intermediate normalization layers [87] can be augmented into our framework in the future. Furthermore, there are recently advanced architectures that enable training extremely large and deep NNs, such as ResNet [85] and Transformer [88]. However, the mainstream NN verification tools [59, 77] only accept plain NNs. Verifying these advanced NNs remains an open area.

5.4. Application of Neural Control Barrier Functions to Safe Policy Learning

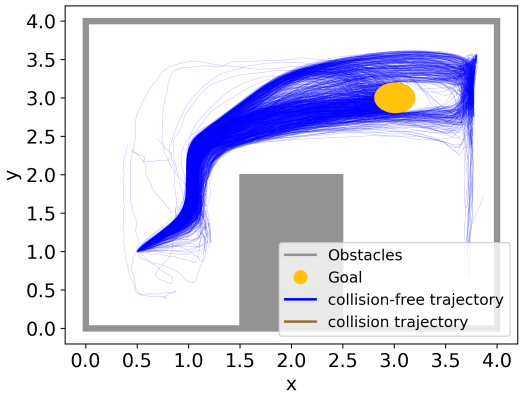
In this section, we use RL to address the 2D navigation task introduced in Section 5.1.2. Let $s_g = [x_g, y_g, 0, 0]$ be the goal state. The step reward is defined as $r_t = -0.01 * \|s - s_g\|$, the terminal reward is $r_{\text{collision}} = -5$ when the robot collides with the obstacles and $r_{\text{goal}} = 10$ when the robot reaches the goal area $X_g = \{s : \|s - s_g\| < \epsilon\}$ where $\epsilon = 0.1$ is the goal tolerance. We use Proximal Policy Optimization [89] to train the agent. To ensure safety during the training, we solve

$$\begin{aligned} u_{\text{safe}} &= \arg\min_{u \in \mathbb{U}_a} \|u - u_{\text{RL}}\|^2 \\ s.t.\ & L_f \hat{h}_w(x) + L_g \hat{h}_w(x) u + \alpha(\hat{h}_w(x)) \geq 0, \end{aligned} \tag{5.3}$$

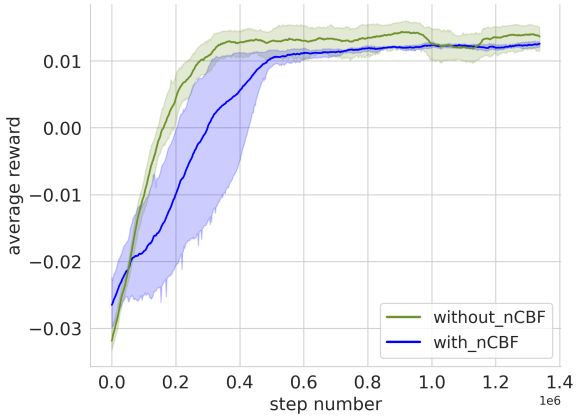
to project the action u_{RL} of the RL policy to the safe action u_{safe} with the least modification.



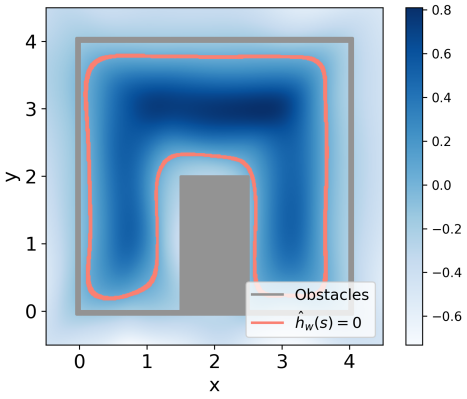
(a) Trajectories during learning without the nCBF



(b) Trajectories during learning with the nCBF



(c) Moving average reward



(d) Heatmap of the nCBF.

Figure 5.6: Training Results for 2D Navigation Task. Figure. 5.6a and Fig. 5.6b show all the trajectories during the RL training. Figure. 5.6c illustrates the moving average reward of every 2048 steps. Figure. 5.6d is the slice of heatmap of $\hat{h}_w(x)$ with velocity $\dot{x} = 0.2, \dot{y} = 0.2$.

We note that, theoretically, this controller guarantees safety with infinite control frequency. However, a continuous controller is not possible to implement on discrete control units. This limits the safety guarantees we may provide. How to address the gap between continuous controllers and their discrete implementations remains an open question. Figure. 5.6a shows all trajectories performed during the training. We can see that several trajectories collide with the obstacles. Note, that the learned policy is not guaranteed to be safe. Figure. 5.6b shows all the training trajectories with the nCBF as a safety filter and no trajectories are colliding with the obstacles. However, we observe that the average reward with the nCBF is larger than for nominal RL without the nCBF in the very early stage but has a slower growth rate and converges to a lower reward level compared with the nominal RL, see Fig. 5.6c. The reason is that the nCBF provides prior knowledge about the environment and the agent could avoid exploring unsafe regions in the early stage and gain a higher reward than the nominal RL. However, the forward invariant set is still suboptimal as discussed in Section 5.3, which means only a suboptimal policy is learned and exploration is restricted. Nevertheless, we believe that provided safety guarantees are beneficial in safety-critical applications.

6

Conclusion

In this chapter, we summarize the method and the results of this thesis as well as the limitations and future research directions.

6.1. Summay

This thesis has addressed the critical challenges in ensuring the safety of autonomous systems under the framework of the Certified Learning (CL). Particularly, we focused on the synthesis and verification of Neural Control Barrier Functions (nCBFs), which guarantee safety through set invariance. Two research questions are addressed throughout this research.

RQ1: How can we find an nCBF that renders the maximum safe area? No general and scalable technique exists for designing Control Barrier Functions (CBFs). Many works rely on the initial guess of function structure or forward invariant set. An improper initial guess usually results in a suboptimal CBF. In this thesis, a newly derived loss function leads to an nCBF which renders a large safe area close to the maximum safe set. Although our method renders a larger safe area than baseline methods, the size of the safe set depends on the proper choice of Neural Network (NN) architectures.

RQ2: How can an nCBF be verified efficiently? Although utilizing NNs as CBFs offers universal approximation capabilities, it necessitates their certification as CBFs to provide safety guarantees. Some works use the Satisfiability Modulo Theory (SMT) to verify the NNs. However, they are restricted to very simple NNs due to expensive computation. In this thesis, we presented a framework that simultaneously synthesizes and verifies continuous nCBFs. To this end, we leveraged bound propagation techniques and the Branch-and-Bound scheme to efficiently verify NNs as CBFs in the continuous state space. In experiments, we showed that our framework verified the nCBF much more efficiently than other state-of-the-art methods. We hope that our framework could have important implications for the deployment of neural controllers in real-world applications.

6.2. Limitations and Future work

While our method provides a systematic way to synthesize and verify nCBFs, several limitations hinder its application to high-dimensional systems and real-world experiments. We categorize these drawbacks into three areas: scalability, safety guarantee, and optimality.

6.2.1. Scalability

The memory requirements and computation time of the Branch-and-Bound Verification scheme still increase almost exponentially with the system dimension. That is the tradeoff we face for exact verification of an NN. In future work, we aim to address the scalability of our framework by implementing a smarter sampling scheme. For instance, the approaches in [90, 91] use evolutionary algorithms to augment training datasets, thereby improving the accuracy of image classifiers. Other works in [79, 92] borrowed the concept of adversarial training [93]. In these approaches, researchers randomly initialize

the counterexamples and push them in the direction of breaking the Control Barrier Condition (CBC). These methods offer potential avenues for enhancing the scalability and efficiency of our framework in future research endeavors.

Instead of pursuing exact verification, an alternative research direction could involve providing only a probability safety guarantee to avoid the need for a massive amount of sampling. The researchers in [94] use the Gaussian Process (GP) to model the properties of an NN and generate counterexamples with low confidence or verify the absence of counterexamples with a certain level of belief. This probabilistic approach offers a more scalable solution by providing a probabilistic assessment of safety rather than requiring exhaustive sampling for exact verification.

6.2.2. Safety in Real World

One must note that the proof of Theorem 1 assumes that the controller $\pi(x) \in K_{cbf}(x)$ is a continuous controller. However, only discrete controllers can be implemented on a real-world robotic system with electrical control unit. Such a continuous-to-discrete gap breaks the safety guarantee that our method provides. To address this issue, [95] proposed a modified QP for a discrete controller but requires the Lipschtz constant of Lie derivatives $L_f h, L_g h$ whose expensive computation is not applicable for real-time performance. To avoid heavy computation, we can consider the maximum Lipschitz constant of Lie derivatives (i.e., worst scenario) or use online learning methods (e.g., GP) to approximate those Lipschitz constants.

6.2.3. Optimality

According to [7], the solution that satisfies the Control Barrier-Value Function Variational Inequality (CBVF-VI), is only a Lipschitz continuous function instead of a differentiable function. Although we use a Tanh-based NN to match the definition of the CBF, the loss based on the CBVF-VI may lead to incorrect training results. What is the property of the optimal CBF that renders the maximum safe set remains an open question.

References

[1] Aaron D. Ames et al. "Control Barrier Functions: Theory and Applications". In: *2019 18th European Control Conference (ECC)*. 2019, pp. 3420–3431. DOI: 10.23919/ECC.2019.8796030.

[2] Charles Dawson, Sicun Gao, and Chuchu Fan. "Safe Control With Learned Certificates: A Survey of Neural Lyapunov, Barrier, and Contraction Methods for Robotics and Control". In: *IEEE Transactions on Robotics* (2023).

[3] Bolun Dai, Prashanth Krishnamurthy, and Farshad Khorrami. "Learning a better control barrier function". In: *2022 IEEE 61st Conference on Decision and Control (CDC)*. IEEE. 2022, pp. 945–950.

[4] Charles Dawson et al. "Safe nonlinear control using robust neural lyapunov-barrier functions". In: *Conference on Robot Learning*. PMLR. 2022, pp. 1724–1735.

[5] Ya-Chien Chang, Nima Roohi, and Sicun Gao. "Neural Lyapunov Control". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/2647c1dba23bc0e0f9cdf75339e120d2-Paper.pdf.

[6] Andrea Peruffo, Daniele Ahmed, and Alessandro Abate. "Automated and formal synthesis of neural barrier certificates for dynamical models". In: *Tools and Algorithms for the Construction and Analysis of Systems: 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27–April 1, 2021, Proceedings, Part I 27*. Springer. 2021, pp. 370–388.

[7] Jason J Choi et al. "Robust control barrier–value functions for safety-critical control". In: *2021 60th IEEE Conference on Decision and Control (CDC)*. IEEE. 2021, pp. 6814–6821.

[8] Huan Zhang et al. "Efficient neural network robustness certification with general activation functions". In: *Advances in neural information processing systems* 31 (2018).

[9] Frederik Baymler Mathiesen, Simeon C Calvert, and Luca Laurenti. "Safety certification for stochastic systems via neural barrier functions". In: *IEEE Control Systems Letters* 7 (2022), pp. 973–978.

[10] Lukas Brunke et al. "Safe learning in robotics: From learning-based control to safe reinforcement learning". In: *Annual Review of Control, Robotics, and Autonomous Systems* 5 (2022), pp. 411–444.

[11] Peter Geibel and Fritz Wysotzki. "Risk-sensitive reinforcement learning applied to control under constraints". In: *Journal of Artificial Intelligence Research* 24 (2005), pp. 81–108.

[12] Núria Armengol Urpí, Sebastian Curi, and Andreas Krause. "Risk-averse offline reinforcement learning". In: *arXiv preprint arXiv:2102.05371* (2021).

[13] Clement Gehring and Doina Precup. "Smart exploration in reinforcement learning using absolute temporal difference errors". In: *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*. 2013, pp. 1037–1044.

[14] Brijen Thananjeyan et al. "Recovery rl: Safe reinforcement learning with learned recovery zones". In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 4915–4922.

[15] Homanga Bharadhwaj et al. "Conservative safety critics for exploration". In: *arXiv preprint arXiv:2010.14497* (2020).

[16] Matteo Turchetta, Andreas Krause, and Sebastian Trimpe. "Robust model-free reinforcement learning with multi-objective Bayesian optimization". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 10702–10708.

[17] Antonio Loquercio et al. "Deep drone racing: From simulation to reality with domain randomization". In: *IEEE Transactions on Robotics* 36.1 (2019), pp. 1–14.

[18] Björn Lütjens, Michael Everett, and Jonathan P How. "Safe reinforcement learning with model uncertainty estimates". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 8662–8668.

[19] Gregory Kahn et al. "Uncertainty-aware reinforcement learning for collision avoidance". In: *arXiv preprint arXiv:1702.01182* (2017).

[20] Teodor Mihai Moldovan and Pieter Abbeel. "Safe exploration in markov decision processes". In: *arXiv preprint arXiv:1205.4810* (2012).
[21] Michael Kearns and Satinder Singh. "Near-optimal reinforcement learning in polynomial time". In: *Machine learning* 49 (2002), pp. 209–232.
[22] Gal Dalal et al. "Safe exploration in continuous action spaces". In: *arXiv preprint arXiv:1801.08757* (2018).
[23] Joshua Achiam et al. "Constrained policy optimization". In: *International conference on machine learning*. PMLR. 2017, pp. 22–31.
[24] John Schulman et al. "Trust region policy optimization". In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.
[25] Tsung-Yen Yang et al. "Projection-based constrained policy optimization". In: *arXiv preprint arXiv:2010.03152* (2020).
[26] Yinlam Chow et al. "Risk-constrained reinforcement learning with percentile risk criteria". In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 6070–6120.
[27] Qingkai Liang, Fanyu Que, and Eytan Modiano. "Accelerated primal-dual policy optimization for safe reinforcement learning". In: *arXiv preprint arXiv:1802.06480* (2018).
[28] Yinlam Chow et al. "Lyapunov-based safe policy optimization for continuous control". In: *arXiv preprint arXiv:1901.10031* (2019).
[29] Harsh Satija, Philip Amortila, and Joelle Pineau. "Constrained markov decision processes via backward value functions". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 8502–8511.
[30] Romain Laroche, Paul Trichelair, and Remi Tachet Des Combes. "Safe policy improvement with baseline bootstrapping". In: *International conference on machine learning*. PMLR. 2019, pp. 3652–3661.
[31] Thiago D Simão and Matthijs TJ Spaan. "Safe policy improvement with baseline bootstrapping in factored environments". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 4967–4974.
[32] Somil Bansal et al. "Hamilton-jacobi reachability: A brief overview and recent advances". In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE. 2017, pp. 2242–2253.
[33] Ian M Mitchell et al. "A toolbox of level set methods". In: *UBC Department of Computer Science Technical Report TR-2007-11* 1 (2007), p. 6.
[34] Goran Frehse et al. "SpaceEx: Scalable verification of hybrid systems". In: *Computer Aided Verification: 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings 23*. Springer. 2011, pp. 379–395.
[35] Alexander B Kurzhanski and Pravin Varaiya. "Ellipsoidal techniques for reachability analysis". In: *Hybrid Systems: Computation and Control: Third International Workshop, HSCC 2000 Pittsburgh, PA, USA, March 23–25, 2000 Proceedings*. Springer. 2002, pp. 202–214.
[36] Antoine Girard. "Reachability of uncertain linear systems using zonotopes". In: *HSCC*. Vol. 3414. Springer. 2005, pp. 291–305.
[37] Bastian Schürmann and Matthias Althoff. "Guaranteeing constraints of disturbed nonlinear systems using set-based optimal control in generator space". In: *IFAC-PapersOnLine* 50.1 (2017), pp. 11515–11522.
[38] Mo Chen et al. "Decomposition of reachable sets and tubes for a class of nonlinear systems". In: *IEEE Transactions on Automatic Control* 63.11 (2018), pp. 3675–3688.
[39] Jérôme Darbon, Gabriel P Langlois, and Tingwei Meng. "Overcoming the curse of dimensionality for some Hamilton–Jacobi partial differential equations via neural network architectures". In: *Research in the Mathematical Sciences* 7 (2020), pp. 1–50.
[40] Somil Bansal and Claire J Tomlin. "Deepreach: A deep learning approach to high-dimensional reachability". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 1817–1824.
[41] Anayo K Akametalu and Claire J Tomlin. "Temporal-difference learning for online reachability analysis". In: *2015 European Control Conference (ECC)*. IEEE. 2015, pp. 2508–2513.
[42] 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.
[43] Kai-Chieh Hsu et al. "Safety and liveness guarantees through reach-avoid reinforcement learning". In: *arXiv preprint arXiv:2112.12288* (2021).

[44] Aaron D Ames et al. “Control barrier function based quadratic programs for safety critical systems”. In: *IEEE Transactions on Automatic Control* 62.8 (2016), pp. 3861–3876.
[45] Quan Nguyen et al. “3d dynamic walking on stepping stones with control barrier functions”. In: *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE. 2016, pp. 827–834.
[46] Xiangru Xu et al. “Realizing simultaneous lane keeping and adaptive speed regulation on accessible mobile robot testbeds”. In: *2017 IEEE Conference on Control Technology and Applications (CCTA)*. IEEE. 2017, pp. 1769–1775.
[47] Amir Ali Ahmadi and Anirudha Majumdar. “Some Applications of Polynomial Optimization in Operations Research and Real-Time Decision Making”. In: *arXiv e-prints*, arXiv:1504.06002 (Apr. 2015), arXiv:1504.06002. DOI: 10.48550/arXiv.1504.06002. arXiv: 1504.06002 [math.OC].
[48] Mohit Srinivasan et al. “Extent-compatible control barrier functions”. In: *Systems & Control Letters* 150 (2021), p. 104895.
[49] Mohit Srinivasan et al. “Synthesis of control barrier functions using a supervised machine learning approach”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 7139–7145.
[50] Desong Du et al. “Reinforcement Learning for Safe Robot Control using Control Lyapunov Barrier Functions”. In: *arXiv preprint arXiv:2305.09793* (2023).
[51] Yujie Yang et al. “Model-free safe reinforcement learning through neural barrier certificate”. In: *IEEE Robotics and Automation Letters* 8.3 (2023), pp. 1295–1302.
[52] Yousef Emam et al. “Safe Reinforcement Learning Using Robust Control Barrier Functions”. In: *arXiv preprint arXiv:2110.05415* (2021).
[53] Zhaojian Li, Uroš Kalabić, and Tianshu Chu. “Safe reinforcement learning: Learning with supervision using a constraint-admissible set”. In: *2018 Annual American Control Conference (ACC)*. IEEE. 2018, pp. 6390–6395.
[54] Xiangru Xu et al. “Correctness guarantees for the composition of lane keeping and adaptive cruise control”. In: *IEEE Transactions on Automation Science and Engineering* 15.3 (2017), pp. 1216–1229.
[55] Alhussein Fawzi, Seyed-Mohsen Moosavi-Dezfooli, and Pascal Frossard. “The robustness of deep networks: A geometrical perspective”. In: *IEEE Signal Processing Magazine* 34.6 (2017), pp. 50–62.
[56] Battista Biggio and Fabio Roli. “Wild patterns: Ten years after the rise of adversarial machine learning”. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, pp. 2154–2156.
[57] Yujia Huang et al. “Training certifiably robust neural networks with efficient local lipschitz bounds”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 22745–22757.
[58] Peter L Bartlett, Dylan J Foster, and Matus J Telgarsky. “Spectrally-normalized margin bounds for neural networks”. In: *Advances in neural information processing systems* 30 (2017).
[59] Zhouxing Shi et al. “Efficiently computing local Lipschitz constants of neural networks via bound propagation”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 2350–2364.
[60] Guy Katz et al. “Reluplex: An efficient SMT solver for verifying deep neural networks”. In: *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*. Springer. 2017, pp. 97–117.
[61] Rudy Bunel et al. “Branch and bound for piecewise linear neural network verification”. In: *Journal of Machine Learning Research* 21.2020 (2020).
[62] Ruediger Ehlers. “Formal verification of piece-wise linear feed-forward neural networks”. In: *Automated Technology for Verification and Analysis: 15th International Symposium, ATVA 2017, Pune, India, October 3–6, 2017, Proceedings 15*. Springer. 2017, pp. 269–286.
[63] Gagandeep Singh et al. “Boosting robustness certification of neural networks”. In: *International conference on learning representations*. 2019.
[64] Mahyar Fazlyab et al. “Efficient and accurate estimation of lipschitz constants for deep neural networks”. In: *Advances in Neural Information Processing Systems* 32 (2019).
[65] Luca Pulina and Armando Tacchella. “Challenging SMT solvers to verify neural networks”. In: *Ai Communications* 25.2 (2012), pp. 117–135.
[66] Xiaowei Huang et al. “Safety verification of deep neural networks”. In: *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*. Springer. 2017, pp. 3–29.
[67] Lily Weng et al. “Towards fast computation of certified robustness for relu networks”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 5276–5285.

[68] Gagandeep Singh et al. "An abstract domain for certifying neural networks". In: *Proceedings of the ACM on Programming Languages* 3.POPL (2019), pp. 1–30.
[69] Weiming Xiang, Hoang-Dung Tran, and Taylor T Johnson. "Reachable set computation and safety verification for neural networks with relu activations". In: *arXiv preprint arXiv:1712.08163* (2017).
[70] Weiming Xiang, Hoang-Dung Tran, and Taylor T Johnson. "Output reachable set estimation and verification for multilayer neural networks". In: *IEEE transactions on neural networks and learning systems* 29.11 (2018), pp. 5777–5783.
[71] Timon Gehr et al. "Ai2: Safety and robustness certification of neural networks with abstract interpretation". In: *2018 IEEE symposium on security and privacy (SP)*. IEEE. 2018, pp. 3–18.
[72] Changliu Liu et al. "Algorithms for verifying deep neural networks". In: *Foundations and Trends® in Optimization* 4.3-4 (2021), pp. 244–404.
[73] Christian Szegedy et al. "Intriguing properties of neural networks". In: *arXiv preprint arXiv:1312.6199* (2013).
[74] Matt Jordan and Alexandros G Dimakis. "Exactly computing the local lipschitz constant of relu networks". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 7344–7353.
[75] Aritra Bhowmick, Meenakshi D'Souza, and G Srinivasa Raghavan. "LipBaB: Computing exact Lipschitz constant of ReLU networks". In: *Artificial Neural Networks and Machine Learning–ICANN 2021: 30th International Conference on Artificial Neural Networks, Bratislava, Slovakia, September 14–17, 2021, Proceedings, Part IV 30*. Springer. 2021, pp. 151–162.
[76] Huan Zhang, Pengchuan Zhang, and Cho-Jui Hsieh. "Recurjac: An efficient recursive algorithm for bounding jacobian matrix of neural networks and its applications". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 5757–5764.
[77] Jacob Laurel et al. "A dual number abstraction for static analysis of Clarke Jacobians". In: *Proceedings of the ACM on Programming Languages* 6.POPL (2022), pp. 1–30.
[78] Nicholas Boffi et al. "Learning stability certificates from data". In: *Conference on Robot Learning*. PMLR. 2021, pp. 1341–1350.
[79] Simin Liu, Changliu Liu, and John Dolan. "Safe control under input limits with neural control barrier functions". In: *Conference on Robot Learning*. PMLR. 2023, pp. 1970–1980.
[80] David G. Luenberger and Yinyu Ye. *Linear and Nonlinear Programming*. Cham, Switzerland: Springer International Publishing, Nov. 2021. ISBN: 978-3-030-85450-8. URL: https://link-springer-com.tudelft.idm.oclc.org/book/10.1007/978-3-030-85450-8.
[81] Matthew Streeter and Joshua V Dillon. "Automatically Bounding the Taylor Remainder Series: Tighter Bounds and New Applications". In: *arXiv preprint arXiv:2212.11429* (2022).
[82] Christopher Brix et al. *First Three Years of the International Verification of Neural Networks Competition (VNN-COMP)*. 2023. arXiv: 2301.05815 [cs.LG].
[83] Alessandro Abate et al. "Counterexample guided inductive synthesis modulo theories". In: *International Conference on Computer Aided Verification*. Springer. 2018, pp. 270–288.
[84] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural networks* 2.5 (1989), pp. 359–366.
[85] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.
[86] Kaiming He et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.
[87] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. pmlr. 2015, pp. 448–456.
[88] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).
[89] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).
[90] Daniel Ho et al. "Population Based Augmentation: Efficient Learning of Augmentation Policy Schedules". In: *arXiv* (May 2019). DOI: 10.48550/arXiv.1905.05393. eprint: 1905.05393.
[91] Jason Liang et al. "Regularized Evolutionary Population-Based Training". In: *arXiv* (Feb. 2020). DOI: 10.48550/arXiv.2002.04225. eprint: 2002.04225.

[92] Marc Rigter, Bruno Lacerda, and Nick Hawes. "RAMBO-RL: Robust Adversarial Model-Based Offline Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo et al. Vol. 35. Curran Associates, Inc., 2022, pp. 16082–16097. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/6691c5e4a199b72dffd9c90acb63bcd6-Paper-Conference.pdf.

[93] Hoki Kim, Woojin Lee, and Jaewook Lee. "Understanding catastrophic overfitting in single-step adversarial training". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 9. 2021, pp. 8119–8127.

[94] Shromona Ghosh et al. "Verifying controllers against adversarial examples with bayesian optimization". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 7306–7313.

[95] Joseph Breeden, Kunal Garg, and Dimitra Panagou. "Control barrier functions in sampled-data systems". In: *IEEE Control Systems Letters* 6 (2021), pp. 367–372.

[96] Kenji Doya. "Reinforcement learning in continuous time and space". In: *Neural computation* 12.1 (2000), pp. 219–245.

[97] Max Schwenzer et al. "Review on model predictive control: An engineering perspective". In: *The International Journal of Advanced Manufacturing Technology* 117.5-6 (2021), pp. 1327–1349.

[98] J.B. Rawlings, D.Q. Mayne, and M. Diehl. *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2017. ISBN: 9780975937730. URL: https://books.google.nl/books?id=MrJctAEACAAJ.

A

Background

This chapter is designed for researchers who are outside the robotics community and not familiar with CMDP, MPC, HJI-RA, and CBF. We first present the general symbols and terminologies in Section A.1. Then the CMDP will be introduced later in Section A.2. The background knowledge on HJI-RA, CBF, and MPC are presented in Section A.3 A.4 A.5, respectively.

A.1. Common Terminologies

A.1.1. System Model

A transition model $P : \mathbb{X} \times \mathbb{U} \times \mathbb{X} \to [0, 1]$ denotes the probability $Pr(x'|x, u)$ of transitioning to state x' given current state and action pair (x, u), where $\mathbb{X}$ and $\mathbb{U}$ are state space and action space. Here, we consider a robot with a physical structure whose dynamics can be represented by the following continuous dynamic model:

$$\dot{x}(t) = f(x(t), u(t), w(t)), \tag{A.1}$$

or discrete dynamic model with time step Δt

$$x_{k+1} = f_k(x_k, u_k, w_k), \tag{A.2}$$

where $k \in \mathbb{Z}_{\geq 0}$ is the discrete index, $x(t), x_k \in \mathbb{X}$ are the states and $u(t), u_k \in \mathbb{U}$ are the actions, f, f_k denote the dynamic model of the robot, $w(t) \in \mathbb{W}(t), w_k \in \mathbb{W}(k)$ is the process noise distributed according to a bounded distribution $\mathbb{W}(t)$ or $\mathbb{W}(k)$. Then the deterministic transition model P can be written as:

$$P(x, u, x') = \begin{cases} 1 & \text{if } x' = f_k(x, u, w) \\ 0 & \text{otherwise} \end{cases} \tag{A.3}$$

We assume direct access to measurements of the state $x(t)$, x_k and neglect the problem of state estimation. Such a system can be easily extended to stochastic disturbance by using Gaussian Process(GP) to learn the disturbance.

A.1.2. Reward Function

The robot's task is described by a reward(cost) function $R : \mathbb{X} \times \mathbb{U} \times \mathbb{X} \to \mathbb{R}$, which returns the reward at state x_k with action u_k. We consider the reward along a trajectory with a finite time horizon of N (It becomes the reward along an infinite long trajectory when $N \to \infty$). Given an initial state $\bar{x}_0$ and a control policy π, we can have a trajectory $x_{0:N} = \{x_0, x_1, \cdots, x_N\}$ from transition model (A.2). Then, the return of the trajectory can be computed:

$$J^{\pi}(\bar{x}_0) = \mathbb{E}_{\pi}[\sum_{k=0}^{N} \gamma^k R(x_k, u_k, x_{k+1})]. \tag{A.4}$$

A.1.3. Safety Constraints

Safety constraints ensure the safe operation of the robot and include:

(i) allowable state space $\mathbb{X}_a \subseteq \mathbb{X}$, which define the set of safe operating states

(ii) allowable action space $\mathbb{U}_a \subseteq \mathbb{U}$, which describe the physical limits of actuators

To encode the safety constraints, we define n_c constraint functions: $c_k(x_k, u_k, d_k) \in \mathbb{R}^{n_c}$ with each constraint c_k^j being a real-valued, time-varying function. Starting with the strongest guarantee, we introduce three levels of safety: hard, probabilistic, and soft constraints.

Safety Level III: Constraint Satisfaction Guaranteed. The system satisfies hard constraints:

$$c_k^j(x_k, u_k, d_k) \geq 0, \tag{A.5}$$

for all times $k \in \{0, \cdots, N\}$ and constaint indexes $j \in \{1, \cdots, n_c\}$

Safety Level II: Constraint Satisfaction with Probability p. The system satisfies probabilistic constraints:

$$\Pr(c_k^j(x_k, u_k, d_k) \geq 0) \geq p^j, \tag{A.6}$$

where Pr(.) denotes the probability and $p^j \in (0, 1)$ defines the likelihood of the j-th constraint being satisfied, with $j \in \{1, \cdots, n_c\}$ and for all times $k \in \{0, \cdots, N\}$.

Safety Level I: Constraint Satisfaction Encouraged. The system encourages constraint satisfaction. This can be achieved by adding a penalty term to the objective function that discourages the violation of the constraints with high cost. A non-negative ϵ_j is added to the right-hand side of the inequality (A.5), for all times $k \in \{0, \cdots, N\}$ and $j \in \{1, \cdots, n_c\}$,

$$c_k^j(x_k, u_k, d_k) \geq -\epsilon_j, \tag{A.7}$$

and an appropriate penalty term $l_\epsilon(\epsilon) \geq 0$ with $l_\epsilon(\epsilon) = 0 \Leftrightarrow \epsilon = 0$ is added to the return in (A.4). The vector ϵ includes all elements ϵ_j and is an additional variable of the optimization problem. Alternatively, although $c_k^j(x_k, u_k, w_k)$ is a step-wise quantity, some safe RL approaches only aim to provide guarantees on its expected value $E[\cdot]$ on a trajectory level:

$$J_{c^j} = \mathbb{E}_{\pi}[\sum_{k=0}^{N-1} c_k^j(x_k, u_k, w_k)] \geq \epsilon_j. \tag{A.8}$$

Visualization of three different safety levels is shown in Fig. A.1

A.2. Constrained Markov Decision Process

The MDP is usually used to formulate the RL problem. A MDP is defined by a tuple ($\mathbb{X}$, $\mathbb{U}$, R, P, $\bar{x}_0$, γ), where $\mathbb{X}$ and $\mathbb{U}$ are state space and action space, R represents reward function, P denote the transition model of the system, $\gamma \in [0, 1]$ is a discount factor. $\bar{x}_0$ is the initial state or sometimes it could be written as $\bar{x}_0 \sim \mu$ where μ denotes the initial state distribution. Let $\pi : \mathbb{X} \to \mathcal{P}(\mathbb{U})$ be a stationary policy, with $\pi(u|x)$ denote the probability of selecting action u at state x ($\pi(u|x) = 1$ renders a deterministic policy). We denote the set of all stationary policies by Π, and the nominal RL problem can be formulated as:

$$\pi^* = \arg\max_{\pi \in \Pi} J^{\pi}(\bar{x}_0) = \mathbb{E}_{\pi}[\sum_{k=0}^{N} \gamma^k R(x_k, u_k, x_{k+1})]. \tag{RL}$$

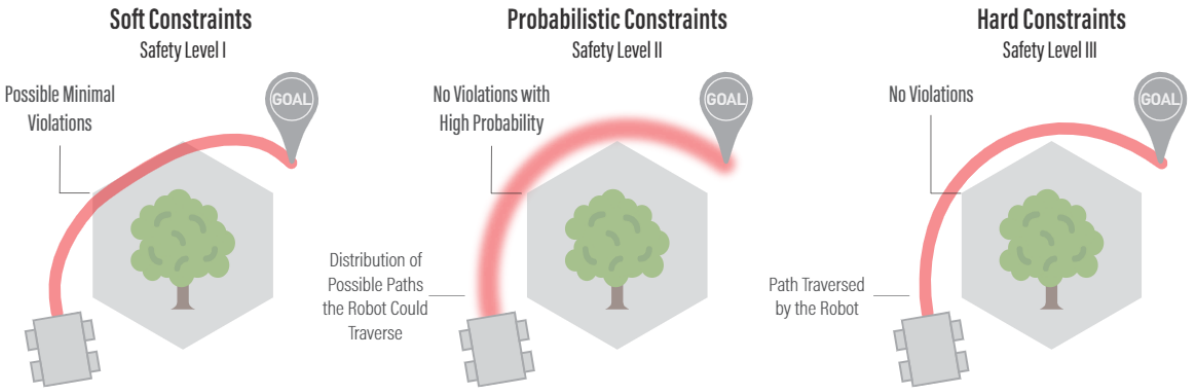


Figure A.1: Illustration of the different safety levels from [10]

The common solution to (RL) is approximating value function based on *Bellman Equation* [96]:

$$V^{\pi}(x) = \mathbb{E}_{u\sim\pi(u|x)}[Q^{\pi}(x,u)], \tag{A.9}$$

$$Q^{\pi}(x,u) = R(x,u,x') + \mathbb{E}_{x'\sim P(x,u,x')}[V^{\pi}(x')], \tag{A.10}$$

where $V^{\pi}(x)$ is the value function that describes the maximum reward one can get from current state x and $Q^{\pi}(x,u)$ is the action-value function that represents the maximum reward one can get when applying action u with current state x. Traditionally, the agent updates the value function and action-value function by interacting with the environment (see Fig. A.2).

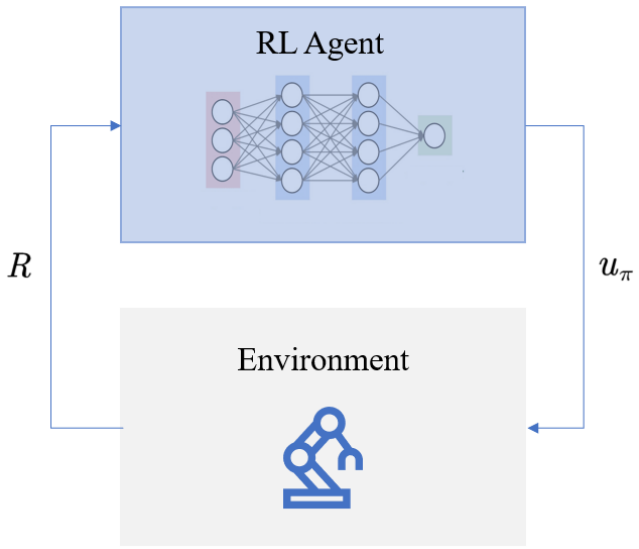


Figure A.2: Illustration of RL. The agent takes actions to manipulate the environment and obtain rewards.

However, plain MDP doesn't take constraints into account. Therefore, CMDP is commonly utilized in safe RL due to its capability to incorporate constraints that express various safety notions. Generally, solving CMDP is a policy search process to maximize reward or minimize cost while respecting some constraints. A CMDP is denoted by a tuple $(\mathbb{X}, \mathbb{U}, R, c, P, \bar{x}_0, \gamma)$, where c represents constraint functions.

Let's define the set of feasible stationary policies for a CMDP:

$$\begin{aligned}\Pi_C = \{\pi \in \Pi :&\text{Safety constraints are satisfied according to either}\\ &\text{Equation (A.5), Equation (A.6), Equation (A.7), Equation (A.8)}, \forall k \in [0, N]\}.\end{aligned} \tag{A.11}$$

Solving a CMDP means searching for a policy that maximizes the expected return of the trajectory while respecting constraints all the time. It can be formulated as the following optimization problem:

$$\pi^* = \arg\max_{\pi \in \Pi_C} J^\pi(\bar{x}_0) = \mathbb{E}_\pi[\sum_{k=0}^{N} \gamma^k R(x_k, u_k, x_{k+1})]. \tag{A.12}$$

Although optimal policies for the finite discrete CMDP with known models can be obtained by LP [23], solving the high-dimensional continuous CMDP is still an open question.

A.3. Hamiltonian-Jacobian Reachability Analysis

The HJI-RA [32] is a verification method for guaranteeing the safety properties of the systems by considering all possible system behaviors and disturbances. For safety-critic applications, we are often interested in the BRT. This is the set of states such that the trajectories that start from this set would eventually reach a target set (obstacles) at any point in the time horizon, and thus become unsafe. Here, we briefly introduce how one can compute a BRT. For example, there is a Pilot who drives his plane to avoid a target area (obstacle). However, a strong Wind appears on the way, which causes aircraft to drift sideways (see Fig. A.3).

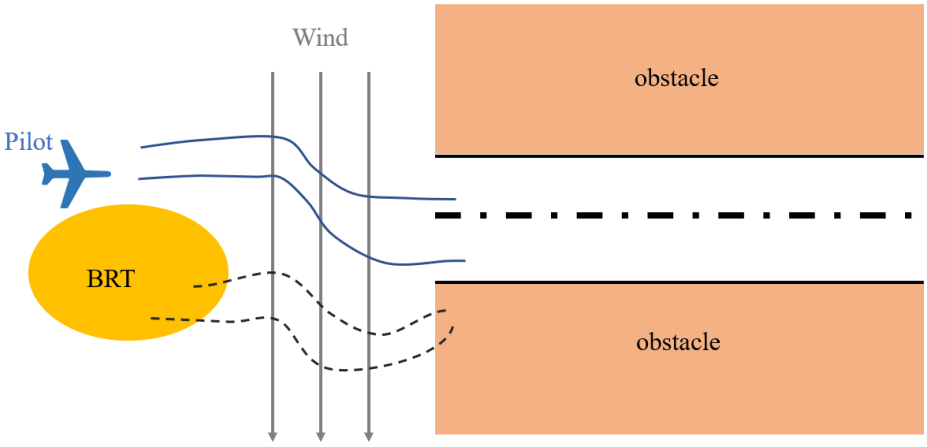


Figure A.3: Illustration of the HJI-RA. The black dash line represents the unsafe trajectory starting from the BRT and the plane will eventually hit the obstacle no matter how hard the Pilot tries

We reformulate the dynamics in Equation (A.1) as:

$$\dot{x}(t) = f(x(t), a(t), b(t)), t \in [t_0, 0], a(t) \in \mathbb{A}, b(t) \in \mathbb{B}, \tag{A.13}$$

where $a(t)$ and $b(t)$ denote the input from Player 1 (Pilot) and Player 2 (Wind), $\mathbb{A}, \mathbb{B}$ are compact action space and t_0 represents the time horizon. Let $\zeta(t; \bar{x}_0, t_0, a(.), b(.)) : [t_0, 0] \rightarrow \mathbb{R}^n$ be the trajectory (solution of Equation (A.13)) given initial state $\bar{x}_0$, time horizon t_0 and control policies $a(.), b(.)$ from two players. We define the target area that we want to avoid as $\mathcal{G}_0 \subseteq \mathbb{R}^n = \{x : g(x) < 0\}$ where $g(x) \geq 0 \Leftrightarrow x \in \mathbb{X}_a$ and $\mathbb{X}_a$ denotes the area that system is allowed to visit. To encode whether the system would enter the target area, we defined a cost function to represent the minimum distance to $\mathcal{G}_0$ along the trajectory:

$$J_{t_0}(\bar{x}_0, a(.), b(.)) = \min_{t \in [t_0, 0]} g(\zeta(t; \bar{x}_0, t_0, a(.), b(.))), \tag{A.14}$$

$J_{t_0}(\bar{x}_0, a(.), b(.)) < 0$ means that the system would collide with an obstacle within the time horizon t_0. However, the trajectory of the system is determined by two control policies $a(.), b(.)$. Therefore, our goal becomes capturing the minimum distance $J_{t_0}(\bar{x}_0, a(.), b(.))$ of the optimal trajectory of the system which is generated when Player 1 (Pilot) maximizes the distance and the worst-case disturbance (Wind)

minimizes the distance. If this optimal trajectory enters the target area, it means that there exists no control that can save the system from a collision. Let a value function represent the minimal distance to the target area $\mathcal{G}_0$ along the optimal trajectory:

$$V(\bar{x}_0, t) = \inf_{b(.)\in\mathbb{B}} \sup_{a(.)\in\mathbb{A}} J_t(\bar{x}_0, a(.), b(.)). \tag{A.15}$$

Using the principle of dynamic programming, it can be shown that the value function $V(\bar{x}_0, t)$ in (A.15) is the solution of the following HJI-VI [40]:

$$\min\{D_t V(x,t) + H(x,t), g(x) - V(x,t)\} = 0, V(x,0) = g(x). \tag{A.16}$$

where D_t and ∇ represent the time and spatial gradients of the value function and Hamiltonian $H(x,t)$ is given by:

$$H(x,t) = \max_{a(.)\in\mathbb{A}} \min_{b(.)\in\mathbb{B}} < \nabla V(x,t), f(x, a(.), b(.)) > . \tag{A.17}$$

Once we obtain the value function $V(\bar{x}_0, t)$ by solving the HJI-VI in (A.16), the BRT is given as the sub-zero level set of the value function:

$$\mathcal{G}(t) = \{x : V(x,t) < 0\}. \tag{A.18}$$

The corresponding optimal safe controller can be derived as:

$$a^*(x) = \arg\max_{a(.)\in\mathbb{A}} \min_{b(.)\in\mathbb{B}} < \nabla V(x,t), f(x, a(.), b(.)) > . \tag{A.19}$$

This controller would be activated when the system hits the boundary of the BRT. We will have a deep review of HJI-RA in Section 2.2.1

A.4. Control Barrier Function

The CBF [1] is a certificate function that describes the safety of the system and provides safety guarantees. Unlike the HJI-RA, which intends to compute BRTs, the CBF renders a forward invariance set and keeps the system inside the safe set (see Fig. A.4).

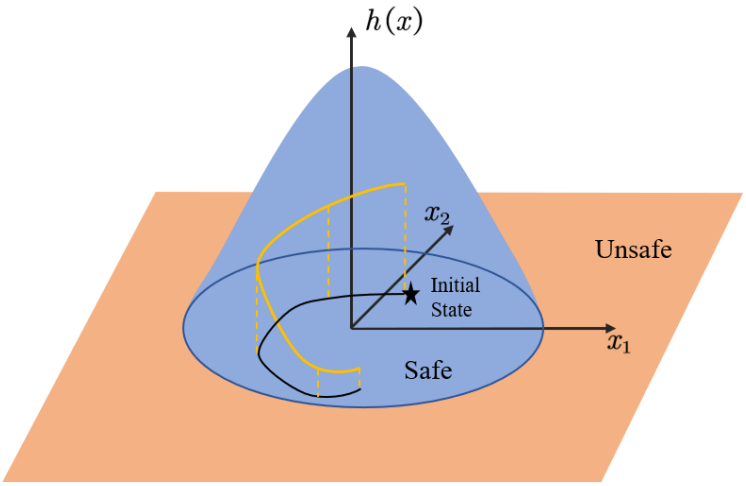


Figure A.4: Illustration of the CBF. The black solid line denotes the trajectory starting from the initial state. The corresponding orange trajectory reflects the evolution of the value of the CBF

Conventionally, the CBF deals with a nonlinear control affine system in Equation (A.20)

$$\dot{x} = f(x) + g(x)u. \tag{A.20}$$

We assume there exists an allowable set of states $\mathbb{X}_a = \{x \in \mathbb{X} : \rho(x) \geq 0\}$ and a continuously differentiable function $h : \mathbb{X} \to \mathbb{R}$. We say that h is a CBF if there exists an extended class $\mathcal{K}_\infty$ function α such that the CBC is satisfied:

$$\sup_{u \in \mathbb{U}} [L_f h(x) + L_g h(x)u] \geq -\alpha(h(x)), \tag{A.21}$$

and 0-superlevel set of h is a subset of the allowable set:

$$\{x \in \mathbb{X} : h(x) \geq 0\} \subseteq \mathbb{X}_a. \tag{A.22}$$

When we find a valid CBF, we can solve a QP to derive a safe controller:

$$u^*_{CBF} = \arg\min_u \ ||u_\pi - u||^2 \tag{CBF-QP}$$

$$\text{s.t.} L_f h(x) + L_g h(x)u \geq -\alpha(h(x)), \tag{A.23a}$$

$$u \in \mathbb{U}, \tag{A.23b}$$

where $u_\pi = \pi(x)$ is the control from the NN or RL policy, u^*_{CBF} is the least modification on u_π but ensure the safety. We have a deep review of CBF in Section 2.2.2

A.5. Model Predictive Control

The MPC has seen significant success in recent decades [97] and has become the primary control method for the systematic handling of system constraints in diverse fields. The MPC scheme relies on a sufficiently accurate model to predict the future behavior of the system (see Fig. A.5).

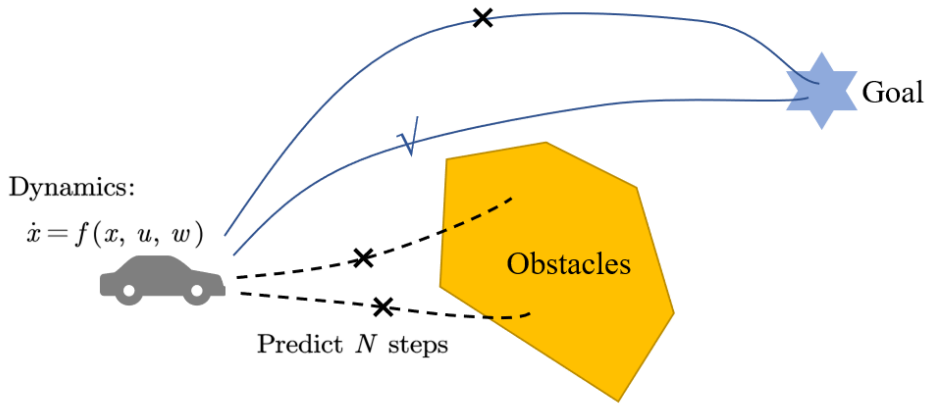


Figure A.5: Illustration of the MPC. The controller will predict N steps ahead and choose the best trajectory that satisfies the constraints and uses less energy

Based on the model, the MPC is also called receding horizon control as an optimization problem that optimizes future trajectory in a short time horizon according to the cost function and the constraints would be solved online iteratively. The optimization problem is formulated as:

$$J^{\pi^*}(\bar{x}_0) = \min_{\pi} \sum_{k=0}^{N-1} l_t(x_k, u_k) + l_N(x_N) \tag{A.24a}$$

$$\text{s.t.discrete dynamics in Equation (A.2)}, \tag{A.24b}$$

$$u_k = \pi(x_k), \tag{A.24c}$$

$$x_0 = \bar{x}_0, \tag{A.24d}$$

$$x_N \in \mathcal{X}_t, \tag{A.24e}$$

$$\begin{aligned}&\text{Safety constraints according to either}\\ &\text{Equation (A.5), Equation (A.6), Equation (A.7), Equation (A.8)}, \forall k \in [0, N],\end{aligned} \tag{A.24f}$$

where $l_t(x_k, u_k)$ denotes the stage cost at each time step, $l_N(x_n)$ denotes the terminal cost at last step, $\bar{x}_0$ is the initial state, $\mathcal{X}_t$ is the terminal set and $\pi(x_k)$ denotes the control policy. At each iteration, the first value of the optimal control sequence would be applied to control the system. The challenge of designing a good MPC lies in the choice of terminal cost and terminal set. These two elements are the cornerstone of proving recursive stability and feasibility. More details can be found in [98].

B

Published Paper

The work of this thesis resulted in a published paper in ECC 2024. The paper is attached below.

Simultaneous Synthesis and Verification of Neural Control Barrier Functions through Branch-and-Bound Verification-in-the-loop Training

Xinyu Wang[1], Luzia Knoedler[1], Frederik Baymler Mathiesen[2], and Javier Alonso-Mora[1]

***Abstract*— Control Barrier Functions (CBFs) that provide formal safety guarantees have been widely used for safety-critical systems. However, it is non-trivial to design a CBF. Utilizing neural networks as CBFs has shown great success, but it necessitates their certification a s C BFs. I n t his work, we leverage bound propagation techniques and the Branch-and-Bound scheme to efficiently v erify t hat a n eural network satisfies t he c onditions t o b e a C BF o ver t he c ontinuous state space. To accelerate training, we further present a framework that embeds the verification s cheme i nto t he t raining l oop to synthesize and verify a neural CBF simultaneously. In particular, we employ the verification s cheme t o i dentify partitions of the state space that are not guaranteed to satisfy the CBF conditions and expand the training dataset by incorporating additional data from these partitions. The neural network is then optimized using the augmented dataset to meet the CBF conditions. We show that for a non-linear control-affine system, our framework can efficiently certify a neural network as a CBF and render a larger safe set than state-of-the-art neural CBF works. We further employ our learned neural CBF to derive a safe controller to illustrate the practical use of our framework.**

I. Introduction

Safety is a critical element of autonomous systems, such as self-driving cars and manipulators that interact with humans. As autonomous systems grow more complex, determining whether they operate safely becomes challenging.

Safety can be formulated via invariance, in the sense that any trajectory originating within an invariant set will never traverse beyond the boundaries of that set. Lately, the use of Control Barrier Functions (CBFs) to derive a forward invariant set has received significant attention in the control and learning community [1]. However, there exists no general and scalable technique for designing CBFs. Therefore, recent works [2], [3] synthesize continuous CBFs using Neural Networks (NNs) as a function template, which are referred to as Neural Control Barrier Functions (nCBFs). Yet, these works rely on an initial guess of the forward invariant set or the function structure of the CBF to synthesize the nCBF. An improper initial guess usually results in a suboptimal nCBF. Constructing an optimal CBF that renders a maximum forward invariant set is challenging. A recent work [4] introduced the Control Barrier-Value Function (CBVF) which is a safe value function and renders the maximum forward invariant set for a chosen time span. In this work, we synthesize a continuous nCBF that approximates the infinite-horizon CBVF and renders a safe set that is close to the maximum forward invariant set.

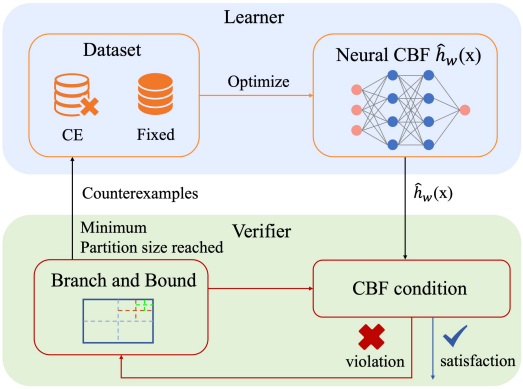


Fig. 1: A schematic overview of the presented Branch-and-Bound Verification-in-the-loop Training. The framework comprises of two key components: the learner and the verifier, which operate sequentially. The learner optimizes the nCBF using a fixed dataset and a Counterexample dataset. The verifier leverages bound propagation techniques and the Branch-and-Bound scheme to refine a partition of the state space until the CBF conditions are satisfied or counterexamples are generated.

Although utilizing NNs as CBFs offers universal approximation capabilities, it necessitates their certification as CBFs to provide safety guarantees. Verifying the NN as an nCBF in the continuous state space presents a significant challenge. Specifically, since the NN is trained using a finite set of data points, it will only be verified on those points. Outside the certified points, safety is no longer guaranteed. There are works [5], [6] that use the Satisfiability Modulo Theory (SMT) to verify their NNs. However, they are restricted to very simple NNs due to expensive computation. In this work, we leverage bound propagation techniques [7] and the Branch-and-Bound scheme (BBS) to efficiently verify nCBFs. In particular, we partition the state space and utilize linear bound propagation techniques to provide lower and upper bounds of the NN and its Jacobian. These bounds are used to verify if the NN satisfies the conditions to

This paper has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 101017008. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect t hose o f t he E uropean U nion o r t he European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

[1]Xinyu Wang, Luzia Knoedler, and Javier Alonso-Mora are with the Cognitive Robotics Department, Delft University of Technology, 2628 CD Delft, The Netherlands

Frederik Baymler Mathiesen is with the Delft Center for Systems and Control, Delft University of Technology, 2628 CD Delft, The Netherlands f.b.mathiesen@tudelft.nl

be a CBF. The BBS is applied to refine the partition to improve scalability and achieve less conservative bounds. We refer to the above verification scheme as Branch-and-Bound Verification scheme (BBV). This approach is similar to [8], however, we verify CBFs instead of barrier functions. To accelerate training, we embed the BBV into the training loop to synthesize and verify an nCBF simultaneously, which we refer to as Branch-and-Bound Verification-in-the-loop Training (BBVT), see Fig. 1. We show the efficiency of our method and the practical use of nCBFs on an inverted pendulum and a 2D navigation task in a simulation environment.

II. Related Work

Many works use CBFs to ensure the safety of a system [9]–[11]. However, it is non-trivial to construct CBFs. In recent years, new techniques emerged to automatically synthesize CBFs. For a system with polynomial dynamics, a CBF can be obtained by solving a sum-of-squares (SOS) optimization problem [12]. Unfortunately, SOS scales poorly to higher dimensional systems [13]. To address this shortcoming, NNs have been employed to approximate CBFs. They are trained by supervised learning [2], [3], [14] or Reinforcement Learning (RL) with the Actor-Critic framework [15], [16]. However, the quality of the nCBF in those works depends on an initial guess of the forward invariant set, CBF candidate, or exploration strategy. An improper initial guess results in a conservative nCBF with a small forward invariant set. To address the conservativity, in this work, we learn a continuous nCBF that renders a safe set close to the maximum forward invariant set. Furthermore, the training does not require an initial guess.

Commonly, NNs are trained through backpropagation of the empirical loss on a finite set of data points. Therefore, it is important to note that even an empirical loss of zero does not guarantee that the certificate is valid everywhere in the state space. Only a few works have verified their NNs, such as [5], [6], [17], which leverage SMT to provide Counterexamples (CEs) and guarantee the correctness of the synthesis procedure. However, SMT is limited to simple NNs with around 20 neurons in one or two hidden layers due to the need for expensive computation. In contrast to using SMT for exact verification, several efficient NN verification methods using linear bound propagation techniques have been developed [7], [18]. These bounding methods provide a new direction to verify neural certificates. The work in [8] partitions the state space with a BBS and verifies the property of the discrete-time stochastic barrier function for each partition leveraging the method in [7]. Our work extends the BBS of [8] to CBFs for continuous-time deterministic control-affine systems where the control input constraints must be considered and uses the BBV scheme to verify the learned continuous nCBF.

III. Problem Formulation

Given the following continuous-time control-affine system

$$\dot{x} = f(x) + g(x)u, \quad x(0) = x_0, \tag{1}$$

where $x \in \mathbb{X} \subset \mathbb{R}^n$, $u \in \mathbb{U} \subset \mathbb{R}^m$, $f : \mathbb{R}^n \to \mathbb{R}^n$ denotes the autonomous dynamics, and $g : \mathbb{R}^n \to \mathbb{R}^{n\times m}$ denotes the input dynamics. We assume that f, g are Lipschitz continuous and $\mathbb{X}, \mathbb{U}$ are compact sets.

The safety requirement for the system in (1) is encoded via a state admissible set $\mathbb{X}_a \subseteq \mathbb{X}$ and a convex input admissible set $\mathbb{U}_a \subseteq \mathbb{U}$. A safe system stays in the state admissible set for all time. To formally define safety, we use $x_\pi(t; x_0)$ to refer to a trajectory of the system in (1) at time t with initial condition x_0 and control policy $u = \pi(x)$. Safety is then defined as:

Definition 1 (Safety)**.** The system in (1) is *safe* if $x_\pi(t; x_0) \in \mathbb{X}_a$ and $u = \pi(x_\pi(t; x_0)) \in \mathbb{U}_a,\ \forall t \in [0, \infty]$.

However, it should be noted that $\mathbb{X}_a$ is not safe everywhere as there may not exist a control input that transitions a state close to the boundary towards the interior of $\mathbb{X}_a$. A safe set should have the property that if the system starts in the safe set, it stays inside for all time. Towards formally defining this property, let a set $\mathcal{C}$ be defined as the *0-superlevel set* of a continuously differentiable function $h : \mathbb{R}^n \to \mathbb{R}$, *i.e.,*

$$\mathcal{C} = \{x \in \mathbb{X} : h(x) \geq 0\},$$
$$\partial\mathcal{C} = \{x \in \mathbb{X} : h(x) = 0\}.$$

Then forward invariance and a safe set are defined as follows.

Definition 2 (Foward invariance)**.** The set $\mathcal{C}$ is *forward invariant* if for every $x_0 \in \mathcal{C}$, there exists a control policy $u = \pi(x) \in \mathbb{U}_a$ such that the trajectory of system in (1) $x_\pi(t; x_0) \in \mathcal{C},\ \forall t \in [0, \infty]$.

Definition 3 (Safe set)**.** The set $\mathcal{C}$ is a *Safe Set* if $\mathcal{C}$ is *forward invariant* and $\mathcal{C} \subseteq \mathbb{X}_a$.

A CBF renders a safe set and can be used to derive safe control inputs. Before defining CBFs, we must introduce extended class $\mathcal{K}_\infty$ functions. An extended class $\mathcal{K}_\infty$ function is a mapping $\alpha : \mathbb{R} \to \mathbb{R}$ that is strictly increasing and for which $\alpha(0) = 0$ holds. We define a continuous CBF as:

Definition 4 (Control Barrier Function)**.** Let $\mathcal{C} \subseteq \mathbb{X}_a$ be the 0-superlevel set of a continuously differentiable function $h : \mathbb{R}^n \to \mathbb{R}$, then h is a CBF in $\mathbb{X}_a$ for system in (1) if there exists an extended class $\mathcal{K}_\infty$ function α such that

$$\sup_{u\in\mathbb{U}_a} [L_f h(x) + L_g h(x)u] \geq -\alpha(h(x)) \tag{2}$$

for all $x \in \mathbb{X}_a$, where L_f, L_g represent Lie derivatives.

With the definition of a CBF, we may derive sufficient conditions for a safe system. According to the main result in [19], the following theorem holds:

Theorem 1 ([19, Theorem 2])**.** *If function h is a CBF for the system in* (1) *and $\frac{\partial h}{\partial x}(x) \neq 0$ for all $x \in \partial\mathcal{C}$, then any Lipschitz continuous controller $\pi(x) \in K_{cbf}(x)$ with*

$$K_{cbf}(x) = \{u \in \mathbb{U}_a : L_f h(x) + L_g h(x)u + \alpha(h(x)) \geq 0\}. \tag{3}$$

renders the set $\mathcal{C}$ safe. Additionally, the set $\mathcal{C}$ is asymptotically stable in $\mathbb{X}_a$.

With Theorem 1, we are able to ensure the safety of the system in (1) as long as a CBF is found and its gradient does not vanish on $\partial\mathcal{C}$. We show in Section V-D how to obtain a safe policy satisfying (3) using CBFs.

The objective of this work is to automatically synthesize a nCBF and verify it for the continuous state space. The problem is defined as follows.

Problem 1. *Given the system in* (1)*, state admissible set $\mathbb{X}_a$, convex input admissible set $\mathbb{U}_a$ and $\alpha(x) = \gamma x$ where γ is a positive constant, synthesize a nCBF that is denoted by $\hat{h}_w(x)$, where w are the parameters of the NN, and renders set $\mathcal{C}$ safe for the system in* (1)*. This is equivalent to*

$$\hat{\mathcal{C}} \subseteq \mathbb{X}_a, \tag{4a}$$

$$\textit{inequality (2) holds in } \mathbb{X}_a, \tag{4b}$$

where $\hat{\mathcal{C}} = \{x \in \mathbb{X} : \hat{h}_w(x) \geq 0\}$ is the 0-superlevel set of the nCBF.

Remark 1. The condition $\frac{\partial h}{\partial x}(x) \neq 0$ for all $x \in \partial C$ is omitted since it generally holds in our setting as we only consider a Tanh-based Fully-Connected Neural Network (FCNN). More specifically, since $\frac{\partial tanh}{\partial x}(x) \in (0, 1]$ for all x, the condition is only violated if either $w = 0$ or catastrophic cancellation occurs in the linear layers, which will almost surely never happen.

IV. Neural Control Barrier Function Training and Verification

In this work, we design a new empirical loss to synthesize an nCBF, which is introduced in Section IV-A. As the training set only contains a finite set of data points, the CBF conditions may not hold in the continuous state space. Therefore, in Section IV-B, we present the BBV to verify nCBFs. Nevertheless, it is often necessary to iterate through multiple training and verification cycles before successfully learning an nCBF. Thus, we introduce BBVT in Section IV-C, which embeds BBV in the training loop to accelerate training for certifiability.

A. Learning a Neural Control Barrier Function

The primary goal of this work is to train an NN $\hat{h}_w(x)$ until it satisfies conditions (4a) and (4b) and render a large forward invariant set. Towards this end, we leverage the main result in [4, Theorem 3], where a CBVF is shown to recover the maximum safe set subject to safety constraints. Contrary to [4], we are interested in infinite-horizon properties. Thus we extend the time-dependent Control Barrier-Value Function Variational Inequality (CBVF-VI) to the infinite-horizon. Let $h(x)$ denote the infinite-horizon CBVF and $\rho(x) : \mathbb{X} \to \mathbb{R}$ denote the signed-distance function for the set $\mathbb{X}_a$, which is defined as $\rho(x) = \inf_{y \in \mathbb{X}/\mathbb{X}_a} \|y - x\|$ if $x \in \mathbb{X}_a$ and $\rho(x) = -\inf_{y \in \mathbb{X}_a} \|y - x\|$ if $x \in \mathbb{X}/\mathbb{X}_a$. The infinite-horizon CBVF-VI is defined as

$$\begin{aligned} 0 = \min\{&\rho(x) - h(x), \\ &\max_{u \in \mathbb{U}_a} L_f h(x) + L_g h(x) u + \gamma h(x)\}. \end{aligned} \tag{5}$$

We use an NN $\hat{h}_w(x)$ to approximate the infinite-horizon CBVF $h(x)$. Then, the empirical loss is defined as follows:

$$\begin{aligned} \mathcal{L} = &\frac{1}{N_1} \sum_{x \in \mathbb{X}_a} \|\min\{\rho(x) - \hat{h}_w(x), \\ &\sup_{u \in \mathbb{U}_a} L_f \hat{h}_w(x) + L_g \hat{h}_w(x) u + \gamma \hat{h}_w(x) - \lambda\}\| \qquad \text{(6a)} \\ &+ \frac{1}{N_2} \sum_{x \in \mathbb{X}/\mathbb{X}_a} \max\{\hat{h}_w(x) + \lambda, 0\}. \qquad \text{(6b)} \end{aligned}$$

where λ is a small positive constant to encourage the strict satisfaction of the conditions. The loss term (6a) shapes the NN to be the solution of the infinite-horizon CBVF-VI introduced in (5), which encourages the satisfaction of condition (4b). The loss term (6b) ensures that the nCBF is negative in the inadmissible area $\mathbb{X}/\mathbb{X}_a$, which is equivalent to condition (4a). Since the system is control-affine, the optimal solution u^* for $\sup_{u \in \mathbb{U}_a}[L_f \hat{h}_w(x) + L_g \hat{h}_w(x) u]$ must be one of the vertices of $\mathbb{U}_a$. Let $\mathbb{U}_a^{\mathcal{V}}$ denote the vertices of the input admissible set, we choose control input $u^\star = \arg\max_{u \in \mathbb{U}_a^{\mathcal{V}}} L_g \hat{h}_w(x) u$. However, the Lie derivative of $\hat{h}_w(x)$ in the early training stage may not align with the Lie derivative of the true CBVF $h(x)$. This results in an undesirable optimization path and the NN can get stuck at deadlock. The occurrence of a deadlock situation signifies that improvements at certain data points cause constraint violations at other data points, as noted in [20]. To facilitate the training process and avoid deadlocks, we borrow ideas from [2], [6], which use a nominal controller to guide the training. Here, we train another neural network $\hat{h}_\phi$ with the same structure as $\hat{h}_w$ based on the loss of [21] and choose $u^* = \arg\max_{u \in \mathbb{U}_a^{\mathcal{V}}} \hat{h}_\phi(x + (f(x) + g(x)u)\Delta t)$ by simulating one step ahead to guide the training of the nCBF $\hat{h}_w$. To further guide the training, we may integrate the verification procedure with a so-called Counterexample Guided Inductive Synthesis (CEGIS) approach, as described in Section IV-C.

B. Verifying the learned Neural Control Barrier Function

Since the NN is trained on finite data points, one must note that the NN may not satisfy the CBF conditions everywhere in the state space, even if the empirical loss decreases to zero. In fact, condition (4b) may be violated almost everywhere, which means the NN may fail to render a forward invariant set and the safety guarantee no longer exists. In this section, we propose to use the BBV to verify the learned nCBF in the continuous state space. Specifically, our primary goal is to verify the satisfaction of conditions (4a) and (4b).

Before we explain our verification scheme in detail, we introduce some notations first. Let the partition of the state space be denoted as hyperrectangles $\mathbb{B}(x_i, \epsilon_i) = \{x : | x - x_i | \leq \epsilon_i\}$ centered at point $x_i \in \mathbb{X}$ with radius $\epsilon_i \in \mathbb{R}^n$,

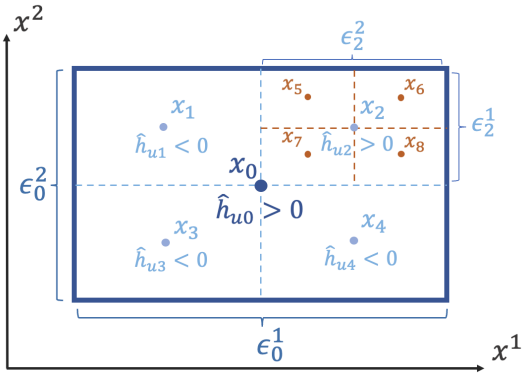


Fig. 2: An example of the BBV in a 2D state space. The scheme starts with a coarse partition $\mathbb{B}(x_0, \epsilon_0)$ and refines it using the Branch-and-Bound scheme. For each hyperrectangle $\mathbb{B}(x_i, \epsilon_i)$, $i = 0, 1, 2, \ldots$, upper bounds for the neural network are computed. In this case, the hyperrectangles $\mathbb{B}(x_0, \epsilon_0)$ and $\mathbb{B}(x_2, \epsilon_2)$ are refined as $\hat{h}_{u0} > 0$, $\hat{h}_{u2} > 0$.

see Fig. 2. Initially, all hyperrectangles have the same radius $\epsilon_i = \epsilon_{init}$. Let $\mathcal{B} = \{\mathbb{B}(x_0, \epsilon_0), \ldots, \mathbb{B}(x_N, \epsilon_N)\}$ denote the set of all hyperrectangles, $\mathcal{B}_{\mathbb{X}/\mathbb{X}_a} \subset \mathcal{B}$ denote the set of hyperrectangles that covers the inadmissible area $\mathbb{X}/\mathbb{X}_a$, and $\mathcal{B}_{\mathbb{X}_a} \subset \mathcal{B}$ denote the set of hyperrectangles that covers the admissible area.

To verify condition (4a), which is equivalent to $\hat{h}_w(x) < 0, \forall x \in \mathbb{X}/\mathbb{X}_a$, we rely on the linear bounds of the NN computed using CROWN [7]. The linear bounds are defined as follows:

$$\hat{h}_{li} \leq \hat{h}_w(x) \leq \hat{h}_{ui},\ x \in \mathbb{B}(x_i, \epsilon_i). \tag{7}$$

We use these linear bounds to certify the satisfaction of condition (4a). In particular, the upper bound $\hat{h}_{ui}$ can be used to check for non-positivity

$$\hat{h}_w(x) \leq \hat{h}_{ui} < 0, x \in \mathbb{B}(x_i, \epsilon_i), \mathbb{B}(x_i, \epsilon_i) \in \mathcal{B}_{\mathbb{X}/\mathbb{X}_a}. \tag{8}$$

However, this upper bound tends to be conservative when $\mathbb{B}(x_i, \epsilon_i)$ covers a large area. Therefore, we leverage the BBS that starts from the coarse partition and refines each hyperrectangle when $\hat{h}_{ui} > 0$ until $\hat{h}_{ui} \leq 0$ or $\epsilon_i \leq t_{gap}$ where $t_{gap} > 0$ is the minimum partition size, see Fig. 2. If condition (8) holds for all hyperrectangles in $\mathcal{B}_{\mathbb{X}/\mathbb{X}_a}$, then the condition (4a) holds in the continuous state space.

Although verifying condition (4a) is simple, verifying condition (4b) $\sup_{u \in \mathbb{U}_a}[L_f \hat{h}_w(x) + L_g \hat{h}_w(x) u] \geq -\gamma \hat{h}_w(x),\ \forall x \in \mathbb{X}_a$ is challenging. For improved readability, we denote $q(x) = \sup_{u \in \mathbb{U}_a}[L_f \hat{h}_w(x) + L_g \hat{h}_w(x) u + \gamma \hat{h}_w(x)]$. Hence, verifying condition (4b) is equivalent to verifying $q(x) \geq 0, \forall x \in \mathbb{X}_a$. Let q_{li} define a lower bound of $q(x)$ for $x \in \mathbb{B}(x_i, \epsilon_i)$. Then the following condition has to hold:

$$q(x) \geq q_{li} \geq 0,\ x \in \mathbb{B}(x_i, \epsilon_i), \mathbb{B}(x_i, \epsilon_i) \in \mathcal{B}_{\mathbb{X}_a}. \tag{9}$$

Similarly to condition (4a), the BBS starts from a coarse partition and refines each hyperrectangle when $q_{li} < 0$ until $q_{li} \geq 0$ or $\epsilon_i \leq t_{gap}$. If condition (9) holds for all hyperrectangles in $\mathcal{B}_{\mathbb{X}_a}$, then condition (4b) holds in the continuous state space.

However, the challenge arises in the computation of q_{li}. The computation of q_{li} can be reframed as an optimization problem within the hyperrectangle $\mathbb{B}(x_i, \epsilon_i)$

$$q_{li} = \min_x\ q(x) \tag{10a}$$
$$\text{s.t. } x \in \mathbb{B}(x_i, \epsilon_i). \tag{10b}$$

The term $q(x)$ is a complex function containing nonlinear dynamic functions f, g, a neural network $\hat{h}_w$ as well as its Jacobian, which renders a constrained Nonlinear Program (NLP) in (10a). The state-of-the-art NLP solver [22] requires gradients of the objective function, which involves computation of the Hessian of the NN. The expensive computation makes it impractical to solve (10a) directly.

Although computing the lower bound of $q(x)$ is quite complex, computing the bound of the components of $q(x)$ separately is much simpler. We can compute the bound of the NN using CROWN [7] and its Jacobian leveraging a recent result in [23] or [24]:

$$\hat{h}_{li} \leq \hat{h}_w(x) \leq \hat{h}_{ui}, \forall x \in \mathbb{B}(x_i, \epsilon_i), \tag{11}$$
$$J_{li} \leq \nabla \hat{h}_w(x) \leq J_{ui}, \forall x \in \mathbb{B}(x_i, \epsilon_i). \tag{12}$$

Furthermore, we can approximate the nonlinear dynamic functions f and g using Taylor Models as done in [25] or sampling:

$$x_{li} \leq f(x) + g(x)u^\star \leq x_{ui}, \forall x \in \mathbb{B}(x_i, \epsilon_i). \tag{13}$$

In (10a), the objective function depends on the variable x and is constrained within the feasible region for x. We simplify (10a) by considering three independent variables subject to independent constraints. This results in

$$q'_{li} = \min_{h,J,x}\ q'(h, J, x) = \langle J, x \rangle + \gamma h \tag{14a}$$
$$\text{s.t. } \hat{h}_{li} \leq h \leq \hat{h}_{ui}, \tag{14b}$$
$$J_{li} \leq J \leq J_{ui}, \tag{14c}$$
$$x_{li} \leq x \leq x_{ui}, \tag{14d}$$

where x denotes the value of $f(x) + g(x)u$, h denotes the value of $\hat{h}_w(x)$ and J denotes the value of $\nabla \hat{h}_w(x)$. When (11), (12), and (13) are over-approximations of the true intervals, it is clear that the optimal solution q'_{li} from (14a) is an over-approximation of the optimal solution q_{li} from (10a), which means $q'_{li} \leq q_{li}$. To efficiently solve (14a), we may compute the optimal solution independently for each term, taking the minimum over the set of vertices.

Although the theoretical complexity of the BBV is still exponential in the dimension of the state space, it improves the scalability in practice. One must note that our method

is a sound verification method instead of a complete one, which means the failure to obtain $\mathcal{B}_{\mathbb{X}/\mathbb{X}_a}$ and $\mathcal{B}_{\mathbb{X}_a}$ that satisfy condition (8), (9) does not imply the invalidation of the nCBF, as we over-approximate the conditions. We want to emphasize that the chosen over-approximation method, CROWN, has been the winning strategy at the Verification of Neural Networks Competition for multiple years [26].

C. Branch and Bound Verification-in-the-loop Training

Although the BBV provides a practical way to certify the NN as nCBF, it requires several training and verification processes until an nCBF is obtained. Therefore, leveraging the information from the verification and ensuring the satisfaction of conditions (4a) and (4b) becomes the task of BBVT. This type of method is also known as CEGIS [27]. See Fig. 1 for an overview of the framework.

We start with the initial fixed training dataset $\mathcal{D}$ that contains a number of uniformly sampled points. During the training procedure, we optimize the NN to decrease the loss in (6) using $\mathcal{D}$. After k epochs, the verifier starts with a coarse partition of the state space. The upper bound $\hat{h}_{ui},\ \forall \mathbb{B}(x_i, \epsilon_i) \in \mathcal{B}_{\mathbb{X}/\mathbb{X}_a}$ and lower bound $q'_{li},\ \forall \mathbb{B}(x_i, \epsilon_i) \in \mathcal{B}_{\mathbb{X}_a}$ are computed. The hyperrectangles, whose $\hat{h}_{ui} \geq 0$ or $q'_{li} \leq 0$, are split until $\epsilon_i \leq t_{gap}$. After reaching the minimum partition size t_{gap}, the hyperrectangles whose $\hat{h}_{ui} \geq 0$ or $q'_{li} \leq 0$ are treated as the violation areas. The center points are added to the CE dataset and the training procedure is repeated until the verifier returns `satisfaction` or the maximum number of iterations $n_{\max}$ is reached.

Note that although the universal approximation theorem in [28] guarantees the existence of $\hat{h}_w(x)$ to be an nCBF that renders maximum forward invariant set, this is under the assumption that the NN has a sufficient number of neurons. The training procedure is not guaranteed to converge to an nCBF, but if the verifier returns `satisfaction`, the NN is an nCBF for the given system in the continuous state space. It is possible to introduce adversarial training, i.e. training on the worst-case state in a region around each sample x, to improve the convergence to a verifiable nCBF [8].

V. Results

In this section, we evaluate our proposed framework on two systems: an inverted pendulum and a 2D navigation task. The experimental setup is introduced in Section V-A. In Section V-B and Section V-C, we provide a comprehensive assessment on the inverted pendulum, addressing the verification efficiency and the size of the safe set, respectively. In Section V-D, we consider a 2D navigation task with nonconvex constraints to display the practical use of our framework and combine the nCBF with RL to achieve safe policy learning. Our code is available on GitHub[1].

We consider the following baseline methods:

- LST: The Level Set Toolbox (LST) [29] generates a safe value function by Hamilton-Jacobian-Issac Reachability Analysis (HJI-RA) over a discrete grid.
- NeuralCLBF: Neural Control Lyapunov Barrier Function (NeuralCLBF) [2] parametrizes the CBF as an NN and optimizes it according to their empirical loss based on (2) and a nominal safe set.
- SMT: [6] trains a neural Lyapunov function with SMT generating counterexamples and ensures the validation of the result. The constraints considered by SMT are conditions (4a) and (4b). To have a fair comparison, the training loss is chosen to be the same as in (6).

[1] `https://github.com/tud-amr/ncbf-simultaneous-synthesis-and-verification`

A. Experimental Setup

1) Inverted Pendulum: Let $s = [\theta, \dot{\theta}] \in \mathbb{X} \subset \mathbb{R}^2$ be the state variable and $u \in \mathbb{U} \subset \mathbb{R}$ be the control input. We consider the state space $\mathbb{X} = \{s : \theta \in [-\pi, \pi], \dot{\theta} \in [-5, 5]\}$ and the input space $\mathbb{U} = \{u : u \in [-12, 12]\}$. The dynamics of the inverted pendulum are given by:

$$\begin{aligned} \dot{\theta} &= \dot{\theta}, \\ \ddot{\theta} &= \frac{3g}{2l} sin(\theta) - \frac{3\beta}{ml^2}\dot{\theta} + \frac{3}{ml^2}u, \end{aligned} \tag{15}$$

where $m = 1$, $b = 0.1$, $g = 9.81$, and $l = 1$. The state admissible set is $\mathbb{X}_a = \{s : \theta \in [-\frac{5\pi}{6}, \frac{5\pi}{6}], \dot{\theta} \in [-4, 4]\}$ and the input admissible set is $\mathbb{U}_a = \mathbb{U}$, see Fig. 3.

2) 2D navigation task: We consider a 2D navigation task in which a point robot should reach a goal position while avoiding obstacles, see Fig. 6a. Let $s = [x, y, \dot{x}, \dot{y}] \in \mathbb{X} \subset \mathbb{R}^4$ be the state variable and $u = [a_x, a_y] \in \mathbb{U} \subset \mathbb{R}^2$ be the control input representing the acceleration along the x-axis and y-axis. The dynamics of the point robot are:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \ddot{x} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} s^T + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a_x \\ a_y \end{bmatrix} \tag{16}$$

We consider the admissible position set $X^1 = \{s : x \in [0, 4], y \in [0, 4]\}$ except the obstacle set $X^2 = \{s : x \in [1.5, 2.5], y \in [0, 2]\}$, together with velocity constraints $X^3 = \{s : \dot{x} \in [-1, 1], \dot{y} \in [-1, 1]\}$. Thus, the state admissible set is $\mathbb{X}_a = X^1 \cup (X^2)^C \cup X^3$, where $(\cdot)^C$ represents the complement of a set. See Figure 6 for a pictorial representation of the set. The input admissible set is $U_a = \{u : a_x \in [-1, 1], a_y \in [-1, 1]\}$.

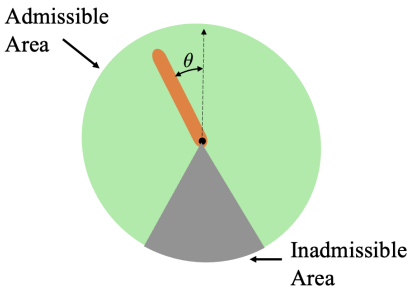
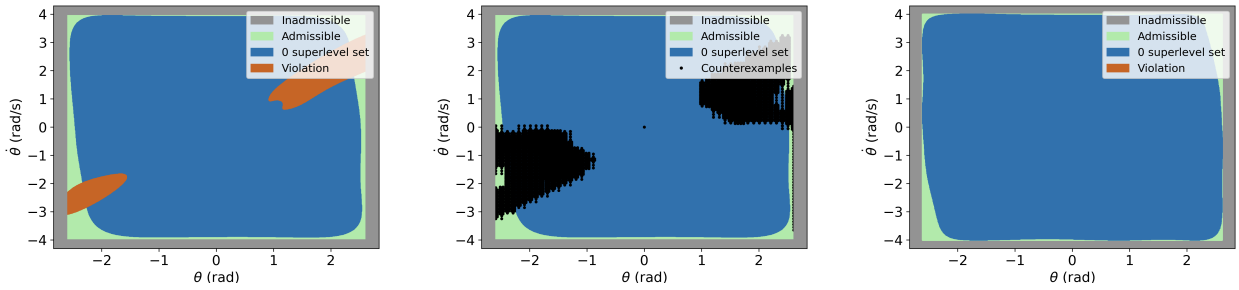


Fig. 3: The workspace of the considered inverted pendulum.



(a) Training without verification-in-the-loop (b) Distribution of counterexamples (c) Training with verification-in-the-loop

Fig. 4: Shapes of 0-superlevel sets of NNs trained with and without BBVT for the inverted pendulum. In Fig. 4a the NN is trained with a fixed dataset and evaluated on a denser testing dataset to showcase that condition (4b) is not satisfied for the continuous state space. Figure 4b shows the counterexamples added to the dataset according to BBVT. Figure 4c showcases that, after training the NN with BBVT, no validations are detected since the NN is an nCBF.

TABLE I: Hyper-parameter for nCBF Training.

γ	0.5	λ	0.05
learning rate r	10^{-3}	learning rate decay β	0.995
verify after every k epochs	20	minimum partition gap t_{gap}	0.005
initial radius ϵ_{init} (inverted pendulum)	[0.2, 0.2]	initial radius ϵ_{init} (2D navigation)	[0.2, 0.2, 0.2, 0.2]
Num. fixed points (inverted pendulum)	10^5	Num. fixed points (2D navigation)	10^6
n_{max}	100		

3) Training Configuration: For both systems, we train the nCBF using Pytorch on NVIDIA A40, and Stochastic Gradient Descent is used as the optimizer to avoid local minima. The used hyper-parameters can be found in Table I. For the inverted pendulum, we choose a Tanh-based FCNN with one hidden layer which consists of 36 neurons. For the 2D navigation task, a larger Tanh-based FCNN is required since the shape of the environment is more complex. Here we choose a Tanh-based FCNN with two hidden layers, each of which consists of 256 neurons.

B. Verification and Efficiency

In this section, we use the inverted pendulum to discuss the certification of the trained NN as an CBF. To showcase the disadvantage of training without verification, we train the nCBF with a fixed data set and stop training after 200 epochs. We then examine the satisfaction of condition (4b) with a denser testing dataset. The 0-superlevel set of the trained NN is shown in blue in Fig. 4a. The orange area indicates the testing data points that violate condition (4b).

We resume the training with the same dataset and use BBVT to augment the training dataset with CEs every k epochs until the verifier returns `satisfaction`. Figure 4b shows the distribution of the CEs after the first verification loop. As we augment the dataset, the verifier returns `satisfaction` after 240 epochs, see Fig. 4c.

To highlight the efficacy of BBVT, we evaluate the training time, verification time, and the ratio of violation areas for our framework and the baseline methods. The results are shown in Table II. We first compare our method with LST [29]. The table shows the results of LST for two different grid gaps, which are 0.2 and 0.05 respectively. It is evident that an increased grid density leads to improved accuracy at the cost of longer computation time. However, a dense grid map is not always possible, since the memory space of LST grows exponentially, which is referred to as the *Curse of Dimensionality*. With a grid gap of 0.05, LST requires 24.41kB memory space, while we only need to store the parameters of the nCBF, which is 1.2kB. This is important for embedded devices such as the control unit on drones.

Then, we compare our method with NeuralCLBF. Due to the lack of a verification process and counterexample data set, the fixed data set for NeuralCLBF contains 10^6 data points in order to have a fair comparison with our method. Since NeuralCLBF learns an nCBF based on a nominal safe set, the training process is assisted by prior knowledge and results in less training time, see Table II. However, there are sparse areas that violate the conditions as discussed in [2] and how these sparse areas grow with the complexity of the system has not been studied yet.

We also compare our method with SMT. However, SMT did not return `satisfaction` until the maximum number of iterations $n_{\max}$ was reached, see Table II. Although there exist some works [5], [6] that use SMT to verify a neural controller, they only use a very simple FCNN with around 9 neurons. In our case, the computation time of SMT grows dramatically since the NN is more complex. Also, SMT can only generate several counterexamples at each iteration, while BBVT generates all the counterexamples in state space $\mathbb{X}$, which is more efficient than SMT.

C. Size of Safe Set

We will compare the size of the forward invariant set derived using our framework and the baseline methods in this section. Since SMT failed to verify the nCBF and LST with a grid gap of 0.2 has a large violation area, we compare our framework only against LST with a grid gap of 0.05 and NeuralCLBF with the nominal safe set being $\mathbb{X}_n = \{s : \|s\| < \frac{3\pi}{4}\}$).

The forward invariant sets derived by the different methods are illustrated in Fig. 5. We see that the size of the forward invariant set from NeuralCLBF is conservative, while our

TABLE II: Verification and Efficiency Comparison for the inverted pendulum. BBVT is compared against LST, NeuralCLBF, and SMT to synthesize an nCBF. LST and NeuralCLBF do not verify their safe value function, which is represented by '-' in columns 3 and 4. To validate the verification process, we calculate the ratio of points that violate condition (4b) on a uniform grid with a size of $10^3 \times 10^3$ within the state space $\mathbb{X}$.

	Stop criteria	Total computation time (s)	Average verification time (s/epoch)	Average generation time (s/per counterexample)	Violation points/testing points (%)
LST(0.2)	value converges	5.34	-	-	1.9
LST(0.05)	value converges	104.48	-	-	0.0064
NeuralCLBF	loss converges	584.6	-	-	0.0013
SMT	max # iter reached	5311.68	14.73	1.34	0.7742
BBVT(ours)	verified	1214.15	16.20	0.004	0.0

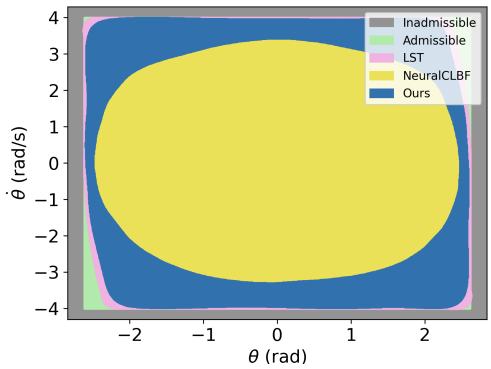


Fig. 5: The forward invariant set of safe value functions obtained by different methods for the inverted pendulum.

method approximates CBVF and renders a safe set that is close to the maximum forward invariant set. Since the forward invariant set of the CBF is always a subset of that from HJI-RA, which is discussed in [4], it is not surprising that LST renders a larger safe set than ours. We note that $\lambda > 0$ in (6a) encourages the satisfaction of the CBF conditions at the expense of rendering a smaller safe set.

D. Application of Neural Control Barrier Functions to Safe Policy Learning

In this section, we use RL to address the 2D navigation task introduced in Section V-A.2. Let $s_g = [x_g, y_g, 0, 0]$ be the goal state. The step reward is defined as $r_t = -0.01 \cdot \|s - s_g\|$, the terminal reward is $r_{\text{collision}} = -5$ when the robot collides with the obstacles and $r_{\text{goal}} = 10$ when the robot reaches the goal area $X_g = \{s : \|s - s_g\| < \epsilon\}$ where $\epsilon = 0.1$ is the goal tolerance. We use Proximal Policy Optimization [30] to train the agent and solve

$$\begin{aligned} u_{\text{safe}} = &\arg\min_{u \in \mathbb{U}_a} \|u - u_{\text{RL}}\|^2 \\ &s.t.\ L_f\hat{h}_w(x) + L_g\hat{h}_w(x)u + \alpha(\hat{h}_w(x)) \geq 0 \end{aligned} \tag{17}$$

to project the action u_{RL} of the RL policy to the safe action u_{safe} with the least modification.

We note that, theoretically, this controller guarantees safety with infinite control frequency. However, a continuous controller is not possible to implement on discrete control units. This limits the safety guarantees we may provide. How to address the gap between continuous controllers and their discrete implementations remains an open question.

Figure 6a shows all trajectories performed during the training and we can see that several trajectories collide with the obstacles, while there are no unsafe trajectories in Figure 6b with nCBF as a safety filter. However, we observe that the average reward with nCBF is larger than for nominal RL without nCBF in the very early stage but has a slower growth rate and converges to a lower reward level compared with nominal RL, see Fig. 6c. The reason is that nCBF provides prior knowledge about the environment and the agent could avoid exploring unsafe regions in the early stage and gain a higher reward than nominal RL. However, the forward invariant set is still suboptimal as discussed in Section V-C, which means only a suboptimal policy is learned and exploration is restricted. Nevertheless, we believe that provided safety guarantees are beneficial in safety-critical applications.

VI. Conclusion

In this work, we presented a framework that simultaneously synthesizes and verifies continuous Neural Control Barrier Functions (nCBFs). To this end, we leveraged bound propagation techniques and the Branch-and-Bound scheme to efficiently verify neural networks as Control Barrier Functions (CBFs) in the continuous state space. In experiments, we showed that our framework efficiently synthesizes an nCBF which renders a larger safe set than state-of-the-art methods without requiring an initial guess.

Since the memory requirements and computation time of the Branch-and-Bound Verification scheme increase exponentially with the system dimension, in future work, we may address the scalability of our framework.

References

[1] C. Dawson, S. Gao, and C. Fan, "Safe control with learned certificates: A survey of neural lyapunov, barrier, and contraction methods for robotics and control," *IEEE Transactions on Robotics*, 2023.

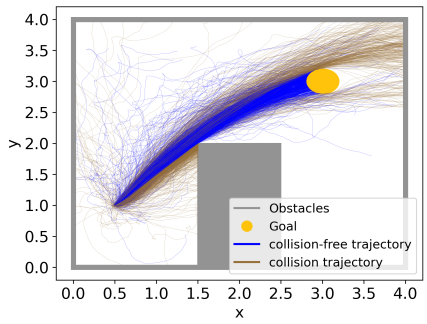
[2] C. Dawson, Z. Qin, S. Gao, and C. Fan, "Safe nonlinear control using robust neural lyapunov-barrier functions," in *Conference on Robot Learning*, pp. 1724–1735, PMLR, 2022.

[3] B. Dai, P. Krishnamurthy, and F. Khorrami, "Learning a better control barrier function," in *2022 IEEE 61st Conference on Decision and Control (CDC)*, IEEE, 2022.

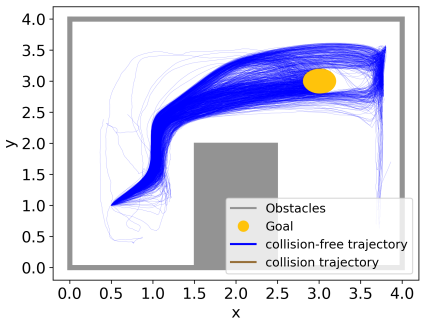
[4] J. J. Choi, D. Lee, K. Sreenath, C. J. Tomlin, and S. L. Herbert, "Robust control barrier–value functions for safety-critical control," in *60th IEEE Conference on Decision and Control (CDC)*, IEEE, 2021.

[5] A. Peruffo, D. Ahmed, and A. Abate, "Automated and formal synthesis of neural barrier certificates for dynamical models," in *Tools and Algorithms for the Construction and Analysis of Systems: 27th International Conference, TACAS 2021*, Springer, 2021.

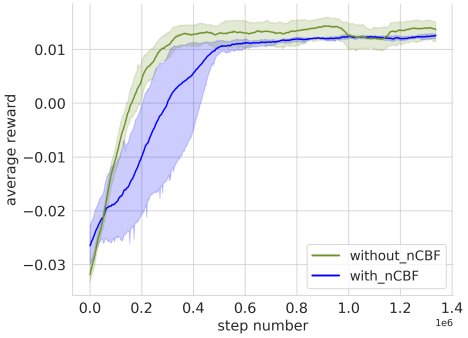
[6] Y.-C. Chang, N. Roohi, and S. Gao, "Neural lyapunov control," in *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019.



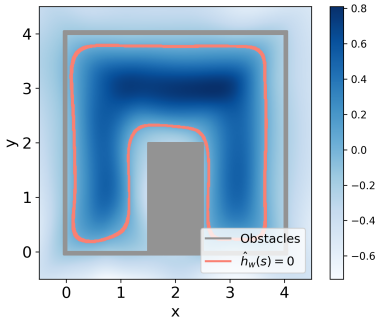
(a) Trajectories during learning without nCBF



(b) Trajectories during learning with nCBF



(c) Moving average reward



(d) Heatmap of the nCBF.

Fig. 6: 6a and 6b show all the trajectories during RL training. 6c illustrates the moving average reward of every 2048 steps. 6d is the slice of heatmap of $\hat{h}_w(x)$ with velocity $\dot{x} = 0.2, \dot{y} = 0.2$.

[7] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel, "Efficient neural network robustness certification with general activation functions," *Advances in neural information processing systems*, vol. 31, 2018.

[8] F. B. Mathiesen, S. C. Calvert, and L. Laurenti, "Safety certification for stochastic systems via neural barrier functions," *IEEE Control Systems Letters*, vol. 7, 2022.

[9] Q. Nguyen, A. Hereid, J. W. Grizzle, A. D. Ames, and K. Sreenath, "3d dynamic walking on stepping stones with control barrier functions," in *IEEE 55th Conference on Decision and Control (CDC)*, IEEE, 2016.

[10] X. Xu, T. Waters, D. Pickem, P. Glotfelter, M. Egerstedt, P. Tabuada, J. W. Grizzle, and A. D. Ames, "Realizing simultaneous lane keeping and adaptive speed regulation on accessible mobile robot testbeds," in *2017 IEEE Conference on Control Technology and Applications (CCTA)*, pp. 1769–1775, IEEE, 2017.

[11] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs for safety critical systems," *IEEE Transactions on Automatic Control*, no. 8, 2016.

[12] A. A. Ahmadi and A. Majumdar, "Some Applications of Polynomial Optimization in Operations Research and Real-Time Decision Making," *arXiv e-prints*, p. arXiv:1504.06002, Apr. 2015.

[13] M. Srinivasan, M. Abate, G. Nilsson, and S. Coogan, "Extent-compatible control barrier functions," *Systems & Control Letters*, 2021.

[14] M. Srinivasan, A. Dabholkar, S. Coogan, and P. A. Vela, "Synthesis of control barrier functions using a supervised machine learning approach," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020.

[15] D. Du, S. Han, N. Qi, H. B. Ammar, J. Wang, and W. Pan, "Reinforcement learning for safe robot control using control lyapunov barrier functions," *arXiv preprint arXiv:2305.09793*, 2023.

[16] Y. Yang, Y. Jiang, Y. Liu, J. Chen, and S. E. Li, "Model-free safe reinforcement learning through neural barrier certificate," *IEEE Robotics and Automation Letters*, vol. 8, no. 3, pp. 1295–1302, 2023.

[17] N. Boffi, S. Tu, N. Matni, J.-J. Slotine, and V. Sindhwani, "Learning stability certificates from data," in *Conference on Robot Learning*, pp. 1341–1350, PMLR, 2021.

[18] L. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, L. Daniel, D. Boning, and I. Dhillon, "Towards fast computation of certified robustness for relu networks," in *International Conference on Machine Learning*, pp. 5276–5285, PMLR, 2018.

[19] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control barrier functions: Theory and applications," in *2019 18th European Control Conference (ECC)*, 2019.

[20] S. Liu, C. Liu, and J. Dolan, "Safe control under input limits with neural control barrier functions," in *Conference on Robot Learning*, pp. 1970–1980, PMLR, 2023.

[21] J. F. Fisac, N. F. Lugovoy, V. Rubies-Royo, S. Ghosh, and C. J. Tomlin, "Bridging hamilton-jacobi safety analysis and reinforcement learning," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8550–8556, IEEE, 2019.

[22] D. G. Luenberger and Y. Ye, *Linear and Nonlinear Programming*. Cham, Switzerland: Springer International Publishing, Nov. 2021.

[23] Z. Shi, Y. Wang, H. Zhang, J. Z. Kolter, and C.-J. Hsieh, "Efficiently computing local lipschitz constants of neural networks via bound propagation," *Advances in Neural Information Processing Systems*, vol. 35, pp. 2350–2364, 2022.

[24] J. Laurel, R. Yang, G. Singh, and S. Misailovic, "A dual number abstraction for static analysis of clarke jacobians," *Proceedings of the ACM on Programming Languages*, vol. 6, no. POPL, pp. 1–30, 2022.

[25] M. Streeter and J. V. Dillon, "Automatically bounding the taylor remainder series: Tighter bounds and new applications," *arXiv preprint arXiv:2212.11429*, 2022.

[26] C. Brix, M. N. Müller, S. Bak, T. T. Johnson, and C. Liu, "First three years of the international verification of neural networks competition (vnn-comp)," 2023.

[27] A. Abate, C. David, P. Kesseli, D. Kroening, and E. Polgreen, "Counterexample guided inductive synthesis modulo theories," in *International Conference on Computer Aided Verification*, pp. 270–288, Springer, 2018.

[28] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, no. 5, 1989.

[29] I. M. Mitchell *et al.*, "A toolbox of level set methods," *UBC Department of Computer Science Technical Report TR-2007-11*, 2007.

[30] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.