Simultaneous Synthesis and Verification of Neural Control Barrier Functions Through Branch-and-Bound Verification-in-the-Loop Training

Wang, Xinyu; Knoedler, Luzia; Mathiesen, Frederik Baymler; Alonso-Mora, Javier

DOI
10.23919/ECC64448.2024.10591251

Publication date
2024

Document Version
Final published version

Published in
Proceedings of the European Control Conference, ECC 2024

Citation (APA)
Wang, X., Knoedler, L., Mathiesen, F. B., & Alonso-Mora, J. (2024). Simultaneous Synthesis and Verification of Neural Control Barrier Functions Through Branch-and-Bound Verification-in-the-Loop Training. In *Proceedings of the European Control Conference, ECC 2024* (pp. 571-578). IEEE. https://doi.org/10.23919/ECC64448.2024.10591251

2024 European Control Conference (ECC)
June 25-28, 2024. Stockholm, Sweden

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 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.**

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.

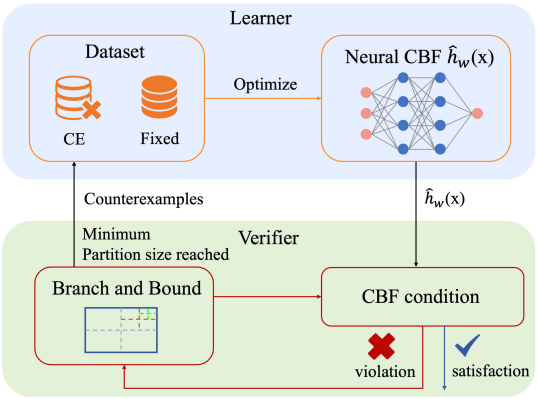


Fig. 1: A schematic overview of the presented Branch-and-Bound Verification-in-the-Loop Training. The framework comprises of two key components: the learner and the verifier, which operate sequentially. The learner optimizes the nCBF using a fixed dataset and a counterexample dataset. The verifier leverages 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 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 be a CBF. The BBS is applied

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 those of the European Union or the 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 x.wang-55@student.tudelft.nl {l.knoedler, j.alonsomora}@tudelft.nl

[2]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

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 an 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 an 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 a consider 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) encourages 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 a 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 $\epsilon_i \in \mathbb{R}^n$,

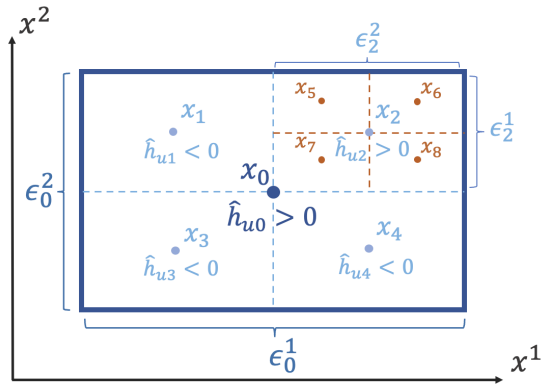


Fig. 2: An example of the BBV in a 2D state space. The scheme starts with a coarse partition $\mathbb{B}(x_0, \epsilon_0)$ and refines it using the Branch-and-Bound scheme. For each hyperrectangle $\mathbb{B}(x_i, \epsilon_i)$, $i = 0, 1, 2, \ldots$, upper bounds for the neural network are computed. 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, the NN $\hat{h}_w$ 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 CEs 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).

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 Fig. 6 for a pictorial representation of the set. The input admissible set is $U_a = \{u : a_x \in [-1, 1], a_y \in [-1, 1]\}$.

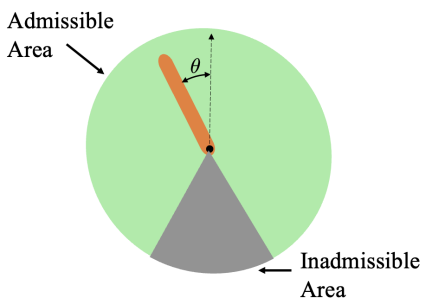
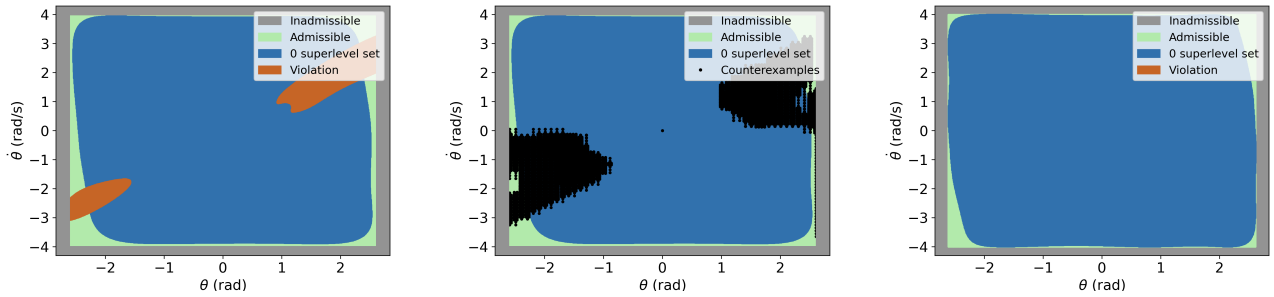


Fig. 3: The workspace of the considered inverted pendulum.

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



(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 CEs 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 a 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 CE 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 CEs at each iteration, while BBVT generates all the CEs 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

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/test 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

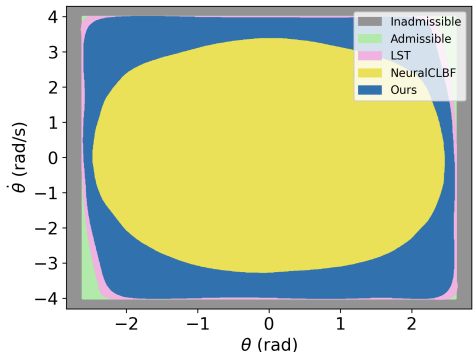


Fig. 5: The forward invariant set of safe value functions obtained by different methods for the inverted pendulum.

invariant set from NeuralCLBF is conservative, while our 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 Fig. 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 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.

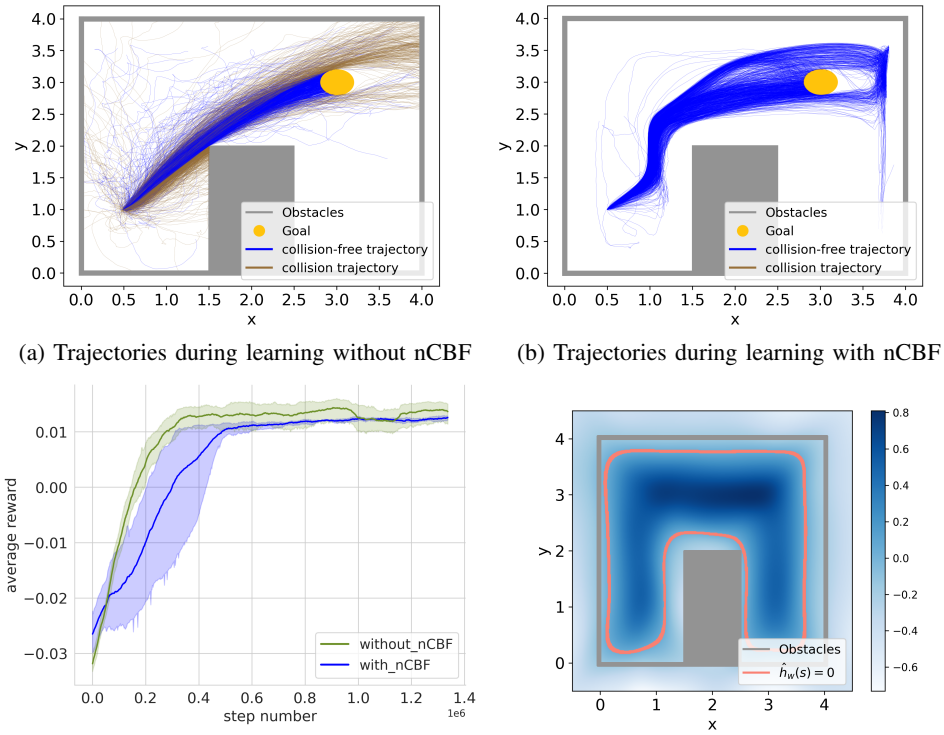
[2] C. Dawson, Z. Qin, S. Gao, and C. Fan, "Safe nonlinear control using robust neural lyapunov-barrier functions," in *Conference on Robot Learning*, pp. 1724–1735, PMLR, 2022.

[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.