Learning Neural Network Barrier Functions with Termination Guarantees

Shaoru Chen[1], Lekan Molu[1], Mahyar Fazlyab[2]

***Abstract*— Barrier functions are a general framework for establishing a safety guarantee for a system. However, there is no general method for finding these functions. To address this shortcoming, recent approaches use self-supervised learning techniques to learn these functions using training data that are periodically generated by a verification procedure, leading to a verification-aided learning framework. Despite its immense potential in automating barrier function synthesis, the verification-aided learning framework does not have termination guarantees and may suffer from a low success rate of finding a valid barrier function in practice. In this paper, we propose a holistic approach to address these drawbacks. With a convex formulation of the barrier function synthesis, we propose to first learn an empirically well-behaved NN basis function and then apply a fine-tuning algorithm that exploits the convexity and counterexamples from the verification failure to find a valid barrier function with *finite-step termination guarantees*: if there exist valid barrier functions, the fine-tuning algorithm is guaranteed to find one in a finite number of iterations. We demonstrate that our fine-tuning method can significantly boost the performance of the verification-aided learning framework on examples of different scales and using various neural network verifiers.**

I. Introduction

Providing safety and stability guarantees is a central goal in controller design for dynamical systems. These guarantees are often established by finding certificate functions. In particular, barrier functions aim to prove that the trajectories of an autonomous dynamical system remain inside a safe set over an infinite time horizon. Similarly, control barrier functions provide a general framework for designing control inputs that render a safe set invariant. Despite their generality, choosing an appropriate functional form and finding its parameters requires either computationally prohibitive methods or case-by-case analysis based on expert knowledge. To mitigate this, recent approaches propose to learn certificate functions from data using function approximators such as neural networks (NNs) [1]. However, to achieve deterministic guarantees, these approaches must be accompanied by a verification procedure, which either proves that the certificate is valid or generates counterexamples that can be added to the dataset for retraining the certificate function candidate [2]–[6]. We refer to this general framework of learning certificate functions with formal guarantees as *verification-aided learning* and summarize its schematics in Fig. 1. Despite the immense potential of this verification-aided learning framework in automating the certificate learning process, there is no guarantee that the updated certificate function will be 'closer' to a valid one in each iteration. Therefore, the alternation between training and verification may never terminate.

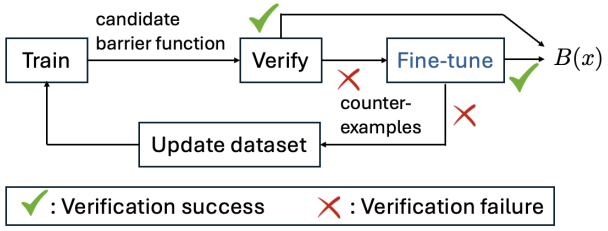


Fig. 1: The verification-aided learning framework with our proposed fine-tuning method (highlighted in blue).

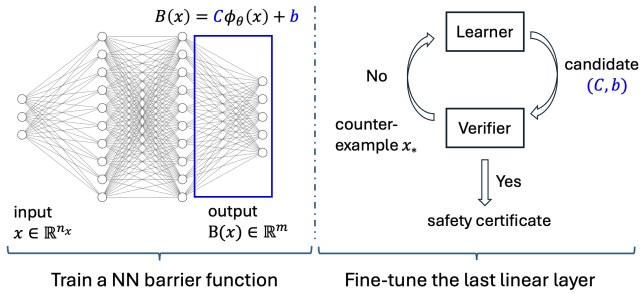


Fig. 2: Illustration of the fine-tuning method. After training the NN barrier function (left), we fine-tune the last linear layer through a counterexample-guided inductive synthesis framework (right) which enjoys termination guarantees.

Motivated by this challenge, in this work, we propose a principled approach to fine-tune the learned NN certificate function candidate with 'monotonic' improvement guarantees. In particular, we consider safety verification of a NN policy with the goal of finding a valid barrier function to certify avoidance of unsafe sets. As illustrated in Fig. 2, after training the NN barrier function, we propose to fine-tune the last linear layer of the NN through a counterexample-guided inductive synthesis (CEGIS) framework [7], [8] with termination guarantees, i.e., if there exist valid barrier functions, the fine-tuning algorithm is guaranteed to find one in a finite number of steps. To achieve such a guarantee, we formulate a NN *vector barrier function*, inspired by the related work in safety analysis for continuous-time systems [9], which maintains the *convexity* of the barrier function formulation and enables flexible tuning of the expressivity of the function

[1] Shaoru Chen and Lekan Molu are with Microsoft Research, 300 Lafayette Street, New York, NY, 10012, USA. Email: {shaoruchen, lekanmolu}@microsoft.com.

[2] Mahyar Fazlyab is with the Mathematical Institute for Data Science, Johns Hopkins University, USA. Email: mahyarfazlyab@jhu.edu.

class, e.g., by adjusting the architecture of the NN. The intuition behind our method in Fig. 2 is that we can empirically learn a good barrier function candidate from data and efficiently close the gap of providing formal guarantees through convex optimization. The **contributions** of this paper are summarized as follows.

1) We propose a novel verification-aided learning framework with a fine-tuning step to learn NN certificate functions with formal guarantees. The proposed fine-tuning approach follows an analytic center cutting-plane method [10] and enjoys finite-step termination guarantees. This enables our framework to benefit from both the learning capacity of NNs and performance guarantees from convex optimization.
2) We formulate a NN vector barrier function for safety analysis of discrete-time autonomous systems. This formulation allows easy tuning of the function class complexity and enables posing the barrier function synthesis problem as a convex one.
3) We demonstrate that the proposed fine-tuning method can significantly boost the success rate and runtime of the verification-aided learning framework on numerous examples of varying scales and with different NN verifiers, namely a mixed-integer programming (MIP)-based verifier [11] and a GPU-accelerated verifier α, β-CROWN [12].

A. Related work

a) Learning certificate functions: Certificate functions such as (control) barrier/Lyapunov functions are powerful tools for certifying the safety/stability of a dynamical system or designing safe/stabilizing controllers. Motivated by the difficulty of finding certificate functions for general nonlinear systems, learning a NN certificate function has been investigated in [13]–[20] with promising results. However, in these works, verification of the learned certificate functions is treated as a separate task, which brings the question of refining the learned NN once the verification fails.

b) Verification of NN certificate functions: NN verification concerns formally certifying the properties of a trained NN [21]. The rapid development of NN verification in recent years has provided us with a rich set of tools with constantly improving scalability and numerical efficiency [12], [22], [23]. Going beyond verification, in this work, we consider synthesizing a NN certificate function using the NN verification tools as a sub-routine.

c) Counterexample-guided synthesis: Counterexample-guided inductive synthesis (CEGIS) [7] is a general framework that utilizes the information or counterexamples from verification failure to progressively refine the solutions. CEGIS has been widely applied to NN certificate function synthesis in [2]–[4], [6], [24], [25]. Although the counterexamples are believed to be informative and useful for the learner, there is often no guarantee that CEGIS will find a feasible solution even if there exists one. From the theoretical perspective, [8], [26], [27] show that such a guarantee can be achieved following a cutting-plane method [10] with a convex parameterization of the certificate function, but they do not consider how such results can boost learning of a NN certificate function.

II. Problem Statement

We consider the dynamical system

$$x_{k+1} = f(x_k, u_k) \tag{1}$$

where $x_k \subseteq \mathbb{R}^{n_x}$ is the state and $u_k \in \mathbb{R}^{n_u}$ is the control input. We assume a policy $u_k = \pi(x_k)$ is given, which gives rise to the autonomous, closed-loop dynamics

$$x_{k+1} = f_\pi(x_k) := f(x_k, \pi(x_k)). \tag{2}$$

In this work, we particularly consider π as a NN policy. Let $\mathcal{X} \subseteq \mathbb{R}^{n_x}$ be a compact set denoting the workspace of the system, and $\mathcal{X}_0, \mathcal{X}_u \subset \mathcal{X}$ denote the set of initial conditions and unsafe sets, respectively. We call system (2) *safe* if all trajectories in the workspace starting from the initial set $\mathcal{X}_0$ will never reach the unsafe set $\mathcal{X}_u$.

Problem 1 (Safety Verification): Given system (2), the workspace $\mathcal{X}$, the initial set $\mathcal{X}_0 \subseteq \mathcal{X}$ and the unsafe set $\mathcal{X}_u \subseteq \mathcal{X}$, show that the system is safe, i.e., if $x_0 \in \mathcal{X}_0$, for all $k \geq 0$, we have $x_k \in \mathcal{X} \Rightarrow x_k \notin \mathcal{X}_u$.

Verifying the safety of the NN policy requires analyzing the behavior of system (2) over an infinite horizon. Following [28], we aim to find a barrier function as a certificate of safety. However, the presence of NNs in the dynamics brings significant challenges in numerical complexity. While a NN barrier function can be parameterized and learned using empirical risk minimization (ERM), a formal verification of the trained NN barrier function is required. In case of verification failure, the whole NN barrier function must be retrained and such a 'train-then-verify' procedure may never terminate.

In this work, we propose a more principled way to learn a NN barrier function with improved efficiency. In Section III, we formulate and train a NN vector barrier function, and propose a fine-tuning method with termination guarantees in Section IV. In Section V, we present the proposed verification-aided learning framework and implementation details. Numerical examples are provided in Section VI. Section VII concludes the paper.

III. NN Vector Barrier Function

First introduced in [28], the barrier function has been a popular method for safety verification of autonomous systems. Although rich theories and applications have been explored for continuous-time systems, barrier functions for discrete-time systems have received less attention. In this section, we introduce formulations of discrete-time barrier functions and propose a discrete-time vector barrier function adapted from [9] to maintain the convexity of the formulation.

A. Scalar barrier functions

Certifying the safety of system (2) requires showing all trajectories $\{x_k\}_{k=0}^{\infty}$ starting from $\mathcal{X}_0$ never reach the unsafe set $\mathcal{X}_u$. Since the infinite horizon trajectory is considered, explicitly evolving the dynamics for verification is intractable. Instead, we look for a numerical certificate that is sufficient for safety verification.

Theorem 1 (Scalar barrier function): Let the system (1) and sets $\mathcal{X}, \mathcal{X}_u, \mathcal{X}_0$ be given. If there is a continuous scalar function $B(x) : \mathcal{X} \mapsto \mathbb{R}$ satisfying

$$B(x) > 0, \forall x \in \mathcal{X}_u, \tag{3a}$$
$$B(x) \leq 0, \forall x \in \mathcal{X}_0, \tag{3b}$$
$$B(f_\pi(x)) \leq \gamma B(x), \forall x \in \mathcal{X}, \tag{3c}$$

where $\gamma \geq 0$, then the safety of system (2) is guaranteed. Such a $B(x)$ satisfying conditions (3a) to (3c) is called a barrier function.

Proof: For the barrier function $B(x)$, we have $x_0 \in \mathcal{X}_0$ and $B(x_0) \leq 0$ by condition (3b). From condition (3c), we have that for all $k > 0$ and $x_0, x_1, \cdots, x_{k-1} \in \mathcal{X}$, $B(x_k) \leq \gamma^k B(x_0) \leq 0$ which indicates that $x_k \notin \mathcal{X}_u$. ■

a) Convex formulation of scalar barrier function: From the definition of barrier function in Theorem 1, it follows that the set of barrier functions is *convex*, i.e., $\lambda B_1(x) + (1-\lambda)B_2(x)$ with $0 \leq \lambda \leq 1$ is a barrier function if both $B_1(x)$ and $B_2(x)$ satisfy the conditions (3a) to (3c). This convexity property facilitates the design of numerically efficient methods to search $B(x)$. For example, the continuous-time counterparts of constraints (3) enable finding $B(x)$ through semidefinite programming for polynomial dynamical systems [28]–[30]. One important feature of the convex barrier function formulation is that if we parameterize a candidate barrier function as $B(x) = \sum_{i=1}^{M} c_i \phi_i(x)$, where $c = [c_1 \ \cdots \ c_M]^\top \in \mathbb{R}^M$ denotes the coefficient vector and $\phi_i(x) : \mathbb{R}^{n_x} \mapsto \mathbb{R}$ denotes a nonlinear basis function, the set of valid barrier functions in the parameter space of c is also convex. In this work, we will consider parameterizing $B(x)$ as a NN [1], i.e.,

$$B(x) = c^\top \phi_\theta(x) + b, \tag{4}$$

where $\phi_\theta(x) : \mathbb{R}^{n_x} \mapsto \mathbb{R}^M$ is an MLP parameterized by θ and c, b can be interpreted as the weights and bias of the last linear layer.

b) Effects of the parameter λ: Despite the simple form of the barrier function conditions (3), the choice of γ in (3c) has a non-trivial impact on the barrier function synthesis. In parallel to the observations made in [30] on continuous-time systems, we discuss the implications of different values of λ for the discrete-time autonomous dynamics (2) below.

1) Case $\gamma = 0$: Denote $\Omega = \{x | B(x) \leq 0 \cap \mathcal{X}\}$ the sublevel set of a barrier function $B(x)$ and $\Omega^{\mathsf{c}} = \mathcal{X} \setminus \Omega$ its complement. When $\gamma = 0$, (3c) requires that $B(x_1) \leq 0$ for all $x_0 \in \mathcal{X}$, i.e., the system (2) evolves into Ω in one step starting from the state space $\mathcal{X}$, and this condition tends to be overly conservative.
2) Case $0 < \gamma < 1$: With a positive scalar γ and a trajectory $\{x_k\}_{k=0}^{\infty}$ contained in $\mathcal{X}$, condition (3c) indicates that $B(x_k) \leq \gamma^k B(x_0)$ for all $x_0 \in \mathcal{X}$. It follows that $\lim_{k\to\infty} B(x_k) \leq 0$ and the sublevel set Ω of $B(x)$ is attractive for all $x_0 \in \mathcal{X}$.
3) Case $\gamma > 1$: The upper bound $\gamma^k B(x_0)$ diverges to $-\infty$ for $x_0 \in \Omega$, which means all trajectories starting inside Ω have to leave $\mathcal{X}$ in a finite number of steps and Ω cannot contain an equilibrium.

It follows that for the set of barrier functions to be non-empty, the choice of γ should depend on the dynamics of the autonomous system (2). Meanwhile, with the scalar barrier function formulation, the flexibility of tuning the hyperparameter γ is limited. This leads to the conservatism of the scalar barrier function formulation.

In the next subsection, we introduce vector barrier functions for safety verification which allow us to search for safety certificates with less conservatism while maintaining the convexity of the barrier function formulation.

B. Vector barrier functions

A vector barrier function $B(x) : \mathbb{R}^{n_x} \to \mathbb{R}^m$ is defined as $B(x) = [B_1(x) \ B_2(x) \ \cdots \ B_m(x)]^\top$ where $B_1(x), \cdots, B_m(x)$ are scalar functions mapping from $\mathbb{R}^{n_x}$ to $\mathbb{R}$. As an adaption of the vector barrier function for continuous-time systems [9], we define the discrete-time vector barrier function in the following lemma.

Lemma 1: Let the system (1) and sets $\mathcal{X}, \mathcal{X}_u, \mathcal{X}_0$ be given. A vector-valued function $B(x) = [B_1(x) \ B_2(x) \ \cdots \ B_m(x)]^\top : \mathcal{D} \mapsto \mathbb{R}^m$ is called a vector barrier function if it satisfies

$$\vee_{i=1}^{m} B_i(x) > 0, \forall x \in \mathcal{X}_u, \tag{5a}$$
$$\wedge_{i=1}^{m} B_i(x) \leq 0, \forall x \in \mathcal{X}_0, \tag{5b}$$
$$B(x_+) \leq AB(x), \forall x \in \mathcal{X}, \tag{5c}$$

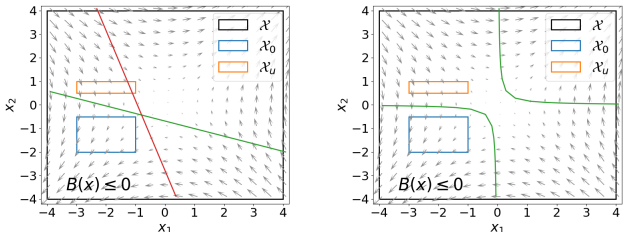
where $A \in \mathbb{R}^{m\times m}$ is a non-negative matrix, i.e., $A \geq 0$, $\vee$ denotes the disjunction operator meaning $\vee_{i=1}^{m} B_i(x) > 0 \Leftrightarrow \exists\ i^* \in \{1, \cdots, m\}$ s.t. $B_{i^*} > 0$, and $\wedge$ denotes the conjunction operator meaning $\wedge_{i=1}^{m} B_i(x) \leq 0 \Leftrightarrow B_i(x) \leq 0, i = 1, \cdots, m$. Then the existence of a vector barrier function $B(x)$ certifies the safety of the system.

Proof: Condition (5b) indicates that $B(x_0) \leq 0$. Since A is a non-negative matrix, we have $B(x_k) \leq A^k B(x_0) \leq 0$ for all $k \geq 0$ as long as $x_k \in \mathcal{X}$. It follows that x_k cannot reach $\mathcal{X}_u$ where at least one entry of $B(x_k)$ is positive. ■

It is obvious that the vector barrier function contains the scalar barrier function defined in (3) as a subclass $m = 1$. However, when $m > 1$, the set of vector barrier functions is not convex due to the disjunction operator in (5a). To recover convexity, we apply the following formulation with stronger conditions:

Theorem 2: The system (1) with the workspace $\mathcal{X}$, the initial set $\mathcal{X}_0$ and the unsafe set $\mathcal{X}_u$ is safe if there exists a

[1]The bias term in (4) can be interpreted as the coefficient of the basis function $\phi_M(x) = 1$.



(a) Vector barrier function. (b) Scalar barrier function.

Fig. 3: The vector barrier function using linear functions (left) and scalar barrier function using a quadratic function (right) for certifying the safety of the system considered in Example 1. The lower-left, encircled regions given by $\{x \mid B(x) \leq 0\}$ separate the trajectories starting from the initial set from the unsafe set.

vector-value function $B(x) : \mathbb{R}^{n_x} \mapsto \mathbb{R}^m$ satisfying

$$B_{i^*}(x) > 0, \forall x \in \mathcal{X}_u, \tag{6a}$$
$$\wedge_{i=1}^m B_i(x) \leq 0, \forall x \in \mathcal{X}_0, \tag{6b}$$
$$B(x_+) \leq AB(x), \forall x \in \mathcal{X}, \tag{6c}$$

where $i^* \in \{1, \cdots, m\}$ is a given number and A is a non-negative matrix.

In the rest of the paper, a vector barrier function is referred to as any $B(x)$ that satisfies conditions (6). It can be easily verified that the set of valid $B(x)$ satisfying (6) is convex. In addition to tuning the function class of each scalar barrier function $B_i(x)$, we can systematically increase the expressivity of the vector barrier function by increasing the dimension m as well. The benefit of the vector barrier function is illustrated in the following example, where the safety of a linear system can be certified by a vector $B(x)$ using only linear functions while a scalar barrier function is required to be at least quadratic.

Example 1: Consider a linear system $x_{k+1} = \left[\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right] x_k$ with the workspace $\mathcal{X}$, initial set $\mathcal{X}_0$, and unsafe set $\mathcal{X}_u$ shown in Fig. 3. There is no linear scalar barrier function satisfying (3) that can certify the safety of the system, while a vector barrier function with dimension 2 using two linear functions $B_1(x), B_2(x)$ and $A = \left[\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right]$ succeeds (Fig. 3a). To find a scalar barrier function, a quadratic function parameterization is needed as shown in Fig. 3b.

C. Training of NN vector barrier functions

We now parameterize a NN vector barrier function as

$$B(x) = C\phi_\theta(x) + b, \tag{7}$$

where $\phi_\theta(x) : \mathbb{R}^{n_x} \mapsto \mathbb{R}^m$ is a multi-layer perceptron (MLP) whose weights and biases are denoted by the parameter θ, and $C \in \mathbb{R}^{m \times M}, b \in \mathbb{R}^m$ are the weights and bias of the last linear layer, respectively. Here we highlight the parameters of the last linear layer which we will fine-tune in the next section. As highlighted in the discussion of the scalar barrier function, the hyperparameter A in (6c) has a complex dependence on the underlying dynamics (2) and a feasible value is generally difficult to find. We propose to train the NN barrier function (7) jointly with the hyperparameter A using the samples collected from the workspace $\mathcal{X}$. Without loss of generality, we fix $i^* = 1$ in constraint (6a).

We consider the following empirical barrier function risk

$$\begin{aligned} L(\theta, C, b, A) = &\frac{1}{N_0}\sum_{i=1}^{N_0} \mathbf{1}^\top \text{ReLU}(B(x_0^i)) + \\ &\frac{1}{N_u}\sum_{i=1}^{N_u} e_1^\top \text{ReLU}(-B(x_u^i)) + \\ &\frac{1}{N}\sum_{i=1}^{N} \mathbf{1}^\top \text{ReLU}(B(f_\pi(x^i)) - AB(x^i)), \end{aligned} \tag{8}$$

where $\mathbf{1}$ denotes the vector of all ones, $e_1 = [1\ 0\ \cdots\ 0]^\top$ denotes the first standard basis, $\{x_0^i\}_{i=1}^{N_0}$, $\{x_u^i\}_{i=1}^{N_u}$, $\{x^i\}_{i=1}^{N}$ are the N_0 samples from the initial set $\mathcal{X}_0$, N_u samples from the unsafe set $\mathcal{X}_u$, and N samples from the workspace $\mathcal{X}$, respectively. Then, projected gradient descent (PGD) is applied to minimize $L(\theta, C, b, A)$ subject to $A \geq 0$ to generate a NN vector barrier function candidate $B(x)$. In the next section, we will address the problem of verifying the validity of the trained NN $B(x)$ and updating $B(x)$ in case of verification failure.

IV. Fine-Tuning through Convex Optimization

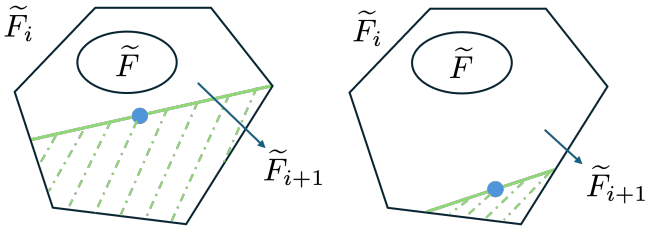
With the trained NN barrier function candidate $B(x)$ and the parameter A, we can verify if $B(x)$ satisfies all the constraints in (6) using NN verification tools. However, in case of verification failure, retraining of $B(x)$ is required and this 'train-then-verify' process may never terminate with a valid barrier function. To address this issue, in this section, we propose a CEGIS method to only update the last linear layer (C, b) for the following reasons:

1) Empirically, it is reasonable to hypothesize that, after sufficient training, the NN vector barrier function candidate $B(x)$ is close to being valid and only minor adjustment is needed.
2) Theoretically, if there exists a feasible pair (C, b) such that $B(x)$ is valid, our CEGIS algorithm is guaranteed to find a feasible pair in a finite number of iterations.

Our algorithm of fine-tuning the last linear layer consists of a learner, which proposes a candidate (C, b), and a verifier, which verifies the validity of the candidate (C, b) or falsifies it with a counterexample state. Similar to [8], by designing the interaction between the learner and verifier according to the cutting-plane method from convex optimization, we achieve the finite-step termination guarantee. We elaborate on the algorithm design in the following subsections.

A. Feasible set of the last layer

After training the NN vector barrier function, we fix the parameter A and the subnetwork $\phi_\theta(x)$ as the basis function. For notational simplicity, we denote $W = \text{vec}(C, b)$ as the vectorized parameter of the last linear layer of $B(x)$. Our



(a) Candidate at the center. (b) Randomly chosen candidate.

Fig. 4: As shown on the left figure, proposing a candidate solution (blue dot) at the center of the localization set is guaranteed to remove a large portion of the search space (shaded area) after verification, while selecting the candidate in an uncontrolled manner (see the right figure) may lead to minimal search space improvement.

goal is to find a W such that $B(x)$ satisfies all the constraints in (6). We denote the set of feasible parameters of W as

$$\mathcal{F} = \{W \in \mathbb{R}^{m\times(M+1)} \mid B(x) \text{ satisfies (6).}\} \tag{9}$$

and call $\mathcal{F}$ the *target set*. We observe that when the state is fixed, constraints (6) are all linear constraints in W. Therefore, the target set $\mathcal{F}$ is described by an (uncountably) infinite number of linear constraints and is convex. Our goal is to either find a feasible point in $\mathcal{F}$, which gives a valid barrier function and proves the safety of the system (2), or certify that $\mathcal{F}$ is empty, which means retraining of the basis function $\phi_\theta(x)$ or the parameter A is needed.

Although being convex, the target set $\mathcal{F}$ is defined by infinitely many linear constraints which pose numerical challenges. This motivates a sampling-based method to over-approximate $\mathcal{F}$ by considering the barrier function conditions (6) only on a set of finite samples $\mathcal{S} = \{\mathcal{S}_0, \mathcal{S}_u, \mathcal{S}_\mathcal{X}\}$ where $\mathcal{S}_0, \mathcal{S}_u, \mathcal{S}_\mathcal{X}$ are collections of a finite number of states in $\mathcal{X}_0, \mathcal{X}_u, \mathcal{X}$, respectively. With the sample set $\mathcal{S}$, an over-approximation of the target set is given by

$$\begin{aligned}\widetilde{\mathcal{F}} = \{W | &B_{i^*}(x) > 0 \ \forall x \in \mathcal{S}_u, \wedge_{i=1}^m B_i(x) \le 0 \ \forall x \in \mathcal{S}_0,\\ &B(f_\pi(x)) \le AB(x) \ \forall x \in \mathcal{S}_\mathcal{X}\}.\end{aligned} \tag{10}$$

Obviously, $\widetilde{\mathcal{F}}$ is an over-approximation of $\mathcal{F}$, and finding a feasible point in $\widetilde{\mathcal{F}}$ reduces to solving a tractable convex feasibility problem. For a given set of samples $\mathcal{S}$, if $\widetilde{\mathcal{F}}$ is shown to be empty, we certify that the target set is empty as well, i.e., $\mathcal{F} = \emptyset$. However, if $\widetilde{\mathcal{F}}$ is non-empty, we have no guarantee that any feasible point $W \in \widetilde{\mathcal{F}}$ will generate a valid barrier function even as we increase the size of the sample set $\mathcal{S}$ asymptotically to infinity.

To fix this issue, we propose a sampling strategy that consists of a learner and a verifier based on cutting planes with which we can expand the sample set $\mathcal{S}$ iteratively.

B. Sampling strategy based on cutting-plane methods

Cutting-plane methods are iterative algorithms that find a feasible point in a convex target set $\mathcal{F}$ or certify that $\mathcal{F}$ is empty. At each iteration, we maintain an over-approximation $\widetilde{\mathcal{F}} \supseteq \mathcal{F}$ called a localization set. If $\widetilde{\mathcal{F}} = \emptyset$, then we conclude $\mathcal{F} = \emptyset$ and terminate the iteration. Otherwise, we query the 'cutting-plane oracle' at a point $W \in \widetilde{\mathcal{F}}$, for which the oracle either certifies that $W \in \mathcal{F}$ or returns a separating hyperplane that separates W and the target set $\mathcal{F}$. In the former case, we terminate the iteration with a feasible $W \in \mathcal{F}$; in the latter case, we update the localization set by adding the separating hyperplane constraint and repeat this process. The intuition behind choosing the candidate solution as the analytic center is illustrated in Fig. 4.

We design our sampling strategy according to the analytic center cutting-plane method (ACCPM). In the ACCPM, the query point W is chosen as the analytic center [31] of the localization set $\widetilde{\mathcal{F}}$. Our proposed method consists of a learner, which proposes barrier function candidates based on a set of samples, and a verifier, which serves as a cutting-plane oracle and updates the sample set with counterexamples.

1) The learner: Note that for the candidate barrier function $B(x)$, scaling the parameter W does not affect the satisfaction of conditions (6). Without loss of generality, we search for a feasible parameter W in the bounded set $-R\mathbf{1} \le W \le R\mathbf{1}$ with a constant $R > 0$. Then, for a sample set $\mathcal{S} = \{\mathcal{S}_0, \mathcal{S}_u, \mathcal{S}_\mathcal{X}\}$ and the corresponding localization set $\widetilde{\mathcal{F}}$, the learner solves the following convex program of finding the analytic center [31]:

$$\begin{aligned}\underset{W}{\text{minimize}} \ &-\sum_{x\in\mathcal{S}_u} \log(B_{i^*}(x)) - \sum_{i=1}^m \sum_{x\in\mathcal{S}_0} \log(-B_i(x))\\ &-\sum_{i=1}^m \sum_{x\in\mathcal{S}_\mathcal{X}} \log(a_i^\top B(x) - B_i(f_\pi(x)))\\ &+\sum_{i=1}^{(M+1)m} (-\log(W_i + R) - \log(R - W_i))\end{aligned} \tag{11}$$

where $a_i^\top$ denotes the i-th row of A and W_i denotes the i-th entry of the vectorized parameter. If the localization set $\widetilde{\mathcal{F}}$ is empty, then we certify that $\mathcal{F}$ is empty. If not, the learner proposes the barrier function candidate $B(x; W^*)$ to the verifier where W^* is the optimal solution to Problem (11).

2) The verifier: Given a barrier function candidate $B(x; W^*)$ proposed by the learner, the verifier checks if $B(x; W^*)$ satisfies the conditions (6) by solving a set of $2m + 1$ global optimization problems:

$$\min_{x\in\mathcal{X}_u} \ B_{i^*}(x; W^*), \tag{12a}$$

$$\min_{x\in\mathcal{X}_0} \ -B_i(x; W^*), 1 \le i \le m, \tag{12b}$$

$$\min_{x\in\mathcal{X}} \ a_i^\top B(x; W^*) - B_i(f_\pi(x); W^*), 1 \le i \le m. \tag{12c}$$

If the optimal values of problems in (12) are all non-negative, $B(x; W^*)$ is a valid barrier function; otherwise, we extract the optimal solutions $\{x_{u,i^*}^{ce}\}$ of Problem (12a), $\{x_{0,i}^{ce}\}_{i=1}^m$ of Problem (12b), $\{x_i^{ce}\}_{i=1}^m$ of Problem (12c) with negative optimal values and denote them as counterexam-

Algorithm 1 ACCPM

Input: Basis function $\phi_\theta(x)$, parameter $A \geq 0$, initial sample set $\mathcal{S}^{(0)} = \{\mathcal{S}_0^{(0)}, \mathcal{S}_u^{(0)}, \mathcal{S}_\mathcal{X}^{(0)}\}$.
Output: Parameters $W = \text{vec}(C, b)$ of the NN last linear layer.

1: $i = 1$.
2: **while** True **do**
3: $\quad \mathcal{S}^{(i)} \Rightarrow \widetilde{F}_i$ {See (10)}.
4: $\quad$ **if** $\widetilde{F}_i = \emptyset$ **then**
5: $\quad\quad$ Return $W = \emptyset$.
6: $\quad$ **end if**
7: $\quad$ Solve (11) {Call the learner}.
8: $\quad$ Solve (12) {Call the verifier}.
9: $\quad$ **if** $B(x; W_i)$ is a valid barrier function **then**
10: $\quad\quad$ Return $W = W_i$.
11: $\quad$ **else**
12: $\quad\quad$ Update the sample set {See (13)}.
13: $\quad$ **end if**
14: $\quad i = i + 1$.
15: **end while**

ples [2]. Then, we expand the sample set $\mathcal{S}$ by

$$\begin{aligned} &\mathcal{S}_0 \leftarrow \mathcal{S}_0 \cup \{x_{0,i}^{ce}\}_{i=1}^m, \quad \mathcal{S}_u \leftarrow \mathcal{S}_u \cup \{x_{u,i^*}^{ce}\}, \\ &\mathcal{S}_\mathcal{X} \leftarrow \mathcal{S}_\mathcal{X} \cup \{x_i^{ce}\}_{i=1}^m. \end{aligned} \tag{13}$$

The alternation between the learner and the verifier is summarized in Algorithm 1. Since our CEGIS follows the design of ACCPM, the convergence of Algorithm 1 can be established below.

Theorem 3: For the basis function $\phi_\theta(x)$, if the target set $\mathcal{F}$ is non-empty and contains a full dimensional ball of radius $\epsilon > 0$ in the parameter space of the last linear layer, Algorithm 1 is guaranteed to find one feasible point in $\mathcal{F}$ in at most $O((m(M+1))^2/\epsilon^2)$ iterations.

Proof: Note that the dimension of the parameter space of the last linear layer is $m(M+1)$. The construction of the learner and the verifier follows the analytic center cutting-plane method and the finite-step termination guarantee of Algorithm 1 directly follows from [32]. ■

V. Verification-Aided Learning

Taking the barrier function $B(x)$ as an example, we summarize the verification-aided learning framework in Algorithm 2 that underpins the existing works [3], [4], [6], [13]. Particularly, in our work, we apply the fine-tuning method in Line 7, and a warm-starting method from [33] in Line 15 to improve the success rate of learning a barrier function. In this section, we explain the implementation details of Algorithm 2.

[2]For notational simplicity, we let x_{u,i^*}^{ce}, $x_{0,i}^{ce}$, or x_i^{ce} be empty if the optimal value of its corresponding problem in (12) is not negative.

[3]This step is recommended in [33] since for toolboxes such as PyTorch different network parameters are initialized randomly using different mechanisms.

Algorithm 2 Verification-aided learning

Input: NN barrier function $B(x)$, initial sample set $\mathcal{S}$, maximum number of iteration N, shrinking weight $\lambda \in [0, 1]$, noise scale $\sigma \in [0, 1]$.
Output: Trained NN barrier function $B(x)$, safety certificate.

1: **for** $i = 1, \cdots, N$ **do**
2: $\quad$ Train $B(x)$ on $\mathcal{S}$ using gradient descent {See (8)}.
3: $\quad$ Verify $B(x)$. {See (12)}
4: $\quad$ **if** Verification succeeds **then**
5: $\quad\quad$ Return $B(x)$ and a safety certificate.
6: $\quad$ **else**
7: $\quad\quad$ Fine-tune the last linear layer by Algorithm 1.
8: $\quad\quad$ **if** Algorithm 1 is feasible **then**
9: $\quad\quad\quad$ Return tuned $B(x)$ and a safety certificate.
10: $\quad\quad$ **end if**
11: $\quad\quad$ Extract counterexamples $\mathcal{S}_{ce}$ for $B(x)$.
12: $\quad$ **end if**
13: $\quad$ Augment the counterexample set $\mathcal{S}_{ce}$.
14: $\quad$ Update the sample set $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{S}_{ce}$.
15: $\quad B(x) =$ Shrink-and-Perturb($B(x)$, λ, σ) {See Alg. 3}.
16: **end for**
17: Return $B(x)$ without any certificate.

Algorithm 3 Shrink-and-Perturb [33]

Input: NN barrier function $B(x)$, shrinking weight $\lambda \in [0, 1]$, noise scale $\sigma \in [0, 1]$.
Output: Adjusted $B(x)$.

Randomly initialize a NN $\widetilde{B}(x)$ that has the same architecture as $B(x)$ [3].
for each learnable parameter W in the NN $B(x)$ **do**
$\quad$ Select the corresponding parameter $\widetilde{W}$ from $\widetilde{B}$.
$\quad W \leftarrow \lambda W + \sigma \widetilde{W}$.
end for

A. Train and verify

In Line 2 of Algorithm 2, the NN barrier function $B(x)$ is first trained on a sample set by minimizing the empirical risk (8) using gradient descent. Then, the trained $B(x)$ is passed to a complete verifier in Line 3 which solves a mixed-integer linear program (MILP) for piecewise linear networks [8] or uses satisfiability modulo theory (SMT) solvers such as dReal [34] for NNs with tanh or sigmoid activation [25]. The algorithm terminates immediately when the verification is a success. Otherwise, in Line 7 we fine-tune the last linear layer of the trained $B(x)$ using Algorithm 1 to further search for a feasible barrier function.

Since calling the verifier to generate counterexamples can be costly, we first apply PGD to search for counterexamples to falsify the barrier function candidate before doing formal verification in both Line 3 of Algorithm 2 and Line 8 of Algorithm 1. If the PGD attack finds valid counterexamples, we skip calling the computationally costly verifier and proceed with these counterexamples.

B. Augment the counterexamples

As discussed in [3, Section 4.4], the counterexamples generated by the verifiers are often scarce. Since adding additional samples does not harm the soundness of CEGIS and is in general beneficial for training the NN barrier function, augmenting the counterexamples through sampling has been proposed [3]–[5]. In particular, we sample a batch of states around each counterexample in the set $\mathcal{S}_{ce}$ (Line 13) and use them as the initial points for the PGD attack. The updated states after performing PGD are added to $\mathcal{S}_{ce}$.

C. Warm-starting NN training

During the training of the barrier function, we want $B(x)$ to not only achieve zero training error shown in (8), but also zero test error which means the satisfaction of constraints (6). While it is natural to warm-start the barrier function using its parameters learned from the previous iteration in Algorithm 2, Ash et al. [33] found that in practice this mechanism often leads to a poorer generalization of $B(x)$ than using fresh random initialization. This implies that warm-starting $B(x)$ from its previous solution can lead to frequent verification failure due to poor generalization.

Motivated by the fact that the training data gets expanded incrementally in verification-aided learning, we propose to use the shrink-and-perturb trick proposed in [33] to initialize $B(x)$ in each iteration as shown in Algorithm 3. Shrink-and-perturb is a simple trick to achieve a good balance between generalization improvement and training cost when the training data arrives in a stream and is therefore well-suited for continual learning or active learning tasks. In shrink-and-perturb, the network parameters from the previous training iteration are shrunk toward zero and perturbed by scaled noises from a randomly initialized network. We refer the readers to [33] for a detailed discussion of this trick.

D. Neural network verifiers

In our experiments, we primarily focus on the training of ReLU networks which is in fact a piecewise affine function. To make a comprehensive evaluation of the role of fine-tuning in verification-aided learning, in our experiments, we apply two complete verification methods for ReLU networks. The first method is based on mixed-integer programming [11] which solves the verification problem by encoding the ReLU network using mixed-integer linear constraints. The second method is based on the GPU-accelerated, specialized NN verifier α, β-CROWN [12], which combines efficient linear bounds propagation [22], [35] and the Branch-and-Bound algorithm for complete verification. α, β-CROWN is the state-of-the-art NN verifier and has achieved outstanding performance in verifying the robustness of deep NN image classifiers. In this work, we apply it to a control task and deploy it as a complete verifier in Algorithm 2.

VI. Experiments

In this section, we numerically compare the performance of the verification-aided learning framework (Algorithm 2) with and without the fine-tuning step on two examples drawn from [36] [4]: a double integrator and a 6D quadrotor, both controlled by ReLU networks. We denote Algorithm 2 with our proposed fine-tuning step as `Fine-tuning` and that without as `Verification-only`. For a fixed setup of the domain $\mathcal{X}$, initial set $\mathcal{X}_0$, and unsafe set $\mathcal{X}_u$, we run Algorithm 2 multiple times with different random seeds and evaluate the performance of our proposed fine-tuning method by comparing (i) the success rates of finding a valid barrier function and (ii) the runtime of `Fine-tuning` and `Verification-only`. In what follows, our experiments show that `Fine-tuning` can consistently and largely improve both the success rate and runtime compared with `Verification-only` over multiple runs and for different NN verifiers (see Table I).

Example	NN Verifier	`Fine-tuning`	`Verification-only`
Double integrator	MIP	**100**%	75%
	α, β-CROWN	**87.5**%	0%
6D quadrotor	α, β-CROWN	**100**%	75%

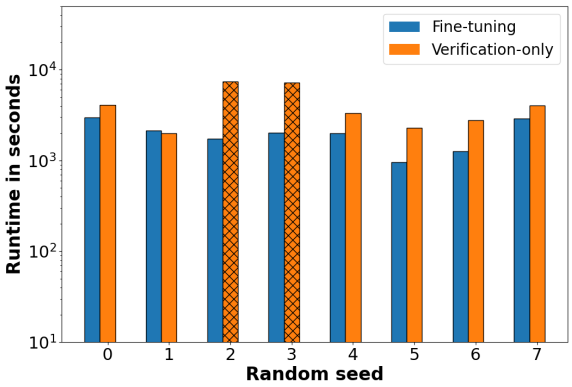
TABLE I: Success rate comparison of `Fine-tuning` and `Verification-only` in learning a valid NN barrier function. For each problem setup, we ran Algorithm 2 using different random seeds for 8 times. The timeout threshold is 2 hours.

Our codes are publicly available at https://github.com/ShaoruChen/Neural-Barrier-Function. The experiments using a MIP-based verifier were run on the Apple M1 chip with Gurobi v10.0.1 [37] as the solver, and the experiments using α, β-CROWN were run on a Nvidia Tesla T4 GPU. CVXPY [38] and the ECOS [39] solver were applied to find the analytic center in (11). In each example, we applied the same set of algorithmic parameters such as the training epochs, the number of augmented counterexample samples, the scaling factors in shrink-and-perturb, etc., to `Fine-tuning` and `Verification-only`. The details of these algorithmic parameters can be found in the configuration files in our code repository.

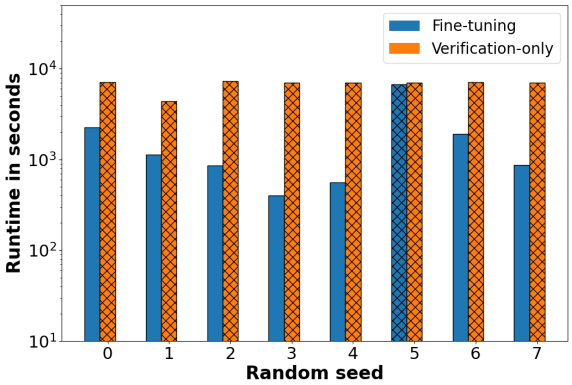
A. Double integrator

A discrete-time double integrator system $x_{k+1} = Ax_k + Bu_k$ is considered where $A = \left[\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right]$ and $B = \left[\begin{smallmatrix} 0.5 \\ 1 \end{smallmatrix}\right]$. The controller $\pi(x)$ is a ReLU network that has been trained to approximate an MPC policy and has an architecture $2-10-5-1$. The workspace, initial set, and unsafe set are given by $\mathcal{X} = \{x \in \mathbb{R}^2 \mid [-3 \ \ -3]^\top \leq x \leq [3 \ 3]^\top\}$, $\mathcal{X}_0 = \{x \in \mathbb{R}^2 \mid [2.5 \ \ -0.5]^\top \leq x \leq [2.8 \ 0]^\top\}$, $\mathcal{X}_u = \{x \in \mathbb{R}^2 \mid [1.5 \ \ -0.25]^\top \leq x \leq [1.8 \ 0]^\top\}$, respectively. The NN barrier function is parameterized as a ReLU network of architecture $2-30-20-10-5$. Next, we run Algorithm 2 with a timeout limit of 2 hours and report the comparison results in Fig. 5a and 5b using the MIP and α, β-CROWN-based NN verifiers, respectively. The shaded bars denote experiments that ran out of time

[4] In our experiments, we used the trained ReLU NN controller from https://github.com/o4lc/ReachLipBnB for safety verification.



(a) Runtime comparison using MIP-based verifier.



(b) Runtime comparison using α, β-CROWN.

Fig. 5: Runtime comparison of Algorithm 2 with (`Fine-tuning`) and without (`Verification-only`) the fine-tuning step for the double integrator example. The MIP-based (top) and α, β-CROWN-based (bottom) NN verifiers are applied. The shaded bars indicate failures to find a valid barrier function within the 2 hrs runtime limit.

without successfully finding a barrier function. The success rates of `Fine-tuning` and `Verification-only` are summarized in Table I. We observe that `Fine-tuning` achieves better success rates than `Verification-only` for both NN verifiers.

B. 6D quadrotor

Following from [36], the discretized 6D quadrotor dynamics with sampling time $\Delta t = 0.1$s is given by $x_{k+1} = Ax_k + Bu_k + c$ with $A = I_{6\times 6} + \Delta t \begin{bmatrix} 0_{3\times 3} & I_{3\times 3} \\ 0_{3\times 3} & 0_{3\times 3} \end{bmatrix}$, $B = \Delta t \begin{bmatrix} 0 & 0 & 0 & g & 0 & 0 \\ 0 & 0 & 0 & 0 & -g & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}^\top$, and $c = \Delta t \begin{bmatrix} 0_{5\times 1} \\ -g \end{bmatrix}$. Here g denotes the value of the gravitational acceleration, and $I_{n\times m}$, $0_{n\times m}$ denote the identity and zero matrices of dimension $n \times m$, respectively. As discussed in [40], the control input u_k is a nonlinear function of the pitch, roll, and thrust of the quadrotor. A ReLU NN controller of architecture $6 - 32 - 32 - 32 - 3$ is trained to approximate the MPC policy. In this example, the workspace, initial set, and unsafe

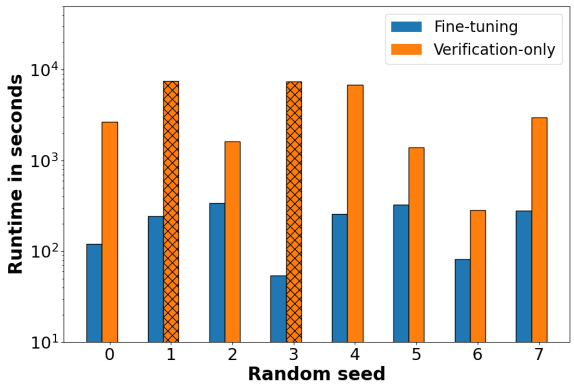


Fig. 6: Runtime comparison of Algorithm 2 with (`Fine-tuning`) and without (`Verification-only`) the fine-tuning step for the 6D quadrotor example. The α, β-CROWN-based (bottom) NN verifier is applied. The shaded bars indicate failures to find a valid barrier function within the 2 hrs runtime limit.

set are given by $\mathcal{X} = \{x \in \mathbb{R}^6 \mid [4.0\ 4.0\ 2.5\ -1.0\ -1.0\ -1.0]^\top \leq x \leq [5.0\ 5.0\ 3.5\ 1.0\ 1.0\ 1.0]^\top\}$, $\mathcal{X}_0 = \{x \in \mathbb{R}^6 \mid [4.69\ 4.65\ 2.975\ 0.9499\ -0.0001\ -0.0001]^\top \leq x \leq [4.71\ 4.75\ 3.025\ 0.9501\ 0.0001\ 0.0001]^\top\}$, $\mathcal{X}_u = \{x \in \mathbb{R}^6 \mid [4.4\ 4.3\ 2.9\ 0.95\ -0.1\ -0.1]^\top \leq x \leq [4.45\ 4.35\ 3.0\ 1.0\ 0.1\ 0.1]^\top\}$, respectively. The NN barrier function $B(x)$ is parameterized as a ReLU network of architecture $6 - 100 - 80 - 60 - 40 - 20 - 10$. We compare the success rate and runtime of `Fine-tuning` and `Verification-only` in Fig. 6 using α, β-CROWN as the verifier [5] and observe that `Fine-tuning` consistently outperforms `Verification-only` in both success rate and runtime.

VII. Conclusion

We proposed a NN vector barrier function to certify the safety of NN-controlled systems. To achieve a formal safety guarantee, a verification-aided learning framework is required which iteratively learns the NN barrier function using counterexamples generated by the NN verifier. Motivated by the fact that such "learn-then-verify" iteration may take long to find a valid NN barrier function, we proposed a fine-tuning method for verification-aided learning which enjoys finite-step termination guarantees and demonstrated that our method can consistently and significantly improve the success rate of synthesizing valid NN barrier functions over numerous experiments.

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.

[5]For the considered NN size, verifying even a single barrier function candidate using MIP easily took more than 2 hours and exceeded the timeout threshold in our experiments.

[2] A. Abate, D. Ahmed, M. Giacobbe, and A. Peruffo, "Formal synthesis of lyapunov neural networks," *IEEE Control Systems Letters*, vol. 5, no. 3, pp. 773–778, 2020.

[3] A. Abate, D. Ahmed, A. Edwards, M. Giacobbe, and A. Peruffo, "Fossil: a software tool for the formal synthesis of lyapunov functions and barrier certificates using neural networks," in *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control*, 2021, pp. 1–11.

[4] H. Dai, B. Landry, L. Yang, M. Pavone, and R. Tedrake, "Lyapunov-stable neural-network control," in *Robotics: Science and Systems XVII, Virtual Event, July 12-16, 2021*, D. A. Shell, M. Toussaint, and M. A. Hsieh, Eds., 2021. [Online]. Available: https://doi.org/10.15607/RSS.2021.XVII.063

[5] Q. Zhao, X. Chen, Z. Zhao, Y. Zhang, E. Tang, and X. Li, "Verifying neural network controlled systems using neural networks," in *Proceedings of the 25th ACM International Conference on Hybrid Systems: Computation and Control*, 2022, pp. 1–11.

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

[7] A. Solar-Lezama, L. Tancau, R. Bodik, S. Seshia, and V. Saraswat, "Combinatorial sketching for finite programs," in *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, 2006, pp. 404–415.

[8] S. Chen, M. Fazlyab, M. Morari, G. J. Pappas, and V. M. Preciado, "Learning lyapunov functions for hybrid systems," in *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control*, 2021, pp. 1–11.

[9] A. Sogokon, K. Ghorbal, Y. K. Tan, and A. Platzer, "Vector barrier certificates and comparison systems," in *International Symposium on Formal Methods*. Springer, 2018, pp. 418–437.

[10] D. S. Atkinson and P. M. Vaidya, "A cutting plane algorithm for convex programming that uses analytic centers," *Mathematical programming*, vol. 69, no. 1-3, pp. 1–43, 1995.

[11] V. Tjeng, K. Y. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," in *International Conference on Learning Representations*, 2018.

[12] S. Wang, H. Zhang, K. Xu, X. Lin, S. Jana, C.-J. Hsieh, and J. Z. Kolter, "Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification," *Advances in Neural Information Processing Systems*, vol. 34, pp. 29 909–29 921, 2021.

[13] Y.-C. Chang, N. Roohi, and S. Gao, "Neural lyapunov control," *Advances in neural information processing systems*, vol. 32, 2019.

[14] W. Jin, Z. Wang, Z. Yang, and S. Mou, "Neural certificates for safe control policies," *CoRR*, vol. abs/2006.08465, 2020. [Online]. Available: https://arxiv.org/abs/2006.08465

[15] A. Robey, H. Hu, L. Lindemann, H. Zhang, D. V. Dimarogonas, S. Tu, and N. Matni, "Learning control barrier functions from expert demonstrations," *arXiv preprint arXiv:2004.03315*, 2020.

[16] Z. Qin, K. Zhang, Y. Chen, J. Chen, and C. Fan, "Learning safe multi-agent control with decentralized neural barrier certificates," in *International Conference on Learning Representations*, 2020.

[17] L. Lindemann, H. Hu, A. Robey, H. Zhang, D. Dimarogonas, S. Tu, and N. Matni, "Learning hybrid control barrier functions from data," in *Conference on Robot Learning*. PMLR, 2021, pp. 1351–1370.

[18] N. Gaby, F. Zhang, and X. Ye, "Lyapunov-net: A deep neural network architecture for lyapunov function approximation," in *2022 IEEE 61st Conference on Decision and Control (CDC)*. IEEE, 2022, pp. 2091–2096.

[19] S. Zhang, Y. Xiu, G. Qu, and C. Fan, "Compositional neural certificates for networked dynamical systems," in *Learning for Dynamics and Control Conference*. PMLR, 2023, pp. 272–285.

[20] W. Xiao, T.-H. Wang, R. Hasani, M. Chahine, A. Amini, X. Li, and D. Rus, "Barriernet: Differentiable control barrier functions for learning of safe robot control," *IEEE Transactions on Robotics*, 2023.

[21] C. Liu, T. Arnon, C. Lazarus, C. Strong, C. Barrett, M. J. Kochenderfer *et al.*, "Algorithms for verifying deep neural networks," *Foundations and Trends® in Optimization*, vol. 4, no. 3-4, pp. 244–404, 2021.

[22] K. Xu, Z. Shi, H. Zhang, Y. Wang, K.-W. Chang, M. Huang, B. Kailkhura, X. Lin, and C.-J. Hsieh, "Automatic perturbation analysis for scalable certified robustness and beyond," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1129–1141, 2020.

[23] H. Zhang, S. Wang, K. Xu, L. Li, B. Li, S. Jana, C.-J. Hsieh, and J. Z. Kolter, "General cutting planes for bound-propagation-based neural network verification," *Advances in Neural Information Processing Systems*, vol. 35, pp. 1656–1670, 2022.

[24] H. Dai, B. Landry, M. Pavone, and R. Tedrake, "Counter-example guided synthesis of neural network lyapunov functions for piecewise linear systems," in *2020 59th IEEE Conference on Decision and Control (CDC)*. IEEE, 2020, pp. 1274–1281.

[25] A. Peruffo, D. Ahmed, and A. Abate, "Automated and formal synthesis of neural barrier certificates for dynamical models," in *International conference on tools and algorithms for the construction and analysis of systems*. Springer, 2021, pp. 370–388.

[26] H. Ravanbakhsh and S. Sankaranarayanan, "Learning control lyapunov functions from counterexamples and demonstrations," *Autonomous Robots*, vol. 43, pp. 275–307, 2019.

[27] S. Chen, M. Fazlyab, M. Morari, G. J. Pappas, and V. M. Preciado, "Learning region of attraction for nonlinear systems," in *2021 60th IEEE Conference on Decision and Control (CDC)*. IEEE, 2021, pp. 6477–6484.

[28] S. Prajna and A. Jadbabaie, "Safety verification of hybrid systems using barrier certificates," in *International Workshop on Hybrid Systems: Computation and Control*. Springer, 2004, pp. 477–492.

[29] S. Prajna, A. Jadbabaie, and G. J. Pappas, "A framework for worst-case and stochastic safety verification using barrier certificates," *IEEE Transactions on Automatic Control*, vol. 52, no. 8, pp. 1415–1428, 2007.

[30] H. Kong, F. He, X. Song, W. N. Hung, and M. Gu, "Exponential-condition-based barrier certificate generation for safety verification of hybrid systems," in *International Conference on Computer Aided Verification*. Springer, 2013, pp. 242–257.

[31] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[32] Y. Ye, "Complexity analysis of the analytic center cutting plane method that uses multiple cuts," *Mathematical Programming*, vol. 78, no. 1, pp. 85–104, 1996.

[33] J. Ash and R. P. Adams, "On warm-starting neural network training," *Advances in neural information processing systems*, vol. 33, pp. 3884–3894, 2020.

[34] S. Gao, S. Kong, and E. M. Clarke, "dreal: An smt solver for nonlinear theories over the reals," in *Automated Deduction–CADE-24: 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings 24*. Springer, 2013, pp. 208–214.

[35] 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.

[36] T. Entesari, S. Sharifi, and M. Fazlyab, "Reachlipbnb: A branch-and-bound method for reachability analysis of neural autonomous systems using lipschitz bounds," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 1003–1010.

[37] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2023. [Online]. Available: https://www.gurobi.com

[38] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.

[39] A. Domahidi, E. Chu, and S. Boyd, "Ecos: An socp solver for embedded systems," in *2013 European control conference (ECC)*. IEEE, 2013, pp. 3071–3076.

[40] H. Hu, M. Fazlyab, M. Morari, and G. J. Pappas, "Reach-sdp: Reachability analysis of closed-loop systems with neural network controllers via semidefinite programming," in *2020 59th IEEE Conference on Decision and Control (CDC)*. IEEE, 2020, pp. 5929–5934.