White Rose Research Online

eprints@whiterose.ac.uk
https://eprints.whiterose.ac.uk
Universities of Leeds, Sheffield and York

Deposited via The University of York.

White Rose Research Online URL for this paper:
https://eprints.whiterose.ac.uk/id/eprint/200199/

Preprint:
Zhao, Hengjun, Zeng, Xia, Chen, Taolue et al. (2020) Learning Safe Neural Network Controllers with Barrier Certificates. [Preprint]

Reuse
Items deposited in White Rose Research Online are protected by copyright, with all rights reserved unless indicated otherwise. They may be downloaded and/or printed for private study, or other acts as permitted by national copyright laws. The publisher or other rights holders may allow further reproduction and re-use of the full text version. This is indicated by the licence information on the White Rose Research Online record for the item.

Takedown
If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

UNIVERSITY OF LEEDS
University of Sheffield
UNIVERSITY of York

arXiv:2009.09826v1 [eess.SY] 18 Sep 2020

Learning Safe Neural Network Controllers with Barrier Certificates

Hengjun Zhao[1], Xia Zeng[1], Taolue Chen[2], Zhiming Liu[1], and Jim Woodcock[1,3]

[1] School of Computer and Information Science, Southwest University
{zhaohj2016, xzeng0712, zhimingliu88}@swu.edu.cn
[2] Department of Computer Science, University of Surrey
taolue.chen@surrey.ac.uk
[3] Department of Computer Science, University of York
jim.woodcock@york.ac.uk

Abstract. We provide a novel approach to synthesize controllers for nonlinear continuous dynamical systems with control against safety properties. The controllers are based on neural networks (NNs). To certify the safety property we utilize barrier functions, which are represented by NNs as well. We train the controller-NN and barrier-NN simultaneously, achieving a verification-in-the-loop synthesis. We provide a prototype tool nncontroller with a number of case studies. The experiment results confirm the feasibility and efficacy of our approach.

Keywords: Continuous dynamical systems; Controller synthesis; Neural networks; Safety verification; Barrier certificates

1 Introduction

Controller design and synthesis is one of the most fundamental problems in control theory. In recent years, especially with the boom of deep learning, there has been considerable research activities in the use of neural networks (NNs) for control of nonlinear systems [24,10]. NNs feature the versatile representational ability of nonlinear maps and fast computation, making them an ideal candidate for sophisticated control tasks [27]. Typical examples include self-driving cars, drones, and smart cities. It is noteworthy that many of these applications are safety-critical systems, where safety refers to, in a basic form, that the system cannot reach a dangerous or unwanted state. For control systems in a multitude of Cyber-Physical-System domains, designing *safe* controllers which can guarantee safety behaviors of the controlled systems is of paramount importance [32,3,33,12,38,5,6,43,17,39].

Typically, when a controller is given, formal verification is required to certify its safety. Our previous work [44] has dealt with the *verification* of continuous dynamical systems by the aid of neural networks. In a nutshell, we follow a deductive verification methodology therein by synthesizing a barrier function, the existence of which suffices to show the safety of the controlled dynamical

system. The crux was to use neural networks to represent the barrier functions, spurred by the well-known universal approximation theorem [22] which assures the expressibility of NNs.

It is imperative to realize that verification or certification of an existing controller does not lend itself to effective and efficient *construction* of controllers, which is the main focus of the current work. Following a correctness-by-design methodology, we aim to synthesize controllers which can guarantee that the controlled system is safe. This question is considerably more challenging and perhaps more interesting from a system engineering perspective. To this end we adopt a data-driven approach for the design of controllers which are to be represented as an NN. A key issue of controller synthesis is to provide a formal guarantee of the quality for the obtained controller, of which safety is arguably the most fundamental. A common practice is to first come up with a controller and then to verify it against desired properties. An interesting innovation of our work is, however, to integrate the synthesis and verification in a unified, data-driven framework, which is enabled by our earlier work by using NNs as a certification mechanism. At a high level, our approach for the controller synthesis will produce two neural networks simultaneously, i.e., one is used to represent the controller (henceforth referred to as controller-NN), and the other is used to represent the barrier function (henceforth referred to as barrier-NN). The synergy of the two NNs, supported by an additional verification procedure to make sure the learned barrier-NN is indeed a barrier certificate, provides the desired safety guarantee for the synthesized controller.

Our method follows a data-driven framework in the sense that both NNs are trained from datasets. For that purpose, we generate training sets and propose specifically designed loss functions which are the key towards application of standard learning algorithms for NNs. In terms of the learned NN controllers, we find that they usually respect safety constraints, but may exhibit poor performance in terms of, e.g., stability. To further improve the synthesized controllers, we propose a number of approaches such as imposing a larger safety region, stability-aware loss functions, and bounded control inputs (via the Hardtanh activation function).

In general, the advantages of our approach are threefold: (1) the approach is data-driven, requiring considerably less control theory expertise; (2) the approach can support non-linear control systems and safety properties, owing to the representation power of neural networks; and (3) the approach can achieve verification-in-the-loop synthesis, owing to the co-synthesis of controller and barrier functions, which can be seamlessly integrated to provide a correctness-by-design controller as well as its certification.

The main contributions of the paper are summarized as follows:

- We put forward a learning-based framework to synthesize controllers as well as the associated safety certification. This is largely a data-driven approach, with little prior knowledge required, and enjoys great flexibility to effectively handle nonlinear (beyond polynomial) dynamics of ODEs.

– We instantiate the framework by using new class of activation functions. Moreover, we demonstrate how to generate training set, and to construct loss functions of neural networks. We also provide practical methods to formally verify the learnt barrier certificates represented as neural networks.
– We carry out proof-of-concept case studies to showcase the efficacy of the approach.

1.1 Related Work

Our work on learning and verifying NN controllers with barrier certificates is closely related to two categories of research, i.e. *safety critical control by machine learning* and *formal verification of neural networks*. Note that the discussions below are necessarily non-exhaustive as a reasonably detailed discussion requires an independent survey.

Safety Critical Control by Machine Learning. Research work in this category has been emerging in the past years. They differ in: (1) the overall learning framework, e.g. reinforcement learning (RL) or supervised learning; (2) the component to be learned (especially by NN), e.g., the system model, the feedback control policy, or the safety certificate; (3) the kind of safety certificate, e.g., control Lyapunov function (CLF) or control barrier function (CBF) [2]. A verification-in-the-loop RL algorithm was proposed in [8] to learn safe NN controllers for known system dynamics using CBFs; an end-to-end safe RL architecture was developed by combining model-free RL control, model-based CBF control, and model learning in [5]; CLFs and CBFs are integrated into the episodic learning framework and RL framework with an emphasis on model uncertainties in [37,38,6]; CBFs are integrated with imitation learning to train safe NN controllers in [43]. For all the above work, CLFs or CBFs are assumed to be given, at least in a parametric form. For CLFs or CBFs synthesis, a demonstrator-learner-verifier framework was proposed in [31] to learn polynomial CLFs for polynomial nonlinear dynamical systems; a special type of neural network was designed in [33] as candidates for learning Lyapunov functions; a supervised learning approach was proposed in [4] to learn neural network Lyapunov functions and linear control policies; data-driven model predictive control (MPC) exploiting neural Lyapunov function and neural network dynamics model was proposed in [12,26]. For multi-agent systems, barrier function has recently been applied for safe policy synthesis on POMDP models [1]. The computer science community has dealt with the issue of safe controller learning in different ways from above: for example, a logical-proof based approach was proposed in [14] towards safe RL; a synthesis framework capable of synthesizing deterministic programs from neural network policies was proposed in [45] and so formal verification techniques for traditional software systems can be applied. Compared with these works, our approach has the following features which make it unique:

– controller and safety certificate are both represented and learned by NNs of general structure; no prior knowledge or initial guess is required;

– training data generation is based on state space sampling, and therefore trajectory simulation is not needed;
– although the method is based on known dynamics, we believe it is possible to extend it with dynamics learning by introducing the third NN representing system dynamics [25].

Formal Verification of Neural Networks. This has attracted considerable research efforts in recent years, and the general problem is NP-hard [20]. A large body of research focuses on the robustness issue of neural networks. In particular, given an input subject to (adversarial) perturbations, one intends to determine whether the output of the neural network (e.g., the classification result) is invariant to these perturbations. Essentially, this is to estimate the output range of a given neural network on a compact set. There are now a wide range of methods including constraint-solving based approaches [20], optimization based approaches [13,41,42], abstract interpretation based approaches [29,23], etc. Furthermore, recently work has been done for verification of control systems with neural network components [11,18,36,9,39]. The main technique is reachability analysis of the closed-loop system, either by finite-state abstraction [36], or by interval-(or other abstract domain)-based reachable set approximation [11,18,39]. Usually reachable set computation can only verify safety up to a finite time horizon, and the approximation error of reachable set may explode. Contrarily, we adopt the deductive approach based on barrier certificate, following and improving the line of work in [40].

1.2 Outline

The rest of this paper is organized as follows: some preliminary knowledge is provided in Section 2 for self-containedness; the main steps of our approach is presented in Section 3 with a running example for demonstration; various improvements of the synthesized controllers are discussed in Section 4; implementation and experiment details are given in Section 5; the paper is concluded by Section 6. We note that a preliminary version is accepted by SETTA 2020 as a short paper under the same title.

2 Preliminaries

Throughout this paper, $\mathbb{R}$ denotes the set of real numbers. For any natural number n, let $[n] = \{1, \cdots, n\}$.

2.1 Constrained Continuous Dynamical System

A continuous dynamical system is modeled by a system of first-order ordinary differential equations (ODEs) $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$, where

– $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T \in \mathbb{R}^n$ is a column vector, $\dot{\mathbf{x}}$ denotes the derivative of $\mathbf{x}$ with respect to the time variable t, and

– $\mathbf{f}(\mathbf{x}) : \Omega \to \mathbb{R}^n$ is a vector field $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \cdots, f_n(\mathbf{x}))^T$ defined on an open subset $\Omega \subseteq \mathbb{R}^n$.

We assume that $\mathbf{f}$ satisfies the *local Lipschitz condition*, which ensures that, given $\mathbf{x} = \mathbf{x}_0$, there exists a time $\mathcal{T} > 0$ and a unique time trajectory $\mathbf{x}(t) : [0, \mathcal{T}) \to \mathbb{R}^n$ such that $\mathbf{x}(0) = \mathbf{x}_0$. In the sequel, the trajectory is denoted by $\mathbf{x}(t, \mathbf{x}_0)$.

A *constrained continuous dynamical systems* (CCDS) is represented by $\Gamma = (\mathbf{f}, X_D, X_I, X_U)$, where

– $\mathbf{f} : \Omega \to \mathbb{R}^n$ is the vector field,
– $X_D \subseteq \Omega$ is an evolution constraint (or system domain),
– $X_I \subseteq X_D$, and
– $X_U \subseteq X_D$.

For CCDSs, the following problem is widely investigated in safety critical applications.

Definition 1 (Safety Verification). *A CCDS $\Gamma = (\mathbf{f}, X_D, X_I, X_U)$ is safe if $\forall \mathbf{x}_0 \in X_I$ and $\forall t \geq 0, \mathbf{x}(t, \mathbf{x}_0) \in X_D$ implies $\mathbf{x}(t, \mathbf{x}_0) \notin X_U$, i.e., the system never reaches X_U from X_I.*

2.2 Controlled CCDS

In this paper, we consider *controlled CCDS* $\Gamma = (\mathbf{f}, X_D, X_I, X_U)$ with continuous dynamics defined by

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) \\ \mathbf{u} = \mathbf{g}(\mathbf{x}) \end{cases}, \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{u} \in U \subseteq \mathbb{R}^m$ are the feedback control inputs, and $\mathbf{f} : \mathbb{R}^{n+m} \to \mathbb{R}^n$ and $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^m$ are locally Lipschitz continuous. The problem we considered in this paper is defined as follows.

Definition 2 (Safe Controller Synthesis). *Given a controlled CCDS $\Gamma = (\mathbf{f}, X_D, X_I, X_U)$ with $\mathbf{f}$ defined by (1), design a locally continuous feedback control law $\mathbf{g}$ such that the closed-loop system Γ with $\mathbf{f} = \mathbf{f}(\mathbf{x}, \mathbf{g}(\mathbf{x}))$ is safe, i.e. the system never reaches X_U from X_I under control $\mathbf{u} = \mathbf{g}(\mathbf{x})$.*

2.3 Barrier Certificate

Given a system Γ, a barrier certificate is a real-valued function $B(\mathbf{x})$ over the states of the system satisfying the condition that $B(\mathbf{x}) \leq 0$ for any reachable state $\mathbf{x}$ and $B(\mathbf{x}) > 0$ for any state in the unsafe set X_U. If such a function $B(\mathbf{x})$ exists, one can easily deduce that the system can *not* reach a state in the unsafe set from the initial set [28,30]. In this paper, we will certify the safety of a synthesized controller by generating barrier certificates.

There are several different formulations of barrier certificates without explicit reference to the solutions of the ODEs [28,21,7,35]. we will adopt what are called *strict barrier certificate* [34] conditions.

Theorem 1 (Strict barrier certificate). *Given a system* $\Gamma = (\mathbf{f}, X_D, X_I, X_U)$*, if there exists a continuously differentiable function* $B : X_D \to \mathbb{R}$ *s.t.*

1. $B(\mathbf{x}) \leq 0$ *for* $\forall \mathbf{x} \in X_I$
2. $B(\mathbf{x}) > 0$ *for* $\forall \mathbf{x} \in X_U$
3. $\mathcal{L}_{\mathbf{f}} B(\mathbf{x}) < 0$ *for all* $\mathbf{x} \in X_D$ *s.t.* $B(\mathbf{x}) = 0$*,*

then the system Γ *is safe, and such* B *is a barrier certificate.*

Note that in the above third condition, $\mathcal{L}_{\mathbf{f}} B$ is the *Lie derivative* of B w.r.t. $\mathbf{f}$, that is, the inner product of $\mathbf{f}$ and the gradient of B:

$$\mathcal{L}_{\mathbf{f}} B(\mathbf{x}) = (\nabla B) \cdot \mathbf{f}(\mathbf{x}) = \sum_{i=1}^{n} \left(\frac{\partial B}{\partial x_i}(\mathbf{x}) \cdot f_i(\mathbf{x}) \right). \tag{2}$$

2.4 Neural Networks

In this paper, both the synthesized control law $\mathbf{g}$ and the barrier certificate B are represented by (feed-forward artificial) neural networks (NNs). We introduce some basic notions here. A typical NN consists of a number of interconnected neurons which are organized in a layered structure. Each neuron is a single processing element that responds to the weighted inputs received from other neurons (cf. Fig. 1.)

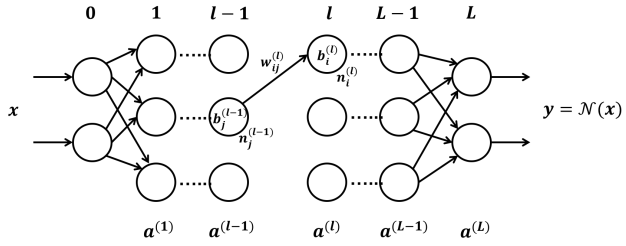


Fig. 1. The structure of a multilayer feed-forward artificial neural network

In general, an NN represents a function $\mathcal{N}(\mathbf{x})$ on the input $\mathbf{x}$ and can be represented as a composition of its layers. We normally reserve 0 and L for the indices of the input and the output layer respectively, and all of the other layers in between are hidden layers. In this paper, we use superscripts to index layer-specific variables. In particular, the layer l comprises neurons $n_i^{(l)}$ for $i \in [d^{(l)}]$, where $d^{(l)}$ is the dimension of the layer l. Neuron $n_j^{(l-1)}$ of the layer $l-1$ is connected with neuron $n_i^{(l)}$ of layer l by a directed edge with weight $w_{ij}^{(l)} \in \mathbb{R}$. Each neuron $n_i^{(l)}$ of layer $l \in [L]$ is associated with a *bias* $b_i^{(l)} \in \mathbb{R}$ and an activation function $a_i^{(l)} : \mathbb{R} \to \mathbb{R}$. Usually the neurons in the same layer

has identical activation functions, denoted by $a^{(l)}$. Commonly used activation functions include ReLU (rectified linear unit, i.e., $\max(0, x)$ for $x \in \mathbb{R}$), sigmoid, hyperbolic tangent, etc.

Denote the input vector to the NN by $\mathbf{x} \in \mathbb{R}^{d^{(0)}}$. Let the output vector of the l-th layer be $\mathbf{x}^{(l)}$. Then $\mathbf{x}^{(0)} = \mathbf{x}$. We introduce the vector variable $\mathbf{z}^{(l)}$ to denote the input vector to the l-th layer for $l \in [L]$. Thus the *forward propagation equations* of an NN can be defined as

$$\begin{cases} \mathbf{x}^{(0)} = \mathbf{x} \\ \mathbf{z}^{(l)} = \mathbf{W}^{(l)} \cdot \mathbf{x}^{(l-1)} + \mathbf{b}^{(l)} & \text{for } l \in [L] \\ \mathbf{x}^{(l)} = a^{(l)}(\mathbf{z}^{(l)}) & \text{for } l \in [L] \\ \mathbf{y} = \mathcal{N}(\mathbf{x}) = \mathbf{x}^{(L)} \end{cases}, \tag{3}$$

where $\mathbf{W}^{(l)}$ is a matrix of dimension $d^{(l)} \times d^{(l-1)}$, $\mathbf{b}^{(l)}$ is a $d^{(l)}$-dimensional column vector, and $a^{(l)}$ is taken as an element-wise function for a vector input.

Training of NN is usually through *backward propagation*, during which the parameters $\mathbf{W}$'s and $\mathbf{b}$'s are learned through an optimization algorithm (e.g., stochastic gradient descent, SGD for short) applied on the training set [15].

3 Methodology

The framework of our safe controller learning approach is demonstrated in Fig. 2. Given a controlled CCDS $\Gamma = (\mathbf{f}, X_D, X_I, X_U)$, the basic idea of the proposed

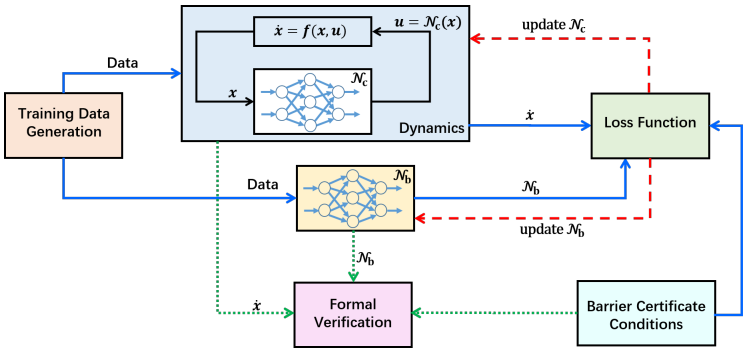


Fig. 2. The framework of safe neural network controller synthesis

approach is to represent the controller function $\mathbf{g}$ as well as the safety certificate function B by two NNs, i.e. $\mathcal{N}_\mathrm{c}$ and $\mathcal{N}_\mathrm{b}$ respectively. Then we formulate the barrier certificate conditions as per Theorem 1 w.r.t. $\mathcal{N}_\mathrm{b}$ and the closed-loop dynamics $\mathbf{f}(\mathbf{x}, \mathcal{N}_\mathrm{c}(\mathbf{x}))$ into a loss function, and then train the two NNs together on a generated training data set until the loss is decreased to 0. The resulting two NNs are the controller and barrier certificate candidates. To overcome

the limitations of data-driven approach, formal verification (SMT solver in this paper) is performed on the synthesized candidates to show that the barrier certificate conditions are indeed satisfied. The blue (solid), red (dashed), and green (dotted) arrows in Fig. 2 shows the information flow of forward propagation, backward propagation, and formal verification, respectively. Next, before giving more detailed steps of our approach, we first introduce a running example.

Example 1 (Dubins' Car [40,8]). The control objective is to steer a car with constant velocity 1 to track a path, here the X-axis in the postive direction. The states of the car are the x, y position and the driving direction θ, which can be transformed to the distance error d_e and angle error θ_e between the current position and the target path (see Fig. 3). The controlled CCDS $\Gamma = (\mathbf{f}, X_D, X_I, X_U)$ is:

$$\mathbf{f}: \begin{bmatrix} \dot{d_\mathrm{e}} \\ \dot{\theta_\mathrm{e}} \end{bmatrix} = \begin{bmatrix} \sin(\theta_\mathrm{e}) \\ -u \end{bmatrix}, \quad \text{where } u \text{ is the scalar control input}$$

- X_D: $\{(d_\mathrm{e}, \theta_\mathrm{e}) \in \mathbb{R}^2 | -6 \le d_\mathrm{e} \le 6, -7\pi/10 \le \theta_\mathrm{e} \le 7\pi/10\}$;
- X_I: $\{(d_\mathrm{e}, \theta_\mathrm{e}) \in \mathbb{R}^2 | -1 \le d_\mathrm{e} \le 1, -\pi/16 \le \theta_\mathrm{e} \le \pi/16\}$;
- X_U: the complement of $\{(d_\mathrm{e}, \theta_\mathrm{e}) \in \mathbb{R}^2 | -5 \le d_\mathrm{e} \le 5, -\pi/2 \le \theta_\mathrm{e} \le \pi/2\}$ in X_D.

Figure 4 shows 50 simulated trajectories on the x-y plane from random initial states in X_I using our learned NN controller u. The two red horizontal lines are the safety upper and lower bounds (± 5) for y (the same bounds as d_e). In the rest of this paper, we will use Example 1 to demonstrate our safe controller synthesis approach.

3.1 The Structure of $\mathcal{N}_\mathbf{c}$ and $\mathcal{N}_\mathbf{b}$

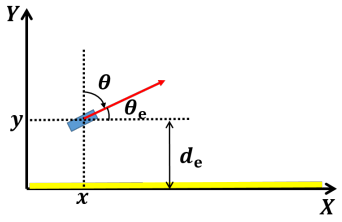


Fig. 3. States of Dubins' car: $d_\mathrm{e} = y$, $\theta_\mathrm{e} = \frac{\pi}{2} - \theta$

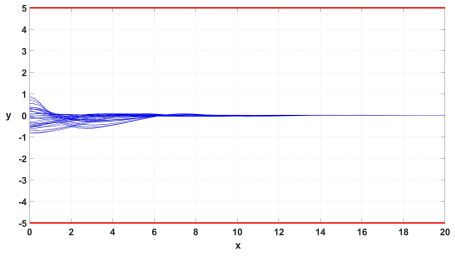


Fig. 4. Simulated car trajectories with learned NN controller

We first fix the structure of $\mathcal{N}_\mathrm{c}$ and $\mathcal{N}_\mathrm{b}$ as follows, assuming that in the controlled CCDS Γ, $\mathbf{x}$ and $\mathbf{u}$ are of n and m dimension respectively, e.g. $n = 2, m = 1$ for Example 1.

- **Input layer** has n neurons for both $\mathcal{N}_\mathrm{c}$ and $\mathcal{N}_\mathrm{b}$;
- **Output layer** has m neurons for $\mathcal{N}_\mathrm{c}$ and one single neuron for $\mathcal{N}_\mathrm{b}$;
- **Hidden layer**: there is no restriction on the number of hidden layers or the number of neurons in each hidden layer; for Example 1, the structures are fixed as: $\mathcal{N}_\mathrm{c}$ has one hidden layer with 5 neurons, and $\mathcal{N}_\mathrm{b}$ has one hidden layer with 10 neurons;
- **Activation function**: considering the inherent requirement of local Lipschitz continuity for $\mathcal{N}_\mathrm{c}$ and the inherent requirement of differentiability for $\mathcal{N}_\mathrm{b}$, and considering the simplicity of formal verification, we adopt ReLU, i.e. $a(x) = \max(0, x)$, and *Bent-ReLU* [44], i.e.,

$$a(x) = 0.5 \cdot x + \sqrt{0.25 \cdot x^2 + 0.0001} \tag{4}$$

  as activation functions for hidden layers of $\mathcal{N}_\mathrm{c}$ and $\mathcal{N}_\mathrm{b}$ respectively (the Lipschitz continuity of ReLU is by [19]); the activation function of the output layer is the identity map for both $\mathcal{N}_\mathrm{c}$ and $\mathcal{N}_\mathrm{b}$.

3.2 Training Data Generation

In our training algorithm, training data are generated by sampling points from the domain X_D, initial set X_I, and unsafe region X_U of the considered system Γ. No simulation of the continuous dynamics is needed. The simplest sampling method is to grid the super-rectangles bounding X_D, X_I, X_U with a fixed mesh size, and then filter out those points not satisfying the constraints of X_D, X_I, X_U. For example, we generate a mesh with $2^8 \times 2^8$ points from X_D for Example 1. The obtained three finite data sets are denoted by S_D, S_I, and S_U.

3.3 Loss Function Encoding

Given S_I, S_U, and S_D, the loss function for training $\mathcal{N}_\mathrm{c}$ and $\mathcal{N}_\mathrm{b}$ can be expressed as

$$L(S_D, S_I, S_U) = c_1 \cdot \sum_{\mathbf{x} \in S_I} L_1(\mathbf{x}) + c_2 \cdot \sum_{\mathbf{x} \in S_U} L_2(\mathbf{x}) + c_3 \cdot \sum_{\mathbf{x} \in S_D} L_3(\mathbf{x}) \tag{5}$$

with

$$\begin{array}{l} L_1(\mathbf{x}) = \mathrm{ReLU}(\mathcal{N}_\mathrm{b}(\mathbf{x}) + \varepsilon_1) \quad \text{for } \mathbf{x} \in S_I\,, \\ L_2(\mathbf{x}) = \mathrm{ReLU}(-\mathcal{N}_\mathrm{b}(\mathbf{x}) + \varepsilon_2) \quad \text{for } \mathbf{x} \in S_U\,, \\ L_3(\mathbf{x}) = \mathrm{ReLU}\big(\mathcal{L}_\mathbf{f}\mathcal{N}_\mathrm{b}(\mathbf{x}) + \varepsilon_3\big) \text{ for } \mathbf{x} \in \{\mathbf{x} \in S_D : |\mathcal{N}_\mathrm{b}(\mathbf{x})| \le \varepsilon_4\} \end{array} \tag{6}$$

denoting the sub-loss functions encoding the three conditions of Theorem 1, and c_1, c_2, c_3 the three positive constant weight coefficients for the sub-losses L_1, L_2, L_3 respectively. The basic idea is to impose a positive (resp., zero) penalty to those sampled points that violate (resp., satisfy) barrier certificate conditions. The $\varepsilon_1, \varepsilon_2, \varepsilon_3$ in (6) are three small non-negative tolerances, the role of which is to get the non-sampled points around the sampled data to have zero loss as well. The ε_4 in (6) is a small positive constant characterizing a narrow belt region around the zero-level set of $\mathcal{N}_\mathrm{b}$, since we cannot sample data on the level set exactly. Note that in the above expression L_3, $\mathbf{f}$ is $\mathbf{f}(\mathbf{x}, \mathcal{N}_\mathrm{c}(\mathbf{x}))$.

3.4 The Training Process

We adopt a modified SGD optimization technique for training the two NNs $\mathcal{N}_\mathrm{c}$ and $\mathcal{N}_\mathrm{b}$. That is, we partition the training data sets S_D, S_I, S_U into mini-batches and shuffle the list of batches to gain some randomness effect, rather than shuffling the whole training data set. For each mini-batch of data, the loss is calculated according to (5) and the the weights and biases of the two NNs are updated by a gradient descent step through backward propagation. To start the training, we must first specify the ε_1 to ε_4 in the loss function, as well as hyper-parameters such as number of restarts n_restart, number of epoches n_epoch, number of mini-batches n_batch, and learning rate l_r, etc. For Example 1, we set $n_\mathrm{restart} = 5$, $n_\mathrm{epoch} = 100$, $n_\mathrm{batch} = 4096$ and $l_r = 0.1$. The choices of ε_1 to ε_4 will be presented in the following subsection. The training process terminates when the loss is decreased to 0 on all mini-batches or the number of restarts exceeds n_restart.

3.5 Formal Verification

The rigorousness of the NNs resulted from 0 training loss is not guaranteed since our approach is data-driven, that is, the three conditions in Theorem 1 are not necessarily satisfied by $\mathcal{N}_\mathrm{c}$ and $\mathcal{N}_\mathrm{b}$. Therefore we resort to formal verification to guarantee the correctness our synthesized controllers. To preform the verification, we replace $\mathbf{f}$ and B in the conditions of Theorem 1 by $\mathbf{f}(\mathbf{x}, \mathcal{N}_\mathrm{c}(\mathbf{x}))$ and $\mathcal{N}_\mathrm{b}$, and try to show that the negation of the conjunction of the three conditions, i.e.

$$\begin{aligned} & \exists \mathbf{x}.\, \mathbf{x} \in X_I \wedge \mathcal{N}_\mathrm{b}(\mathbf{x}) > 0 \\ \vee\ & \exists \mathbf{x}.\, \mathbf{x} \in X_U \wedge \mathcal{N}_\mathrm{b}(\mathbf{x}) \leq 0 \\ \vee\ & \exists \mathbf{x}.\, \mathbf{x} \in X_D \wedge \mathcal{N}_\mathrm{b}(\mathbf{x}) = 0 \wedge \mathcal{L}_{\mathbf{f}(\mathbf{x}, \mathcal{N}_\mathrm{c}(\mathbf{x}))} \mathcal{N}_\mathrm{b}(\mathbf{x}) \geq 0 \end{aligned} \tag{7}$$

is UNSATISFIABLE. Due to the high degree of nonlinearity in $\mathbf{f}$ and $\mathcal{N}_\mathrm{b}$ of (7), its satisfiability is resolved by the interval-propagation based nonlinear SMT solver iSAT3.[4] To speed up the verification process, we compute piece-wise linear approximations (with interval error bounds) of Bent-ReLU function and its derivative, and replace their occurrences in $\mathcal{N}_\mathrm{b}$ and $\mathcal{L}_\mathbf{f}\mathcal{N}_\mathrm{b}$ by the linear approximations. As a result, there are three issues that may affect the efficiency and effectiveness of formal verification:

- The tolerances chosen for loss function encoding in (5) and (6);
- The piece-wise linear approximation error of Bent-ReLU function and its derivative;
- The interval splitting width for iSAT3.

For the third issue, we usually set the minimal splitting width option --msw to 0.001 for iSAT3. The first and second issues are addressed in the following two paragraphs.

[4] https://projects.informatik.uni-freiburg.de/projects/isat3/

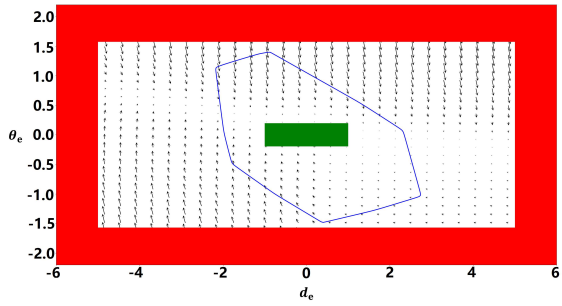
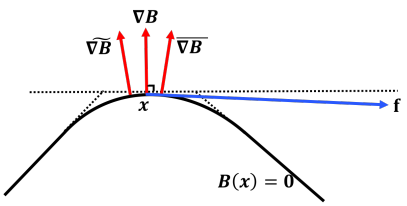
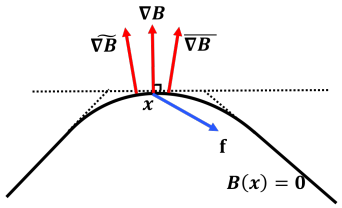


Fig. 5. Learned and verified NN controller and barrier certificate for Example 1: the inner (green) and outer (red) shaded areas are the initial and unsafe regions, black arrows in the white area are the closed-loop vector fields $\mathbf{f}(\mathbf{x}, \mathcal{N}_c(\mathbf{x}))$, and the blue curve surrounding the inner shaded box is the zero-level set of $\mathcal{N}_b$

Pre-training and Fine-tuning. The success of synthesis and formal verification heavily relies on the choices of the four constants ε_1 to ε_4 in (5) and (6). Generally, small tolerances are preferred for faster training, while larger tolerances are preferred for formal verification to compensate for the errors caused by activation function linearization and interval arithmetic computation. In practice, we adopt a pre-training and fine-tuning combination strategy. That is, we start with small positive ε_4 and zero ε_1 to ε_3 to perform the initial training. If the pre-trained NNs failed formal verification, they are iteratively refined by gradually increasing the tolerances. For Example 1, the first controller and barrier certificate are synthesized with $\varepsilon_4 = 0.01$ and $\varepsilon_1 = \varepsilon_2 = \varepsilon_3 = 0$, for which formal verification fails, while the fine-tuned controller and barrier certificate are successfully verified when ε_3 was increased to 0.01 (see Fig. 5).



(a) Negative $\mathcal{L}_\mathbf{f} B$ with large $|\mathcal{L}_\mathbf{f} B|$



(b) Negative $\frac{\mathcal{L}_\mathbf{f} B}{\|\nabla B\| \cdot \|\mathbf{f}\|}$ with large $\frac{|\mathcal{L}_\mathbf{f} B|}{\|\nabla B\| \cdot \|\mathbf{f}\|}$

Fig. 6. The sign of normalized Lie derivative is robust to Bent-ReLU linearization errors

Adding Normalized Lie Derivative in Loss Encoding. Larger tolerances in the loss function (5) and (6) are not always useful for formal verification. To see this, consider checking unsatisfiability of the third condition of (7). Noting that $\mathcal{L}_{\mathbf{f}}B = \nabla B \cdot \mathbf{f} = \|\nabla B\|\|\mathbf{f}\| \cos\theta_{\nabla B,\mathbf{f}}$, where $\|\cdot\|$ denotes the Euclidean norm and $\theta_{\nabla B,\mathbf{f}}$ denotes the angle between ∇B and $\mathbf{f}$, Fig. 6(a) illustrates a situation that a point $\mathbf{x}$ on the zero-level set of a barrier candidate B has negative Lie derivative, since $\theta_{\nabla B,\mathbf{f}}$ is slightly larger than $\frac{\pi}{2}$ at $\mathbf{x}$. Moreover, it can be concluded that $\mathcal{L}_{\mathbf{f}}B(\mathbf{x}) < -\varepsilon_3$ for very large ε_3 since $\|f\|$ is large. However, formal verification of the negative Lie derivative condition would be very hard at $\mathbf{x}$, where ∇B has a large approximation error due to linearization. For instance, if the approximated $\nabla B(\mathbf{x})$ ranges from $\overline{\nabla B}$ to $\overline{\nabla B}$, then formal verification becomes impossible since $\theta_{\overline{\nabla B},\mathbf{f}} < \frac{\pi}{2}$ which makes the Lie derivative positive. The reason for such a phenomenon is that negative $\mathcal{L}_{\mathbf{f}}B$ does not necessarily force the span angle of ∇B and $\mathbf{f}$ to be large, so the sign of $\mathcal{L}_{\mathbf{f}}B$ is not robust to approximation noises of ∇B. The problem can be resolved by introducing additional sub-loss function specifying normalized Lie derivative into the loss function (5) as follows:

$$L_4(\mathbf{x}) = \mathrm{ReLU}\big(\tfrac{\mathcal{L}_{\mathbf{f}}\mathcal{N}_{\mathrm{b}}(\mathbf{x})}{\|\nabla\mathcal{N}_{\mathrm{b}}\|\cdot\|\mathbf{f}\|} + \varepsilon_5\big), \text{ for } \mathbf{x} \in \{\mathbf{x} \in S_D : |\mathcal{N}_{\mathrm{b}}(\mathbf{x})| \leq \varepsilon_4\} \tag{8}$$

where ε_4 are defined in (6) and ε_5 is a non-negative constant. By (8), if a barrier certificate is synthesized with zero L_4 value and enough large ε_5, then the angle between $\nabla\mathcal{N}_{\mathrm{b}}$ and $\mathbf{f}$ would be large enough to tolerant large approximation errors of gradient (cf. Fig. 6(b)).

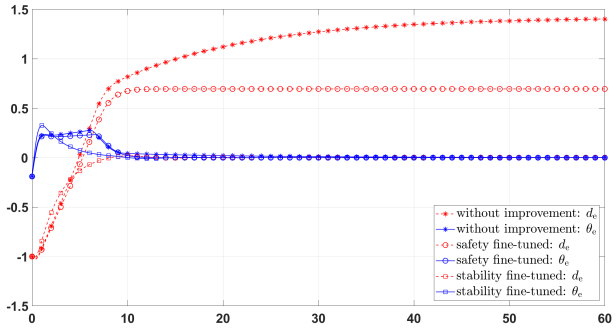


Fig. 7. Simulations of Dubins' car from $(-1, -0.19)$ with different NN controllers for comparison of stability performance

4 Improvement of the Learned Controllers

The controller synthesized and verified in the last section is guaranteed to be safe. However, it may perform poorly regarding properties such as stability. As an

illustration, we simulate the Dubins' car system from initial state $d_e = -1, \theta_e = -0.19$ using the NN controller corresponding to Fig. 5. The changes of d_e and θ_e within 60 time units are shown in Fig. 7 by $*$-marked dashed (d_e) or solid (θ_e) lines. It is obvious that the car has a large distance error although it is still within safety bounds (± 5). We therefore propose a series of ways to improve the performance of synthesized controllers in this section.

4.1 Larger Safety Margin

The first improvement is to gradually increase the safety margin specified by the ε_2 constant in the loss function (5) and (6) by iterative fine-tuning. For example, when ε_2 is increased to 0.8, a NN controller $\mathcal{N}_c$ and the corresponding barrier $\mathcal{N}_b$ are synthesized and shown in Fig. 8. The simulation performance of $\mathcal{N}_c$ is shown in Fig. 7 by $\circ$-marked dashed (d_e) or solid (θ_e) lines. It is obvious that distance error is reduced compared to the controller of Fig. 5.

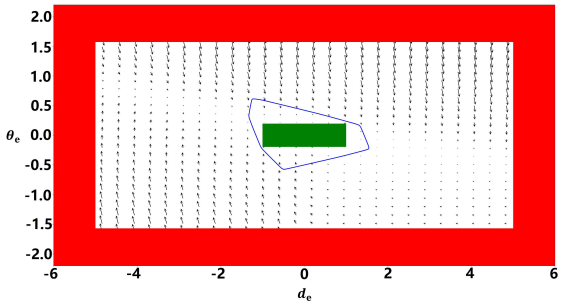


Fig. 8. NN controller learned and verified for Example 1 with larger safety margin: $\varepsilon_1 = 0.02$, $\varepsilon_2 = 0.8$, $\varepsilon_3 = 0.01$, $\varepsilon_4 = 0.05$; the inner (green) and outer (red) shaded areas are the initial and unsafe regions, black arrows in the white area are the closed-loop vector fields $\mathbf{f}(\mathbf{x}, \mathcal{N}_c(\mathbf{x}))$, and the blue curve surrounding the inner shaded box is the zero-level set of $\mathcal{N}_b$

4.2 Asymptotic Stability

Figure 7 shows that using the NN controller with larger safety margin, the distance error of the Dubins' car stabilizes at a value larger than 0.5, which is not desirable. To further reduce the distance error in the long time, we introduce additional loss terms into the loss function to express asymptotic-stability-like properties. Suppose that $\mathbf{x}_o$ is an expected equilibrium point of the system, that is, $\mathbf{f}(\mathbf{x}_o, \mathcal{N}_c(\mathbf{x}_o)) = \mathbf{0}$. For example, the system in Example 1 is expected to stabilize with 0 distance and angle errors and so $\mathbf{x}_o$ is $(0, 0)$. Then we define the sub-loss functions for asymptotic stability as:

$$\begin{aligned} L_5(\mathbf{x}) &= \mathrm{ReLU}\big(-\|\mathbf{f}(\mathbf{x}, \mathcal{N}_c(\mathbf{x}))\| + \varepsilon_6\big) \quad \text{for } \mathbf{x} \in \{\mathbf{x} \in S_D : \|\mathbf{x} - \mathbf{x}_o\| > \varepsilon_7\}, \\ L_6(\mathbf{x}) &= \mathrm{ReLU}\big(\|\mathbf{f}(\mathbf{x}, \mathcal{N}_c(\mathbf{x}))\| - \varepsilon_8\big) \quad \text{for } \mathbf{x} = \mathbf{x}_o \end{aligned} \tag{9}$$

where $\varepsilon_6, \varepsilon_7, \varepsilon_8$ are three small non-negative constants. The basic idea of L_5, L_6 is to impose such constraints that the closed-loop vector field $\mathbf{f}(\mathbf{x}, \mathcal{N}_c(\mathbf{x}))$ has negligible norm at the asymptotically stable point $\mathbf{x}_o$, and strictly positive norm outside a neighborhood of $\mathbf{x}_o$ with radius ε_7. By choosing $\varepsilon_7 = 0.1$, $\varepsilon_6 = 0.05$, $\varepsilon_8 = 0.001$ we obtain a fine-tuned $\mathcal{N}_c$ whose simulation performance is shown in Fig. 7 by □-marked dashed (d_e) or solid (θ_e) lines, which demonstrate good asymptotic stability property. We also fix $\varepsilon_8 = 0.001, \varepsilon_6 = 0.05$ and compare the performances of $\mathcal{N}_c$ obtained from different ε_7 values. The simulation results are shown in Fig. 9. It can be roughly concluded that decreasing ε_7 will have an effect of increasing the *overshoot* and decreasing the *settling time* of the simulated traces. An intuitive explanation of such effects is that by L_5, shrinking ε_7 increases $\|\mathbf{f}\|$ near $\mathbf{x}_o$, and thus trajectories approaches $\mathbf{x}_o$ quickly but may overshoot.

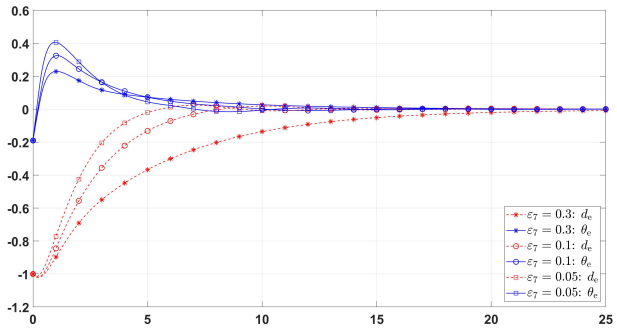


Fig. 9. Comparison of NN controllers learned using L_5 and L_6 losses with $\varepsilon_6 = 0.05$, $\varepsilon_8 = 0.001$ for Example 1: all simulations are from initial state $(-1, -0.19)$; dashed and solid lines represent d_e and θ_e traces respectively; simulations corresponding to controllers learned with $\varepsilon_7 = 0.3, 0.1, 0.05$ are marked by $*$, $\circ$, and □ respectively

Comparison with LQR Controllers. To further evaluate the performance of synthesized NN controllers, we linearize the Dubins' car system near $\mathbf{x}_o = (0, 0)$ and then compute the classic LQR (linear quadratic regulator [16]) controllers for the linearized system. Preliminary experiment shows that for fixed Q and R matrices in the LQR controller computation, by tuning the values of ε_6 and ε_7, we can obtain NN controllers with comparable performances to LQR controllers (cf. Fig. 10).

Remark 1. NNs controllers are in principle much more expressive than linear controllers such as LQR, and so it is interesting to investigate better ways of loss function encoding and controller tuning to gain superior NN controllers to linear controllers (e.g. LQR) in future.

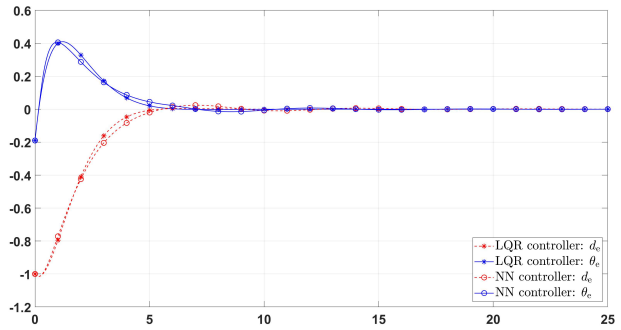


Fig. 10. Simulation of NN and LQR controllers with initial state $(-1, -0.19)$ for Example 1: the NN controller is synthesized with $\varepsilon_6 = \varepsilon_7 = 0.05, \varepsilon_8 = 0.001$, and the LQR controller is synthesized with Q the 2-dimensional identity matrix and $R = 1$; dashed and solid lines represent d_e and θ_e traces respectively, and traces simulated with LQR and NN controllers are marked by $*$ and $\circ$ respectively

4.3 Bounded Control Inputs

In practice, the control input $\mathbf{u}$ to system (1) cannot take arbitrary values but are bounded within a compact set U. Therefore it is necessary to consider how to synthesize bounded NN controllers for practical applications. Actually this can be achieved simply by replacing the identity activation function in the output layer of $\mathcal{N}_c$ (cf. Section 3.1) by any activation with bounded range, say hyperbolic tangent function. For ease of formal verification, we adopt a piece-wise linear activation *Hardtanh* for the output layer of $\mathcal{N}_c$, that is,

$$a^{(L)}(x) = c \cdot \max\big(-1, \min(1, x)\big)$$

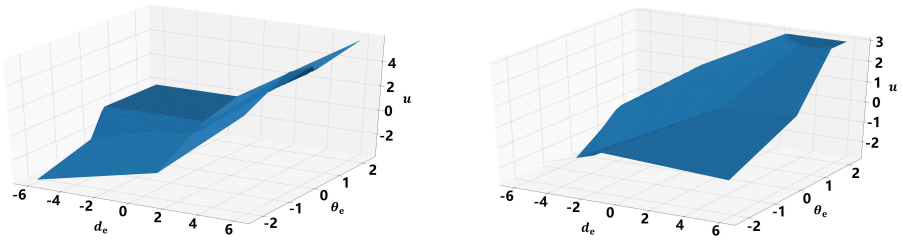
with c a positive constant, which restricts the output of $\mathcal{N}_c$ to be within $[-c, c]$ for each dimension. For Example 1, by choosing $c = 3$ we learned a bounded NN controller as shown in Fig. 11(b). In our experiment, the Hardtanh activation can either be applied in the pre-training or fine-tuning process.

5 Implementation and Experiments

Given a controlled CCDS $\Gamma = (\mathbf{f}, X_D, X_I, X_U)$ and generated training data set S_D, S_I, S_U, in the most general form, the loss function we adopted for training safe NN controllers is:

$$L(S_D, S_I, S_U) = c_1 \sum_{\mathbf{x}\in S_I} L_1(\mathbf{x}) + c_2 \sum_{\mathbf{x}\in S_U} L_2(\mathbf{x}) + \sum_{\mathbf{x}\in S_D} \Big(c_3 L_3(\mathbf{x}) + c_4 L_4(\mathbf{x}) + c_5 L_5(\mathbf{x})\Big) + c_6 L_6(\mathbf{x}_o) \tag{10}$$

where $\mathbf{x}_o$ is the equilibrium point, $L_1, L_2, L_3, L_4, L_5, L_6$ are defined in (6), (8) and (9), c_1, c_2, c_3 are defined in (5), and c_4, c_5, c_6 are non-negative constant sub-loss



(a) Unbounded NN controller (b) Bounded NN controller

Fig. 11. Plotting of surfaces of unbound or bounded NN controllers for Example 1 over X_D

weights. Thus there are totally 6 sub-loss weights denoted by $\mathbf{c} = (c_1, c_2, \ldots, c_6)$ for short; besides, there are 8 tolerances in (10) denoted by $\boldsymbol{\varepsilon} = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_8)$ for short. Our implementation and experiments are conducted based on (10) and related notations.

5.1 The Training Algorithm

The main algorithm for training a safe NN controller is presented in Algorithm 1, which can be explained as follows:

Algorithm 1 Safe NN-Controller Training Algorithm

Input: $\Gamma = (\mathbf{f}, X_D, X_I, X_U)$, n_{restart}, n_{epoch}, n_{batch}, l_r, $\mathbf{c}$, $\boldsymbol{\varepsilon}$;
Output: $\mathcal{N}_{\text{c}}$, $\mathcal{N}_{\text{b}}$;

```
N_c, N_b = nn_construct(Γ);
data_gen(Γ);
for i = 1 to n_restart do
    initialize(N_c, N_b);
    for j = 1 to n_epoch do
        L_epoch = 0;
        for k = 1 to n_batch do
            L_epoch += compute_batch_loss(c, ε);
            update(N_c, N_b, l_r);
        end for
        if decide_success(L_epoch) then
            return N_c, N_b;
        end if
    end for
end for
```

– $n_{\text{restart}}, n_{\text{epoch}}, n_{\text{batch}}$ and l_r are hyper-parameters for training (cf. Section 3.4); in all our case studies, n_{restart} and n_{batch} are fixed at 5 and 4096 respectively;

- nn_construct() in Line 1 is to construct the structure of $\mathcal{N}_c$ and $\mathcal{N}_b$ (cf. Section 3.1); in all our case studies, $\mathcal{N}_c$ has one hidden layer with 5 neurons, and $\mathcal{N}_b$ has one hidden layer with 10 neurons;
- data_gen() in Line 2 is to generate batches of training data (cf. Section 3.2);
- initialize() in Line 4 is to initialize weights and biases of $\mathcal{N}_c$ and $\mathcal{N}_b$ by Gaussian distribution;
- compute_batch_loss() in Line 8 is to compute the loss value on each batch of data using the input $\mathbf{c}, \boldsymbol{\varepsilon}$ (cf. Section 3.3 and (10));
- update() in Line 9 is to update $\mathcal{N}_c$ and $\mathcal{N}_b$ using *gradient descent* with step-size l_r;
- decide_success() in Line 11 is to decide the termination condition, which involves checking whether the epoch loss L_{epoch} reaches 0.

We have implemented a prototype tool nncontroller[5] based on the Pytorch[6] platform. Given a problem description and a set of user-specified parameters (cf. Algorithm 1), nncontroller automatically learns a safe NN controller with a NN barrier certificate, and generate script files as the input to iSAT3 for formal verification. We have applied nncontroller to a number of cases in the literature [40,8,45]. All experiments are performed on a laptop workstation running Ubuntu 18.04 with Intel i7-8550u CPU and 32GB memory. The details of cases studies are presented in the following sub-section.

5.2 Experiment Results

In addition to the running example, we have synthesized and verified NN controllers using nncontroller for the following cases.

Example 2 (Inverted Pendulum [45]). The controlled CCDS $\Gamma = (\mathbf{f}, X_D, X_I, X_U)$ is:

$$\mathbf{f}: \begin{bmatrix} \dot{\theta} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} \omega \\ \frac{g}{l}(\theta - \frac{\theta^3}{6}) + \frac{1}{ml^2}u \end{bmatrix},$$

where $m = 1$ and $l = 1$ denote the pendulum mass and length respectively, $g = 9.8$ is the gravitational acceleration, u is the scalar control input maintaining the pendulum upright, and

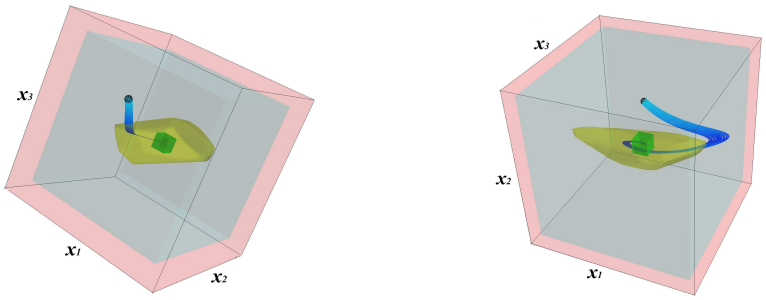
- X_D: $\{\theta, \omega) \in \mathbb{R}^2 | -\pi/2 \leq \theta \leq \pi/2,\ -\pi/2 \leq \omega \leq \pi/2\}$;
- X_I: $\{\theta, \omega) \in \mathbb{R}^2 | -\pi/9 \leq \theta \leq \pi/9,\ -\pi/9 \leq \omega \leq \pi/9\}$;
- X_U: the complement of $\{\theta, \omega) \in \mathbb{R}^2 | -\pi/6 \leq \theta \leq \pi/6,\ -\pi/6 \leq \omega \leq \pi/6\}$ in X_D.

Example 3 (Duffing Oscillator [45]). The controlled CCDS $\Gamma = (\mathbf{f}, X_D, X_I, X_U)$ is:

$$\mathbf{f}: \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} y \\ -0.6y - x - x^3 + u \end{bmatrix},$$

where u is the scalar control input that regulates the system's trajectories to $(0, 0)$, and

[5] Publically available at: https://github.com/zhaohj2017/FAoC-tool
[6] https://pytorch.org/



(a) The bicycle model

(b) The academic 3D model

Fig. 12. Learned and verified NN controllers and barriers for Example 4 and 5: for both cases, the innermost cube (green) represents the initial set, the outermost cube (pink) represents the system domain, and the space between the outermost and the middle cube (grey) is the unsafe region; the irregular surface (yellow) surrounding the innermost cube is the zero-level set of synthesized NN barrier; the curves (blue) approaching the origin are simulated system trajectories

- X_D: $\{x, y) \in \mathbb{R}^2 | -6 \leq x \leq 6, -6 \leq y \leq 6\}$;
- X_I: $\{x, y) \in \mathbb{R}^2 | -2.5 \leq x \leq 2.5, -2 \leq y \leq 2\}$;
- X_U: the complement of $\{x, y) \in \mathbb{R}^2 | -5 \leq x \leq 5, -5 \leq y \leq 5\}$ in X_D.

Example 4 (Bicycle Steering [8]). The control objective is to balance a bicycle. The states of the bicycle are (x_1, x_2, x_3) which denote the tilt angle, the angular velocity of tilt, and the handle bar angle with body respectively. The controlled CCDS $\Gamma = (\mathbf{f}, X_D, X_I, X_U)$ is:

$$\mathbf{f}: \begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \end{bmatrix} = \begin{bmatrix} x_2 \\ \frac{ml}{J}(g \sin x_1 + \frac{v^2}{b} \cos x_1 \tan x_3) \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{amlv}{Jb} \cdot \frac{\cos x_1}{\cos^2 x_3} \\ 1 \end{bmatrix} u\,,$$

where u is the scalar control input, $m = 20$ is the mass, $l = 1$ is the height, $b = 1$ is the wheel base, $J = \frac{mb^2}{3}$ is the moment of inertia, $v = 10$ is the velocity, $g = 10$ is the acceleration of gravity, $a = 0.5$, and

- X_D: $\{(x_1, x_2, x_3) \in \mathbb{R}^3 | -2.2 \leq x_1 \leq 2.2, -2.2 \leq x_2 \leq 2.2, -2.2 \leq x_3 \leq 2.2\}$;
- X_I: $\{(x_1, x_2, x_3) \in \mathbb{R}^3 | -0.2 \leq x_1 \leq 0.2, -0.2 \leq x_2 \leq 0.2, -0.2 \leq x_3 \leq 0.2\}$;
- X_U: the complement of $\{(x_1, x_2, x_3) \in \mathbb{R}^3 | -2 \leq x_1 \leq 2, -2 \leq x_2 \leq 2, -2 \leq x_3 \leq 2\}$ in X_D.

By introducing $\tilde{u}$ such that $u = \tilde{u} \cos^2 x_3 - 20 \cos x_3 \sin x_3$, the original $\mathbf{f}$ is transformed equivalently into

$$\tilde{\mathbf{f}}: \begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \end{bmatrix} = \begin{bmatrix} x_2 \\ 30 \sin x_1 + 15 \tilde{u} \cos x_1 \\ \tilde{u} \cos^2 x_3 - 20 \cos x_3 \sin x_3 \end{bmatrix}.$$

A NN controller representing $\tilde{u}$ was learned and verified for the transformed system $(\tilde{\mathbf{f}}, X_D, X_I, X_U)$ (cf. Fig. 12(a)).

Example 5 (Academic 3D [8]). The controlled CCDS $\mathit{\Gamma} = (\mathbf{f}, X_D, X_I, X_U)$ is:

$$\mathbf{f}: \begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \end{bmatrix} = \begin{bmatrix} x_3 + 8x_2 \\ -x_2 + x_3 \\ -x_3 - x_1^2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u\,, \quad \text{where } u \text{ is the scalar control input}$$

- X_D: $\{(x_1, x_2, x_3) \in \mathbb{R}^3 | -2.2 \le x_1 \le 2.2, -2.2 \le x_2 \le 2.2, -2.2 \le x_3 \le 2.2\}$;
- X_I: $\{(x_1, x_2, x_3) \in \mathbb{R}^3 | -0.2 \le x_1 \le 0.2, -0.2 \le x_2 \le 0.2, -0.2 \le x_3 \le 0.2\}$;
- X_U: the complement of $\{(x_1, x_2, x_3) \in \mathbb{R}^3 | -2 \le x_1 \le 2, -2 \le x_2 \le 2, -2 \le x_3 \le 2\}$ in X_D.

A NN controller was successfully learned and verified for $\mathit{\Gamma}$ (cf. Fig. 12(b)).

Table 1. Key parameters for pre-training and fine-tuning by nncontroller (cf. Algorithm 1 and Remark 2)

E.g.	n_e	l_r	$\mathbf{c}$	$\boldsymbol{\varepsilon}$	$\mathbf{c}^v$	$\boldsymbol{\varepsilon}^v$
1	100	0.1	$(1,1,1,0,0,0)$	$(0,0,0,0.01,\cdot,\cdot,\cdot,\cdot)$	$(1,1,1,0,0,0)$	$(0,0,0.01,0.01,\cdot,\cdot,\cdot,\cdot)$
2	100	0.1	$(1,1,1,0,0,0)$	$(0,0,0,0.01,\cdot,\cdot,\cdot,\cdot)$	$(1,1,1,0,0,0)$	$(0.01,0,0.02,0.01,\cdot,\cdot,\cdot,\cdot)$
3	100	0.01$\rightsquigarrow$0.1	$(1,1,1,0,0,0)$	$(0,0,0,0.05,\cdot,\cdot,\cdot,\cdot)$	$(1,1,1,0,0,0)$	$(0,0,0,0.05,\cdot,\cdot,\cdot,\cdot)$
4	200	0.01$\rightsquigarrow$0.2	$(1,1,0.1,0.1,0,0)$	$(0,0,0,0.02,0,\cdot,\cdot,\cdot)$	$(1,1,0.1,0.1,0.01,0.01)$	(0,0,0.35,0.02,0.35,0.1,0.1,0.01)
5	200	0.01$\rightsquigarrow$0.2	$(1,1,0.1,0.1,0,0)$	$(0,0,0,0.02,0,\cdot,\cdot,\cdot)$	$(1,1,0.1,0.1,0.01,0.01)$	(0.01,0.01,0.15,0.02,0.1,0.1,0.2,0.01)

Table 2. Time costs of synthesis and verification by nncontroller and iSAT3 (cf. Remark 3)

E.g.	run 1		run 2		run 3		run 4		run 5		learning	verification
	time	n_r	time	n_r	time	n_r	time	n_r	time	n_r	avg. cost	cost
1	21.11	0	15.04	0	14.98	0	65.25	0	15.37	0	26.35	8.27
2	478.29	1	168.75	0	292.96	0	111.55	0	43.89	0	219.09	15.24
3	60.59	0	72.47	0	64.64	0	48.08	0	851.49	1	219.45	4.71
4	752.63	1	1528.07	2	499.83	0	122.64	0	924.41	1	765.52	1344.50
5	240.94	0	301.22	0	2522.14	3	1001.66	1	390.25	0	891.24	6070.83

The key parameters used by nncontroller for our experiments are summarized in Table 1, and the time costs of synthesis and verification by nncontroller and iSAT3 are summarized in Table 2.

Remark 2. In Table 1, n_e is a shorthand for n_{epoch}, $\cdot$ means the corresponding parameter is not applicable, $\rightsquigarrow$ means we adopt a self-adaptive learning rate scheduling strategy, and the superscript v means that the weight coefficients $\mathbf{c}^v$ and parameters $\boldsymbol{\varepsilon}^v$ are for the fine-tuned controllers, which are formally verified.

Remark 3. In Table 2, all time costs are measured in seconds; the time cost of NN controller training is not deterministic since the NN models are initialized randomly and the batches of training data are shuffled during the training process, and therefore we record the time costs of 5 separate runs of the training algorithm and compute the averaged cost; n_{r} denotes how many times we restart the algorithm when no NN controller is learned within the specified number of training epochs, i.e. n_{epoch}; the last column corresponds to time costs of formal verification for the NN controllers and barriers obtained with the $\mathbf{c}^{\mathrm{v}}$ and $\boldsymbol{\varepsilon}^{\mathrm{v}}$ parameters in Table 1 for each case.

Remark 4. Comparison of time costs of our experiment with related work such as [8,45] is not straightforward since we train two NNs simultaneously, while [8] requires user-provided barrier functions and [45] requires pre-trained NN controllers as their inputs. However, considering the number of layers and neurons (we use one hidden layer with 5 neurons and ReLU activations for $\mathcal{N}_{\mathrm{c}}$ uniformly), it can be asserted that our synthesized NN controllers have much simpler structure than [8,45].

6 Conclusion

We have proposed a new approach to synthesize neural network controllers for nonlinear continuous dynamical systems with control against safety properties. Our approach features in verification-in-the-loop synthesis: we simultaneously train the controller and its certificate, which we use barrier functions, represented by an NN as well. We have provided a prototype tool nncontroller with a number of case studies. The experiment results have confirmed the feasibility and efficacy of our approach.

Future work includes experimenting on different sampling and training strategies to reduce the data set size and to improve the training efficiency, as well different verification methods/tools other than interval SMT solvers. We anticipate that these would potentially further improve the scalability of our approach. We also plan to extend our approach to other properties such as reachability coupled with cost/reward based optimality as what has been done in optimal control.

Acknowledgements. We thank Prof. Jyotirmoy V. Deshmukh for explanations on the bicycle model of Example 4.

H. Zhao was supported partially by the National Natural Science Foundation of China (No. 61702425, 61972385); X. Zeng was supported partially by the National Natural Science Foundation of China (No. 61902325), and "Fundamental Research Funds for the Central Universities" (SWU117058); T. Chen is partially supported by NSFC grant (No. 61872340), and Guangdong Science and Technology Department grant (No. 2018B010107004), the Overseas Grant of the State Key Laboratory of Novel Software Technology (No. KFKT2018A16), the Natural Science Foundation of Guangdong Province of China (No. 2019A1515011689); Z. Liu was supported partially by the National Natural Science Foundation of China (No. 61672435, 61732019, 61811530327), and Capacity Development Grant of Southwest University (SWU116007); J. Woodcock was partially supported by the research grant from Southwest University.

References

1. Ahmadi, M., Singletary, A., Burdick, J.W., Ames, A.D.: Safe policy synthesis in multi-agent POMDPs via discrete-time barrier functions. In: 2019 IEEE 58th Conference on Decision and Control (CDC). pp. 4797–4803. IEEE (2019)
2. Ames, A.D., Coogan, S., Egerstedt, M., Notomista, G., Sreenath, K., Tabuada, P.: Control barrier functions: Theory and applications. In: 2019 18th European Control Conference (ECC). pp. 3420–3431 (2019)
3. Berkenkamp, F., Turchetta, M., Schoellig, A.P., Krause, A.: Safe model-based reinforcement learning with stability guarantees. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. p. 908919. NIPS17, Curran Associates Inc., Red Hook, NY, USA (2017)
4. Chang, Y.C., Roohi, N., Gao, S.: Neural Lyapunov control. In: Advances in Neural Information Processing Systems 32, pp. 3245–3254. Curran Associates, Inc. (2019)
5. Cheng, R., Orosz, G., Murray, R.M., Burdick, J.W.: End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks. In: The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019. pp. 3387–3395. AAAI Press (2019)
6. Choi, J., Castaeda, F., Tomlin, C.J., Sreenath, K.: Reinforcement learning for safety-critical control under model uncertainty, using control Lyapunov functions and control barrier functions (2020)
7. Dai, L., Gan, T., Xia, B., Zhan, N.: Barrier certificates revisited. Journal of Symbolic Computation **80**, 62–86 (2017)
8. Deshmukh, J.V., Kapinski, J., Yamaguchi, T., Prokhorov, D.: Learning deep neural network controllers for dynamical systems with safety guarantees: Invited paper. In: 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). pp. 1–7 (2019)
9. Dreossi, T., Fremont, D.J., Ghosh, S., Kim, E., Ravanbakhsh, H., Vazquez-Chanlatte, M., Seshia, S.A.: VerifAI: A toolkit for the formal design and analysis of artificial intelligence-based systems. In: Computer Aided Verification. pp. 432–442. Springer International Publishing (2019)
10. Duan, Y., Chen, X., Houthooft, R., Schulman, J., Abbeel, P.: Benchmarking deep reinforcement learning for continuous control. In: Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016. JMLR Workshop and Conference Proceedings, vol. 48, pp. 1329–1338. JMLR.org (2016)
11. Dutta, S., Chen, X., Sankaranarayanan, S.: Reachability analysis for neural feedback systems using regressive polynomial rule inference. In: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC. pp. 157–168 (2019)
12. Dutta, S., Jha, S., Sankaranarayanan, S., Tiwari, A.: Learning and verification of feedback control systems using feedforward neural networks. IFAC-PapersOnLine **51**(16), 151 – 156 (2018), 6th IFAC Conference on Analysis and Design of Hybrid Systems ADHS 2018
13. Dutta, S., Jha, S., Sankaranarayanan, S., Tiwari, A.: Output range analysis for deep feedforward neural networks. In: NASA Formal Methods. pp. 121–138. Springer International Publishing (2018)
14. Fulton, N., Platzer, A.: Safe reinforcement learning via formal methods: Toward safe control through proof and learning. In: Proceedings of the Thirty-Second AAAI

Conference on Artificial Intelligence, (AAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018. pp. 6485–6492. AAAI Press (2018)
15. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. The MIT Press (2016)
16. Hespanha, J.P.: Linear Systems Theory. Princeton University Press, second edn. (2018)
17. Ivanov, R., Carpenter, T.J., Weimer, J., Alur, R., Pappas, G.J., Lee, I.: Case study: verifying the safety of an autonomous racing car with a neural network controller. In: HSCC '20: 23rd ACM International Conference on Hybrid Systems: Computation and Control, Sydney, New South Wales, Australia, April 21-24, 2020. pp. 28:1–28:7. ACM (2020)
18. Ivanov, R., Weimer, J., Alur, R., Pappas, G.J., Lee, I.: Verisig: verifying safety properties of hybrid systems with neural network controllers. In: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2019. pp. 169–178 (2019)
19. Jordan, M., Dimakis, A.G.: Exactly computing the local Lipschitz constant of ReLU networks (2020)
20. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient smt solver for verifying deep neural networks. In: International Conference on Computer Aided Verification. pp. 97–117. Springer (2017)
21. Kong, H., He, F., Song, X., Hung, W.N., Gu, M.: Exponential-condition-based barrier certificate generation for safety verification of hybrid systems. In: Proceedings of the 25th International Conference on Computer Aided Verification (CAV). pp. 242–257. Springer (2013)
22. Leshno, M., Lin, V.Y., Pinkus, A., Schocken, S.: Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. Neural Networks **6**(6), 861 – 867 (1993)
23. Li, J., Liu, J., Yang, P., Chen, L., Huang, X., Zhang, L.: Analyzing deep neural networks with symbolic propagation: Towards higher precision and faster verification. In: Static Analysis. pp. 296–319. Springer International Publishing (2019)
24. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. In: 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings (2016)
25. Manek, G., Kolter, J.Z.: Learning stable deep dynamics models (2020)
26. Mittal, M., Gallieri, M., Quaglino, A., Salehian, S.S.M., Koutnk, J.: Neural Lyapunov model predictive control (2020)
27. Poznyak, A., EN, S., Yu, W.: Differential Neural Networks for Robust Nonlinear Control. World Scientific (2001)
28. Prajna, S., Jadbabaie, A., Pappas, G.J.: A framework for worst-case and stochastic safety verification using barrier certificates. IEEE Transactions on Automatic Control **52**(8), 1415–1429 (2007)
29. Pulina, L., Tacchella, A.: An abstraction-refinement approach to verification of artificial neural networks. In: Computer Aided Verification. pp. 243–257 (2010)
30. Ratschan, S.: Converse theorems for safety and barrier certificates. IEEE Transactions on Automatic Control **63**(8), 2628–2632 (2018)
31. Ravanbakhsh, H., Sankaranarayanan, S.: Learning control Lyapunov functions from counterexamples and demonstrations. Autonomous Robots **43**(2), 275–307 (2019)
32. Ray, A., Achiam, J., Amodei, D.: Benchmarking safe exploration in deep reinforcement learning, `https://cdn.openai.com/safexp-short.pdf`

33. Richards, S.M., Berkenkamp, F., Krause, A.: The Lyapunov neural network: Adaptive stability certification for safe learning of dynamic systems. CoRR **abs/1808.00924** (2018), http://arxiv.org/abs/1808.00924
34. Sloth, C., Pappas, G.J., Wisniewski, R.: Compositional safety analysis using barrier certificates. In: Proc. of the Hybrid Systems: Computation and Control (HSCC). pp. 15–24. ACM (2012)
35. Sogokon, A., Ghorbal, K., Tan, Y.K., Platzer, A.: Vector barrier certificates and comparison systems. In: Formal Methods. pp. 418–437 (2018)
36. Sun, X., Khedr, H., Shoukry, Y.: Formal verification of neural network controlled autonomous systems. In: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2019. pp. 147–156 (2019)
37. Taylor, A.J., Dorobantu, V.D., Le, H.M., Yue, Y., Ames, A.D.: Episodic learning with control Lyapunov functions for uncertain robotic systems. In: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 6878–6884 (2019)
38. Taylor, A., Singletary, A., Yue, Y., Ames, A.: Learning for safety-critical control with control barrier functions (2019)
39. Tran, H.D., Yang, X., Lopez, D.M., Musau, P., Nguyen, L.V., Xiang, W., Bak, S., Johnson, T.T.: NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In: 32nd International Conference on Computer-Aided Verification (CAV) (2020)
40. Tuncali, C.E., Kapinski, J., Ito, H., Deshmukh, J.V.: Invited: Reasoning about safety of learning-enabled components in autonomous cyber-physical systems. In: 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC). pp. 1–6 (2018)
41. Weng, T., Zhang, H., Chen, H., Song, Z., Hsieh, C., Daniel, L., Boning, D.S., Dhillon, I.S.: Towards fast computation of certified robustness for relu networks. In: Proceedings of the 35th International Conference on Machine Learning, ICML 2018. pp. 5273–5282 (2018)
42. Xiang, W., Tran, H., Johnson, T.T.: Output reachable set estimation and verification for multi-layer neural networks. CoRR **abs/1708.03322** (2017)
43. Yaghoubi, S., Fainekos, G., Sankaranarayanan, S.: Training neural network controllers using control barrier functions in the presence of disturbances (2020)
44. Zhao, H., Zeng, X., Chen, T., Liu, Z.: Synthesizing barrier certificates using neural networks. In: HSCC '20. pp. 25:1–25:11. ACM (2020)
45. Zhu, H., Xiong, Z., Magill, S., Jagannathan, S.: An inductive synthesis framework for verifiable reinforcement learning. In: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 686–701. PLDI 2019, Association for Computing Machinery, New York, NY, USA (2019)