Learning Deep Neural Network Controllers for Dynamical Systems with Safety Guarantees

JYOTIRMOY V. DESHMUKH, University of Southern California
JAMES P. KAPINSKI, Toyota R&D
TOMOYA YAMAGUCHI, Toyota R&D
DANIL PROKHOROV, Toyota R&D

There is recent interest in using deep neural networks (DNNs) for controlling autonomous cyber-physical systems (CPSs). One challenge with this approach is that many autonomous CPS applications are safety-critical, and is not clear if DNNs can proffer safe system behaviors. To address this problem, we present an approach to modify existing (deep) reinforcement learning algorithms to guide the training of those controllers so that the overall system is safe. We present a novel verification-in-the-loop training algorithm that uses the formalism of barrier certificates to synthesize DNN-controllers that are safe by design. We demonstrate a proof-of-concept evaluation of our technique on multiple CPS examples.

ACM Reference Format:
Jyotirmoy V. Deshmukh, James P. Kapinski, Tomoya Yamaguchi, and Danil Prokhorov. 2020. Learning Deep Neural Network Controllers for Dynamical Systems with Safety Guarantees. 1, 1 (June 2020), 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The design of control and decision-making software for cyber-physical systems (CPSs) such as unmanned aerial vehicles, ground vehicles and general robots is a highly challenging problem. Spurred by the exciting developments in the field of deep learning, recently, there has been burgeoning interest the learning-oriented paradigm for designing such software controllers. A typical learning oriented approach involves the following steps: (1) pick a (parameterized) control architecture that lends itself to an efficient learning procedure; (2) setup a physical or virtual experiment to generate system behaviors; (3) define cost functions to quantify performance of the system with respect to specific behaviors; and finally, (4) employ an optimization scheme to pick the optimal controller parameters with respect to the specified cost function. Due to the growing importance and expressive power of *deep learning* [12], *deep neural networks* (DNNs) have been proposed as an effective parameterized control architecture. A number of techniques included under the umbrella of *Deep Reinforcement Learning* have been used to effectively learn controllers from user-defined cost functions encoding desired system behavior. However, there has been limited research on techniques that formally reason about the safety of such DNN-controlled dynamical systems, or that seek to incorporate safety as part of the learning process. This is, however, a challenging problem as models of autonomous CPSs often represent highly nonlinear dynamical systems, not to mention the challenge of reasoning about inherently nonlinear DNNs.

In this paper, we describe a new approach to training DNN-based controllers for CPS applications such that the system behaviors are guaranteed to satisfy a given safety requirement. The key idea in our approach is to incorporate safety requirements into the RL-based training using three main steps: (1) identify an *a priori* safety invariant that

James P. Kapinski is now an employee of Amazon.
Authors' addresses: Jyotirmoy V. Deshmukh, University of Southern California, jyotirmoy.deshmukh@usc.edu; James P. Kapinski, Toyota R&D, jimkapinski@gmail.com; Tomoya Yamaguchi, Toyota R&D, tomoya.yamaguchi@toyota.com; Danil Prokhorov, Toyota R&D, danil.prokhorov@toyota.com.

2020. Manuscript submitted to ACM

we require the DNN-controlled closed-loop system satisfy, (2) modify the cost function used in RL to bias the search towards safe controllers, and finally, (3) use an automated reasoning tool to verify system safety. In the last step, if the verification fails, the resulting counterexample is used to improve the cost function, and then the system is re-trained. The process repeats until the automated reasoning tool produces a proof of safety. We demonstrate our technique through a proof-of-concept tool on three nonlinear dynamical system examples.

2 NEURAL NETWORKS AND DYNAMICAL SYSTEMS

Closed-loop Control System. A closed-loop control model of a CPS consists of two main parts: a *plant model* representing the physical components of the system, and a *controller*. Depending on the modeling assumptions, closed-loop control systems can be either continuous-time and modeled by ordinary differential equations (ODEs) or discrete-time and modeled using recursive difference equations. In this paper, we describe the problem we wish to solve assuming a continuous-time model, and comment on extensions to discrete-time.

A plant model is characterized by an n-dimensional state variable, $\mathbf{x} = (x_1, \ldots, x_n)$, and the m-dimensional control input to the plant, $\mathbf{u} = (u_1, \ldots, u_m)$. At any given time t, the instantaneous configuration of the plant is $\mathbf{x}(t)$. The set of all states of the plant is denoted by X and is assumed to be some compact subset of $\mathbb{R}^n$. Similarly, the input to the plant at time t, $\mathbf{u}(t)$, is assumed to take values from the set U, which is assumed to be a compact subset of $\mathbb{R}^m$, where $m \leq n$. We assume that the controller is *stateless*, meaning that at any time t, the output of the controller (i.e., the input to the plant $\mathbf{u}(t)$) depends only on the current state of the plant. We assume an *affine-control form* for the given system as in [17]:

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t)) + g(\mathbf{x}(t)) \cdot \mathbf{u}(t) \tag{1}$$

$$\mathbf{u}(t) = k(\mathbf{x}(t)) \tag{2}$$

Here, f and g are locally Lipschitz functions, k is some arbitrary nonlinear function representing the nonlinear controller, and $\dot{\mathbf{x}}(t)$ denotes the time-derivative of $\mathbf{x}$ at time t. Henceforth, we abuse terminology and use the symbol $\mathbf{x}$ to denote both a state variable and a value assigned to the state variable (at an unspecified time).

DNN Controllers. Control design using deep learning has been gaining significant interest in recent years [12, 20]. This includes techniques for deep RL [22] and deep imitation learning [36, 42]. These techniques use a DNN to output control actions based on the current plant state (and possibly exogenous inputs from the environment). A DNN [20] is a universal function approximator mapping a multi-dimensional input to a multi-dimensional output. For DNN controllers, at each time step t, the input to the DNN is the state of the system $\mathbf{x}(t)$, and the output is the control action $\mathbf{u}(t) = \eta(\mathbf{x}(t))$, which yields the following closed-loop dynamics:

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t)) + g(\mathbf{x}(t)) \cdot \eta(\mathbf{x}(t)). \tag{3}$$

Figure 1 illustrates a DNN-controlled closed-loop system.

An L-layer DNN ($L > 2$) is defined recursively: layer 1 is the input layer, layer L is the output layer, while layers $2, \ldots, L-1$ are the hidden layers. For $\ell \in [2, L]$, the ℓ^{th} contains d_ℓ neurons, the output of the j^{th} neuron in the ℓ^{th} layer is denoted $v_{\ell,j}$, and V_ℓ is the vector of d_ℓ outputs. For $\ell \in [2, L]$, the ℓ^{th} layer is parameterized by a $d_\ell \times d_{\ell-1}$ *weight matrix* W_ℓ and a *bias vector* B_ℓ of size d_ℓ. Note that here, $d_1 = n$ and $d_L = m$. For $\ell \in [2, L]$, V_ℓ is given by the expression $\sigma_\ell(W_\ell \cdot V_{\ell-1} + B_\ell)$, where σ_ℓ is a suitable activation function applied component-wise to its arguments. There are many activation functions used in the literature; popular ones include the rectified linear unit (ReLU), the hyperbolic

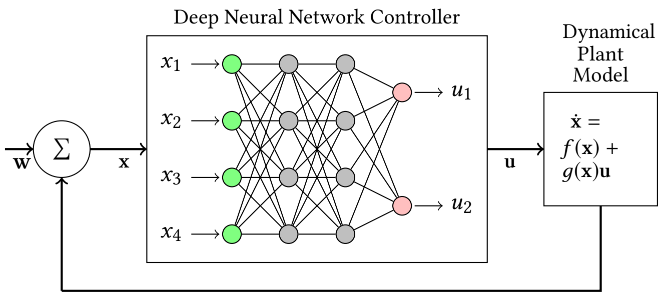

Fig. 1. DNN-controlled closed-loop system model

tangent function, the sigmoid activation function and many other variants. DNNs with the topology described above are called *fully connected networks*. In certain kinds of NNs (such as convolutional NNs [19]), the inter-layer connections may be sparse. Typically, DNNs used for control use the identity function as the activation function for the output layer, while activations for hidden layers could be application-specific.

Reinforcement Learning (RL) and Deep RL. Traditional RL [35] is used to learn the optimal control strategy (or *policy*) for a given closed-loop system, where the desired system behavior is indicated through a hand-crafted reward function that assigns a real-valued reward to each pair $(\mathbf{x}, a)$, where $\mathbf{x} \in X$ is a state and $a \in U$ is an action possible from $\mathbf{x}$. The optimal controller obtained using RL chooses an action such that the expected reward from each state in an infinite run of the system is maximized. Such techniques are considered *model-based* techniques as they assume a probabilistic transition system model such as a Markov Decision Process, and typically use dynamic programming based approaches to obtain the optimal policy [7]. Model-free RL uses physical or virtual tests to simultaneously learn the system structure and estimate the expected reward from each state. Deep RL techniques use DNNs to output an action taken from a given plant state, an expected reward from each state action pair, or a combination of the two [14, 22, 23]. In this paper, we assume a single DNN that maps a state to the resulting control action.

Problem Definition. For the dynamical system specified in Eq. (1), let Φ denote the trajectory function, where $\Phi(x_{\text{init}}, t)$ is a solution to initial value problem of the ODE represented by Eq. (1) with $\mathbf{x}(0) = x_{\text{init}}$. We use $\Phi(x_{\text{init}}, [0, T])$ to denote the function of time $\mathbf{x}(t)$ that satisfies Eq. (1) for $0 \leq t \leq T$. We now define the main problems that we wish to solve. Let X_{init} and X_{unsafe} be arbitrary subsets of X that respectively denote the set of initial values and the set of unsafe values for the state variable $\mathbf{x}$.

Definition 2.1 (DNN Controllers with safety guarantees). Given initial states $X_{\text{init}} \subseteq X$, and unsafe states $X_{\text{unsafe}} \subseteq X$, we say that the system described by Eq. (1) is *safe* iff $\forall x_{\text{init}} \in X_{\text{init}}$, and $\forall t \in [0, \infty)$, $\Phi(x_{\text{init}}, t) \cap X_{\text{unsafe}} = \emptyset$. A controller η is said to be a DNN controller with safety guarantees, if it ensures safety of Eq. (1) when $k = \eta$.

Barrier Certificates. We now present a safety verification technique based on continuous-time inductive invariants, also known as barrier certificates. This technique can be used to prove that a given closed-loop system is safe. To define a barrier certificate formally, we first introduce some terminology. Given a vector field h from X to X, and a function F from X to $\mathbb{R}$, the Lie derivative of F along h, denoted $L_h F(\mathbf{x})$ is defined as $\frac{\partial F}{\partial \mathbf{x}} \cdot h(\mathbf{x})$. The quantity $L_h F(\mathbf{x})$ gives the rate of change of F along the flow defined by h.

Definition 2.2 (Barrier Certificate). A *barrier certificate* describes a safe, initialized, and forward invariant set using a *barrier function* B, which is a differentiable function from X to $\mathbb{R}$. Given a DNN-controlled system of the form specified in Eq. (3) (i.e. $\dot{\mathbf{x}} = f(\mathbf{x}(t)) + g(\mathbf{x}(t))\eta(\mathbf{x}(t))$), the barrier function has the following properties:

(1) $\forall \mathbf{x} \in x_{\text{init}}$: $B(\mathbf{x}) < 0$,
(2) $\forall \mathbf{x} \in X_{\text{unsafe}}$: $B(\mathbf{x}) > 0$,
(3) $\forall \mathbf{x}$ s.t. $B(\mathbf{x}) = 0 : L_f B(\mathbf{x}) + L_g B(\mathbf{x})\eta(\mathbf{x}) < 0$.

3 VERIFICATION-IN-THE-LOOP RL

We propose a new technique to learn a control policy that guarantees the resulting closed-loop system satisfies barrier certificate conditions. We first introduce some terminology.

Definition 3.1 (Safe and Initialized Sets). A set C is called *safe* if $C \cap X_{\text{unsafe}} = \emptyset$. A set C is called *initialized* if $I \subseteq C$.

Given a safe and initialized set, we would like to design a controller that guarantees that the boundary of the set serves as a barrier certificate. Unfortunately, it is not always possible to find a controller that is guaranteed to keep a given closed-loop system within a safe and initialized set, as the following example illustrates.

Example 3.2. Let C be the unit circle in $\mathbb{R}^2$, $C = \{\mathbf{x}|B(\mathbf{x}) = 0\}$, where $B(\mathbf{x}) = x_1^2 + x_2^2 - 1$, let $I = (0, 0)$ and $X_{unsafe} = \{\mathbf{x}|B(\mathbf{x}) \geq 10\}$, and consider the following system dynamics:

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} k(u(t)), \tag{4}$$

where k and u are scalar functions. The set C is safe and initialized, but no controller policy exists that will guarantee that the boundary of C serves as a barrier certificate. To see this, consider the point on the boundary of C, $\mathbf{x} = (0, 1)$, and observe that at this point $L_f B(\mathbf{x}) + L_g B(\mathbf{x})\eta(\mathbf{x}) = 2x_1^2 + 2x_2^2 + 2x_1 k(u(t)) = 2$. In other words, regardless of what form k takes, Condition 3 from Def. 2.2 is not satisfiable by any controller, at this point.

In order to guarantee that there *exists* a controller that keeps the closed-loop system within the safe and initialized set, we need an additional notion known as *forward invariance* of the set. Essentially, adding the notion of forward invariance allows us to construct an argument similar to a barrier certificate that proves system safety.

Definition 3.3 (Forward Invariant Set). A set $C \subseteq X$ is called forward invariant for a system of the form given in Eq. (3), if $\forall$ states $x_{\text{init}} \in C$, and $\forall$ time $t \geq 0$, $\Phi(x_{\text{init}}, t) \in C$.

We say that a set C is a *controlled invariant set* (CIS) if for any system trajectory starting in an initial state x_{init}, at all times $t \geq 0$, there exists some control input $\mathbf{u} \in U$ that guarantees that C is *forward invariant*. A related notion from [41] is that of a *control barrier function* (CBF).

Definition 3.4 (Control Barrier Function). A differentiable function B satisfying the following conditions is called a control barrier function (CBF) if $|\{\mathbf{x} \mid B(\mathbf{x}) \leq 0\}| \neq \emptyset$, B satisfies the barrier conditions for X_{init} and X_{unsafe} states, and: $(L_g B(\mathbf{x}) = 0) \Rightarrow (L_f B(\mathbf{x}) < 0)$.

Our main idea is that given a control barrier function B implicitly defining the safe and initialized control invariant set $B(\mathbf{x}) \geq 0$, we can find a neural network controller η that is guaranteed to keep the system with this set.

Algorithm 1: Verification-in-the-loop Reinforcement Learning

```
Input: Controlled Dynamical System $M(\eta(\cdot))$,
       Control Barrier Function $C$,
       Performance Cost Function $\mu$,
       Maximum training iterations maxIter,
       Initial Set of States $X_{\text{init}}$,
       Bounds $W$: weights and $B$: biases
Output: Verified and Trained NN Controller/Failure
1  iter ← 0
2  $w_0, b_0$ ← uniform_random($W, B$)
3  while (iter < maxIter) and not(isVerified) do
4      $\tilde{\mu}(x_{\text{init}}) \leftarrow \mu(x_{\text{init}})$ + cbf_penalty($x_{\text{init}}$)
5      $w^{\star}_{\text{iter}}, b^{\star}_{\text{iter}} \leftarrow \min_{w \in W, b \in B} \max_{x_{\text{init}} \in X} \tilde{\mu}\,(\Phi_{\text{sim}}(x_{\text{init}}, [0, \ldots, T]))$
6      isVerified, $cex$ ← checkBarrier($M(\eta(w^{\star}_{\text{iter}}, b^{\star}_{\text{iter}})), C$)
7      if not(isVerified) then
8          $X = X \cup \{cex\}$
9          iter ← iter + 1
10     end
11 end
12 if isVerified then return verified controller $\eta(w^{\star}_{\text{iter}}, b^{\star}_{\text{iter}})$
13 else return failure
```

Finding Control Barrier Functions. We note that the condition above allows checking if a given function is a valid CBF *independent* of an actual controller. However, these conditions may be too permissive, in the sense that they may correspond to unbounded control actions to keep the system within the barrier. To place constraints on the possible control action, we can use the following modified condition (see [41]):

$$\left(\sup_{\mathbf{x} \in X} \inf_{\mathbf{u} \in U} L_f B(\mathbf{x}) + L_g B(\mathbf{x})\mathbf{u} \right) < 0 \tag{5}$$

In either formulation, if we assume that the CBF $B(\mathbf{x})$ has a polynomial form, and if the functions $g(\mathbf{x})$ and $f(\mathbf{x})$ are also polynomials, and if U and X are semi-algebraic sets, then we can use Sum-of-Squares (SoS) optimization techniques to obtain a CBF [27–29].

There is little work on finding a CBF if the system dynamics are non-polynomial. We can use an SMT-based approach to finding CBFs. The intuition is that finding the CBF reduces to an *exists-for all* synthesis problem that can be solved using a counterexample-guided inductive synthesis procedure such as the ones used in EF-SMT solvers [8, 18]. We also note that in certain cases, it may be reasonable to assume that the CBF is provided by the user. The reason is that as the CBF is fundamentally connected with the safety and controllability of the system, it is possible that the user has sufficient domain knowledge to provide a candidate invariant set (or a CBF). Given such a set, we propose to check if the supplied set is indeed a CBF using dReal [10].

Reinforcement Learning for Safe Controllers. The procedure in Algorithm 1 upon terminates finds a DNN controller of a given architecture that is guaranteed to keep the system within the invariant described by the supplied control barrier function C. In Line 2, we randomly sample the permissible weight/bias space of the neural network to initialize its weights w and biases b. We assume a function μ that characterizes a certain cost that the user may have specified for the given RL algorithm[1] that maps a given initial state to a cost incurred from the behavior from that initial state[2]. We use the function Φ_{sim} to denote a numerical simulation of the ODE represented by Eq. (1) over suitable time points in the interval $[0, T]$. In other words, it represents a discrete sampling the trajectory function Φ at chosen time-points in $[0, T]$.

In Line 4, we modify the cost function μ to $\tilde{\mu}$, by adding the value cbf_penalty(x_{init}) that assigns a penalty for violating the given barrier condition. The exact form of the penalty function is discussed later in this section. We then use a black-box optimizer to search over the weights (W) and biases (B) to find weights $w^{\star}_{\text{iter}}$ and biases $b^{\star}_{\text{iter}}$ that minimize the effective cost $\tilde{\mu}$. During each cost function evaluation (i.e., during simulations from each initial condition in X) the weights and biases of the NN controller are fixed. The next step is to verify whether the system containing the controller with the optimal weights/biases identified in iteration iter, $\eta(w^{\star}_{\text{iter}}, b^{\star}_{\text{iter}})$, is safe. To check for safety, we check whether the system satisfies the third barrier certificate condition C (as specified in Definition 2.2)[3]. If the condition is satisfied, then we terminate the loop with a verified controller; if not, we get a counterexample, i.e., we find a point cex such that $B(cex) = 0$ and the Lie derivative of $B(\mathbf{x})$ or $L_f B(cex) + L_g B(cex) \cdot \eta(cex)$, is positive. In Line 8, we add this counterexample to the set of initial conditions X.

Remark 3.1 (Monotonicity of training). *We note that adding the counterexample state to the set X causes the training to be* monotonic*, i.e. new counterexamples will continue to account for previous counterexamples. This is because the function picks the maximum penalty incurred from* any *initial state in the set X. Thus, for the penalty function to have a low value, the penalties evaluated on* all *initial states in X are required to be low. This ensures that the controller* does not degrade *w.r.t. the safety requirement across iterations.*

Penalty Functions. A crucial element in Algorithm 1 is the penalty function cbf_penalty that maps an initial state x_{init} to some real value. The key requirement for a penalty function is that it should make it very expensive for the controller to violate a safety requirement. The idea of penalty functions is inspired by two methods in constrained optimization: the eponymous *penalty function method*, and the *barrier function method*[4]. The main idea is that when trying to minimize a function $q(\mathbf{x})$, subject to a constraint $r(\mathbf{x}) < 0$, we can instead try to minimize a function of the form $q(\mathbf{x}) + h(r(\mathbf{x}))$, where $h(r(\mathbf{x})) \geq 0$, and $|h(r(\mathbf{x}))| >> |q(\mathbf{x})|$ if $r(\mathbf{x}) \geq 0$, and $h(r(\mathbf{x})) \leq |q(\mathbf{x})|$ if $r(\mathbf{x}) < 0$. Below we discuss some penalty functions that we have explored in the context of barrier certificates.

State-based Maximal penalty function. Consider function max_penalty mapping an initial state to a penalty in $\mathbb{R}$:

$$\left(1 + \max\left(0, \underline{L_f B(x_{\text{init}}) + L_g B(x_{\text{init}})\eta(x_{\text{init}})}\right)\right)^{2\ell} - 1 \tag{6}$$

Here, ℓ is a positive integer. If the barrier condition is violated (i.e., when the underlined term is positive) we get a quantity that is large and positive. If the underlined term is negative, then the above function returns 0. The magnitude of the positive quantity is dependent on the integer ℓ picked to exceed any user-supplied performance cost function μ.

Exponential trajectory-based cost Function. Typically, cost functions quantifying performance that are used to train DNN controllers in an RL procedure compute a suitable cost on a given system trajectory. We explore a similar cost function that approximates the Lie derivative of the barrier function using finite differences. Recall that $\Phi_{\text{sim}}(x_{\text{init}}, \Delta)$

[1]In RL parlance, we would use the term *reward*, instead of cost, but these are interchangeable by changing their sign.
[2]Note that here we define μ as a cost for the entire trajectory, and effectively equally penalizing actions taken from all states in the trajectory; this effectively becomes a state-based reward if we only perform single-step simulations.
[3]We do not check the first two conditions, as the chosen CBF guarantees that C is initialized and safe.
[4]The use of the word barrier is coincidental, and does not have a direct relation with barrier certificates.

denotes the state value at time Δ. Consider the following cost function:

$$\text{exp\_penalty}(x_{\text{init}}) = e^{B(\Phi(x_{\text{init}},\Delta))+(\Delta-1)B(x_{\text{init}})} \tag{7}$$

The above function is inspired by the notion of *zeroing barrier functions* from [41]. Here, the authors require that $\dot{B}(\mathbf{x}) < \alpha(B(\mathbf{x}))$, where α is a class κ function, meaning it is strictly increasing and zero at the origin[5]. If we consider α to be the identity function and consider a finite differences approximation of $\dot{B}(\mathbf{x})$, then the following should hold:

$$\frac{B(\Phi(x_{\text{init}},\Delta)) - B(\mathbf{x})}{\Delta} < B(x_{\text{init}}).$$

Rearranging and capturing this constraint as an *exponential penalty function* [24], we arrive at (7).

Remark 3.2. *We remark that our procedure has two distinct phases: the penalty-function driven search for a DNN controller, and the verification phase. The first sub-routine can be combined with any generic model-free deep RL technique. The second sub-routine requires the knowledge of a system model. In this paper, we assume a precise dynamical model of the system. We observe that it is possible to extend this to a dynamical model with nondeterministic uncertainties. It is also interesting to consider extensions to a system with stochastic uncertainties, where the proof strategies would be different. For example in [5, 6, 9], the authors assume a Gaussian Process based dynamical model and derive guarantees on the resulting RL procedure using Lyapunov theory and barrier certificates (in a similar vein to work in this paper).*

Checking barrier conditions. Finally, we note that to check if the barrier condition is valid for the chosen controller, it is enough to show the unsatisfiability of the following query:

$$(B(\mathbf{x}) = 0) \ \wedge \ ((L_f B(\mathbf{x}) + L_g B(\mathbf{x}) \cdot \eta(\mathbf{x})) > -\varepsilon) \tag{8}$$

In the above query, ε is a tiny small positive number. If the above query is unsatisfiable, it implies that for all $\mathbf{x}$ s.t. $B(\mathbf{x}) = 0$, the Lie derivative $\dot{B}(\mathbf{x})$ is negative by a robust margin of ε, thereby proving the barrier condition. In order to perform this check, we use the nonlinear SMT solver dReal [10]. This is a δ-sat SMT solver, meaning that if it returns the answer unsat, then the query is truly unsatisfiable, otherwise it returns a hyperbox of size δ that may contain a counterexample. We use the center of this hyperbox as a counterexample in Algorithm 1.

Remark 3.3 (Extension to discrete-time dynamical systems). *While our discussion in this paper has been centered on continuous-time dynamical systems, practical control systems are often better modeled as discrete-time dynamical systems*[6]*. We can use arguments similar to those for barrier certificates for discrete-time dynamical systems. The key difference is that instead of checking a condition on the Lie derivative of the barrier function $B(\mathbf{x})$, we directly check whether the zero level set of the barrier function is an invariant set. In other words, we check for the validity of the following condition:*

$$\forall \mathbf{x} : (B(\mathbf{x}) \leq 0) \implies (B(f(\mathbf{x}) + g(\mathbf{x})\eta(\mathbf{x})) \leq 0) \tag{9}$$

We can check the validity of the above condition by checking for the unsatisfiability of its negation as before.

4 EXPERIMENTAL RESULTS

We have implemented the procedure described in Algorithm 1 in a Matlab® tool called Miyagi. Miyagi contains a customized neural network library, an interface with Matlab® for performing numerical simulations, and an interface

[5]Note that the formulation provided here is modified from [41]. It includes a sign change, as [41] considers barrier functions that *increase* over time, as compared to our barriers, which decrease.

[6]In other words, we use the recurrence equations $\mathbf{x}(n+1) = f(\mathbf{x}(n)) + g(\mathbf{x}(n))k(\mathbf{x}(n))$.

to dReal to perform verification of the barrier conditions. We have benchmarked the performance of the tool on three example systems, which we describe below. For each example, we summarize the results of training in Table 1. As the first example has 2 states, we illustrate the training process on the first example.

Dubins' Car Model. The Dubins car is a model of a 4-wheeled robot moving in a plane. The states are x and y position and θ, which is the angle of the front wheels with respect to the car body. An NN controller is used to steer the car along a given path. The purpose of the controller is to cause the car to follow the given fixed reference trajectory. We follow the steps from [37] to put the example in the form required by Eq. (1). As described in [38], for a constant input trajectory, the system dynamics can be described in terms of the difference between the desired trajectory and the actual trajectory.

$$\begin{bmatrix} \dot{d}_{err} \\ \dot{\theta}_{err} \end{bmatrix} = \begin{bmatrix} V \sin(\theta_{err}) \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix} u$$

Safety Requirement and Controller Setup. We defined a set of possible initial conditions for the system to be $X_{\text{init}} = [-1.0, 1.0] \times [-\pi/16, \pi/16]$. We also define the set of *unsafe* system states X_{unsafe} to be the complement of the set $[-5.0, 5.0] \times [-\pi/2, \pi/2]$. We select a single layer DNN with 10 neurons in the hidden layer with an activation function that is mathematically equivalent to the hyperbolic tangent or $\tanh$ activation function. We use the Matlab® implementation for this function (known as tansig), where $\text{tansig}(x) = \frac{2}{1+e^{-2x}} - 1$. We choose a function $B(\mathbf{x}) = \mathbf{x}^T P \mathbf{x} - c$ as the desired barrier function, where we select a P matrix and a constant c in a way that ensures that the resulting function is a control barrier function. Thus, we ensure that for the set of initial states, $B(\mathbf{x}) \leq 0$ and for unsafe states, $B(\mathbf{x}) > 0$, and that the CBF condition in Def. 3.4 holds for the chosen P matrix and c. We repeat this example with multiple configurations for the DNN controller, to show the flexibility of our approach; the results are summarized in Table 1.

Academic 3D Model. Next, we consider the following nonlinear dynamical system with 3 states in the plant:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \end{bmatrix} = \begin{bmatrix} x_3 + 8x_2 \\ -x_2 + x_3 \\ -x_3 - x_1^2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u$$

Safety Requirement and Controller Setup. The initial set of the system requires each state $x_i \in [-0.2, 0.2]$, and the unsafe set is the complement of the set $[-2, 2]^3$. The chosen DNN has one hidden layer and 10 neurons, with the rectified linear unit (ReLU) activation function (i.e. $\sigma(a) = \max(a, 0)$). We designed a control barrier function using techniques from control theory. We first considered the linear approximation of the system around the equilibrium point, and obtained a Lyapunov function for these linear dynamics. Then, we verified that the level set of this Lyapunov function is a valid CBF. We omit the actual CBF used for brevity.

Bicycle Steering Model. Finally, we consider an example (obtained from [31]) describing a controller to balance a bicycle by modulating the turning angle of its steering handle. The system dynamics are described as follows:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \end{bmatrix} = \begin{bmatrix} x_2 \\ \frac{m\ell}{J}(g \sin(x_1) + \frac{v^2}{b} \cos(x_1) \tan(x_3)) \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{am\ell v}{Jb} \frac{\cos(x_1)}{\cos^2(x_3)} \\ 1 \end{bmatrix} \mathbf{u}$$

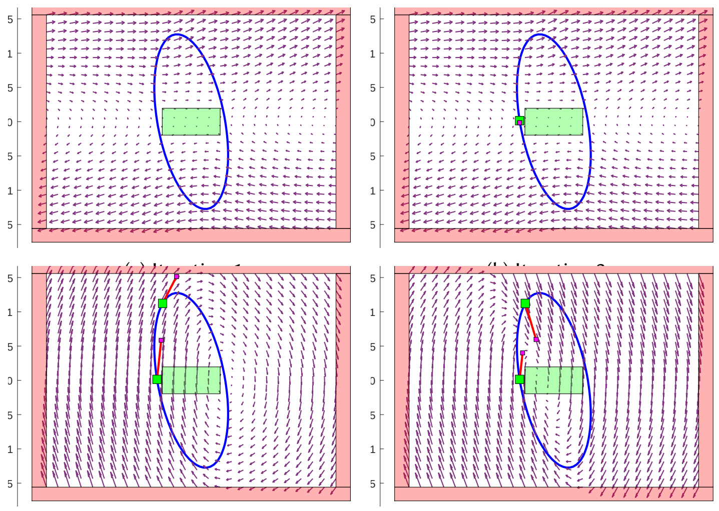
(c) Iteration 3 (d) Iteration 4

Fig. 2. SMT-driven training example. In each figure, we denote the vector field of the closed-loop system using the trained DNN controller in that iteration. The red (resp. green) region represents unsafe (resp. initial) states. The blue ellipse is the selected CBF. The figures show the behavior of the closed loop system at the start of each iteration (i.e., the first figure is for the untrained DNN). Red traces indicate counterexamples provided by dReal in each iteration. Note that in each iteration, the training causes the controller to successfully "adjust" the vector field so that trajectories that left the barrier in the previous iteration are now directed inside the barrier.

In the above, equations the states x_1, x_2 and x_3 respectively denote the tilt angle of the bicycle, the angular velocity with which the bicycle tilts, and the current angle made by the handle bars with the bicycle body respectively. The control input u is the angular velocity applied at the handle bars. The constants used in the above equations are $m = 20$kg (mass), $\ell = 1$m (bicycle height), $b = 1$m (wheel base), $J = \frac{1}{3}mb^2$ (moment of inertia), $v = 10$ m/s (constant linear velocity for the bike), and $g = 10$ m/s^2 (acceleration due to gravity).

Safety Requirement and Controller Setup. We require the bicycle to always remain within $\pm\frac{\pi}{3}$ units for each of the states, and assume that the initial state set is $\pm\frac{\pi}{30}$ units. We select a DNN with two hidden layers, each containing 10 neurons. We choose a barrier certificate that we prove to be a CBF using dReal. The chosen CBF is of the form $B(\mathbf{x}) = \mathbf{x}^T P\mathbf{x} - 1$, where P = [0.722 -0.1528 -0.1250; -0.2778 0.8472 -0.1250; -0.2778 0.1528 0.5694].

The above results table indicates that our technique for training DNN controllers scales well with the number of hidden layers and neurons in each layer. The runtime is dominated by the time required for each dReal query. The technique has an exponential dependence on the number of state variables. At this time, the technique is unable to successfully verify a DNN controller for a nonlinear dynamical system with 4 states (modeling a problem for balancing a pole on a moving cart). The bottleneck for the procedure is the scalability of dReal, which is likely to improve as the nonlinear constraint solving technology matures.

Remark 4.1. *There are a few other hyper-parameters in our implementation that are tunable. We use the particle swarm optimizer in Matlab® to find the weights and biases, and the number of function evaluations, initial swarm size, and number of parallel threads to use (if any) are all hyper-parameters. In our experiments, we did some basic tuning of these parameters*

Model	Act.	L/d	$\|w\| + \|b\|$	Runtime (seconds)		Iter.
				dReal	Optimizer	
Dubins'	tansig	1/10	41	7.6	2.7	2
	tansig	2/5	51	8.7	4.2	2
	tansig	3/5	81	13.4	8.9	4
	ReLu	1/10	41	7.6	3.1	2
Academic	ReLu	1/10	51	307	1152	37
	tansig	2/5	56	192	1553	47
Bicycle	tansig	2/10	161	699	175	35

Table 1. Results on Verification-in-the-loop Reinforcement Learning. The column Act. denotes the activation function, L and d respectively denote the number of layers and neurons per layer, $|w| + |b|$ denotes the search dimension for the optimizer, which is the total number of DNN parameters (weights and biases). The number of iterations denotes the total number of macro verification-in-the-loop iterations required for Algorithm 1 to find a verified controller.

but did not try to find the best hyper-parameters to obtain optimal results. Another hyper-parameter is the robustness margin used for dReal queries, which we fix to 10^{-4} *for all examples. We used the exponential penalty function for the Dubins' car experiments using* tansig *activation, while we used the maximal penalty function for all other experiments.*

5 CONCLUSIONS AND FUTURE WORK

Related Work. In our paper, we use the general theory of CBFs developed in these papers to guide our verification-in-the-loop RL procedure. The basic theory of CBFs has been developed in previous work [40], and CBFs have been used to develop traditional controllers in various CPS applications including bipedal robots, drone swarms and autonomous driving [2, 26, 33, 41]. Recent work on safety in RL has been addressed using different approaches across research communities such as statistical learning formal methods and control theory. In the learning area, research has focused on variance reduction for maximizing rewards and encoding of domain-specific safety constraints into RL [3, 11, 32]. Formal methods approaches have explored the use of temporal logics like LTL to specify safety objectives, and (model-based) RL techniques to satisfy these objectives [15, 16, 30, 39].

Another related area is that of reward shaping, where designers try to manually identify state-based reward functions. In safety-critical systems, reward shaping is an important problem as ill-crafted rewards can lead to reward hacking, i.e., where the RL algorithm learns a controller that performs unsafe or unrealistic actions, even though it maximizes the expected total reward [4, 21]. Reward shaping [13], seeks to minimize reward hacking, and has been addressed using ideas like inverse reinforcement learning (IRL) [1], potential-based reward shaping [25], or combinations of the above [34]. These techniques require demonstrations of desired behavior from the user or domain-specific insights, which may not always be available. In contrast, our design of the penalty function is derived directly from safety conditions and results in better reward shaping for safety-critical applications.

Finally, we remark that in earlier work by some of the co-authors, we investigated a counterexample-guided procedure to find a barrier certificate for a given DNN controller [37], whereas in this paper, we fix the (control) barrier certificate, and find a safe DNN controller.

Conclusion. We present a verification-in-the-loop RL procedure to learn DNN controllers that guarantee safety of a given dynamical system. The key idea in this paper is to use control barrier functions (CBFs) to *a priori* construct a safety argument for the system, and then use the CBF to impose penalties on undesired behaviors when learning the controller. We use nonlinear satisfiability solving to produce either proofs of safety or counterexamples to further guide our RL algorithm, that guarantees that upon termination the resultant control is provably safe.

REFERENCES

[1] Pieter Abbeel and Andrew Y. Ng. Apprenticeship Learning via Inverse Reinforcement Learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, ICML '04, New York, NY, USA, 2004. ACM.

[2] Aaron D Ames, Xiangru Xu, Jessy W Grizzle, and Paulo Tabuada. Control barrier function based quadratic programs for safety critical systems. *IEEE Trans. Aut. Control*, 62(8):3861–3876, 2016.

[3] Haitham Bou Ammar, Rasul Tutunov, and Eric Eaton. Safe policy search for lifelong reinforcement learning with sublinear regret. In *ICML*, pages 2361–2369, 2015.

[4] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

[5] Felix Berkenkamp and Angela P Schoellig. Safe and robust learning control with gaussian processes. In *2015 European Control Conference (ECC)*, pages 2496–2501. IEEE, 2015.

[6] Felix Berkenkamp, Matteo Turchetta, Angela Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. In *Advances in neural information processing systems*, pages 908–918, 2017.

[7] Dimitri P. Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA, 1995.

[8] Chih-Hong Cheng, Saddek Bensalem, Harald Ruess, Natarajan Shankar, and Ashish Tiwari. Efsmt: A logical framework for the design of cyber-physical systems. *Cyber-Physical System Architectures and Design Methodologies (CPSArch)*, 2014.

[9] Richard Cheng, Gábor Orosz, Richard M Murray, and Joel W Burdick. End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks. In *Proc. of AAAI*, 2019.

[10] Sicun Gao, Soonho Kong, and Edmund M Clarke. dreal: An smt solver for nonlinear theories over the reals. In *International Conference on Automated Deduction*, pages 208–214. Springer, 2013.

[11] Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[13] Marek Grześ. Reward shaping in episodic reinforcement learning. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '17, pages 565–573, Richland, SC, 2017. International Foundation for Autonomous Agents and Multiagent Systems.

[14] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[15] Ernst Moritz Hahn, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi, and Dominik Wojtczak. Omega-regular objectives in model-free reinforcement learning. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 395–412. Springer, 2019.

[16] Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Logically-constrained reinforcement learning. *arXiv preprint arXiv:1801.08099*, 2018.

[17] Hassan K Khalil. Nonlinear systems. 2014.

[18] Soonho Kong, Armando Solar-Lezama, and Sicun Gao. Delta-decision procedures for exists-forall problems over the reals. In *International Conference on Computer Aided Verification*, pages 219–235. Springer, 2018.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[20] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[21] Jan Leike, Miljan Martic, Victoria Krakovna, Pedro A Ortega, Tom Everitt, Andrew Lefrancq, Laurent Orseau, and Shane Legg. AI Safety gridworlds. *arXiv preprint arXiv:1711.09883*, 2017.

[22] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[23] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[24] S.G. Nash and A. Sofer. *Linear and Nonlinear Programming*. McGraw-Hill series in industrial engineering and management science. McGraw-Hill, 1996.

[25] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pages 278–287, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[26] Petter Nilsson and Aaron D Ames. Barrier functions: Bridging the gap between planning from specifications and safety-critical control. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 765–772, 2018.

[27] Antonis Papachristodoulou and Stephen Prajna. Analysis of non-polynomial systems using the sum of squares decomposition. In *Positive polynomials in control*, pages 23–43. Springer, 2005.

[28] Antonis Papachristodoulou and Stephen Prajna. A tutorial on sum of squares techniques for systems analysis. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 2686–2700. IEEE, 2005.

[29] Stephen Prajna, Antonis Papachristodoulou, and Pablo A Parrilo. Introducing sostools: A general purpose sum of squares programming solver. In *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, volume 1, pages 741–746. IEEE, 2002.

[30] Dorsa Sadigh, Eric S Kim, Samuel Coogan, S Shankar Sastry, and Sanjit A Seshia. A learning based approach to control synthesis of markov decision processes for linear temporal logic specifications. In *53rd IEEE Conference on Decision and Control*, pages 1091–1096. IEEE, 2014.

[31] Henning Schmidt, Karl Henrik Johannson, Krister Jacobsson, Bo Wahlberg, Per Hägg, and Elling W. Jacobsen. *EL2620: Nonlinear Control*. KTH Electrical Engineering, September 2012.

[32] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*, 2016.

[33] Eric Squires, Pietro Pierpaoli, and Magnus Egerstedt. Constructive barrier certificates with applications to fixed-wing aircraft collision avoidance. In *2018 IEEE Conference on Control Technology and Applications (CCTA)*, pages 1656–1661. IEEE, 2018.

[34] Halit Bener Suay, Tim Brys, Matthew E. Taylor, and Sonia Chernova. Learning from Demonstration for Shaping Through Inverse Reinforcement Learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, AAMAS '16, pages 429–437, Richland, SC, 2016. International Foundation for Autonomous Agents and Multiagent Systems.

[35] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[36] Lei Tai, Jingwei Zhang, Ming Liu, Joschka Boedecker, and Wolfram Burgard. A survey of deep network solutions for learning control in robotics: From reinforcement to imitation. *arXiv preprint arXiv:1612.07139*, 2016.

[37] Cumhur Erkan Tuncali, Hisahiro Ito, James Kapinski, and Jyotirmoy V Deshmukh. Reasoning about safety of learning-enabled components in autonomous cyber-physical systems. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2018.

[38] Cumhur Erkan Tuncali, James Kapinski, Hisahiro Ito, and Jyotirmoy V Deshmukh. Reasoning about safety of learning-enabled components in autonomous cyber-physical systems. In *Proc. of the Design Automation Conference*, 2018.

[39] Min Wen, Rüdiger Ehlers, and Ufuk Topcu. Correct-by-synthesis reinforcement learning with temporal logic constraints. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4983–4990. IEEE, 2015.

[40] Peter Wieland and Frank Allgöwer. Constructive safety using control barrier functions. *IFAC Proceedings Volumes*, 40(12):462–467, 2007.

[41] Xiangru Xu, Paulo Tabuada, Jessy W. Grizzle, and Aaron D. Ames. Robustness of control barrier functions for safety critical control. *IFAC-PapersOnLine*, 48(27):54 – 61, 2015. Analysis and Design of Hybrid Systems ADHS.

[42] Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.