NeuralExplorer: State Space Exploration of Closed Loop Control Systems Using Neural Networks

Manish Goyal and Parasara Sridhar Duggirala
{manishg,psd}@cs.unc.edu

Department of Computer Science, University of North Carolina at Chapel Hill

Abstract. In this paper, we propose a framework for performing state space exploration of closed loop control systems. Our approach involves approximating sensitivity and a newly introduced notion of inverse sensitivity by a neural network. We show how the approximation of sensitivity and inverse sensitivity can be used for computing estimates of the reachable set. We then outline algorithms for performing state space exploration by generating trajectories that reach a neighborhood. We demonstrate the effectiveness of our approach by applying it not only to standard linear and nonlinear dynamical systems, but also to nonlinear hybrid systems and also neural network based feedback control systems.

Keywords: State space exploration, Sensitivity, Inverse Sensitivity, Neural Networks, Testing, Approximation, Falsification

1 Introduction

Advances in hardware and software have made it easier to integrate sophisticated control algorithms in embedded devices. While such control algorithms might improve the performance of the system, they make the task of verification and validation very challenging. In a typical work flow, after deploying the control algorithm, the control designer generates a few test cases and checks if the specification is satisfied. Given that the state space is continuous and the dynamics are often nonlinear, finding the trajectory that violates the specification is similar to searching for a needle in a haystack. For example, consider a regulation application where the output of the control system is required to eventually converge to a set point s within the error threshold of δ. Therefore, the output should remain in the interval $[s - \delta, s + \delta]$ after a specified settling time. The control designer would first test the control algorithm by generating a test suite. If all of the executions in the test suite satisfy the specification, the control designer would like to generate test cases that are close to violating the specification. Given the nonlinearity of the dynamics, the control designer does not have a method to generate the next test input that results in a higher value of error than observed in the test suite.

In some instances, the designer can encode the property as a temporal logic formula and use off-the-shelf falsification tools for generating a trajectory that violates the specification. Such an approach has a few drawbacks. First, falsification tools are geared towards finding a trajectory that violates the given specification, not necessarily to help the control designer in state space exploration. Second, if the specification (error threshold δ or the settling time) is changed, the results from the falsification analysis are no longer useful. Finally, the falsification tools require the specification to be provided in a temporal logic such as signal temporal logic or metric temporal logic. While such specification might be useful in the verification and deployment phase, they are a hindrance during the design and exploration phase. Currently there are no tools that aid control designer in performing systematic testing of closed loop control systems.

In this paper, we present *NeuralExplorer*, a technique for performing state space exploration of closed loop control systems using neural networks. NeuralExplorer can supplement the testing procedure by helping the designer generate test cases that reach a target or a neighborhood. The artifact that helps us in this endeavor is *sensitivity*. Informally, sensitivity of a closed loop system is the change in the trajectory of the system as a result of perturbing the initial condition. The backward time notion of sensitivity is called *inverse sensitivity*. Given a sample set of trajectories, we train a neural network to approximate the sensitivity and inverse sensitivity functions. These neural networks are then used to generate a trajectory (or trajectories) that reaches a destination (or a neighborhood around it).

Our framework has three primary advantages. First, since NeuralExplorer relies only on the sample test cases, it does not require a model of the system and can be applied to a black-box systems. Second, since sensitivity is a fundamental property of the closed loop system, approximating it using a neural network is generalizable to trajectories that are beyond the test cases generated by the control designer. Third, a control designer can develop intuition about the convergence and divergence of trajectories by querying the neural network. In evaluating our framework, we were not only able to perform state space exploration for standard linear and nonlinear dynamical systems, but also for nonlinear hybrid systems and neural network based feedback control systems. We believe that NeuralExplorer is useful for generating corner cases and supplements some of the existing testing and reachable set computation procedures.

2 Related Work

Reachability analysis is often employed for proving the safety specification of safety critical control system [11, 4]. Some of the recent works in this domain are SpaceEx [22], Flow* [9], CORA [3] and HyLAA [7]. These techniques use a symbolic representation for the reachable set of states. While these are useful for proving that the safety specification is satisfied, generating counterexamples using reachability analysis is still an area of research [24].

For generating trajectories that violate a given safety specification, falsification techniques are widely applied [19, 15]. In these techniques, the required specification is specified in a temporal logic such as Metric Temporal Logic (MTL) [31] or Signal Temporal Logic (STL) [35, 32]. Given a specification, falsification techniques generate several sample trajectories and use various heuristics [38, 2, 43, 48, 12, 23] for generating trajectories that violate the specification. Prominent tools in this domain include S-Taliro [5] and Breach [13].

Bridging falsification and reachability are simulation driven verification methods [14, 16, 28, 20]. These methods compute an upper bound on the sensitivity of the trajectories and compute an overapproximation of the reachable set using sample trajectories. While these techniques bridge the gap between falsification and verification, they suffer from curse of dimensionality. That is, the number of trajectories generated might increase exponentially with the dimensions. C2E2 [17], and DryVR [21] are some of the well known tools in this domain.

Given the rich history of application of neural networks in control [36, 33, 37] and the recent advances in software and hardware platforms, neural networks are now being deployed in various control tasks. As a result, many verification techniques are now being developed for neural network based control systems [30, 47, 44, 18]. Additionally, techniques for verification of neural networks deployed in other domains have been proposed [46, 45, 27].

In this paper, we use neural networks to approximate an underlying property of sensitivity and inverse sensitivity. We believe that this is a valid approach because recently, many neural network based frameworks for learning the dynamics or their properties have been proposed [39, 40, 34, 42, 8, 41].

3 Preliminaries

We denote the elements of the state space as x to be elements in $\mathbb{R}^n$. Vectors are denoted as v. We denote the dynamics of the plant as

$$\dot{x} = f(x, u) \tag{1}$$

Where x is the state space of the system that evolves in $\mathbb{R}^n$ and u is the input space in $\mathbb{R}^m$.

Definition 1 (Unique Trajectory Feedback Functions). *A feedback function $u = g(x)$ is said to be unique trajectory feedback function if the closed loop system $\dot{x} = f(x, g(x))$ is guaranteed existence and uniqueness of the solution for the initial value problem for all initial points $x_0 \in \mathbb{R}^n$.*

Notice that for a feedback function to give a unique trajectory feedback, it need not be differentiable. From the sufficient conditions of ODE solutions, it is sufficient if $g(x)$ is continuous and is lipschitz.

Definition 2 (Trajectories of Closed Loop System). *Given a unique trajectory feedback function $u = g(x)$, a trajectory of closed loop system $\dot{x} = f(x, g(x))$, denoted as $\xi_g(x_0, t)$ ($t \geq 0$), is the solution of the initial value problem of the differential equation $\dot{x} = f(x, g(x))$ with initial condition x_0. We often drop the feedback function g when it is clear from the context.*

We extend the notion of trajectory to include backward time trajectories as well. Given $t > 0$, the backward time trajectory $\xi_g(x_0, -t) = x$ such that $\xi_g(x, t) = x_0$. We denote backward time trajectory as $\xi^{-1}(x_0, t)$.

Given $x_0, x_1 \in \mathbb{R}^n$ and $t > 0$ such that $\xi(x_0, t) = x_1$, then $\xi^{-1}(x_1, t) = x_0$. It is trivial to observe that $\xi^{-1}(\xi(x_0, t), t) = x_0$.

Definition 3 (Sensitivity of Trajectories). *Given an initial state x_0, vector v, and time t, the sensitivity of the trajectories, denoted as $\Phi(x_0, v, t)$ is defined as.*

$$\Phi(x_0, v, t) = \xi(x_0 + v, t) - \xi(x_0, t).$$

Informally, sensitivity is the vector difference between the trajectories starting from x_0 and $x_0 + v$ after time t. We extend the definition of sensitivity to backward time trajectories as

$$\Phi^{-1}(x_0, v, t) = \xi^{-1}(x_0 + v, t) - \xi^{-1}(x_0, t).$$

We call $\Phi^{-1}(x_0, v, t)$ as inverse sensitivity function. Informally, inverse sensitivity function gives us the perturbation of the initial condition that is required to displace the trajectory passing through x_0 by v. Observe that $\xi(\xi^{-1}(x_0, t) + \Phi^{-1}(x_0, v, t), t) = x_0 + v$.

For general nonlinear differential equations, analytic representation of the trajectories of the ODEs need not exist. If the closed loop system is a smooth function, then the infinite series for the trajectories is given as

$$\xi(x_0, t) = x_0 + \mathcal{L}_f(x_0)t + \mathcal{L}_f^2(x_0)\frac{t^2}{2!} + \mathcal{L}_f^3(x_0)\frac{t^3}{3!} + \dots. \quad (2)$$

Where $\mathcal{L}_f^i$ is the i^{th} order Lie-derivative over the field $f(x, g(x))$ at the state x_0. Hence, one can write the sensitivity function as

$$\Phi(x_0, v, t) = v + (\mathcal{L}_f(x_0 + v) - \mathcal{L}_f(x_0))t + (\mathcal{L}_f^2(x_0 + v) - \mathcal{L}_f^2(x_0))\frac{t^2}{2!} + \dots. \quad (3)$$

$\Phi^{-1}(x_0, v, t)$ is obtained by substituting $-f$ for f in Equation 3. When the closed loop dynamics is linear, i.e., $\dot{x} = Ax$, it is easy to observe that $\Phi(x_0, v, t) = e^{At}v$, $\Phi^{-1}(x_0, v, t) = e^{-At}v$ where e^{At} (e^{-At}) is the matrix exponential of the matrix At ($-At$). Observe that for linear systems, the inverse sensitivity function is independent of the state x_0. For nonlinear dynamical systems, one can truncate the infinite series up to a specific order and obtain an approximation. However, for hybrid systems that have state based mode switches, or for feedback functions where the closed loop dynamics is not smooth or is discontinuous, such an infinite series expansion is hard to compute. The central idea in this paper is to approximate Φ and Φ^{-1} using a neural network and perform state space exploration using such neural networks.

4 Neural Network Approximations of Sensitivity and Inverse Sensitivity

Given a domain of operation $D \subseteq \mathbb{R}^n$, one can generate a finite set of trajectories for testing the system operation in D. Often, these trajectories are generated using numerical ODE solvers which return trajectories sampled at a regular time step. For approximating sensitivity and inverse sensitivity, we generate a finite number of time bounded trajectories where the step size, time bound, and the number of trajectories are specified by the user. The trajectories can be generated either according to a probability distribution specified by the user or from specific initial configurations provided by her.

Given a sampling of a trajectory at regular time interval with step size h, i.e., $\xi(x_0, 0), \xi(x_0, h), \xi(x_0, 2h), \ldots, \xi(x_0, kh)$, we make two observations. First, any prefix or suffix of this sequence is also a trajectory, albeit, of a shorter duration. Hence, from a given set of trajectories, one can generate more *virtual trajectories* that have shorter duration. Second, given two trajectories (real or virtual) starting from initial states x_1 and x_2, ($x_1 \neq x_2$), We have the two following observations.

$$\Phi(x_1, x_2 - x_1, t) = \xi(x_2, t) - \xi(x_1, t) \tag{4}$$

$$\Phi^{-1}(\xi(x_1, t), \xi(x_2, t) - \xi(x_1, t), t) = x_2 - x_1. \tag{5}$$

Given an initial set of trajectories, we generate virtual trajectories and use Equations 4 and 5 for generating all tuples $\langle x_0, v, t, v_{sen}\rangle$ and $\langle x_0, v, t, v_{isen}\rangle$ such that $v_{sen} = \Phi(x_0, v, t)$ and $v_{isen} = \Phi^{-1}(x_0, v, t)$. This data is then used for training and evaluation of the neural network to approximate the sensitivity and inverse sensitivity functions. We denote these networks as NN_{Φ} and $NN_{\Phi^{-1}}$ respectively.

4.1 Evaluation on Standard Benchmarks

For approximating the sensitivity and inverse sensitivity functions, we pick a standard set of benchmarks consisting of nonlinear dynamical systems, hybrid systems, and control systems with neural network feedback functions. Most of the benchmarks are taken from standard hybrid systems benchmark suite [1, 29, 6]. The benchmarks `Brussellator`, `Lotka`, `Jetengine`, `Buckling`, `Vanderpol`, `Lacoperon`, `Roesseler`, `Steam`, `Lorentz`, and `Coupled vanderpol` are continuous nonlinear systems, where `Lorentz` and `Roesseler` are *chaotic* as well. `SmoothHybrid-Oscillator` and `HybridOscillator` are nonlinear hybrid systems. The remaining benchmarks `Mountain Car` and `Quadrotor` are selected from [30], where the state feedback controller is given in the from of neural network.

For each benchmark, we generated a given number (typically 30 or 50) of trajectories, where the step size for ODE solvers and the time bound are provided by the user. We do not know how much data is required to obtain a required amount of accuracy. The trade offs between the amount of data required, training time, and accuracy of the approximation is a subject of future research. The data used for training the neural network is collected as described in previous subsection. We use 90% of the data for training and 10% for testing.

We used Python Multilayer Perceptron implemented in `Keras` [10] library with Tensorflow as the backend. The network has 8 layers with each layer having 512 neurons. The optimizer used is stochastic gradient descent. The network is trained using Levenberg-Marquardt backpropagation algorithm optimizing the mean absolute error loss function, and the Nguyen-Widrow initialization.

The activation function used to train the network is **relu** for all benchmarks except `Mountain car` for which **sigmoid** performs better because the NN controller is sigmoid-based. Note that the choice of hyper-parameters such as number of layers and neurons, the loss and activation functions is empirical, and is motivated by our prior work [25]. We evaluate the network performance using root mean square error (MSE) and mean relative error (MRE) metrics. The training and evaluation are performed on a system running Ubuntu 18.04 with a 2.20GHz Intel Core i7-8750H CPU with 12 cores and 32 GB RAM. The network training time, MSE and MRE for learning inverse sensitivity function are given in Table 1. The reader is addressed to [26] for the training performance of the neural network tasked with learning sensitivity function.

Benchmark		Dims	Step size (sec)	Time bound	Training Time (min)	MSE	MRE
Continuous Nonlinear Dynamics	Brussellator	2	0.01	500	67.0	1.01	0.29
	Buckling	2	0.01	500	42.0	0.59	0.17
	Lotka	2	0.01	500	40.0	0.50	0.13
	Jetengine	2	0.01	300	34.0	1.002	0.26
	Vanderpol	2	0.01	500	45.50	0.23	0.23
	Lacoperon	2	0.2	500	110.0	1.8	0.46
	Roesseler	3	0.02	500	115.0	0.44	0.07
	Lorentz	3	0.01	500	67.0	0.48	0.08
	Steam	3	0.01	500	58.0	0.13	0.057
	C-Vanderpol	4	0.01	500	75.0	0.34	0.16
Hybrid/ NN Systems	HybridOsc.	2	0.01	1000	77.0	0.31	0.077
	SmoothOsc.	2	0.01	1000	77.5	0.23	0.063
	Mountain Car	2	-	100	10.0	0.005	0.70
	Quadrotor	6	0.01	120	25.0	0.0011	0.16

Table 1: **Learning inverse sensitivity function.** Parameters and performance of neural network tasked with learning inverse sensitivity function. The set of benchmarks includes nonlinear dynamical, hybrid and neural network based feedback control systems. Time bound is number of steps for which the system simulation is computed.

Benchmark	Dims	Iteration count = 1			Iteration count = 5		
		d_a	d_r	Time (ms)	d_a	d_r	Time (ms)
Brussellator	2	[0.19 - 1.87]	[0.23 - 0.74]	11.38	[0.003- 0.22]	[0.01 - 0.12]	31.34
Buckling	2	[1.67 - 11.52	[0.17 - 0.45]	13.61	[0.36- 2.09]	[0.06 - 0.31]	34.51
Lotka	2	[0.08 - 0.24]	[0.21 - 0.45]	12.38	[0.02 - 0.07]	[0.09 - 0.22]	34.28
Jetengine	2	[0.05 -0.20]	[0.19 - 0.28]	15.96	[0.0004 - 0.05]	[0.006 - 0.14]	38.26
Vanderpol	2	[0.29 - 0.58]	[0.16 - 0.66]	12.34	[0.03 - 0.18]	[0.04 - 0.16]	34.02
Lacoperon	2	[0.03 - 0.13]	[0.12 - 0.28]	17.18	[0.003 - 0.03]	[0.02 - 0.16]	37.34
Roesseler	3	[0.72 - 2.02]	[0.20 - 0.34]	16.08	[0.21 - 0.63]	[0.06 - 0.14]	38.26
Lorentz	3	[1.24 - 5.60]	[0.29 - 0.58]	24.72	[0.20 - 0.70]	[0.05 - 0.17]	60.18
Steam	3	[1.59 - 5.21]	[0.31 - 0.67]	8.68	[0.41 - 1.8]	[0.08 - 0.30]	69.80
C-Vanderpol	4	[0.87 - 1.72]	[0.34 - 0.60]	17.44	[0.20 - 0.40]	[0.07 - 0.18]	44.86
HybridOsc.	2	[0.28 - 0.92]	[0.13 - 0.29]	16.70	[0.03 - 0.31]	[0.01 - 0.10]	45.82
SmoothOsc.	2	[0.37 - 1.09]	[0.13- 0.23]	52.22	[0.04 - 0.42]	[0.02 - 0.18]	136.72
Mountain Car	2	[0.004 - 0.24]	[0.08 - 0.22]	138.90	[0.0002 - 0.005]	[0.03 - 0.12]	266.76
Quadrotor	6	[0.014 -1.09]	[0.10 - 0.67]	284.96	[0.004 - 0.04]	[0.02 - 0.13]	668.78

Table 2: **Evaluations.** The results of reachTarget after iteration count 1 and 5. We compute average absolute distance d_a and relative distance d_r over 500 iterations of our algorithm for each benchmark. Additionally, a range of values is obtained for d_a and d_r by performing the evaluation on 10 different targets.

5 Space Space Exploration Using Neural Network Approximation

In this section, we present various applications in the domain of state space exploration using the neural network approximation of sensitivity and inverse sensitivity. The goal of state space exploration is to search for trajectories that satisfy or violate a given specification. In this paper, we primarily concern ourselves with a safety specification, that is, whether a specific trajectory reaches a set of states labelled as *unsafe*. In order to search for such trajectories, we present four different algorithms that use neural networks that approximate sensitivity and inverse sensitivity. The main reason for providing a variety of such algorithms is to demonstrate the flexibility of the framework and the wide variety of ways in which it can be used. The reader is referred to [26] for additional experimental results of these techniques.

5.1 Reaching a Specified Destination Using Inverse Sensitivity Approximation

In the course of state space exploration, after testing the behavior of the system for a given set of test cases, the control designer might choose to explore the system behavior that reaches a destination or approaches the boundary condition for safe operation. Given a domain of operations D, we assume that the designer provides a desired target state z (with an error threshold of δ) that is reached by a trajectory at time t. Our goal is to generate a trajectory ξ such that $\xi(t)$ visits a state in the δ neighborhood of z.

Our approach for generating the target trajectory is as follows. First, we generate a random trajectory ξ from the initial state x, and compute the difference vector of target state z and $\xi(t)$ (i.e., $z - \xi(t)$). We now use the neural network approximation of the inverse sensitivity function and estimate the perturbation required in the initial set such that the trajectory after time t goes through z (i.e., $NN_{\Phi^{-1}}(\xi(t), z-\xi(t), t)$). Since the neural network can only approximate the inverse sensitivity function, the trajectory after the perturbation (i.e., $x + NN_{\Phi^{-1}}(\xi(t), z - \xi(t), t)$) need not visit δ neighborhood of the destination. However, we can repeat the procedure until a threshold on the number of iterations is reached or the δ threshold is satisfied. The pseudo code of this procedure, denoted as reachTarget, is given in Algorithm 1.1

In Algorithm 1.1, $\xi(x, .)$ is the simulation generated by ξ for the state x; x_0 is the simulation state at time t; $v^{-1} \triangleq \Phi^{-1}(x_0, v, t)$ is the inverse sensitivity function which is learned using neural network $NN_{\Phi-1}$. The *absolute distance* d_a is the euclidean distance between simulation state at time t and the destination z. d_r is the *relative distance* with respect to the initial absolute distance d_{init}. Since v^{-1} is an estimate of the perturbation required in the initial set, a new anchor trajectory with initial state $x + v^{-1}$ is generated and the new distance d_a between $\xi(x, t)$ and z is computed. The **while** loop runs until either δ threshold is reached or iteration count I is exhausted.

Evaluation of reachTarget on Standard Benchmarks We evaluate the performance of reachTarget algorithm by picking a random target state z in the domain of interest and let it generate a trajectory that goes through the neighborhood ($\delta = 0.01$ or 0.001) of the target at a specified time t. We use random target states in order to evaluate the performance of the search procedure in the entire domain and not bias it to a specific

input : System simulator ξ, time bound T, trained neural network $NN_{\Phi^{-1}}$, time instance $t \leq T$, destination state $z \in D$, iteration count I, initial set θ, and threshold δ

output: State $x \in \theta$, d_r, $d_a \triangleq ||\xi(x,t) - z||_2$ such that $d_a \leq \delta$

1 $x \leftarrow x_{random} \in \theta$; $i \leftarrow 1$;
2 $x_0 \leftarrow \xi(x,t)$; $d_a \leftarrow ||x_0 - z||_2$;
3 $d_{init} \leftarrow d_a$; $d_r \leftarrow 1$;
4 **while** $(d_a > \delta)$ & $(i \leq I)$ **do**
5 $\quad v \leftarrow x_0 - z$;
6 $\quad v^{-1} \leftarrow NN_{\Phi^{-1}}(x_0, v, t)$;
7 $\quad x \leftarrow x + v^{-1}$; $x_0 \leftarrow \xi(x,t)$;
8 $\quad d_a \leftarrow ||x_0 - z||_2$; $i \leftarrow i + 1$;
9 **end**
10 $d_r \leftarrow \frac{d_a}{d_{init}}$;
11 **return** (x, d_r, d_a);

Algorithm 1.1: reachTarget. Finding an initial state from which the simulation goes within δ-neighborhood of destination z at time t. $\|\cdot\|_2$ is the $l2$-norm. The algorithm returns best candidate for the falsifying initial state, absolute distance d_a, and relative distance d_r wrt initial d_a.

sub-space. Typically, reachTarget executes the loop in lines 4-9 for 10 times before reaching the target. In Table 2, we present the relative and absolute distance between the target and the state reached by the trajectory generated by reachTarget after one or five iterations of the loop. The demonstration of the procedure is shown in Fig. 1.

We now discuss a few variations of our algorithm and their evaluation approaches. *Uncertainty in time:* The control designer might not be interested in reaching the target at a precise time instance as long as it lies within a bounded interval of time. In such cases, one can iterate reachTarget for every step in this interval and generate a trajectory that approaches the target. Consider the designer is interested in finding the maximum distance (or, height) the car can go to on the left hill in `Mountain Car`. By providing an ordered list of target states and a time interval, she can obtain the maximum distance as well the time instance at which it achieves the maxima. If there is no state in the given initial set from which the car can go to a particular target, the approach, as a side effect, can also provide a suitable initial candidate that takes the car as close as possible to that target. Similarly, in `Quadrotor`, one can find an initial state from which the system can go to a particular location during a given time interval.
Generalization: Based on our `Mountain Car` experiment, we observed that, for the given initial set, the maximum distance the car can achieve on the left hill is approx. 1.17. However, even after expanding the initial set from [-0.55, -0.45][0.0, 0.0] to [-0.60, -0.40][0.0, 0.0], our approach finds the maximum achievable distance (1.3019) such that the car can still reach on the top of the right hill (shown in Fig. 2). This shows that our neural network is able to generalize the inverse sensitivity over trajectories that go beyond the test cases considered during the training process.
Discussion: It can be observed from Table 2 that our technique is capable of achieving below 10% relative distance in almost all cases after 5 iterations. That is, the trajec-

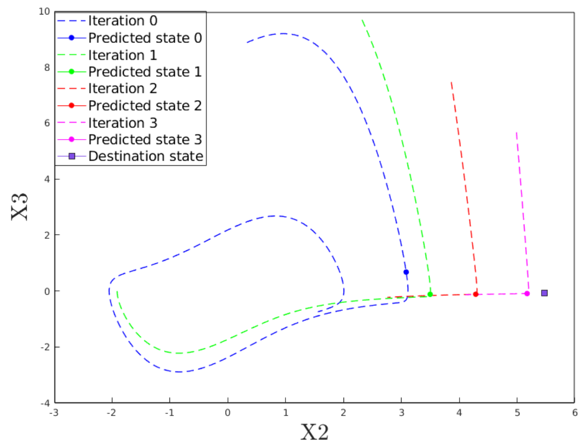


Fig. 1: **Illustration of reachTarget**. We highlight the result of executing reachTarget on `Coupled Vanderpol`. Iteration 0 is the trajectory from x_{random}. Subsequent 3 trajectories are labeled as Iteration 1, 2 and 3 respectively. As shown, with each iteration, the trajectory moves closer to the destination.

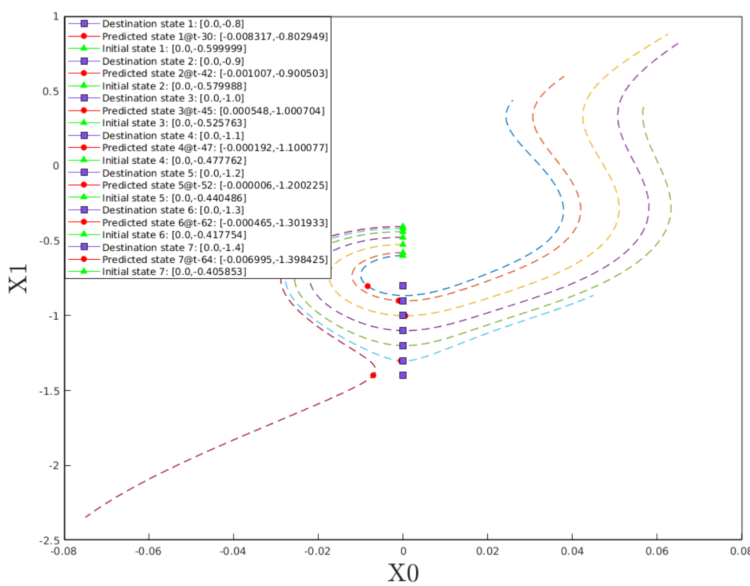
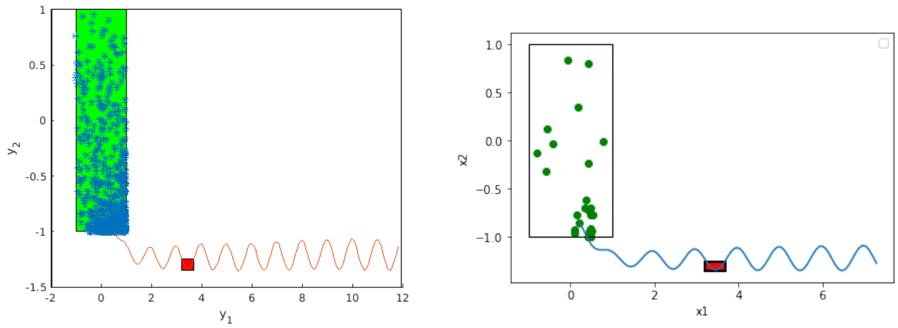


Fig. 2: **Generalization** Computing the maximum distance the car can achieve on the left hill after expanding the initial set.



(a) S-Taliro (b) NeuralExplorer

Fig. 3: **Falsification in `Simulated Annealing` using S-Taliro and NeuralExplorer.** The red box in each of the figures denotes the unsafe set and the other box denotes the initial set. Each of the points in the initial set represents a sample trajectory generated by the falsification engine.

tory generated by reachTarget algorithm after 5 iterations is around 10% away from the target than the initial trajectory. This was the case even for chaotic systems, hybrid systems, and for control systems with neural network components. While training the neural network might be time taking process, the average time for generating new trajectories that approach the target is very fast (less than a second for all cases). The high relative distance in some cases might be due to high dimensionality or large distance to the target which may be reduced further with more iterations.

5.2 Falsification of Safety Specification

One of the widely used methods for performing state space exploration are falsification methods [43, 38]. Here, the specification is provided in some temporal logic such as Signal or Metric Temporal Logic [35, 31]. The falsifier then generates a set of test executions and computes the *robustness* of trajectory with respect to the specification. It then invokes heuristics such as stochastic global optimization for discovering a trajectory that violates the specification.

Given an unsafe set U, we provide a simple algorithm to falsify safety specifications. We generate a fixed number (m) of random states in the unsafe set U. Then, using the reachTarget sub-routine, generate trajectories that reach a vicinity of the randomly generated states in U. We terminate the procedure when we discover an execution that enters the unsafe set U. For `Simulated annealing` benchmark, we compare the number of trajectories generated by S-Taliro with the trajectories generated using inverse sensitivity in Fig. 3. S-Taliro takes 121 sec with quadratic optimization and 11 sec with analytical distance computation, NeuralExplorer obtains a falsifying trajectory in 2.5 sec. Similar performance gain is observed in a few other benchmarks.

Falsification using approximation of inverse sensitivity enjoys a few advantages over other falsification methods. First, since our approach approximates the inverse sensitivity, and we use the reachTarget sub-routine; if the approximation is accurate to a certain degree, each subsequent trajectory generated would make progress towards the destination. Second, if the safety specification is changed slightly, the robustness of the trajectories with respect to new specification and the internal representation for the stochastic optimization solver has to be completely recomputed. However, since our trajectory generation does not rely on computing the robustness for previously generated samples, our algorithm is effective even when the safety specification is modified.

The third and crucial advantage of our approach lies when the falsification tool does not yield a counterexample. In those cases, the typical falsification tools cannot provide any geometric insight into the reason why the property is not satisfied. However, using an approximation of inverse sensitivity, the system designer can envision the required perturbation of the reachable set in order to move the trajectory in a specific direction. This geometric insight would be helpful in understanding why a specific trajectory does not go into the unsafe set.

Considering these advantages, the results demonstrated in Fig. 3 should not be surprising. We also would like to mention that these advantages come at the computational price of training the neural networks to approximating the inverse sensitivity. We observed that in some other examples, S-Taliro terminates with a falsification trajectory faster than our approach. The reasons for such cases and methods to improve falsification using NeuralExplorer are a topic of future work.

5.3 Density Based Search Methods For State Space Exploration

One of the most commonly used technique for performing state space exploration is generation of trajectories from a set of random points generated using an apriori distribution. Based on the proximity of these trajectories to the *unsafe* set, this probability distribution can further be refined to obtain trajectories that move closer to the unsafe set. However, one of the computational bottlenecks for this is the generation of trajectories. Since the numerical ODE solvers are sequential in nature, the refinement procedure for probability distribution is hard to accelerate.

For this purpose, one can use the neural network approximation of sensitivity to *predict* many trajectories in an embarassingly parallel way. Here, a specific set of initial states for the trajectories are generated using a pre-determined distribution. Only a few of the corresponding trajectories for the initial states are generated using numerical ODE solvers. These are called as *anchor trajectories*. The remainder of trajectories are not generated, but rather predicted using the neural network approximation of sensitivity and anchor trajectories. That is, $\xi(x_i, t) + \Phi_{NN}(x_i, x_j - x_i, t)$. Additionally, the designer has the freedom to choose only a subset of the initial states for only a specific time interval for prediction and refine the probability distribution for generating new states. This would also allow us to specifically focus on a time interval or a trajectory without generating the prefix of it. An example of predictive trajectory generation for performing *reachability analysis* on `Vanderpol` oscillator is provided in Fig. 4.

5.4 Density Based Search for Falsification

Similar to the inverse sensitivity based falsification, one can use the density based search space method for generating trajectories that reach a destination and violate a safety

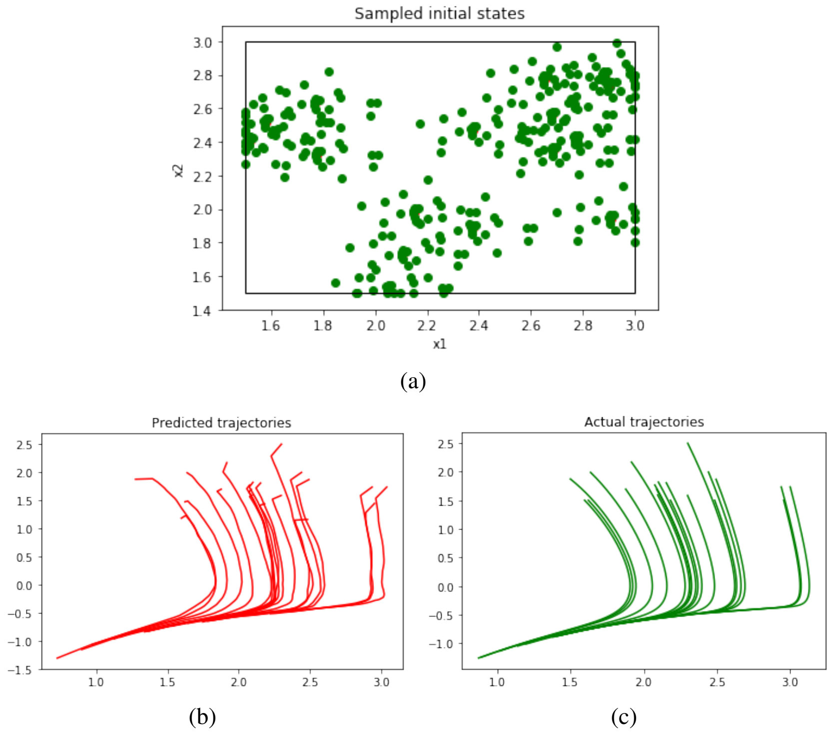
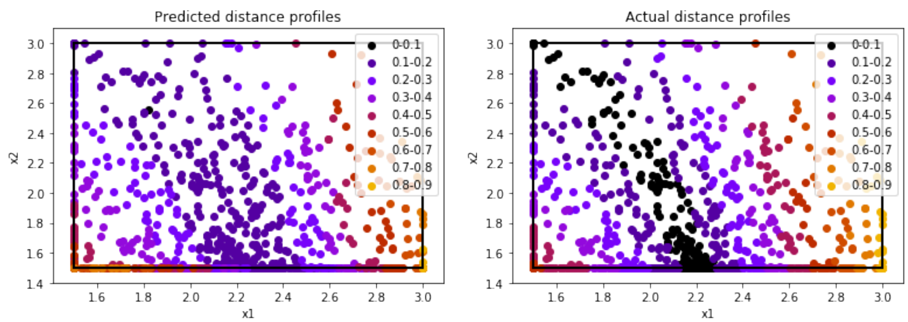


Fig. 4: **State space exploration on `Vanderpol`**. A cluster of points is sampled in the neighborhood of a reference state. Actual trajectories as well as predicted trajectories obtained by the neural network which approximates sensitivity function are shown.

specification. The forward density based search procedure would work as follows. First, an anchor trajectory is generated and time instances of this trajectory that are closer to the unsafe set are identified. Then a set of new initial states are generated according to an apriori decided distribution. Instead of generating the trajectories from these initial states using ODE solvers, the *predicted trajectories* using the anchor trajectory and neural network approximation of sensitivity are generated specifically for the time intervals of interest. Then, the initial state with the predicted trajectory that is closest to the unsafe set is chosen and a new anchor trajectory from the selected initial state is generated. This process of generating anchor trajectory, new distribution of initial states is continued until you reach within the given threshold around the unsafe set. Demonstration of this procedure for `Vanderpol` system is shown in Fig. 5. Notice that this approach gives an underlying intuition about the geometric behavior of neighboring trajectories.

A similar method for density based estimation using inverse sensitivity approximation can also be devised. Instead of sampling the initial set, the density based method for inverse sensitivity generates random states around the unsafe set to be reached and then, using reachTarget, explores states in the initial set that reach these unsafe configurations at a particular time instance. In addition, it maintains the distance of each trajectory from the unsafe set. In this manner, one can classify states in the initial set based on their respective trajectories' distances to the unsafe set. This results into a density map that can provide some geometric insights about initial configurations. An example of such a density map generated is given in Fig. 6. In Fig 6(a), the trajectories starting from the states in the bottom left side of the initial set either go into the unsafe set or are much closer to it compared to the states in the upper right side. Also, observe how the density map changes by changing the unsafe specification in Fig. 6(b).



(a) (b)

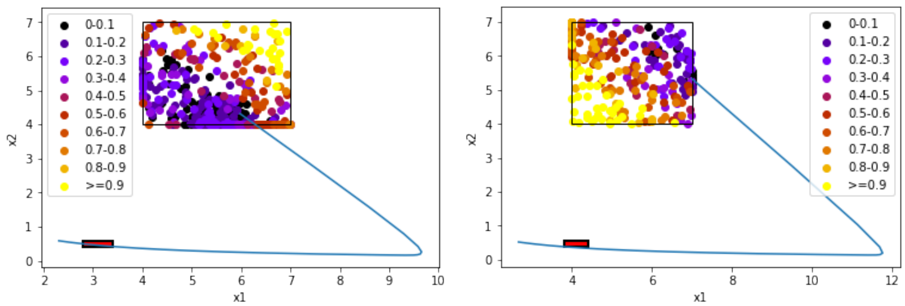
Fig. 5: **Density based search for *falsification* using sensitivity in `Vanderpol`.** The perturbation in the neighborhood of reference state are greedily chosen in an iterative manner so as to minimize the distance to unsafe state. The sampled states are classified based on their euclidean distance to the unsafe state.

6 Conclusion and Future Work

We presented NeuralExplorer framework for state space exploration of closed loop control systems using neural network. Our framework depends on computing neural network approximations of two key properties of a dynamical system called sensitivity and inverse sensitivity. We have demonstrated that for standard benchmarks, these functions can be learned with less than 20% relative error. We demonstrated that our method can not only be applied to standard nonlinear dynamical systems but also for control systems with neural network as feedback functions.

Using these approximations of sensitivity and inverse sensitivity, we presented new ways to performing state space exploration. We also highlighted the advantages of the falsification methods devised using the approximations. Additionally, we demonstrated that our techniques give a geometric insight into the behavior of the system and provide more intuitive information to the user, unlike earlier black box methods. We believe that these techniques can help the system designer in search of the desired executions. [1]

[1] All the code and examples for the evaluations performed in this section are available at `https://github.com/mag16154/NeuralExplorer`



(a) Brusselator: For original unsafe spec

(b) Brusslator: For a different unsafe spec

Fig. 6: **Density based search for *falsification* using inverse sensitivity in `Brusselator`**. The initial states explored in the falsification process are colored according to their distance to the unsafe set. These color densities help in identifying regions in the initial set potentially useful for falsification. Notice the difference in the color densities as we select a difference unsafe spec.

In future, we intend to extend this work to handle more generic systems such as feedback systems with environmental inputs. We believe such a black-box method for generating adversarial examples can be integrated into generative adversarial training for training neural networks for control applications.

Acknowlegements This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-19-1-0288 and National Science Foundation (NSF) under grant numbers CNS 1739936, 1935724. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force or National Science Foundation.

References

1. Benchmarks of continuous and hybrid systems. `https://ths.rwth-aachen.de/research/projects/hypro/benchmarks-of-continuous-and-hybrid-systems/`, benchmarks of continuous and hybrid systems
2. Abbas, H., Fainekos, G.E.: Linear hybrid system falsification through local search. In: Automated Technology for Verification and Analysis, 9th International Symposium, ATVA 2011, Taipei, Taiwan, October 11-14, 2011. Proceedings. pp. 503–510 (2011)
3. Althoff, M.: An introduction to cora 2015. In: Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems (2015)
4. Alur, R., Dang, T., Ivančić, F.: Progress on reachability analysis of hybrid systems using predicate abstraction. In: Maler, O., Pnueli, A. (eds.) Hybrid Systems: Computation and Control (2003)
5. Annpureddy, Y., Liu, C., Fainekos, G.E., Sankaranarayanan, S.: S-taliro: A tool for temporal logic falsification for hybrid systems (2011)

6. Bak, S., Beg, O.A., Bogomolov, S., Johnson, T.T., Nguyen, L.V., Schilling, C.: Hybrid automata: from verification to implementation. International Journal on Software Tools for Technology Transfer (2019)
7. Bak, S., Duggirala, P.S.: Hylaa: A tool for computing simulation-equivalent reachability for linear systems. In: Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control (2017)
8. Chen, R.T.Q., Rubanova, Y., Bettencourt, J., Duvenaud, D.: Neural ordinary differential equations (2018)
9. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: An analyzer for non-linear hybrid systems. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification (2013)
10. Chollet, F., et al.: Keras. `https://github.com/fchollet/keras` (2015)
11. Dang, T., Maler, O.: Reachability analysis via face lifting. In: Henzinger, T.A., Sastry, S. (eds.) Hybrid Systems: Computation and Control (1998)
12. Deshmukh, J.V., Fainekos, G.E., Kapinski, J., Sankaranarayanan, S., Zutshi, A., Jin, X.: Beyond single shooting: Iterative approaches to falsification. In: American Control Conference, ACC (2015)
13. Donzé, A.: Breach, A toolbox for verification and parameter synthesis of hybrid systems. In: Computer Aided Verification, 22nd International Conference, CAV 2010. Proceedings (2010)
14. Donzé, A., Maler, O.: Systematic simulation using sensitivity analysis. In: International Workshop on Hybrid Systems: Computation and Control (2007)
15. Donzé, A., Maler, O.: Robust satisfaction of temporal logic over real-valued signals. In: Formal Modeling and Analysis of Timed Systems - 8th International Conference, FORMATS 2010. Proceedings (2010)
16. Duggirala, P.S., Mitra, S., Viswanathan, M.: Verification of annotated models from executions. In: Proceedings of the Eleventh ACM International Conference on Embedded Software (2013)
17. Duggirala, P.S., Potok, M., Mitra, S., Viswanathan, M.: C2E2: a tool for verifying annotated hybrid systems. In: HSCC (2015)
18. Dutta, S., Chen, X., Jha, S., Sankaranarayanan, S., Tiwari, A.: Sherlock - A tool for verification of neural network feedback systems: demo abstract. In: HSCC (2019)
19. Fainekos, G.E., Pappas, G.J.: Robustness of temporal logic specifications for continuous-time signals. TCS (2009)
20. Fan, C., Mitra, S.: Bounded verification with on-the-fly discrepancy computation. In: International Symposium on Automated Technology for Verification and Analysis. pp. 446–463 (2015)
21. Fan, C., Qi, B., Mitra, S., Viswanathan, M.: Dryvr: Data-driven verification and compositional reasoning for automotive systems. In: CAV (1) (2017)
22. Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: Spaceex: Scalable verification of hybrid systems. In: Proc. 23rd International Conference on Computer Aided Verification (CAV) (2011)
23. Ghosh, S., Sadigh, D., Nuzzo, P., Raman, V., Donzé, A., Sangiovanni-Vincentelli, A.L., Sastry, S.S., Seshia, S.A.: Diagnosis and repair for synthesis from signal temporal logic specifications. In: Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control (2016)
24. Goyal, M., Duggirala, P.S.: On generating A variety of unsafe counterexamples for linear dynamical systems. In: ADHS (2018)
25. Goyal, M., Duggirala, P.S.: Learning robustness of nonlinear systems using neural networks (2019), `https://sites.google.com/view/dars2019/home`
26. Goyal, M., Duggirala, P.S.: Neuralexplorer: State space exploration of closed loop control systems using neural networks. CoRR **abs/2007.05685** (2020)

27. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: CAV (1) (2017)
28. Huang, Z., Mitra, S.: Proofs from simulations and modular annotations. In: Proceedings of the 17th international conference on Hybrid systems: computation and control (2014)
29. Immler, F., Althoff, M., Chen, X., Fan, C., Frehse, G., Kochdumper, N., Li, Y., Mitra, S., Tomar, M.S., Zamani, M.: ARCH-COMP18 category report: Continuous and hybrid systems with nonlinear dynamics. In: ARCH@ADHS (2018)
30. Ivanov, R., Weimer, J., Alur, R., Pappas, G.J., Lee, I.: Verisig: verifying safety properties of hybrid systems with neural network controllers (2018)
31. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-time systems (1990)
32. Kyriakis, P., Deshmukh, J.V., Bogdan, P.: Specification mining and robust design under uncertainty: A stochastic temporal logic approach. ACM Trans. Embed. Comput. Syst. (2019)
33. Lewis, F., Jagannathan, S., Yesildirak, A.: Neural network control of robot manipulators and non-linear systems. CRC Press (1998)
34. Long, Z., Lu, Y., Ma, X., Dong, B.: PDE-net: Learning PDEs from data. In: Proceedings of the 35th International Conference on Machine Learning. pp. 3208–3216 (2018)
35. Maler, O., Nickovic, D., Pnueli, A.: Checking temporal properties of discrete, timed and continuous behaviors. In: Pillars of computer science (2008)
36. Miller, W., Werbos, P., Sutton, R.: Neural Networks for Control. A Bradford book, MIT Press (1995), `https://books.google.com/books?id=prjMtIr\_yT8C`
37. Moore, K.L.: Iterative learning control for deterministic systems. Springer Science & Business Media (2012)
38. Nghiem, T., Sankaranarayanan, S., Fainekos, G., Ivancić, F., Gupta, A., Pappas, G.J.: Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems. In: Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control (HSCC 2010) (2010)
39. Pan, S., Duraisamy, K.: Long-time predictive modeling of nonlinear dynamical systems using neural networks. Complexity **2018**
40. Pathak, J., Hunt, B., Girvan, M., Lu, Z., Ott, E.: Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach. Phys. Rev. Lett. (2018)
41. Phan, D., Paoletti, N., Zhang, T., Grosu, R., Smolka, S.A., Stoller, S.D.: Neural state classification for hybrid systems. In: Proceedings of the Fifth International Workshop on Symbolic-Numeric Methods for Reasoning About CPS and IoT (2019)
42. Raissi, M., Perdikaris, P., Karniadakis, G.E.: Multistep neural networks for data-driven discovery of nonlinear dynamical systems (2018)
43. Sankaranarayanan, S., Fainekos, G.E.: Falsification of temporal properties of hybrid systems using the cross-entropy method. In: Hybrid Systems: Computation and Control, 2012 (2012)
44. 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 (2019)
45. Sun, Y., Huang, X., Kroening, D., Sharp, J., Hill, M., Ashmore, R.: Structural test coverage criteria for deep neural networks. ACM Trans. Embedded Comput. Syst. (2019)
46. Tjeng, V., Xiao, K.Y., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. In: ICLR (Poster). OpenReview.net (2019)
47. Tran, H.D., Cai, F., Diego, M.L., Musau, P., Johnson, T.T., Koutsoukos, X.: Safety verification of cyber-physical systems with reinforcement learning control. ACM Trans. Embed. Comput. Syst. (2019)
48. Zutshi, A., Deshmukh, J.V., Sankaranarayanan, S., Kapinski, J.: Multiple shooting, cegar-based falsification for hybrid systems. In: Proceedings of the 14th International Conference on Embedded Software (2014)