NeuralExplorer: Data Driven State Space Exploration

Learning for Dynamics & Control (L4DC) 2020

Manish Goyal and Parasara **Sridhar** Duggirala

University of North Carolina at Chapel Hill

June 12, 2020

Outline

1. Introduction
2. Motivation
3. Preliminaries
4. Technique
5. Evaluations
6. Discussion

Introduction

1. Integration of software into control processes has lead to the deployment of sophisticated control algorithms in safety critical scenarios.
2. Improvements in software and hardware platforms for evaluating neural networks have made it easier to integrate them in embedded devices.

Challenge

Increased complexity of software in control tasks makes testing and validation of the closed loop systems very hard.

Introduction

Existing techniques can broadly be categorized as

1. Testing
2. Falsification
3. Reachability analysis
4. Data driven verification

Testing

- Technique for checking whether a given control system satisfies its specification.
- After designing the feedback function, the control designer generates a few test cases to check for their validity.

Not exhaustive

Given that the state space is continuous, finding the trajectory that violates the specification is extremely hard.

Falsification

- Geared towards finding a trajectory that violates the given specification
- Requires the specification to be given in temporal logic
- Software tools: Breach and S-Taliro

Challenges

It does not help in state space exploration and falsification results may not be useful across runs.

Reachability analysis

- Employed for proving safety of a safety critical system
- Uses a symbolic representation of the reachable set
- Exhaustive and provide guarantees by typically using over-approximation
- Software tools: CORA, SpaceEx, Flow*, and HyLaa

Data driven verification

- Bridges falsification and reachability
- Computes an upper bound on the sensitivity of trajectories
- Obtains over-approximation of the reachable set
- Software tools: C2E2 and DryVR

Challenge

Both reachability and data driven verification suffer from the curse of dimensionality as the complexity of the system grows.

Motivation

- Wide adoption of neural networks in various domains
- Availability of huge amount of data
- In many cases, model of a system is either not available or is complex
- Sensitivity of a closed loop system can be approximated using neural networks and used to obtain corner cases

Preliminaries

Plant dynamics

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

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

Unique trajectory feedback function

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 $x_0 \in \mathbb{R}^n$.

Preliminaries

Closed loop system trajectory

Given a unique trajectory feedback function $u = g(x)$, a trajectory of closed loop system $\dot{x} = f(x, g(x))$, denoted as $\xi(x_0, t)$ ($t \geq 0$), is the solution of the initial value problem of the differential equation $\dot{x} = f(x, g(x))$ with initial condition x_0.

Backward time trajectory

Given $t > 0$, the backward time trajectory (denoted as $\xi^{-1}(x_0, t)$) is defined as $\xi(x_0, -t) = x$ such that $\xi(x, t) = x_0$.

Preliminaries

Sensitivity

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

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

Inverse Sensitivity

Given an initial state x_0, vector v, and time t, the inverse sensitivity of the trajectories, denoted as $\Phi^{-1}(x_0, v, t)$ is defined as.

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

Approximating inverse sensitivity

The training of the neural network $NN_{\Phi^{-1}}$ is performed as follows.

- Given a domain of operation $D \subseteq \mathbb{R}^n$, we generate a finite set of trajectories for testing the system operation in D.
- Given two trajectories starting from initial states x_1 and x_2, $(x_1 \neq x_2)$, we have

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

- For the initial set of trajectories, we generate tuples $\langle x_0, v, t, v_{isen} \rangle$ such that $v_{isen} = \Phi^{-1}(x_0, v, t)$.
- These tuples ar used for training and evaluation of neural network $NN_{\Phi^{-1}}$ to approximate Φ^{-1}.

$NN_{\Phi^{-1}}$ Training results

Benchmark		Dims	Training Time (min)	MSE	MRE
Continuous Dynamics	Brussellator	2	67.0	1.31	0.21
	Jetengine	2	82.0	1.48	0.34
	Vanderpol	2	77.50	0.63	0.094
	Lorentz	3	67.0	0.67	0.08
	Steam	3	65.0	0.32	0.045
	Roesseler	3	184.0	0.64	0.062
	C-Vanderpol	4	134.0	0.25	0.04
Hybrid/ NN Systems	HybridOsc.	2	77.0	0.58	0.077
	Mountain Car	2	10.0	5.8e-5	0.70
	Quadrotor	6	25.0	8e-5	0.16

Table 1: Approximating inverse sensitivity. MSE and MRE are mean squared error and mean relative error respectively.

State space exploration

Objective

Given a desired target state z (with an error threshold of ϵ) and time t, the goal is to generate a trajectory ξ using $NN_{\Phi^{-1}}$ such that $\xi(t)$ visits a state in the ϵ neighborhood of the target z.

Algorithm

The approach denoted as `reachTarget` for generating the desired trajectory consists of the following steps.

1. Generate a random trajectory ξ from the initial set θ, and compute the difference vector of target set z and $\xi(t)$.
2. Use the $NN_{\Phi^{-1}}$ to estimate the perturbation required in the initial set such that the trajectory after time t goes through z.
3. Since the neural network can only approximate the inverse sensitivity function, the new trajectory after the perturbation may not visit ϵ neighborhood of z.
4. The procedure is repeated until either a threshold on the number of iteration is reached or ϵ threshold is satisfied.

`reachTarget` Illustration

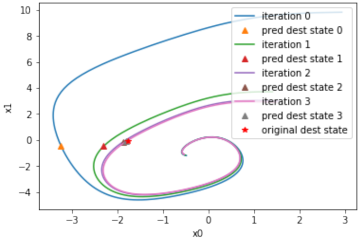

Figure 1: Jetengine

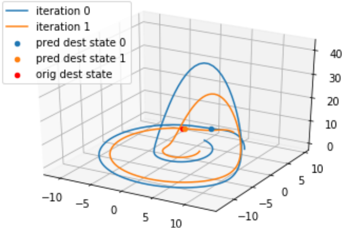

Figure 2: Roesseler

Evaluation results

Benchmark	Iteration count = 1		Iteration count = 5	
	d_a	d_r	d_a	d_r
Brussellator	[0.19 - 1.87]	[0.23 - 0.74]	[0.003 - 0.22]	[0.01 - 0.12]
Jetengine	[0.05 -0.20]	[0.19 - 0.28]	[$4e^{-4}$ - 0.05]	[0.006 - 0.14]
Vanderpol	[0.29 - 0.58]	[0.16 - 0.66]	[0.03 - 0.18]	[0.04 - 0.16]
Lorentz	[1.24 - 5.60]	[0.29 - 0.58]	[0.20 - 0.70]	[0.05 - 0.17]
Steam	[1.59 - 5.21]	[0.31 - 0.67]	[0.41 - 1.8]	[0.08 - 0.30]
Roesseler	[0.72 - 2.02]	[0.20 - 0.34]	[0.21 - 0.63]	[0.06 - 0.14]
C-Vanderpol	[0.87 - 1.72]	[0.34 - 0.60]	[0.20 - 0.40]	[0.07 - 0.18]
HybridOsc.	[0.28 - 0.92]	[0.13 - 0.29]	[0.03 - 0.31]	[0.01 - 0.10]
MountainCar	[$4e^{-3}$ - 0.24]	[0.08 - 0.22]	[$2e^{-4}$ - $5e^{-3}$]	[0.03 - 0.12]
Quadrotor	[0.014 -1.09]	[0.10 - 0.67]	[0.004 - 0.04]	[0.02 - 0.13]

Table 2: Evaluation results of `reachTarget` for iteration count 1 and 5. For each case, we run the experiment 500 times and compute the average absolute distance d_a and relative distance d_r from z.

Multiple targets

Given multiple targets, the trajectories obtained in the proximity of each target are shown.

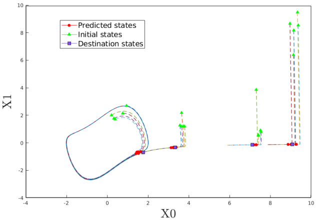

Figure 3: Vanderpol

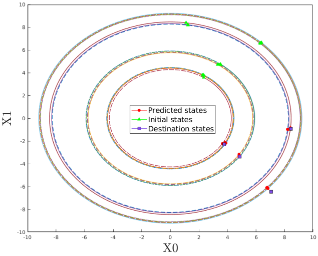

Figure 4: Hybrid Oscillator

Discussion

- Our technique is capable of achieving below 20% relative distance in almost all cases after 5 iterations.
- That is, the trajectory generated by `reachTarget` algorithm after 5 iterations is around 20% away from the target than the initial trajectory.
- The high relative distance in some cases might be due to high dimensionality or large initial distance to the target which may be reduced further with more iterations.
- While training the neural network might be time taking process, the average time for generating new trajectories that approach the target is very fast.

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 the `reachTarget` algorithm for every step in this interval and generate a trajectory that approaches a target.

Uncertainty in time

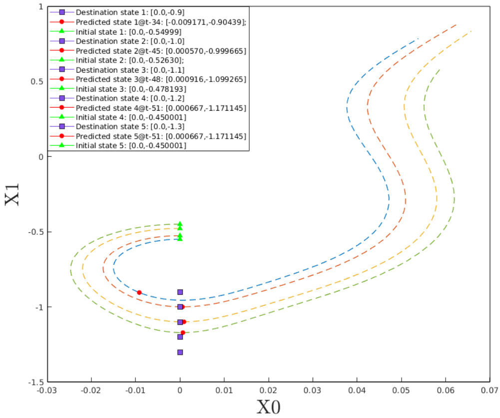

Figure 5: Uncertainty in time illustrated on Mountain Car

Falsification: S-Taliro vs NeuralExplorer

These are some experimental results of performing safety specification violation using our approach and S-Taliro.

- Brusselator

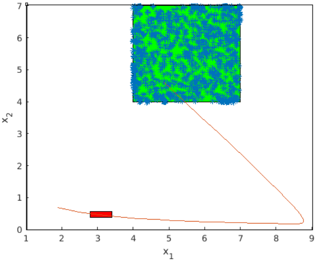

Figure 6: S-Taliro

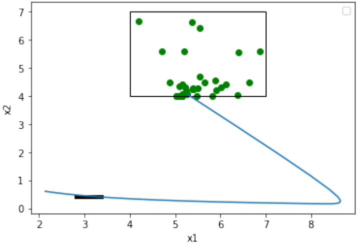

Figure 7: NeuralExplorer

Falsification: S-Taliro vs NeuralExplorer

- Vanderpol

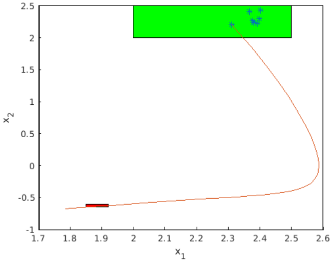

Figure 8: S-Taliro

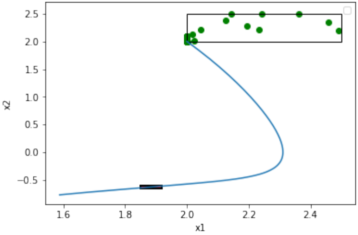

Figure 9: NeuralExplorer

Future Work

- To extend this work to handle more generic systems such as feedback systems with environmental inputs.
- To analyze cases where S-Taliro terminates with a falsification trajectory faster than our approach and find methods to improve falsification using NeuralExplorer.