Learning Robustness of Nonlinear Systems Using Neural Networks

Manish Goyal
manishg@cs.unc.edu

Parasara Sridhar Duggirala
psd@cs.unc.edu

Department of Computer Science
University of North Carolina at Chapel Hill

Abstract

Physical processes are often modelled using nonlinear differential equations. For systems that are not chaotic, we often expect the behaviors to be robust with respect to perturbation of the initial configuration. That is, perturbations in the initial configuration would not contribute to wide deviations in the behavior. In this ongoing work, we intend to learn this robustness called as perturbation function, which represents the effect of perturbation of initial configuration on the state of trajectory, using neural networks. We outline the results of training perturbation functions and its applications in state space exploration and counterexample generation.

1 Introduction

Physical processes are often modelled as differential equations over a state space $\mathcal{X} \subseteq \mathbb{R}^n$. The right hand side of these differential equations is often a nonlinear function of the current state.

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

Solutions of initial value problem of the differential equation (such as given in Equation 1) are called trajectories, denoted as $\xi(x_0, t)$, where, x_0 is the initial state and t is the time. In general, a closed form expression for the trajectories of a differential equation, does not exist. Hence, we often rely on numerical approximations of the trajectories for understanding the behavior of the system starting from an initial configuration x_0.

Safety verification problem of such a system entails answering the following question. Given an initial set of states $\Theta \subset \mathcal{X}$ (often uncountable), bounded time $t > 0$, and unsafe set $U \subset X$, does there exist a trajectory ξ starting from Θ, that visits U within t time. Static analysis techniques such as Flow* [4], CORA [1], d/dt [2] use a symbolic representation of the initial set and compute an artifact called *reachable set* that is an overapproximation of all the states visited by all the trajectories starting from Θ. An alternative set of techniques called dynamic analysis techniques (used in tools Breach [5], C2E2 [6], HyLAA [3]) generate a sample set of trajectories and compute the overapproximation from the sample trajectories.

Reachable set computation using C2E2 relies on *discrepancy function* that describes the divergence of trajectories as a function of the initial state and time. Given a system, a function (such as in Equation 1) $\beta : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is a discrepancy function if and only if

$$\forall x_1, x_2 \in \Theta, |\xi(x_1, t) - \xi(x_2, t)| \leq \beta(x_1, x_2, t). \tag{2}$$

Using this discrepancy function, the sample trajectories of nonlinear systems are bloated to compute an overapproximation of the reachable set.

For systems that have a contraction metric or have an incremental Lyapunov function, one can construct a discrepancy function from them. In some cases, one can use matrix measures for computing the discrepancy functions for complex nonliner dynamics. More recently, for black-box systems, discrepancy functions that provide statistical guarantees can be inferred using machine learning techniques.

In this ongoing work, we intend to learn the robustness of trajectories with respect to perturbations of the initial configuration. We call this function as perturbation function. We believe that this perturbation function is useful in more directed state space exploration for inferring the safety specification.

2 Perturbation Function: Training and Evaluation

Definition 1 *Given a system (such as in Equation 1), perturbation function* $P : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ *is such that, given* $x_0, v \in \mathbb{R}^n$ *and* $t \in \mathbb{R}_{\geq 0}$,

$$P(x_0, v, t) = \xi(x_0 + v, t) - \xi(x_0, t) \tag{3}$$

Intuitively, the perturbation function returns the difference between the trajectories from an initial state x_0 and its perturbation $x_0 + v$ at time t. For linear systems, where $f(x)$ is a linear transformation of the current state, represented as Ax, the closed form solution of the IVP is given as $\xi(x_0, t) = e^{At}x_0$ where $e^{At} = I + \frac{At}{1!} + \frac{(At)^2}{2!} + \frac{(At)^3}{3!} + \dots$ is the matrix exponential. For such systems, the perturbation function $P(x_0, v, t) = e^{At}(x_0 + v) - e^{At}x_0 = e^{At}v$ is independent of the initial state. That is, the effect of perturbation on the trajectories is the same, irrespective of the initial configuration. For nonlinear systems, since the closed form solution might not exist, the perturbation function might not have a closed form representation.

In this ongoing work, we intend to learn the perturbation function from sample trajectories. For this purpose, we use neural networks (NNs). NNs have been very effective functional approximators in the domain of image processing, natural language processing, and predicting chaos. In this work, we evaluate how well the neural networks can approximate the perturbation function.

Training: We select a standard set of benchmarks [1] for training the Neural Networks. Our benchmarks consists of both linear and nonlinear continuous systems and linear hybrid systems. For each system, we generate a fixed number of trajectories (20 in this case) for a desired time bound (1000 steps) with time step 0.01. We use 15 trajectories for training and 5 for evaluation of NN. For all pairs of points x_1 and x_2 in any two trajectories ξ_1 and ξ_2, we train the perturbation function as $\hat{P}(x_1, x_2 - x_1, t) = \xi(x_2, t) - \xi(x_1, t)$.

We use the Python Multilayer Perception Regressor from sklearn library. The solver function used is **adam** which is default and refers to a stochastic gradient-based optimizer. Each layer in a 4-layer network has 100 neurons; whereas, in a 8-layer network, first 4 layers have 100 neurons each and the rest 4 have 50 neurons each. The NN is trained using Levenberg-Marquardt backpropagation algorithm optimizing the mean square error loss function, and the Nguyen-Widrow initialization. The activation functions used to train the network are **tanh**, **sigmoid**, and **relu** respectively. 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 with 4 layers takes total ∼10 minutes for its training and testing, and the total time taken by a network with 8 layers is approximately 20 minutes.

Evaluation: The experimental results are given in Table 1. We evaluate the neural network based on average relative error ($\frac{|P(x,v,t) - \hat{P}(x,v,t)|}{|\hat{P}(x,v,t)|}$). We hope the relative error would be less than 10%, as a rough indication that the estimates are relatively close to the actual values. Although in some cases the relative error is higher than the desired value, the performance for some nonlinear benchmarks such as Vanderpol, Brussellator and Jetengine is less than 5%. These systems are known to be contactring and hence might be easy to learn. Among activation functions, both **sigmoid** and **tanh** outperform **relu** in most cases. Additionally, sigmod performs very well in the case of linear dynamics such as Gravity, Smooth Hybrid Linear Oscillator, and Hybrid Linear Oscillator. It might indicate that a NN with sigmod activation functions might be more suitable for learning matrix exponentials. The experiments also indicate that more layers does not necessarily translate into better performance (at least, not with the same data).

These results show promise that learning *perturbation function* using NNs is a promising direction. In future, we plan to explore other aspects of training NNs such as using CNNs, overfitting, appropriate number of hyper-parameters, etc.

3 Applications and Future Work

We believe that using the approximation of perturbation function, and sample simulations, we can compute a statistical approximation of reachable set for black box systems. Additionally, using the gradients from

[1] https://ths.rwth-aachen.de/research/projects/hypro/benchmarks-of-continuous-and-hybrid-systems/

Benchmark	Simulations = 20					
	tanh		sigmoid		relu	
	4	8	4	8	4	8
Brussellator	0.18697	0.79865	0.12718	1.3	1.17704	**0.11594**
Gravity	0.03921	0.03098	**0.01184**	0.38827	0.03865	0.02092
Lotka	**0.13751**	0.71537	0.43694	1.23553	0.52826	0.41853
Jetengine	**0.20198**	0.29759	0.27452	0.40289	0.63742	1.23015
Vanderpol	0.05516	0.05390	0.14771	0.02216	0.04205	**0.01663**
Lacoperon	0.08206	0.06901	**0.02794**	0.08680	0.04704	0.03994
Regular Oscillator	**0.02696**	0.04208	0.05348	2.73357	0.10722	0.15042
Smooth Hybrid LinearOsc.	0.07679	0.23966	**0.02873**	0.42826	0.09168	0.06982
Roesseler	**0.21178**	0.63130	0.48688	0.44327	0.29933	0.28454
Buckling	0.82935	0.62144	**0.48411**	66.891	0.9082	0.54257
Lorentz	2.30147	2.24524	2.94222	3.94334	**0.93714**	1.01919
Steam	**0.27901**	0.60035	1.13542	2.09367	0.32180	0.58326
Hybrid LinearOsc	0.21142	0.11011	**0.04765**	0.61595	0.26715	0.10176
Spring Pendulum	1.14354	1.88636	0.57823	1.31859	**0.47352**	0.98459

Table 1: **Experiments:** The metrics used for performance is average relative error.

the neural network, we can perform adaptive state space exploration.

This preliminary work leaves a lot of scope for future results. First, we would like to understand the most suitable activation functions for learning matrix exponential linear transformations. We would also like to learn the minimum size of the neural network for achieving a specific performance. We would also like to investigate whether NNs perform inherently better for contracting dynamics as opposed to chaotic dynamics. We are also interested in exploring the appropriate NN architecture and loss functions that are suitable for learning the perturbation function.

References

[1] M. Althoff. An introduction to cora 2015. In *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems*, 2015.

[2] E. Asarin, T. Dang, and O. Maler. The d/dt tool for verification of hybrid systems. In *International Conference on Computer Aided Verification*, 2002.

[3] S. Bak and P. S. Duggirala. 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.

[4] X. Chen, E. Ábrahám, and S. Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *International Conference on Computer Aided Verification*, 2013.

[5] A. Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *International Conference on Computer Aided Verification*, 2010.

[6] P. S. Duggirala, S. Mitra, M. Viswanathan, and M. Potok. C2e2: A verification tool for stateflow models. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2015.