Final Project For ECE228 Track Number # 2

Group Number #25

Matt Taylor
University of California, San Diego
mat028@ucsd.edu

Abstract

Our proposed method simultaneously learns a controller and a Lyapunov function to certify stability of a system without relying on full system dynamics. It is assumed that a linearized solution is known and non-linear dynamics of the system are incorporated by using finite difference from trajectories. We test our solution on the Gymnasium inverted pendulum environment and show that certification with neural networks is still possible even without complete knowledge of full system dynamics. This is a step in the right direction for better control for systems with unknown nonlinear dynamics. [GitHub Repository]

Evaluation

I certify that I have filled the evaluation.

1 Introduction

Traditional control algorithms thrive in real-world engineering applications due to their relative simplicity and theoretical guarantees. Methods such as linear quadratic regulator (LQR) work extremely well when a system has well defined linear dynamics. LQR and other control methods like proportional–integral–derivative (PID) also require tunable parameters. For LQR, this involves tuning the Q and R matrices. Q represent a cost on the system state while R represents the cost associated with controller effort. For PID, this involves tuning the controller gains, K_p, K_i, and K_d. Manual tuning is not much of an issue for systems with small state spaces, however, when a state and input space become large, it becomes increasingly difficult to tune these parameters. In addition, LQR and PID controllers linearize the system around an equilibrium point. Ideally, a more robust controller would capture the nonlinearities in a system while still providing nice theoretical guarantees on stability.

One idea is to use a controller with learnable weights. Deep neural networks are universal function approximators and can be used to approximate complicated non-linear relationships [6]. Unfortunately, "black-box" AI models are plagued with lack of explainability and theoretical guarantees, which is one of the main reasons why these methods still have not been fully adopted in the controls community today.

Our proposed method simultaneously learns a controller and a Lyapunov function to certify stability of the system without relying on the full dynamics of the system. It is assumed that the LQR solution is known and non-linear dynamics of the system are incorporated by using finite difference from trajectories.

Preprint. Under review.

2 Background

A dynamical system can be defined as the change in state with respect to time. Often this is represented with $f_u(x)$ or $f(x, u)$ where x is the current state and u is a controller input.

$$\dot{x} = \frac{\mathrm{d}x}{\mathrm{d}t} = f(x, u) \tag{1}$$

This section only briefly mentions key ideas from control theory. Please see [2] for more details on state space models, LQR, and theoretical stability guarantees.

2.1 State Space Model

A state-space model is a set of ordinary differential equations that can be written by Equation (2). Here, A and B are constant matrices that are found by taking the Jacobian of the dynamics with respect to the state and input evaluated at equilibrium.

$$\dot{x} \approx Ax + Bu \tag{2}$$

It's important to note that the state space model is linear despite most dynamical systems being non-linear in real world engineering applications. As we'll see, methods like LQR can still use this linearized system to provide robust solutions to real world control problems.

2.2 Linear Quadratic Regulator

LQR finds the optimal controller gain k in order to minimize a quadratic loss objective. The loss objective J is shown below in Equation (3). Q represent a tunable cost on the system state while R represents the cost associated with controller effort.

$$J = \int_0^\infty \left(x^T Q x + u^T R u\right) dt \tag{3}$$

The gain, k, influences the controller input itself, where:

$$u = -kx \tag{4}$$

The gain matrix can easily be solved for by minimizing this cost function and solving the Continuous Algebraic Riccati Equations [2]. Because state space model is linear, the theoretical guarantees for the controller gain found by LQR is only valid close to an equilibrium point. In our work, this $-k$ value is used as an initial weight in the linear layer of our neural network for the controller similar to [3].

2.3 Lyapunov Function and Lie Derivative

A Lyapunov function, V, is a scalar function used to prove the stability of a system. The Lyapunov function should always be greater than 0, evaluate to 0 at the equilibrium point, x_0, and have a Lie derivative less than 0. One can think of the Lyapunov function as an energy landscape, where the minimum energy is at the equilibrium point.

The Lie derivative of a Lyapunov function, V, is defined below. It measures the rate of change of V along the direction of the system dynamics [3]. Here, $x^{(j)}$ represents indices for the dimension of the state itself. This is an important distinction from x_i which we define as the i^{th} state sample given as input to the model.

$$\mathrm{L}_f V(x) = \langle \nabla V(x), f \rangle = \sum_{j=1}^{n} \frac{\partial V}{\partial x^{(j)}} \frac{\mathrm{d}x^{(j)}}{\mathrm{d}t} \tag{5}$$

If the Lyapunov function is continuously differentiable and Equation (6) is met for all inputs, then the system is asymptotically stable at the origin and V is a true Lyapunov function [3].

$$V(x_0) = 0 \qquad \text{and} \qquad \forall x \in \mathcal{D} \backslash \{x_0\}, \quad V(x) > 0 \qquad \text{and} \qquad \mathrm{L}_f V(x) < 0 \tag{6}$$

3 Related work

Recent works have learned certificates with neural networks as a way to add theoretical guarantees to control algorithms. The three most common methods for certification are Lyapunov functions, barrier functions, and contraction metrics. Lyapunov functions (used in this work) certify stability of a system, barrier functions certify safety, and contraction metrics certifies a controller's ability to track arbitrary trajectories [4].

A recent work by Boffi et al. [1] learns certificates from fixed controller using trajectory data. Learning from trajectories means no reliance on any system dynamics which is a huge advantage of the paper.

The closest work to ours is by Chang et al. [3] which uses a learner to find the necessary control input and Lyapunov function - and a falsifier which adds counterexamples to the dataset during training. This setup is a great start for certifiable neural networks, however, there are two key assumptions that aren't true for more complicated dynamical systems. The first is that the true Lie derivative is known which requires knowledge of nonlinear system dynamics as shown in Equation (5). Additionally, it is assumed that the neural network controller needs a good initialization. Often this means that the weights for the controller are set to the gain matrix weights found by an LQR solution. This way, the controller can start with a linearized solution and learn more robust nonlinear control while simultaneously learning the lyapunov function. Our work also assumes a linearized initialization, however, we do not assume access to the full nonlinear system dynamics beyond the state space model.

Finally, model-free reinforcement solutions have been proposed which seek to find certificates and a robust controller. Han et al. [5] use actor-critic reinforcement learning to simultaneously learn a controller and Lyapunov function. The advantage of using model-free RL is that it requires no knowledge of system dynamics. They also use finite difference to approximate the lie derivative. Unfortunately, RL is a data hungry algorithm and requires lots of compute. In addition, the models are often sample inefficient. It would be better to start from a linearized solution and learn to represent the nonlinear dynamics given this starting point. Qu et al. attempt to do this by combining model-based and model-free RL [7] and it would be interesting to adapt this work to also learn a certificate.

4 Methodology

4.1 Loss Functions

The driving self-supervised loss function used is Lyapunov risk, which incorporates the criteria mentioned in Equation (6). This is taken directly from Chang et al. [3]. The first term is a penalty on the Lyapunov function itself for a given state x_i. The Lyapunov function must be greater than 0 for all non-equilibrium points, so a ReLU is used to add a penalty for negative function outputs. The second term is a penalty on the Lie derivative of the Lyapunov function. This is also a ReLU that adds a penalty for positive Lie derivatives which lead to unstable trajectories. Notice that $L_f V_\theta$ is the true Lie derivative which requires known dynamics. Our work approximates this in two separate ways which is described in the Approximation Methods section. Finally, the last term is a penalty on the equilibrium point x_0. The Lyapunov function evaluated at the equilibrium point must be 0, so a squared error term us used for a penalty.

$$L_N(\theta, u) = \frac{1}{N} \sum_{i=1}^{N} \left(\max\left(-V_\theta\left(x_i\right), 0\right) + \max\left(0,\ \mathrm{L}_f V_\theta\left(x_i\right)\right) \right) + V_\theta^2(x_0), \tag{7}$$

Another loss function used for regularization of the Lyapunov function's is shape-tuning loss. The shape-tuning loss is defined below, where λ is the maximum of the sampled states.

$$L_S = \frac{1}{N}\sum_{i=1}^{N}\left(\|x_i\|_2 - \lambda V_\theta(x_i)\right)^2 \tag{8}$$

This is not a required loss function, however, it was found that adding a penalty for the squared difference between the ℓ_2 norm of the state and a scaled Lyapunov function helped the Lyapunov function to learn a slightly larger region of attraction.

4.2 Model Architecture

A multi-layer feed forward neural network is used with `tanh` activation functions to learn the Lyapunov function. Tanh activation functions are used over `ReLU` activations because the Lyapunov function's Lie derivative requires a smooth function [3]. A separate linear layer is used for the controller weights which is initialized to the negative controller gain from Equation (4). The model takes in a single state, x and returns both the Lyapunov function scalar output and the predicted controller input required for the given state. It's important to note that there is no requirement on how many layers the model can have. For simplicity we chose a 2 layer neural network for the Gymnasium inverted pendulum environment, however a larger network just of easily could have been chosen.

4.3 Sampling-based Falsifier

A sampling-based falsifier is used to add counterexamples to the training dataset to guide the learner towards more robust solutions. Chang et al. used a falsifier with an SMT solver to completely certify the lyapunov conditions within given state bounds [3]. This is a very strong claim and helps to guide the model towards extremely robust solutions, however, it also requires the full dynamics to be known. We propose a sampling-based falsifier that does not provide as strong of claims as an SMT solver but does still guide the learner to learn robust solutions and avoid overfitting to the sampled training data.

The proposed sampling-based falsifier periodically looks at all points in the training dataset where the Lyapunov function is less than 0 and the Lie derivative is greater than 0 and takes the union between them to found all counterexamples. Once counterexamples are found, each counterexample is uniformly sampled around a small neighborhood k times and the k sampled points are added to the training dataset for each counterexample.

4.4 Approximation Methods

Equation (5) shows how the true lie derivative is calculated. This work assumes that the full nonlinear system dynamics are unknown, and that we only have access to trajectories and the linearized dynamics from the state space model.

Two different methods are proposed to estimate the Lie derivative in the loss function. The first method is called "Approximate Dynamics Loss" which approximates the dynamics of the system using a blend of finite difference and the state space model as shown in Equation (9). α is a tunable parameter for the weighted average between the linearized dynamics and finite difference between the current state x_i and next state x_i'. If $\alpha = 1$, then purely the state space model is used which is only a linear approximation close to equilibrium by construction. If $\alpha = 0$, then purely finite difference is used for the system dynamics which captures non-linearities in the dynamics with the only downside being that it is a noisy estimate.

$$\dot{x}_i \approx \alpha\left(Ax_i + Bu_i\right) + (1-\alpha)\frac{x_i' - x_i}{dt} \tag{9}$$

The true Lie derivative in Equation (5) then has the true dynamics substituted with the formulation in Equation (9). The partial derivative of the Lyapunov function with respect to the state can easily be calculated via backpropagation through the neural network used.

$$\mathrm{L}_f V(x) \approx \sum_{j=1}^{n}\frac{\partial V}{\partial x^{(j)}}\left(\alpha\left(Ax + Bu\right) + (1-\alpha)\frac{x' - x}{dt}\right)^{(j)} \tag{10}$$

The second method proposed is called "approximate Lie loss" which seeks to directly approximate the Lie derivative with finite difference rather than the dynamics of the system. In this method, the Lie derivative is first calculated using only the state space model to approximate system dynamics. A weighted sum is then used between this approximate Lie derivative and finite difference between neural network model outputs as shown in Equation (11). This approximation for the Lie derivative is again fed into the Lyapunov risk formulation in place of the true Lie derivative.

$$\mathrm{L}_f V(x) \approx \alpha \left(\sum_{j=1}^{n} \frac{\partial V}{\partial x^{(j)}} (Ax + Bu)^{(j)} \right) + (1 - \alpha) \left(\frac{V(x') - V(x)}{dt} \right) \tag{11}$$

Similar to the approximate dynamics loss, α is a tunable parameter for the weighted average between the Lie derivative from linearized dynamics and the Lie derivative from finite difference of the model output. α must be a value from 0 to 1.

5 Experiments and Results

The following experiments and results are on the Gymnasium Inverted Pendulum environment. The inverted pendulum only contains two states: the angle of the pendulum θ, and the angular velocity $\dot{\theta}$. The goal is to apply a scalar torque to keep the pendulum in the upright position.

Three separate models were trained for comparison using the true Lyapunov loss, approximate dynamics loss, and approximate lie derivative loss. All methods also use the shape-tuning loss. The "true" Lyapunov model is used as a baseline to compare with the other two proposed methods. It's important to note that the only major difference between the "true" model and the results from Chang et al. [3] is that our model does not use an SMT solver to provide rigorous claims on the entire state space. Instead, we apply our sampling-based falsifier for a closer comparison to the two approximation methods.

Figure 1 below shows two separate loss plots. The plot on the left represents the training loss without the sampling-based falsifier and the plot on the right shows with falsification happening at selected frequencies. The peaks in the loss in the second image are from when counterexamples are added to the training dataset. For the true loss shown in blue, the falsification frequency was chosen to be every 100 epochs and for the approximation methods, the falsification frequency was chosen to be every 150 because the approximations took longer to reach relatively low loss. Early stopping was also used during sampling-based falsification when no counterexamples were found 2 times during training.

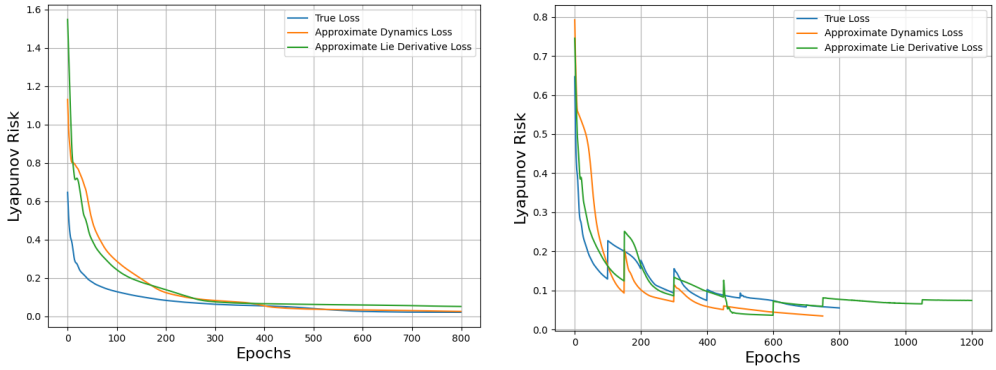

Figure 1: Loss function without sampling-based falsifier (left) and with (right)

It's important to note that while sampling-based falsification did find more robust Lyapunov functions and controllers, it also required tuning to figure out the correct frequency and how many samples to use for each counterexamples. Too many samples and the loss explodes while too few samples means the learner is weakly guided towards more robust solutions.

The learned Lyapunov functions for all three methods are shown below. The model trained with the true Lyapunov risk has the most circular and convex shape which is desirable for the inverted pendulum system. The other two methods (trained with $\alpha = 0.7$) are more oblong in nature but still are convex. It was found that as α decreased below 0.5 (true average) for the approximation methods, the shape of the Lyapunov function became less convex. This is likely because there is more of a reliance on finite difference methods when calculating the Lie derivative which are noisy in nature. An α value > 0.5 seemed to give more stable results.

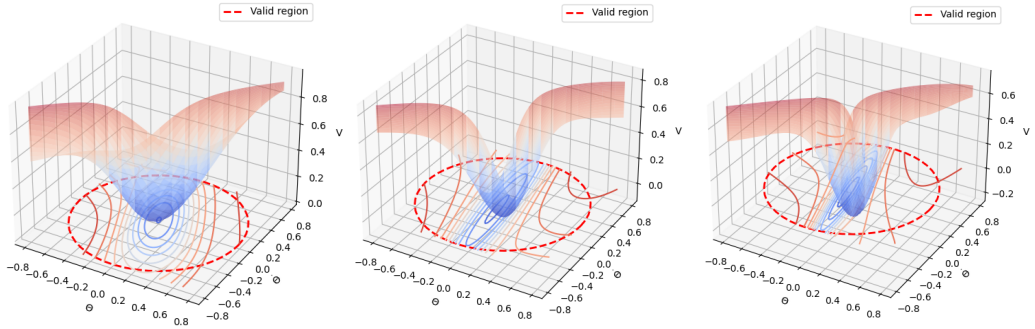

Figure 2: Learned lyapunov functions (true, approximate dynamics, approximate lie)

Figure 3 show an overlay 4 regions of attraction (Lyapunov function level set). The LQR region of attracion is shown in purple and is found with a quadratic representation of the Lyapunov function ($x^T Px$ where P is the solution to the Algebraic Riccati Equations). As expected, the neural network trained with the true Lyapunov risk achieved the largest ROA. Both approximation methods achieved similar regions of attraction and did not perform as well as the true Lyapunov method. This is as expected, because the learned Lyapunov function with these methods is not a true Lyapunov function due to the approximation of the Lie derivatives.

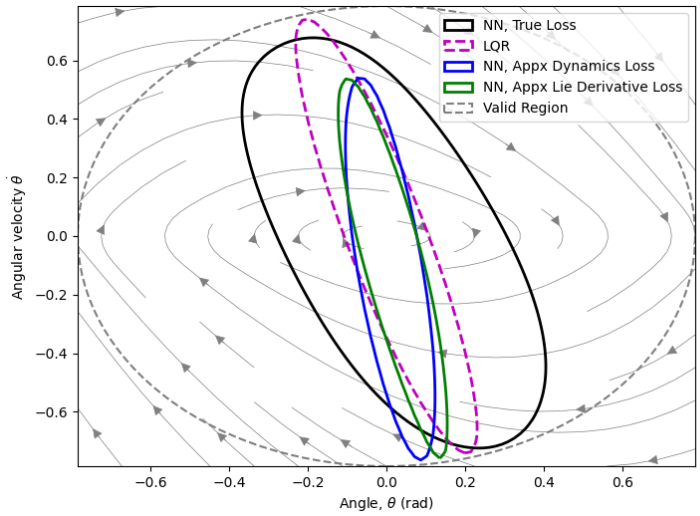

Figure 3: 2D region of attraction comparison over all methods

The Lyapunov level set alone does not fully explain system stability since a sampling-based falsifier was used and both approximation methods do not use the true Lie derivative. Thus, Lie derivatives for random samples enclosed within the level set must be shown to be less than 0 to show that all trajectories point back inside the level set. Figure 4 shows this visualization on the model trained with the true lyapunov loss and sampling based falsifier. As you can see, the Lie derivative is less than 0 for all points inside the level set with the exception of the equilibrium point which is less than a small ϵ value close to 0.

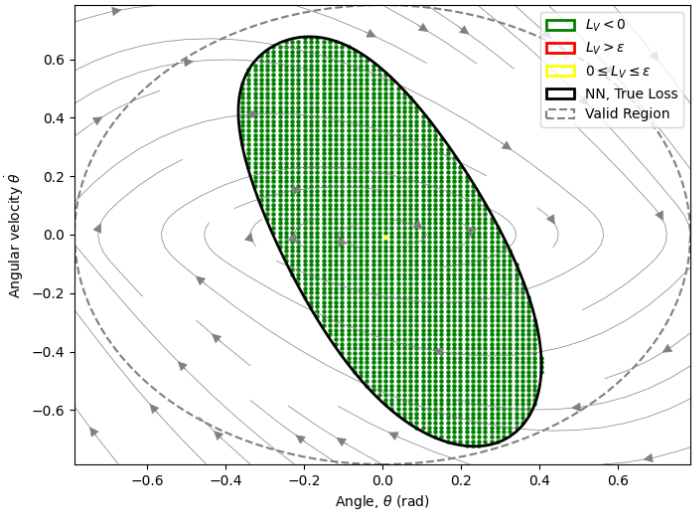

Figure 4: ROA visualization on true loss with lie derivative sampling overlay

Figure 5 shows the lie derivative visualizations for the approximate dynamics and approximate Lie models. As you can see, both of these learned Lyapunov functions also have Lie derivatives less than 0 with the exception of points near equilibrium which are within a small ϵ. While there are no Lie derivative points larger than ϵ in these plots, other training results did show poor results. For example, when $\alpha = 0$ and no state space model was used, a Lyapunov function was incorrectly learned due to the noisiness of finite difference of samples.

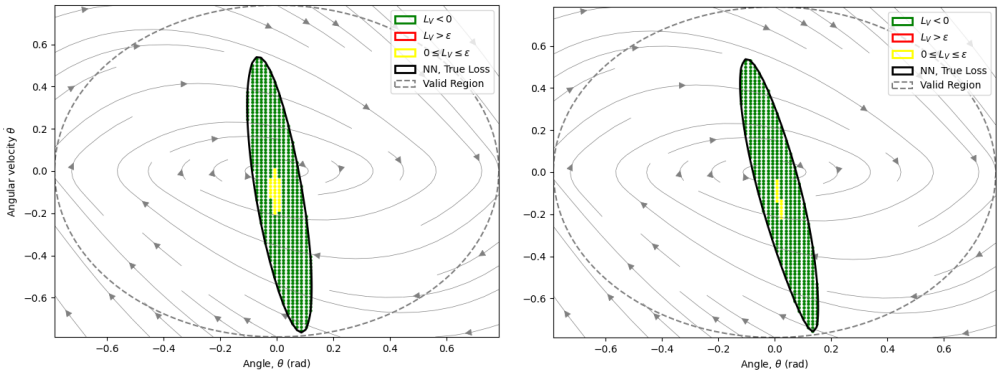

Figure 5: ROA visualization with approximate dynamics (left) and approximate lie (right)

6 Conclusion

A sampling-based falsifier and two separate approximation methods for Lie derivatives were introduced for learning a controller and a Lyapunov function when full system dynamics are not known. A tunable parameter α was used to balance the tradeoff between accurate linearized dynamics close to equilibrium and noisy non-linear dynamics from trajectories. Approximation results were compared to a model trained with the true Lie derivative incorporated in the loss function for comparison.

Future work will look into improving the sampling-based falsifier to find the most important counterexamples during falsification. One limitation of the current falsifier is that counterexamples are naively sampled around a small area without knowledge of previous counterexamples. A sampling-based falsifier that understands the distribution and density of counterexamples could be much more robust than the naive falsifier proposed. Additionally, the proposed method can be tested on other

environments besides Inverted pendulum. One such environment that is in progress is CartPole, which uses four states instead of two.

References

[1] Nicholas M. Boffi et al. "Learning Stability Certificates from Data". In: *CoRR* abs/2008.05952 (2020). arXiv: 2008.05952. URL: https://arxiv.org/abs/2008.05952.

[2] Timothy Bretl. *AE353 Course Notes*. https://tbretl.github.io/ae353-sp22/reference. Accessed: 2024-06-08. 2022.

[3] Ya-Chien Chang, Nima Roohi, and Sicun Gao. *Neural Lyapunov Control*. 2022. arXiv: 2005.00611 [cs.LG].

[4] Charles Dawson, Sicun Gao, and Chuchu Fan. *Safe Control with Learned Certificates: A Survey of Neural Lyapunov, Barrier, and Contraction methods*. 2022. arXiv: 2202.11762 [cs.RO].

[5] Minghao Han et al. "Actor-Critic Reinforcement Learning for Control with Stability Guarantee". In: *CoRR* abs/2004.14288 (2020). arXiv: 2004.14288. URL: https://arxiv.org/abs/2004.14288.

[6] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. URL: https://www.sciencedirect.com/science/article/pii/0893608089900208.

[7] Guannan Qu et al. *Combining Model-Based and Model-Free Methods for Nonlinear Control: A Provably Convergent Policy Gradient Approach*. 2020. arXiv: 2006.07476 [math.OC].