Exploring Model-based Safe Reinforcement Learning and its implementation on High-Dimensional Models

Yu-Wei Wu, Xin Su, Menghan Liu
Department of Electrical Engineering
California Institute of Technology
Pasadena, CA 91106
{ywu7, xssu, mliu2}@caltech.edu

Abstract

In this paper, we mainly focus on the algorithm behind the model-based safe reinforcement learning (RL) and its implementation. Intrigued by the topic we were assigned for the presentation, we examined the feasibility of the safety RL for general LQR-controlled linear system and further explored its implementation on higher dimensional model. The visualization shows that with the safe RL strategy, our agents can remain stable within specific regions and retrieve better restoring ability. However, scalability issue does exist through our implementation on a 4-dimensional inertial wheel pendulum (IWP), which leads to our further discussion.

1 Introduction

1.1 Background

In real world, safety and ability to operate within constraints imposted by the environment are critical prerequisites. In real dynamics, especially robotic system, where systems will often face large uncertainties, failures could cause serious problem which may damage the whole system. To avoid this unsafe behavior, some constraints should be satisfied.

Reinforcement learning is an efficient paradigm for learning optimal policies from experimental data. In order to gain more data points and find the optimal policy, we aim to explore every possible states and action, which may be harmful for some real world systems. The harms make learning algorithms rarely be applied on safety-critical systems in real world.

Safe reinforcement learning (safe RL) introduces a learning algorithm which considers safety constraint. This means our whole learning process should satisfy safety constraint. Safe RL is usually based on real world systems. For example on Atari platform, we do not need safe RL because we could explore any states and actions and find the best policy that gives the maximum reward, and it would not damage the system or lead to serious consequences. When it comes to some realistic examples like self driving car, safe RL becomes significant due to safety reasons. Any policy we apply to the system has to guarantee safety constraints. In this project, we implemented reinforcement learning algorithm with safety guarantees.

1.2 Our Goal

(1) In order to explore more states and actions to gain more experimental data, we aim to expand our safety region, which is the region of state-action pairs that will not cause system crash.

(2) With all the policies that satisfy the safety constraints, we aim to find the policy with the largest reward.

(3) Implement safe RL algorithm on realistic examples: inverted pendulum system, mass-spring-damper system and inertial wheel pendulum system. Expand safety region, update our policy and reward in safety region and compare the new policy with initial policy.

2 Related previous work

There has been much active research about safe reinforcement learning. However, people define safety in different ways. In discrete system, discrete Markov decision processes (MDPs) are one class of tractable models that have been used to maximize rewards while safely explore the discrete state space. Both [6] and [9] introduce algorithms that can safely explore MDPs and never get stuck without safe actions. The main challenge of these methods is that we need have an accurate probabilistic model of the system.

For continuous states and actions, model-free policy optimization algorithms have been successful. For instance, [5] proposed a model-free algorithm that can be implemented in real-world tasks such as robotics. In this scheme, people often guarantee safety as the system satisfying certain constraint with high probability [3]. Specifically, Lyapunov function and region of attraction are often used to verify stability when the system and its controller are known [8]. A recent work [4] used a Lyapunov function for estimating region of attraction. The main challenge of these methods is to find a suitable Lyapunov function for the system.

3 Assumption and Theorem

3.1 Framework Overview

We model the discrete-time, deterministic dynamic system as follow:

$$x_{t+1} = f(x_t, u_t) = h(x_t, u_t) + g(x_t, u_t) \tag{1}$$

where states $x \in \chi \subset R^q$, control actions $u \in \mho \subset R^p$ and a discrete time index t. The true dynamic f contains two parts: $h(x_t, u_t)$ is a known prior model that can be obtained, $g(x_t, u_t)$ represents an unknown model error. In order to obtain measurements of $g(x_t, u_t)$, we learn the true dynamics by driving the system to a series of state-action pairs.
In this project, we start with an initial safety region and initial policy, then collect data points in safety region and find the policy which could satisfies safety constraints as well as expand the safety region. Next, we pick the policy with the largest reward. Last, we update safety region and policy, and further update model error $g(x_t, u_t)$. With the process of collecting data points, we could expand safety region as well as optimize the policy which gives a larger reward.

3.2 Model Assumptions

In order to model the real dynamics, the paper[4] made a few assumptions to solve the problem about the dynamic model.

(1) Real dynamic models are continuous, but our transform function (1) is discrete, so we have to discretize both state and action space into cells, and approximate state-action pairs with the nearest grid point. We assume that the dynamics $h(x_t, u_t)$ and $g(x_t, u_t)$ are Lipschitz continuous with respect to the 1-norm. By Lipschitz continuous function, we could approximate continuous dynamics by discretization model.

(2) The model error $g(x_t, u_t)$ is unknown. In this model, we use Gaussian Process(GP) model to solve this problem, so that although g is unknown, we could build confidence intervals that cover the true function of the dynamic system with high probability.

(3) It is very difficult and complex to guarantee each state-action pair satisfies safety constraints. We characterize the region of attraction by Lyapunov function, which only needs to check one step decrease condition and could guarantee the value function will converge to zero. Then, we need to find a value function of the dynamics that satisfies Lyapunov function. Fortunately, according to the definition of value function:

$$v(x) = r(x, \pi(x)) + v(f(x, \pi(x))) \leq v(f(x, \pi(x))) \tag{2}$$

where $v(x)$ is the value function, $r(x, \pi(x))$ is the cost function, which is strictly positive away from the origin. It already satisfies Lyapunov function that our value function will decrease and converge to zero.

(4) We also assume that we could get initial safety region and policy. It is reasonable because in model-based RL problem, we must have some knowledge about the model, so that we could get s_0 either by simulation or experience. For example, a self-driving car's initial safety region could be defined as driving in the middle of the street at a low speed.

3.3 Algorithm

We first define initial safe set S_0 and initial policy π_0, then the algorithm selects safe, informative state-action pairs within safety region, which can be evaluated without leaving region of attraction (Lyapuno constraints) of the current policy π_n. As we get more data, we can expand the safety region with the data points. Then, we select a set of policies that satisfy safety constraints and we pick one with the largest reward. Next, we update our policy, safety region and GP model. The figure below shows our safe RL algorithm.

Algorithm 1 SAFELYAPUNOVLEARNING

Input: Initial safe policy π_0, dynamics model $\mathcal{GP}(\mu(\mathbf{z}), k(\mathbf{z}, \mathbf{z}'))$
for all $n = 1, \dots$ **do**
  Compute policy π_n via SGD on (7)
  $c_n = \text{argmax}_c\, c$, such that $\forall \mathbf{x} \in \mathcal{V}(c_n) \cap \mathcal{X}_\tau \colon u_n(\mathbf{x}, \pi_n(\mathbf{x})) - v(\mathbf{x}) < -L_{\Delta v}\tau$
  $\mathcal{S}_n = \{(\mathbf{x}, \mathbf{u}) \in \mathcal{V}(c_n) \times \mathcal{U}_\tau \,|\, u_n(\mathbf{x}, \mathbf{u}) \leq c_n\}$
  Select $(\mathbf{x}_n, \mathbf{u}_n)$ within $\mathcal{S}_n$ using (6) and drive system there with backup policy π_n
  Update GP with measurements $f(\mathbf{x}_n, \mathbf{u}_n) + \epsilon_n$

4 Experiment on 2-dimensional Models

4.1 Linear–quadratic Regulator[1]

The safety issue becomes critical in real world system, therefore one biggest challenge in our problem is to define and explore the dynamic model for some physical systems.

First of all, we are going to implement the safe RL algorithm in the case where the system dynamics are described by a set of linear differential equations,which is defined as linear–quadratic(LQ) problem.The solution to LQ problem is provided by the a feedback controller called linear–quadratic regulator(LQR), one of the most fundamental problems in control theory, which is essentially an automated way of finding an appropriate state-feedback controller. And the cost is described by a quadratic function, which is often defined as a sum of the deviations of key measurement. The algorithm thus finds those controller settings that minimize undesired deviations.

The way to use LQR algorithm is by specifying the cost function parameters, and using an iterative process to find the "optimal" controllers produced through simulation and then adjusting the parameters to produce a controller more consistent with design goals.

4.2 Linear systems with state space presentation [2]

The most general state-space representation of a linear system is written in the following form:

$$\dot{x}(t) = A(t) * x(t) + B(t) * u(t) \tag{3}$$

Here, x(·) is called the "state vector", x(t) $\in \mathbb{R}$, y(·) is called the "output vector", y(t) $\in \mathbb{R}$, u(·) is called the "input (or control) vector", u(t) $\in \mathbb{R}$, A(·) is the "state (or system) matrix", B(·) is the "input matrix".

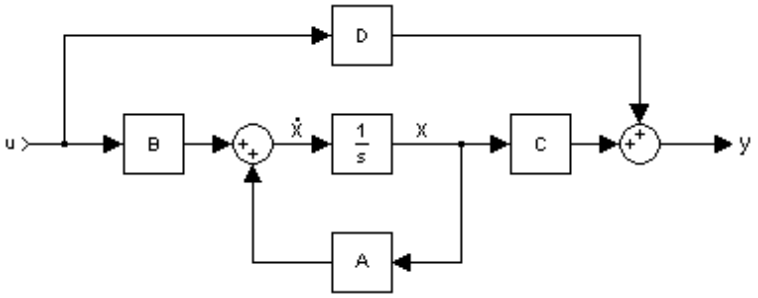

Figure 1: Block diagram representation of the linear state-space equations

The state space and the action space we use in our studies are discretized for further ease of use. Therefore, we have a discrete state vector and its discrete derivative (the derivative of finite discrete function is locally approximate to the one under continuous case).

Another important issue before building up the connection is that we need to linearize our system if we are to use this feedback control system. Although almost every physical system contains nonlinearities, oftentimes its behavior within a certain operating range of an equilibrium point can be reasonably approximated by that of a linear model. For example, given a nonlinear system and its equilibrium point x^* = $[x_1^*...x_n^*]^T$ obtained when $u = u^*$, we denote Δx = x - x^*, Δu = u - u^*. The linearization of the original control function becomes:

$$\dot{\Delta x} = A * \Delta x + B * \Delta u \tag{4}$$

This kind of linearization, which also referred to as a small-signal model, is valid only in a sufficiently small neighborhood of the equilibrium point x^*. The similar procedure will be taken care of in the following examples.

4.3 EXAMPLE 1: Inverted Pendulum System Dynamics

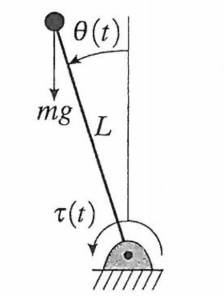

Figure 2: A Typical Inverted Pendulum Model

The previous work from Felix [4] can be illustrated through an inverted pendulum benchmark problem. Under that circumstance, the dynamics are given by m l^2 $\ddot{\varphi}$ = gml sin (φ). φ is the angle, m the mass, g the gravitational constant, and u the torque applied to the pendulum. And the equilibrium point can be obtained by setting u = u^* = 0. In his settings, a combination of linear and Matérn kernels are used in order to capture the model errors that result from parameter and integration errors.

In order to linearize the model, we need to assume that the vertical upward position is only valid when the angle $\theta$is in a small neighborhood of and the angular velocity $\dot{\theta}$ is small. In the implementation, the boundaries for the states and actions are: state[anglemax = 30°, anglevelocitymax = $\sqrt{g\backslash l}$], action= $mgl * \sin(anglemax)$. Also, the state space is discretized into [2001, 1501] grids and the action space into [55, 55].

Recall the fact that the optimal value function is a Lyapunov function for the optimal policy if the dynamics are deterministic. As uncertainty about the dynamics decreases, the value function for the mean dynamics will thus converge to a Lyapunov function.

For the initial policy, we use approximate dynamic programming to compute the optimal policy for the prior mean dynamics. This policy is unstable for large deviations from the initial state and has poor performance. Under this initial, suboptimal policy, the system is stable within a small region of the state-space. Starting from this initial safe set, the algorithm proceeds to collect safe data points and improve the policy. As the uncertainty about the dynamics decreases, the policy improves and the estimated region of attraction increases. At the same time, the control performance improves drastically relative to the initial policy. Overall, the approach enables safe learning about dynamic systems, as all data points collected during learning are safely collected under the current policy.

By using the tkinter package from canvas in Python, we built a small demo to see what would our trained action look like. The gif version is included in the folder called demo.gif. Figure 3 are the screenshots of the progress, from which we can clearly see that, when a disturbance is applied to the pendulum, the learned policy would move it upward to the equilibrium.

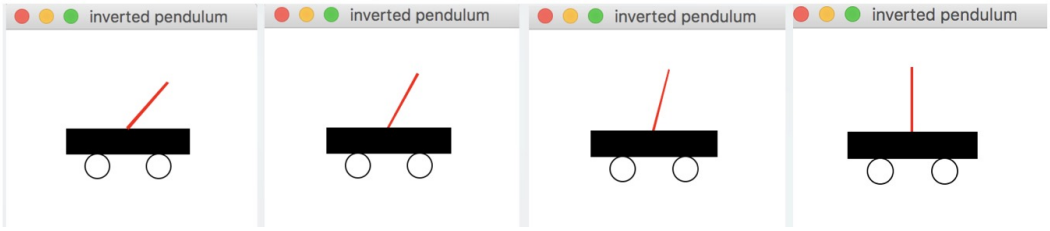

Figure 3: Screenshots of our Trained Policy

4.4 EXAMPLE 2: Mass-Spring-Damper System

Similar to the pendulum system, we testified the safe RL in another LQ problem: the dynamic model of mass spring damper system. The following is a fixed-base configuration with spring and damper in parallel.

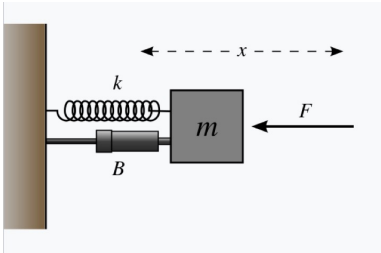

Figure 4: Mass-Spring-Damper System

Applying Newton's law to a free-body diagram of the mass, we can get the 2nd order ODE:

$$m\ddot{x} + c\dot{x} + kx = F \tag{5}$$

Here, k: stiffness of the spring (N/m), c: damping coefficient (kg · m/s2), m: mass of the block (kg), F: external force (N).

Table 1: mass-spring-damper system

Parameters	Value
k	100 N/m
c	2 kg·m/s2
m	0.5kg

Suppose that we have a practical system (parameters shown in the above table). And our goal here is to stabilize the mass when an external force is applied to this system. The equilibrium point is

defined as the balanced point when x = 0. The control force is limited, so that the mass necessarily breaks down beyond a certain displacement.

For the policy, we use a neural network with 32 neurons with ReLU activations. And a quadratic, normalized cost is considered for standard approximate dynamic programming : (x, u) = x^T Qx + u^TRu, where Q and R are positive-definite, to compute the cost-to-go J. Specifically, we use a piecewise-linear triangulation of the state-space as to approximate J. This allows us to quickly verify the assumptions that we made about the Lyapunov function using a graph search. In practice, one may use other function approximators. We optimize the policy via stochastic gradient descent.

After further discretization and linearization, we can examine the optimization process of our policy and how the safety comes into play. The initial safe set lies in the space like this:

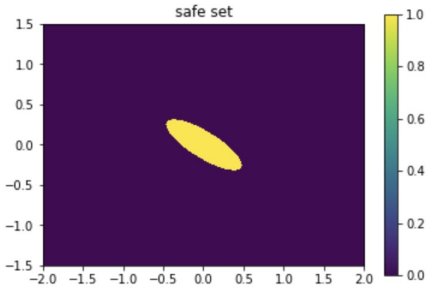

Figure 5: Initial Safe Set

Using dynamic programming and enforcing the decrease condition on the Lyapunov function with a Lagrange multiplier, we can explore our safe set which is close to the current policy by sampling the most uncertain state that does not leave the current level set. The updated safe sets through 5 iterations are as follows.

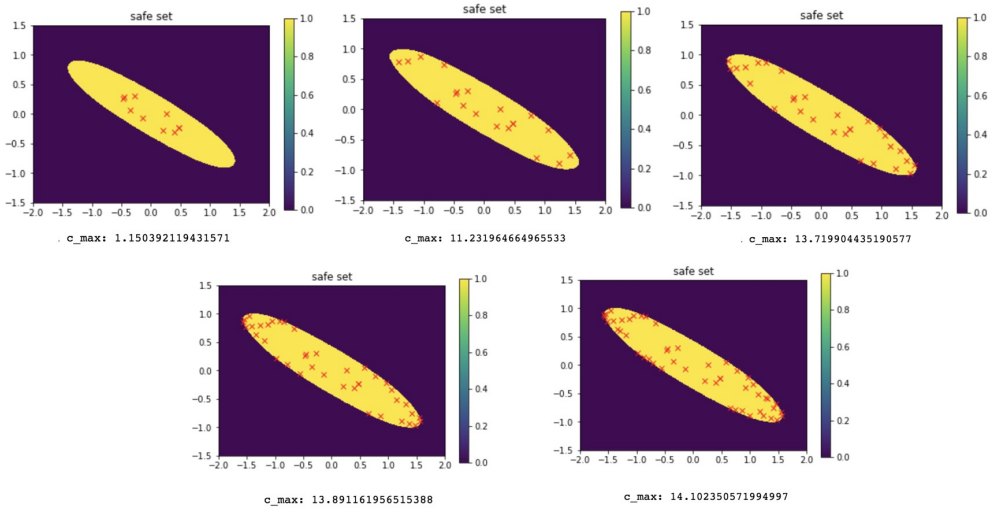

Figure 6: Safe Set After Each Iteration

The difference in performance can be easily illustrated with the following visualizations. Here, we plot the trajectories of the updated states and actions. Furthermore, we plot the resulting value function, policy and the track of the optimized policy on the action space.
Note: The state has the form of [displacement, velocity]. The policy has the form of [force].

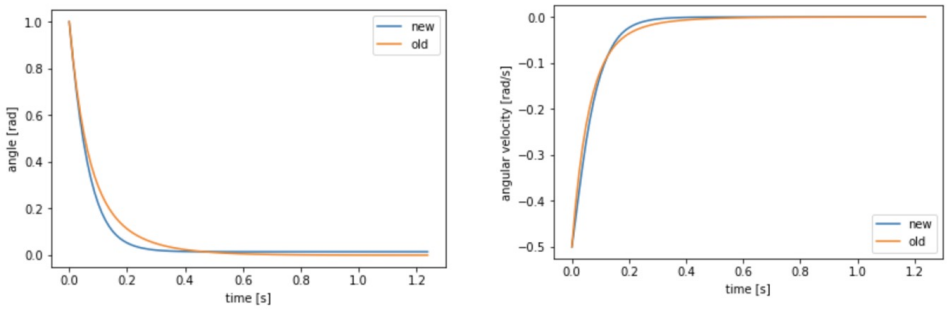

Figure 7: Trajectories of the States

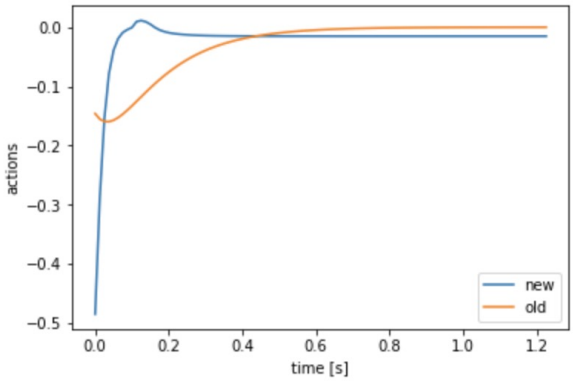

Figure 8: Trajectories of the Action

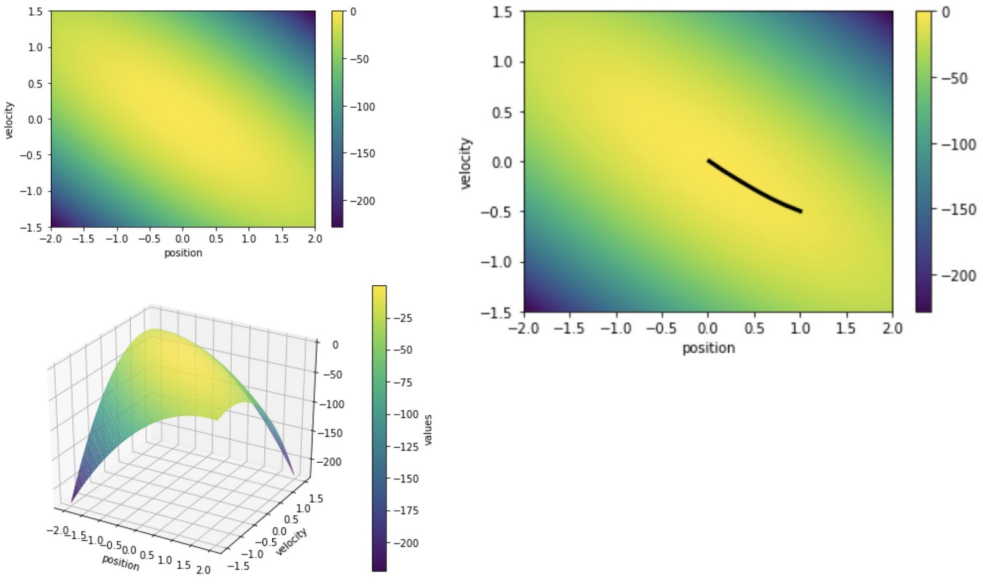

Figure 9: Value Function and the Trajectory of the Optimized Policy

4.4.1 Conclusion

The above visualizations for the process of our updated policy can verify that:

- Our way of finding the maximum possible safe set works: the safe set is expanding with the level value increasing.
- The new strategy has a reward function of [-5.5861], whereas the old value function was [-5.7485], which implies an improvement in the reward.
- Once a disturbance is applied to the system, the new algorithm (the one optimized through safety RL) has a better performance of finding its way back to the initial state, which is defined as a backup plan during the exploration.
- The trajectory of the action shows one possible track for the trained policy. It can be shown from the path that the safety RL algorithm guarantees a policy which would remain inside of the safe set during the entire exploration process.

5 One Higher Dimentional Model: Inertial Wheel Pendulum

This section describes another example we have tried but not really worked out as expected. We will discuss our scheme and the result as well as the possible reasons why the result is not desirable.

We consider the IWP example that can be controlled under LQR. The kinetic model of IWP is shown below. It is an underactuated robot class with two degrees of freedom and one actuator. The first degree of freedom is the angle from vertical (x1 in the figure) and the second is steering wheel angle (x3 in the figure). The control is the current which flows in the dc motor. The motor accelerates the rotating mass and makes it possible to stabilize the system.

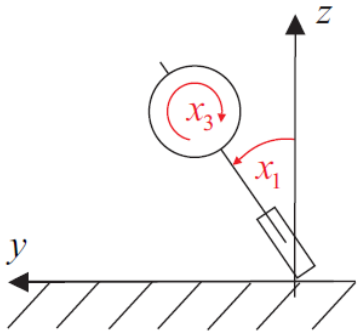

Figure 10: IWP

Referring to [7], we can derive the mathematical model of the system as:

$$\begin{cases} \dot{x_1} = x_2 \\ \dot{x_2} = \varphi_1\varphi_4 u + \varphi_1\varphi_5 x_4 + \varphi_2 x_2 \\ \dot{x_3} = x_4 \\ \dot{x_4} = \varphi_4 u + \varphi_5 x_4 \end{cases}$$

where
x_1 is the angle of the pole from vertical
x_2 is the angular velocity of the pole
x_3 is the angle of the steering wheel
x_4 is the angular velocity of the steering wheel
u is the torque applied to the system by the motor $\varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5$ are constants defined by the the steering wheel inertia, the construction inertia, height and mass and the friction of the system.

With this we can find the state-space representation of the system and apply LQR. Ideally, we can then follow the previous example to construct the model and apply our algorithm to solve for the best policy under safety constraint. However, now we have four dimensions in each state instead of two as in the previous examples, which greatly scales up the the Tensorflow graph and eventually

results in memory error. We tried to reduce the dimension of every part of our algorithm and finally got the model worked with discretization space of only 6 x 6 x 6 x 6 grid size. The problem is we weren't able to effectively expand the safe set with this rough discretization. This manifests one critical drawback of the algorithm, which is that the complexity of the system would be too large for us to solve with higher dimension of state space.

6 Conclusion and future work

In this project, we implement Safe RL algorithm and verify that the algorithm could efficiently expand safety region as well as optimize the policy with larger rewards. We build two models: inverted pendulum model and mass-spring-damper model, which prove that Safe RL algorithm is valid and practical. When we explore high-dimension state space, inertial wheel pendulum model, because of high dimension and large time complexity, we could not explore safe region.

From our implementation and exploration of Safe RL algorithm, we realize there are a few limitations that need more future exploration:

(1) High dimension problem: When the state or action parameters' dimension is larger than two, the algorithm becomes impractical. On one hand, the model itself becomes hard to build. On the other hand, GP model becomes hard to calculate due to large covariance matrix.

(2) Continuous problem: We discretize the state and action space and use discretization space instead of continuous real world dynamics. It's based on the assumption that the dynamics satisfies Lipschitz continuous condition, but in real system it could not be guaranteed.

(3) State reachable problem: We only consider safe constraints in safe region. In real dynamic, some states can be unreachable although it satisfies our safe constraints. So there still exists gap between this Safe RL algorithm and real dynamic learning process.

7 Acknowledgements

We would like to thank Professor Yue and the TAs for their support. We would also like to thank Felix Berkenkamp from ETH Zurich for his kindly support in building up the environment and solving our puzzles about his paper.

References

[1] Linear–quadratic regulator. `https://en.wikipedia.org/wiki/Linear%E2%80%93quadratic_regulator`.

[2] State-space representation. `https://en.wikipedia.org/wiki/State-space_representation`.

[3] J. Achiam, D. Held, A. Tamar, and P. Abbeel. Constrained Policy Optimization. *ArXiv e-prints*, May 2017.

[4] F. Berkenkamp, M. Turchetta, A. P. Schoellig, and A. Krause. Safe Model-based Reinforcement Learning with Stability Guarantees. *ArXiv e-prints*, May 2017.

[5] Y. Li. Deep Reinforcement Learning: An Overview. *ArXiv e-prints*, January 2017.

[6] T. Mihai Moldovan and P. Abbeel. Safe Exploration in Markov Decision Processes. *ArXiv e-prints*, May 2012.

[7] A. Owczarkowski, M. Lis, and P. Kozierski. Tracking control of an inertial wheel pendulum by lqr regulation. In *2014 19th International Conference on Methods and Models in Automation and Robotics (MMAR)*, pages 384–388, Sept 2014.

[8] Theodore J. Perkins and Andrew G. Barto. Lyapunov design for safe reinforcement learning. *J. Mach. Learn. Res.*, 3:803–832, March 2003.

[9] Wolfram Wiesemann, Daniel Kuhn, and Berç Rustem. Robust markov decision processes. *Mathematics of Operations Research*, 38(1):153–183, 2013.