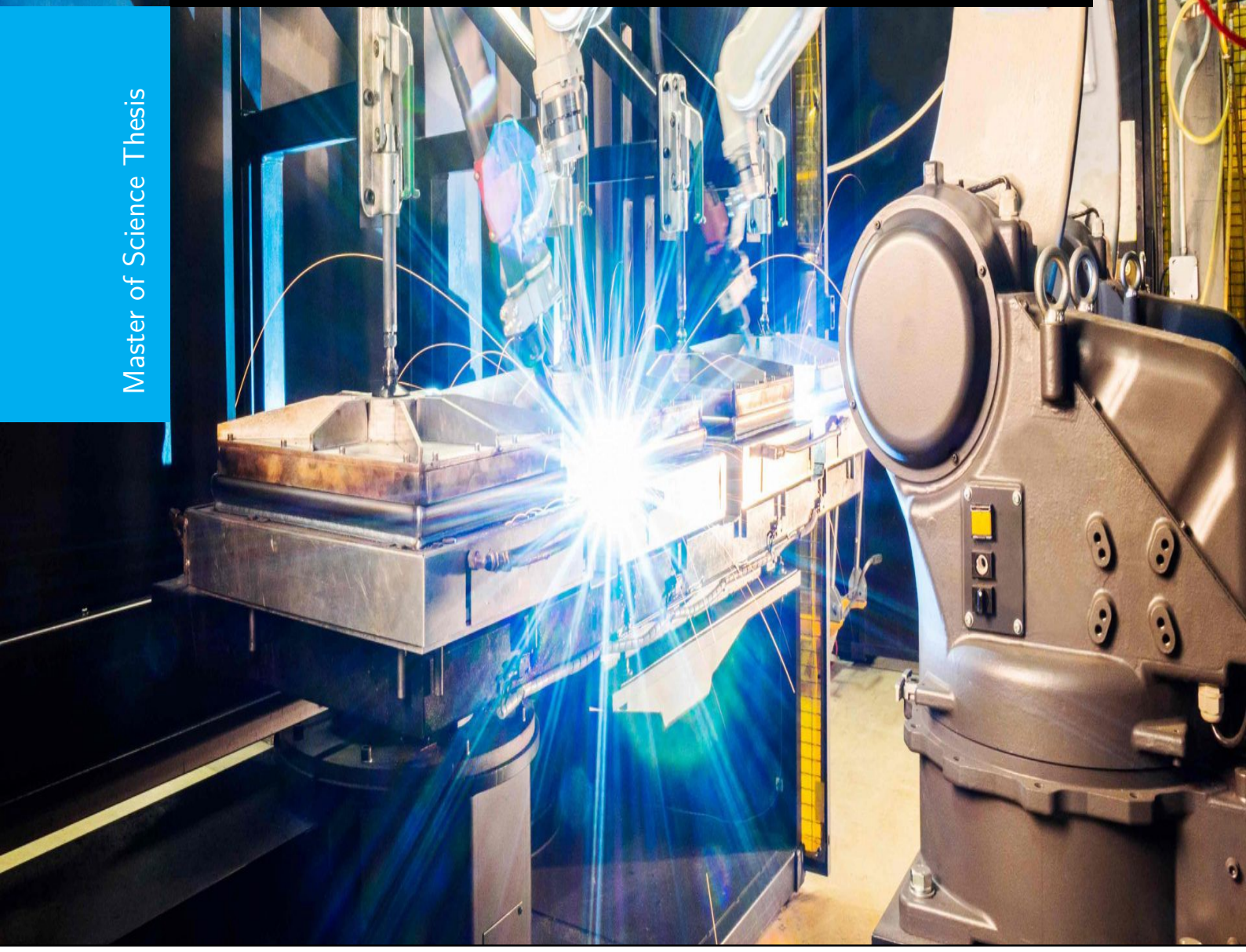
Model Free Reinforcement Learning with Stability Guarantee

Yuan Tian

Master of Science Thesis



TU Delft Delft University of Technology

Delft Cognitive Robotics

Model Free Reinforcement Learning with Stability Guarantee

Master of Science Thesis

For the degree of Master of Science in Vehicle Engineering at Delft University of Technology

Yuan Tian

August 12, 2019

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

Model-free reinforcement learning has proved to be successful in many tasks such as robotic manipulator, video games, and even stock trading. However, as the dynamics of the environment is unmodelled, it is fundamentally difficult to ensure the learned policy to be absolutely reliable and its performance is guaranteed. In this thesis, we borrow the concept of stability and Lyapunov analysis in control theory to design a policy with stability guarantee and assure the guaranteed behaviors of the agent. A novel sample-based approach is proposed for analyzing the stability of a learning control system, and on the basis of the theoretical result, we establish a practical model-free learning framework with provable stability, safety and performance guarantees. In our solution, a Lyapunov function is searched automatically to guarantee the closed-loop system stability, which also guides the simultaneous learning (covering both the policy and value-based learning methods). Our approach is evaluated on a series of discrete and continuous control benchmarks and largely outperforms the state-of-the-art results concerning unconstrained and constrained problems. It is also shown that the algorithm has the ability of recovery to equilibrium under perturbation using the policy with stability guarantee. (Anonymous code is available to reproduce the experimental results[1].) Since sometimes the constraint is hard to define, we introduce a novel method to learn a constraint by representing the bad cases or situations as a distribution, and the constraint is the Wasserstein distance between the distribution.

[1] https://github.com/RLControlTheoreticGuarantee/Guarantee_Learning_Control

Table of Contents

List of Figures

List of Tables

Acknowledgements

Firstly, I would like to thank my supervisor prof. dr. Wei Pan for his guide, help, and support during the whole year. He gives me many helpful and useful suggestions, teaches me how to be an eligible scientific researcher and how to build my own career, which means a lot to me. I also want to thank Hongpeng Zhou and Minghao Han. I met a lot of obstacles during the whole year, they always help me comprehend quickly with patience. And during the thesis work, we submitted a paper to NIPS. It is really a precious experience for me. Most importantly, I want to thank my family, without their support, I can not have this chance to study abroad.

Through the thesis work, I not only learned much state-of-art technology, but also learned how to work out a large specific project, learned how to manage my life. I really enjoy the time very much.

Finally, I want to thank TU Delft, I learn, experience, grow up a lot here, I will never forget the two years.

Delft, University of Technology — Yuan Tian
August 12, 2019

“Don’t waste your life”

Chapter 1

Introduction

1-1 Background

Optimal control aims to find a control policy for a given system for a certain task with the optimality criterion. The task is described as a cost function. An optimal control determinates control and trajectories over a period of time to minimize the cost function[3]. But, optimal control solutions are off-line and require complete knowledge of the system dynamics. Therefore, they are not able to cope with uncertainties and changes in dynamics[4].

On the other hand, reinforcement learning (RL) is a useful substitute. Same with the optimal control, it also uses a cost function as guidance. RL focus on how an agent could interact with its environment and learn a policy could maximize expected cumulative rewards for a task. For model-free reinforcement learning, its advantage over optimal control is that it does not need a predefined controller structure and the dynamic model, which limits the performance of the agent and costs more human effort. Furthermore, it could deal with uncertainties and changes in dynamic.

In recent years, significant progress and success have been made in solving challenging and complicated problems across various fields by using deep reinforcement learning (RL). People use reinforcement learning to play Atari from pixels[1], or training an agent to become the best player in Go [5]. In the domain of AutoML, it made success in neural architecture search or design[6]. In health care, it helps make the decision in the use of medical instruments[7] and drug dose [8].

People believe complex video games could somehow capture the messiness and continuous nature of the real world. OpenAI and DeepMind now are working on two extremely complicated tasks, the Dota 2 and StarCraft2 respectively. Dota 2 is a real-time strategy game played between two teams of five players. It has long time horizons, each game usually ends more than 20000 moves, while the Go usually ends before 150 moves. The state space is partially-observed and high-dimensional, continuous. Even with extraction, there still remain more than 20000 floating-point numbers, and the Go has only 381 enumeration values and fully-observed. The action space is large as well, every step has more than 1000 valid actions at least on average,

and the Go is 35. Besides, the Dota rules are also very complex. Now, OpenAI's Dota agent has defeated the world's top professionals at 1 on 1, has defeated the world's top professionals at 5 on 5, they are going to world's top professionals' team next year [9].

Most importantly, as mentioned before, reinforcement learning is inherent to deal with a control problem. It has been applied to a range of robotic control tasks. In the simulator, the trained agents could learn how to run, jump, crouch[10]. In real-world, the agent could learn a complex door opening skill in only 2 hours[11]. Using reinforcement learning to control a quadrotor in some critical situation shows a better result than common trajectory optimization algorithms with an approximated model[12]. Recently, using policies trained in simulation, the quadrupedal machine could achieves locomotion skills and run in real-world, which could run faster than before and recover from falling even in complex configurations [13].

1-2 Problem Statement and Research Question

Relative to these mentioned AI milestones, my hope is that systems which solve complex video games will be highly general, with applications outside of games.

However, during the literature review and thinking, I found there are still some obstacles need to be overcome.

1-2-1 Model

Reinforcement learning aims to obtain a policy about how agents could interact with the environment and get the highest reward or lowest cost. In other words, agents all need to be trained in an environment, which could be a simulator or the real world. Real-world training is time and cost inefficiency, so most RL applications are trained in the simulator.

But there are still many tasks or physical phenomena cannot be accurately modeled, like the flow field or the real traffic, which means these tasks still hard to deal with. And the reality gap is a serious problem as well, the difference between simulator and real-world can severe decrease the controller performance.

1-2-2 Guarantee

Although an increasing amount of promising results are being produced, RL is still a large black-box that we could not explain or predict the actions and the outcome. Agents may take a temporal undesirable move, running into states unable to recover, hurting itself or human. For example, a self-driving car should always keep away from the region of pedestrian; even if it is perturbed to the edge of the zone of catastrophe accidentally, it still should be able to recover before it's too late. The ability to recover is defined as stability in control theory, which is crucial in safety-related physical plants such as autonomous vehicles, health-care robots and chemical industries. However, due to the formidable challenge, the stability theory for RL is immature and still an open problem [14, 15]. And there is not any guarantee to make sure the policy is absolutely reliable and predictable.

And, RL agents learn a policy by thousands of time trial and mistake, and they explore the world by trying many different policies before converging. So even if we design a reward function that leads an agent to safe policies at optimum, it could still result in unsafe exploration behavior.

1-2-3 Research Questions

With the previous problems, this thesis will focus on the second topic and my main research question is:

How could we train an agent with stability or safe guarantee?

1-3 Contributions

- **Reinforcement Learning with Stability Guarantee** We develop a novel method for analyzing Lyapunov stability for model-free reinforcement learning. We provide a theoretical guarantee for the performance of the stability guaranteed policy. We show that the stability guaranteed policy is more capable of handling large external perturbation compared to those without such guarantees.

- **Safe Reinforcement Learning with UUB Guarantee** With the stability analysis, We develop the Lyapunov-based approach to guarantee the UUB of the system, and eventually achieve the constraint or safe satisfaction in CMDP. And our safety definition is strictly staying inside the safe region confined by the safety cost, rather than keeping the discounted sum of safety cost under certain bounds as in normal CMDP [16], which is far more restricted than others. Our setting is more effective and practical since no discount parameter or bound to be tuned [17]. Our approach can be generally combined with both on-policy and off-policy algorithms. We then introduce several practical learning algorithms for UUB guaranteed policies and our algorithms achieving the state-of-the-art performance in a series of continuous and discrete control benchmarks with safety constraints. Our experiment shows that the algorithm outperforms the existing results in [18, 19], while maintaining zero violation of the state action constraints.

- **Lyapunov function in reinforcement learning** We present several principled ways for constructing Lyapunov functions and evaluate their effectiveness on various control tasks.

- **Reproducible results** Reproducible is a big challenge in reinforcement learning. With good hyper-parameters tunning, reward design and combining proper state-of-art technologies, all the results in this thesis are reproducible. And we provide all the experiment details in 5.

- **Self-Constrained method** Since in many situations, the constraint might be hard to define like the driving a car in real traffic. So, we develop a novel method could let the agent learn a constraint by itself.

1-4 Thesis Outline

This thesis presents the proposed approach and the results from the thesis work. This thesis report is organized as follows: Chapter 2 is the literature study, giving the basic concepts of RL and introduction to the state-of-art RL algorithms and some related works. Chapter 3 introduces the preliminaries used in followed Chapters. Chapter 4 shows the proposed approach. Chapter 5 explains the environment of the experiments and shows the results. Chapter 6 is the conclusion of the thesis work and some thoughts for future research. Chapter 7 is a byproduct of this thesis.

Chapter 2

Literature Survey

This chapter will introduce the basic idea of reinforcement learning (RL), explain some of the state-of-art RL algorithms and some necessary control concepts used in this thesis work.

2-1 Reinforcement Learning

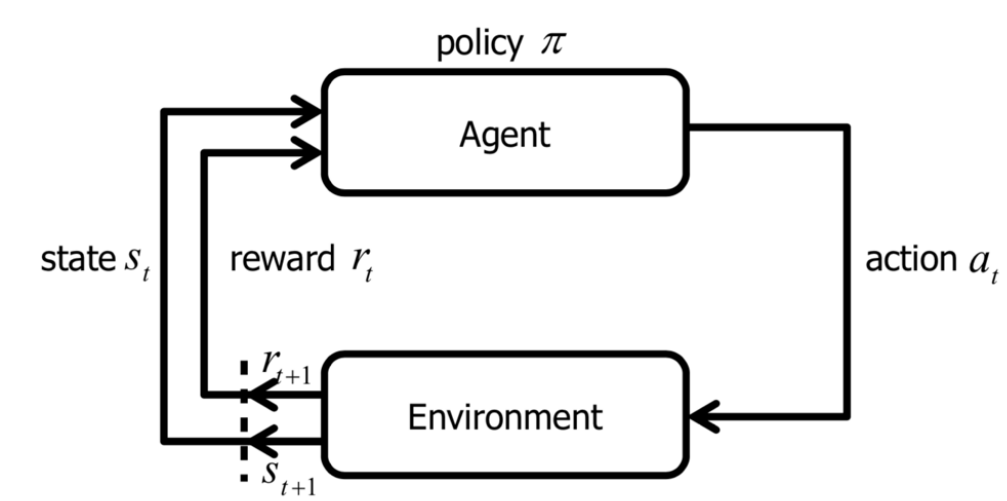


Figure 2-1: RL focus on how an agent could interact with its environment and to learn a policy could maximizes expected cumulative rewards for a task.

2-1-1 Markov Decision Processes

Reinforcement learning focus on how an agent could interact with the environment to achieve largest reward. And the whole process, is subject to Markov property and the Markov decision processes(MDPs).

Markov property means the future state's conditional probability distribution **only** depends on the current state and action instead of the whole history:

$$\mathbb{P}(X_n = x_n \mid X_{n-1} = x_{n-1}, \ldots, X_0 = x_0) = \mathbb{P}(X_n = x_n \mid X_{n-1} = x_{n-1}) \tag{2-1}$$

MDPs are useful for solving optimization problems solved by dynamic programming and reinforcement learning. In reinforcement learning, MDPs could model the problem or task that the agent need to solve. A Markov decision process is a 4-tuple : (S, A, P, R). S is the current state, A is the action that the agent make at the current state, P is the transition probability to the next state that at the current state S with the made action A, R is the received reward when at the current state S and make the action A.

2-1-2 Q Learning

Q learning is one of the simplest algorithm in reinforcement learning. The most basic Q learning could solve the discrete action and state problem like finding path in a grid world. It work as a table, the rows represent discrete state, the columns represent the different actions at each state. And the value for every grid $Q(s, a)$ means the **future discounted reward** at the state s and implement action a:

$$Q_\pi(s_t, a_t) = E_\pi(\sum_{k=0} \gamma^k R_{t+k+1} | S_t = s, A_t = a) \tag{2-2}$$

If the table's value is the ground truth, the agent could make the best action according to the highest value at each column. The only problem is how could we get a good prediction about the table's value. And Q-Learning provides a effective method for this [20].

Before learning begins, The Q table need to initialize an arbitrary fixed value. Then, at each time t,the agent selects an action a_t, observes a reward r_t, enters a new state s_{t+1} ,and Q is updated. The core of the algorithm is a simple value iteration update, using the weighted average of the old value and the new information, the γ is the discount factor for future reward:

$$Q^{new}(s_t, a_t) = (1 - \alpha)Q^{old}(s_t, a_t) + \alpha(r_t + \gamma maxQ(s_{t+1}, a)) \tag{2-3}$$

As we keep improving the policy, it will converge to the optimal policy.

In reinforcement learning, the algorithm could be divided into on-policy algorithms and off-policy algorithms. Off-policy algorithms could let the agent learn off-policy, which means it could just learn from the memory or data, like the Q learning. An on-policy agent must learn by himself. There is an on-policy algorithm which is very similar with Q-learning called SARSA:

$$Q^{new}(s_t, a_t) = (1 - \alpha)Q^{old}(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1})) \tag{2-4}$$

The difference is the SARSA replaces the max future Q by a real $Q(s_{t+1}, a_{t+1})$, the action is exactly the action will do next.

2-1-3 Deep Q Network

The Q-learning is hard to deal with large state space problems, and it is almost impossible to solve the continuous state problems like playing Atari games.

With the development of the deep neural networks(DNN), people find the DNN could help solve many challenging problems like image-based detection, classification and segmentation.

Then in 2015, people use recent advances in training deep neural networks to develop a novel artificial agent, named deep Q-network, that can learn successful policies directly from high-dimensional image inputs using end-to-end reinforcement learning [1]. Basically, DQN

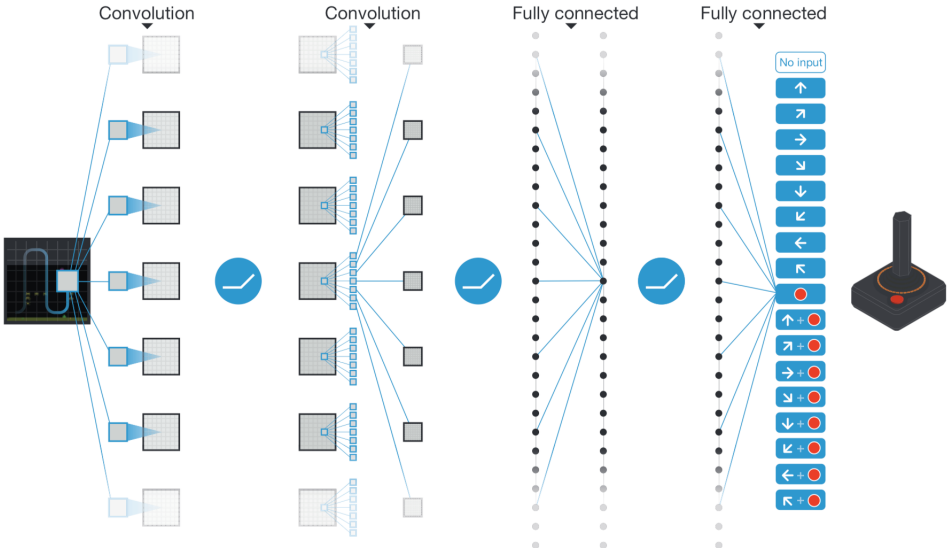


Figure 2-2: DQN [1]

is an advanced version of Q-Learning. So,like the Q-learning, the DQN input the state and output different actions' followed future discounted reward and the DQN build loss function in same sense with the Q update function:

$$L(\theta) = E[Q_{Target} - Q(s, a, \theta)] \tag{2-5}$$

But there are two main problems that make the training not very stable. Firstly, in supervised learning , we always want the data is independent and identically distributed. But in reinforcement learning, the data is always a trajectory, which makes the training unstable. Secondly, DQN learns the values of Q according to 2-5, but the target values for Q depends on Q itself, which means it is chasing a non-stationary target:

$$Q_{Target} = R + \gamma max_a Q(s', a) \tag{2-6}$$

To address these problems, DQN uses two technology : experience replay and target network.

Experience replay: DQN build a place to save or remember memories. While training, it uses a batch random memories instead of the current online experience. This forms an input

dataset which is stable enough for training and more independent of each other and closer to independent and identically distributed.

Target network: DQN create two networks θ' and θ. It use the first one called target network to generate action or get the Q values while the second to optimization. The purpose is to fix the Q-value targets temporarily and make the objective function is not non-stationary.

So, the whole training process is in Fig 2-3 below.

Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For t = 1,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Figure 2-3: DQN training algorithm [1]

2-1-4 Policy Gradient

In reinforcement learning, the algorithm could also be divided into policy-based algorithms and value-based algorithms. Value-based learning uses V or Q value to derive the optimal policy like the Q-Learning. while the policy-base method focuses on the policy itself.

The Q-Learning or value-based can only deal with discrete action problems, but some other tasks need the action space continuous, like balaceing a Cart-Pole. Then, people explored an alternative approach called policy gradient. In policy gradient, $\pi(a|s)$ is the probability of taking the action a given a state s [21].

In policy-based methods, instead of learning a value function that tells us what is the expected sum of rewards given a state and an action, it learn directly the policy function that maps state to action.

And, there is a trick to build the objective function:

$$f(x)\nabla_\theta \log f(x) = \nabla_\theta f(x) \tag{2-7}$$

Then, we replace the $f(x)$ with π :

$$\pi \nabla_\theta \log \pi = \nabla_\theta \pi \tag{2-8}$$

So, the objective function could be :

$$J(\theta) = E_{\tau \sim \pi_\theta(\tau)}[r(\tau)] = \int \pi_\theta r(\tau) d\tau \tag{2-9}$$

The policy gradient becomes:

$$\begin{aligned} \nabla J(\theta) &\approx \int \nabla_\theta \pi_\theta(\tau) r(\tau) d\tau \\ &\approx \int \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) r(\tau) d\tau \\ &\approx E[\nabla_\theta \log \pi_\theta(\tau) r(\tau)] \\ &\approx \frac{1}{N} \sum_i^N (\sum_t^T \nabla_\theta log \pi_\theta(a_{i,t}|s_{i,t}) (\sum_t^T r(s_{i,t}, a_{i,t})) \end{aligned} \tag{2-10}$$

It means to increase the likelihood of a policy if the trajectory results in a high positive reward. On the contrary, to decrease the likelihood of a policy if it results in a high negative reward. When it converges, the best action will have highest probability.

2-1-5 Actor Critic

The PG algorithm is successful, however, some glaring issues with vanilla policy gradients: noisy gradients and high variance. Because we compute the gradient is to sample a bunch of trajectories from the current policy ($\tau \sim \pi_\theta(\tau)$) and average their values. Then the computed gradient becomes dependent on the randomly sampled trajectories. As a result, itâĂŹs going to have variance, since its values depend on the sampled trajectories. This variance can be quite high. This is because each trajectory can take very different paths depending on the states visited and actions sampled from your policy.

And the Actor-Critic structure could improve it. The policy gradient algorithm's objective function has two parts:

$$\nabla J(\theta) \approx \frac{1}{N} \sum_i^N (\sum_t^T \nabla_\theta log \pi_\theta(a_{i,t}|s_{i,t}) (\sum_t^T r(s_{i,t}, a_{i,t})) \tag{2-11}$$

The first part is $\nabla_\theta log \pi_\theta(a_{i,t}|s_{i,t})$, which is a direction vector. Its direction is the steepest way to change the $log \pi_\theta(a_{i,t}|s_{i,t})$. The parameters update in this direction could significant increase or decrease the trajectory $\tau' s$ probability $P(\tau; \theta)$.

The second part is R, which is a scalar. It acts as vector's amplitude. The larger the R is, the larger the $\tau' s$ probability $P(\tau; \theta)$ will be after update.

Intuitively, the trajectory $\tau' s$ reward is like a **critic**. In other words, how a policy will change after update is according to the **critic**.

During my literature research, I found six frequently-used **critic**:

* Trajectory's total reward: $\sum_{t=0}^{\infty} \gamma^(t-1) r_t$
* Total reward after action: $\sum_{t'=t}^{\infty} \gamma^(t'-1) r'_t$
* Total reward after action with baseline: $\sum_{t'=t}^{\infty} \gamma^{(t'-1)} r'_t - b(s_t)$
* Q value: $Q(s_t, a_t)$
* Advantage value: $A(s_t, a_t)$
* TD error: $r_t + \gamma V(st+1) - V(s_t)$

The first three type of **critic**, they used trajectory's cumulative reward. With this, the gradient will have no bias but high variance.

When the critic used the last three types, the algorithm is the classical Actor-Critic structure. These critic will have low variance.

To sum up, Actor-critic combines the policy gradient with function fitting. It use the actor to model the policy and the critic to model V. And the whole training process is :

1 Take action $a \sim \pi_\theta(a|s)$, get (s,a,s',r)

2 Update $V^\pi_{target}(s)$ using $r + \gamma V^\pi_{target}(s')$

3 Evaluate $A^\pi(s,a) = r + r + \gamma V^\pi_{target}(s') - V^\pi_{target}(s)$

4 $\nabla J(\theta) \approx \nabla_\theta log \pi_\theta(a|s) A^\pi(s,a)$

5 $\theta = \theta + \alpha \nabla J(\theta)$

2-1-6 Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) is an algorithm with actor-critic structure, which learns a Q-function and a policy. It uses off-policy data and the Bellman equation to learn the Q-function, and uses the Q-function to learn the policy.

It has same idea with the DQN that uses target network to make the training more stable. DDPG uses four neural networks: a Q network, a deterministic policy network, a target Q network, and a target policy network. The Q network and policy network like the Actor-Critic very much. And the only difference is the Actor directly maps states to actions instead of outputting the probability distribution and sampling.

And it also uses memory to ensure the training data is independent and identically distribute.

In normal policy-based or actor-critic algorithms, the exploration is done via probabilistically. In the DDPG , the exploration is by adding noise to the action output:

$$a = a + \mathcal{N} \tag{2-12}$$

The Pseudocode is:

Algorithm 1 Deep Deterministic Policy Gradient

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
repeat
  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for however many updates **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets

$$y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_\phi(s, a) - y(r, s', d)\right)^2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$

      Update target networks with

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1 - \rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta$$

    end for
  end if
until convergence

Figure 2-4: DDPG Pseudocode

2-1-7 Trust Region Policy Optimization

In the whole policy-base algorithms included the Actor-Critic method, there is always a big problem about the learning rate α:

$$\theta = \theta + \alpha \nabla J(\theta) \tag{2-13}$$

Like I mentioned in 2-1-5, the policy gradient's update equation's second part represents the steepest ascent direction for the rewards, the equation means to update the policy towards that direction. It assumes that the policy's hyperplane is supposed to be flat. If the surface has high curvature, a large step will be a bad move. But, if the step is too small, the model learns too slow. Like the Fig 2-5, the blue diamonds is the current position, the red circle is the optimal point. If the learning rate is too large, it takes step that may lead the policy fall down from the cliff. Mostly, the agent will start from a bad performance situation with a poorly policy. Once the policy collapses, it will take a long time if ever, to recover. And the learning rate is very difficult to tune in reinforcement learning, so the previous policy-based reinforcement learning algorithms always suffer from convergence problem badly.

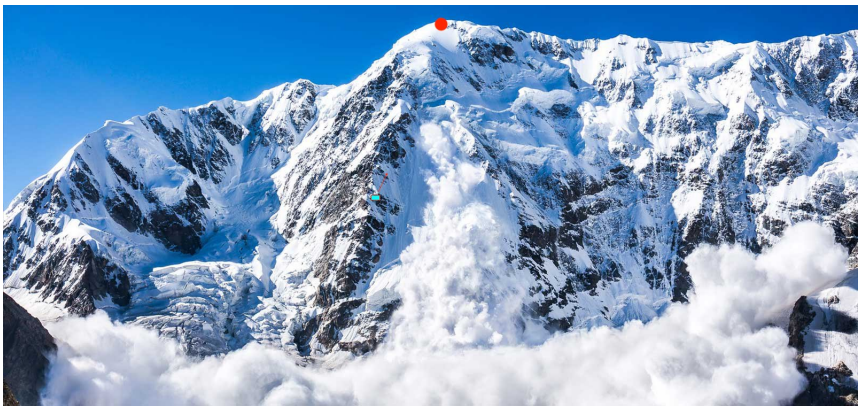


Figure 2-5: Hyperplane example

Then, people are wondering if there is a way could make the agent do better after every update. That is what the the trust region policy optimization(TRPO) could do.

As the TRPO aims to make the reward monotone increasing during every update, a natural thought is to represent the new policy's reward by the old policy's reward added with a other term. With this thought, we could build the objective function 2-14, and its proof is equation 2-15:

$$\eta(\pi_{new}) = \eta(\pi_{old}) + E_{\tau \sim \pi_{new}}(\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t)) \tag{2-14}$$

$$
\begin{aligned}
E_{\tau\sim\pi_{new}}(\sum_{t=0}^{\infty}\gamma^t A_\pi(s_t,a_t)) &= E_{\tau\sim\pi_{new}}((\sum_{t=0}^{\infty}\gamma^t(Q_\pi(s_t,a_t)-V_\pi(s_t))) \\
&= E_{\tau\sim\pi_{new}}(\sum_{t=0}^{\infty}\gamma^t(r+\gamma V_\pi(s_{t+1})-V_\pi(s_t)) \\
&= E_{\tau\sim\pi_{new}}(\sum_{t=0}^{\infty}\gamma^t r_t) + E_{\tau\sim\pi_{new}}(\sum_{t=0}^{\infty}\gamma^{1+t}V_\pi(s_{t+1})-\gamma V_\pi(s_t)) \\
&= \eta(\pi_{new}) + E_{\tau\sim\pi_{new}}(\gamma^n V_\pi(s_n)-V_\pi(s_0)) \\
&= \eta(\pi_{new}) + E_{s_0}(-V_\pi(s_0)) \\
&= \eta(\pi_{new}) - \eta(\pi_{old})
\end{aligned}
\tag{2-15}
$$

But from the equation 2-14, we could find it is impossibble to sample the $\tau \sim \pi_{new}$. So we do an approximation:

$$
L(\pi_{new}) = \eta(\pi_{old}) + \sum_s P(s)\sum_a \pi_{new}(a|s)A_\pi(s_t,a_t) \tag{2-16}
$$

From the equation 2-16, there still is a problem that we could not sample action from the new policy due to the parameters of new policy is unknown. So, we do the next approximation by importance sampling:

* Importance Sampling: In statistics, importance sampling is a general technique for estimating properties of a particular distribution, while only having samples generated from a different distribution than the distribution of interest.

$$
\begin{aligned}
\pi_{new}(a|s)A_\pi(s_t,a_t) &= \sum_a \pi_{old}(a|s)\frac{\pi_{new}(a|s)}{\pi_{old}(a|s)}A_{old}(s,a) \\
&= E_{a\sim\pi_{old}}\frac{\pi_{new}(a|s)}{\pi_{old}(a|s)}A_{old}(s,a)
\end{aligned}
\tag{2-17}
$$

Then, the objective function is :

$$
L_{\pi_{old}}(\pi_{new}) = \eta(\pi_{old}) + E_{s\sim\rho_{old},a\sim\pi_{old}}\frac{\pi_{new}(a|s)}{\pi_{old}(a|s)}A_{old}(s,a) \tag{2-18}
$$

And the policy update objective function could be :

$$
\begin{aligned}
&maximize_\theta(E_{a\sim\pi_{\theta_{old}}}\frac{\pi_{\theta_{new}(a|s)}}{\pi_{\theta_{old}}(a|s)}A_{\theta_{old}}(s,a)) \\
&\text{subject to } D_{KL}^{max}(\theta_{old},\theta)) \le \sigma
\end{aligned}
\tag{2-19}
$$

But the max KL divergence is unsolvable, the last step is using mean KL divergence to approximate the max KL divergence. Finally, the policy update objective function is :

$$
\begin{aligned}
&maximize_\theta(E_{a\sim\pi_{\theta_{old}}}\frac{\pi_{\theta_{new}(a|s)}}{\pi_{\theta_{old}}(a|s)}A_{\theta_{old}}(s,a)) \\
&\text{subject to } D_{KL}^{mean}(\theta_{old},\theta)) \le \sigma
\end{aligned}
\tag{2-20}
$$

And it use conjugate gradient to Solve the update.

2-1-8 Proximal Policy Optimization

PPO is based on some idea with the TRPO, to take the biggest possible improvement at every update, without stepping so far that causes performance collapse. But the TRPO's solution is a very complex second-order methor, it needs large computation and times. And PPO methods are significantly simpler to implement, and empirically seem to perform at least as well as TRPO.

The objective function is :

$$L^{CLIP}(\theta) = E[\min(r(\theta)\hat{A}_t, clip(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{2-21}$$

Which means that we will take the minimum of the clipped and non clipped objective, so the final objective is a lower bound (pessimistic bound) of the unclipped objective. Like the Fig 2-6 below. For example, when the $\hat{A}_t > 0$, it means at this state, the action is better than the average of all the actions in that state. Therefore, we should encourage our new policy to increase the probability of taking that action at that state, to increase the $r(t)$. However, because of the clip, $r(t)$ ill only grows to as much as $1+\epsilon$.

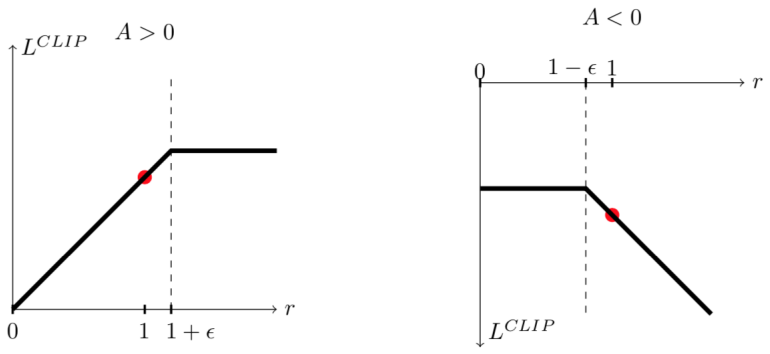


Figure 2-6: Plots showing one term (i.e., a single timestep) of the surrogate function L^{CLIP} as a function of the probability ratio r, for positive advantages left and negative advantages right $\hat{A}$ The red circle on each plot shows the starting point for the optimization, i.e., r = 1 Note that L^{CLIP} sums many of these terms

In baseline [1], the objective function is :

$$L(\theta) = E[L^{CLIP}(\theta) - c_1 L^{VF}(\theta) - c_2 S(\pi_\theta)] \tag{2-22}$$

Where the L^{VF} is squared error value loss :

$$L^{VF}(\theta) = [V(\omega) - V_{target}]^2 \tag{2-23}$$

And the $S(\pi_\theta$ is entropy term, which ensure sufficient exploration.

[1] https://github.com/openai/baselines

2-1-9 Soft Q

In reinforcement learning, there is a step called choosing action. In Q-learning or other value-based algorithm, we used epsilon-greedy schedule for action selection. In policy-based algorithm, , the actions are defined as a probability distribution, according to the current state $p(a|s)$. For example , in continuous control agent, the action probability distribution could be a gaussian distribution with a mean and standard deviation, which could be a neural network [22].

These actions selection will help the agent to know, to explore about the environment with least a priori knowledge about the world. So, it sometimes leads to different ends. For instance, the Fig 2-7 below is a Gym task called HalfCheetah, which the agent focus on how to move faster. With the same reward setting, the agent learned two different policies, one use legs to run, the other use the back to move.

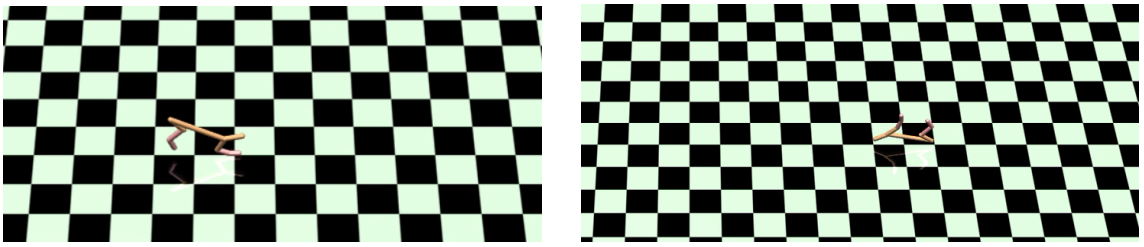


Figure 2-7: Trained HalfCheetah

Considering the role of randomness in action selection. People find a technology could improved exploration and compositionality that allows transferring skills between tasks called soft Q-Learning [23].

In reinforcement learning, the optimal policy maximizes the discounted cumulative reward:

$$\pi^* = \arg max E_\pi (\sum_t^{\inf} \gamma^{t-1} r) \tag{2-24}$$

The Q-value Q(s,a) represents the expected discounted cumulative reward after taking an action a at state s. Consider the robot Fig 2-7. The Q-function may look like the grey curve in Fig 2-8, with two peaks corresponding to the two move methods(back move and leg move). Normal RL approaches will have a policy distribution centered at the maximal Q-value with a variance for exploration. Due to the shape and the one peak of the policy , the agent will fix its policy little by little and ignores the lower method completely. In other words, the normal RL algorithms will obtain a policy like the red curve. But the optimal policy is the greed curve which make the agent could explores all promising states while select the most promising one. And this kind of policy could be like the exponentiated Q:

$$\pi(a|s) \propto \exp Q(s,a) \tag{2-25}$$

With this policy, the agent will be aware of all possible actions that may solve problems, which allows transferring skills and helps the agent adapt to changing situations. And, people shows that the policy defined upon is an optimal solution for the maximum-entropy RL objective:

$$\pi^*_{MaxEnt} = \arg max_\pi E_\pi (\sum_{t=0}^{T} \gamma^t (r_t + \alpha H(\pi(*s_t))) \tag{2-26}$$

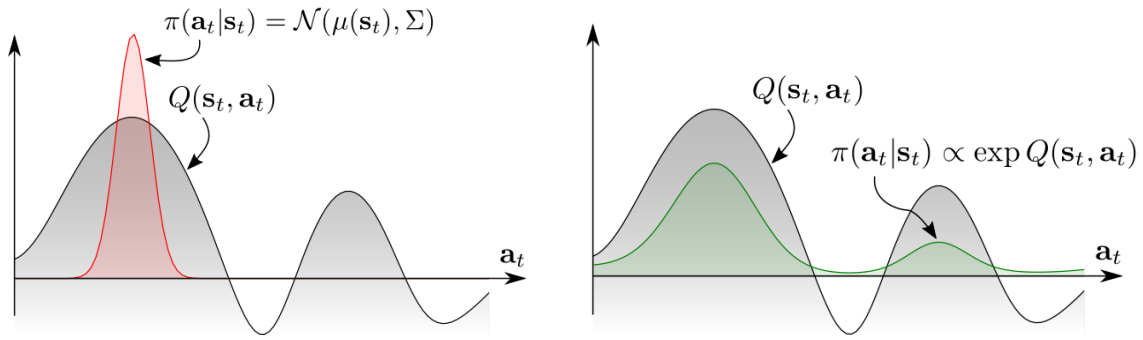


Figure 2-8: A multimodal Q-function

And they defined the soft Q-function by:

$$Q^*_{soft}(s_t, a_t) = r_t + E_{(s_{t+1},\ldots)\sim\rho}(\sum_{l=1}^{\infty} \gamma^l (r_{t+l} + \alpha H(\pi^*_{MaxEnt}(*|s_{t+l}))) \tag{2-27}$$

And the soft value function is:

$$V^*_{soft}(s_t) = \alpha \log \int_A \exp(\frac{1}{\alpha} Q^*_{soft}(s_t, a'))da' \tag{2-28}$$

Then the the optimal policy is

$$\pi^*_{MaxEnt}(a_t|s_t) = \exp(\frac{1}{\alpha} Q^*_{soft}(s_t, a_t) - V^*_{soft}(s_t)) \tag{2-29}$$

The soft Q-function satisfies the soft Bellman equation

$$Q^*_{soft}(s_t, a_t) = r_t + \gamma E_{s_t+1\sim\rho_s}(V^*_{soft}(s_{t+1})) \tag{2-30}$$

The soft Bellman equation have been proved that can hold for the optimal Q-function of the entropy augmented reward function [24].

Soft Q-Iteration

The Soft Q-Iteration is to iteratively update the estimates of V^*_{soft} 2-28 and Q^*_{soft} 2-30 until convergence just like the Q-iteration. But this is hard to be performed in continuous or large state and action spaces, and sampling from the energy-based model in is intractable in general.

Soft Q-Learning

So, a more general solution is soft Q-Learning. First, it converts the problem into a stochastic optimization problem, and the soft value function becomes by doing importance sampling like the TRPO:

$$V^\theta_{soft}(s_t) = \alpha \log E_{qa'}(\frac{\exp(\frac{1}{\alpha} Q^\theta_{soft}(s_t, a'))}{q_{a'}(a')}) \tag{2-31}$$

And the objective function is:

$$J_Q(\theta) = E_{s_t\sim q_{s_t}, a_t\sim q_{a_t}}(\frac{1}{2}(Q^\theta_{soft}(s_t, a_t) - Q^{\theta'}_{soft}(s_t, a_t))^2) \tag{2-32}$$

2-1-10 Soft Actor-Critic

As mentioned before, the maximum entropy Q function is:

$$Q(s_t, a_t) = r_t + \gamma E_{s_t+1 \sim \rho_s}(V(s_{t+1})) \tag{2-33}$$

The entropy term could be :

$$H(\pi(a|s)) = \int \pi(a|s) \log \pi(a|s) da = E_{a \sim \pi} \log \pi(a|s) \tag{2-34}$$

And the Value function is :

$$V(s_t) = E_{a_t \sim \pi}(Q(s_t, a_t) - \alpha \log \pi(a_t|s_t)) \tag{2-35}$$

With the same idea of soft Q-learning, people introduced the soft actor-critic. It uses a parameterized Q network $Q_\theta(s_t, a_t)$ and a policy network $\pi_\phi(a_t|s_t)$. The soft Q-function parameters can be trained to minimize the soft Bellman residual :

$$J_Q(\theta) = E_{s_t, a_t \sim D}(\frac{1}{2}(Q_\theta(s_t, a_t) - (r(s_t, a_t) + \gamma E_{s_{t+1} \sim p}(V_{\theta'} t + 1)))^2) \tag{2-36}$$

So the critic network update function is :

$$\nabla_\theta J_Q(\theta) = \nabla Q_\theta(s_t, a_t)(Q_\theta(s_t, a_t) - (r(s_t, a_t) + \gamma(Q_{\theta'}(s_{t+1}, a_{t+1}) - \alpha \log(\pi_\phi(a_{t+1}|s_{t+1})))))) \tag{2-37}$$

Where the policy network's update objective function is :

$$J_\pi(\phi) = E_{s_t \sim D}(E_{a_t \sim \pi_\phi}(\alpha \log(\pi_\phi(a_t|s_t) - Q_\theta(s_t, a_t))) \tag{2-38}$$

2-1-11 Constrained Policy Optimization

Constrained Policy Optimization (CPO) is a trust region method for constrained RL which approximately enforces the constraints in every policy update. It could constrained reinforcement learning with guarantees for near-constraint satisfaction at each iteration.

The objective function is :

$$\begin{aligned} &maximize_\theta(E_{a \sim \pi_{\theta_{old}}} \frac{\pi_{\theta_{new}(a|s)}}{\pi_{\theta_{old}}(a|s)} A_{\theta_{old}}(s, a)) \\ &\text{subject to } D_{KL}^{mean}(\theta_{old}, \theta)) \leq \sigma, \\ &\text{and } J_{c_i}(\pi_{\theta_{old}}) + \frac{1}{1-\gamma}[E_{a \sim \pi_{\theta_{old}}} \frac{\pi_{\theta_{new}(a|s)}}{\pi_{\theta_{old}}(a|s)} A_{\theta_{old}}(s, a)] \leq d_i \ \forall i \end{aligned} \tag{2-39}$$

Where the first term is same with the TRPO or PPO, the second term is the policy's KL divergence and the third term is the constrain term.

2-2 Control Theory

2-2-1 Stable

Consider an autonomous nonlinear dynamical system

$$\dot{x} = f(x(t)), x(0) = x_0 \tag{2-40}$$

Suppose f has an equilibrium at x_e so that $f(x_e) = 0$ then

This equilibrium is said to be Lyapunov stable, if , for every ϵ, there exists a $\delta > 0$ such that, if $||x(0) - x_e|| < \delta$, then for every $t \geq 0$ we have $||x(t) - x_e|| < \epsilon$

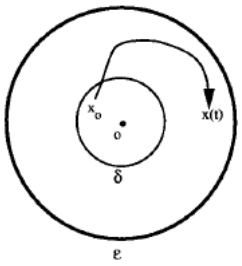


Figure 2-9: Lyapunov stable

2-2-2 Asymptotic stable

Consider an autonomous nonlinear dynamical system

$$\dot{x} = f(x(t)), x(0) = x_0 \tag{2-41}$$

Suppose f has an equilibrium at x_e so that $f(x_e) = 0$ then

This equilibrium is said to be asymptotic stable, if it is stable and there exists there exists a $\delta > 0$ such that if $||x(0) - x_e|| < \delta$, then $\lim_{t\to\infty} \|x(t) - x_e\| = 0$

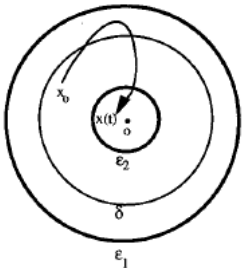


Figure 2-10: Asymptotic stable

2-2-3 Uniformly bounded

The system is said to be uniformly bounded in mean square if there exists $\epsilon > 0$ and there exists a $d(\epsilon) < \infty$, such that for every $t > t_0$, we have

$$\|x(t_0)\| < \epsilon \implies \|x(t)\| < d(\epsilon) \tag{2-42}$$

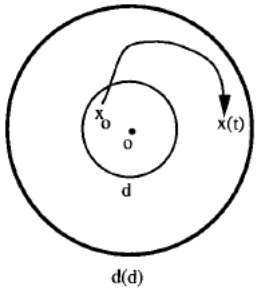


Figure 2-11: Uniformly bounded

2-2-4 Ultimately uniformly bounded

The system is said to be uniformly bounded in mean square if there exists $\epsilon > 0$ and there exists a $W \subset R^n$, such that for every $t > t_0 + T$, we have

$$\|x(t_0)\| < \epsilon \implies \|x(t)\| \in W \tag{2-43}$$

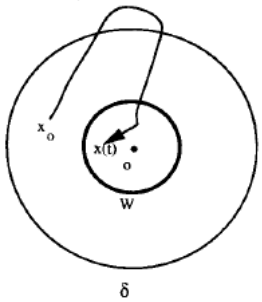


Figure 2-12: Ultimately uniformly bounded

2-3 Auto-encoder

Auto-encoders are an unsupervised learning technique which consists two parts, an encoder and a decorder. Both of these two parts are represented by a neural network. The general structure shows below 2-13.

Basically, the encoder works like PCA, which could extract features. And the performance of the extraction is evaluated by the decoder. The decoder aims to reconstruct the input from the information after encoder. By training encoder and decorder together, the network could learn a good representation of the input.

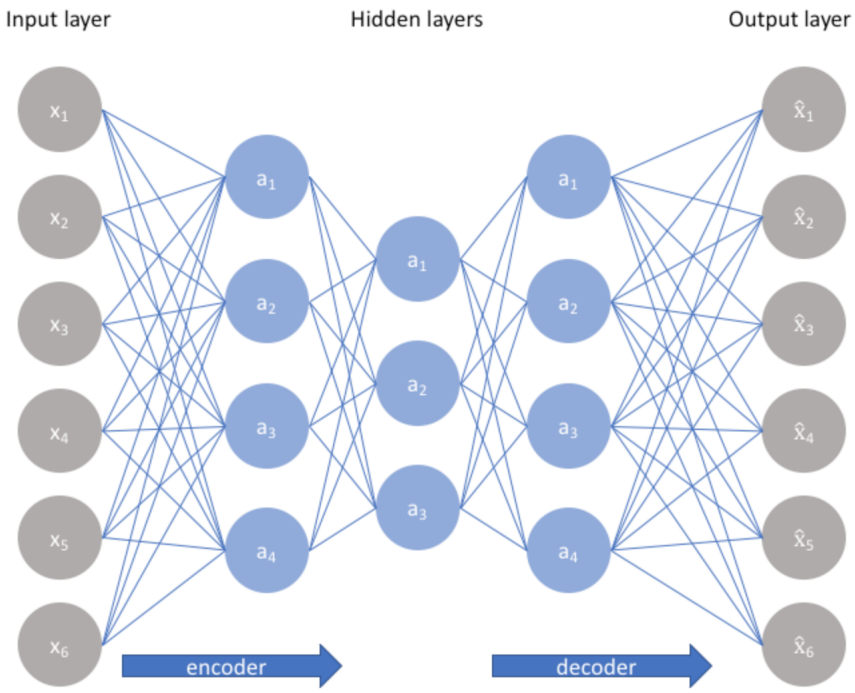


Figure 2-13: Auto-encoder structure

2-3-1 Variational Autoencoder

An Auto-encoder could find some latent state representation of that data. A variational autoencoder (VAE) provides a probabilistic apsect for describing an observation in latent space, see Fig 2-14 below. The encoder encodes the input to a distribution. And the decoder aims to sample a data from a distribution then decode it to original input.

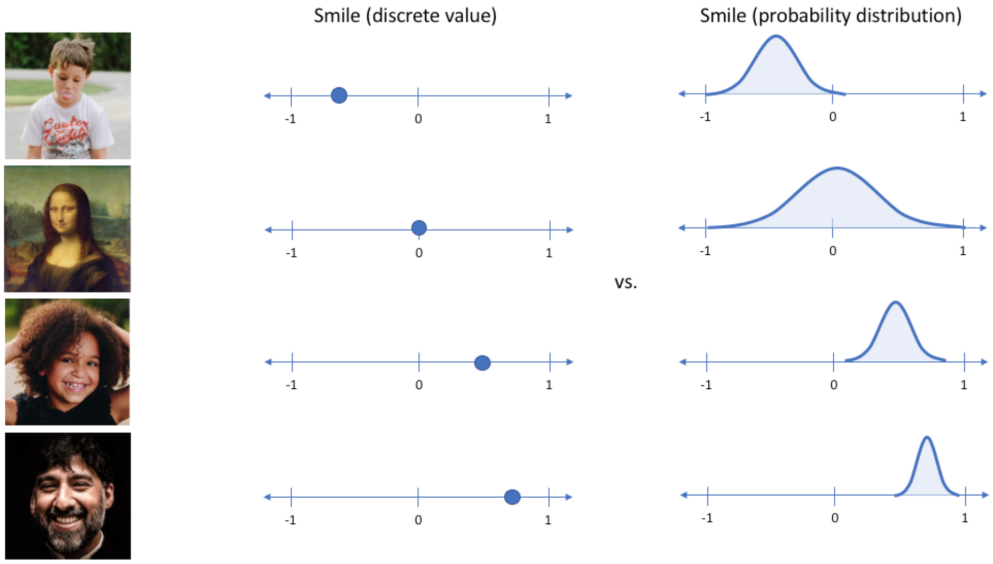


Figure 2-14: AE VS. VAE

Chapter 3

Preliminaries

This chaper will introduce to the notations and concepts of this thesis , which includes the control theory and reinforcement learning.

3-1 MDP and CMDP

A Markov decision process (MDP) is a tuple, (S, A, c, P, ρ), where S is the set of states, A is the set of actions, $c(s,a) \in [0, \infty)$ is the cost function, $P(s'|s,a)$ is the transition probability function, and $\rho(s)$ is the starting state distribution. The CMDP model is a extension of MDP, where the tuple consists of (S, A, r, P, b, ρ). $b(s,a) \in [0, \infty)$ is the constraint function which we want to keep lower than safety threshold $\overline{d}$, while $r(s,a)$ is the reward function. $\pi(a|s)$ is a stationary policy denoting the probability of selecting action a in state s. In addition, the cost functions under stationary policy are defined as $c_\pi(s) \doteq \mathbb{E}_{a\sim\pi} c(s,a)$ and $b_\pi(s) \doteq \mathbb{E}_{a\sim\pi} b(s,a)$ correspondingly.

In RL, we aim to find a policy π which minimizes $J(\pi)$, $J_c(\pi) \doteq \pi \sum_{t=0}^{\infty} \gamma^t c(s_t, a_t)$. Accordingly, for the CMDP problem, the objective is to maximize $J(\pi) \doteq \pi \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)$ while keeping b under $\overline{d}$, which we select to be zero in this paper. Here $\gamma \in [0, 1)$ is the discount factor, τ denotes a trajectory ($\tau = (s_0, a_0, s_1, ...)$), and $\tau \sim \pi$ is shorthand for indicating that the distribution over trajectories depends on π: $s_0 \sim \rho$, $a_t \sim \pi(\cdot|s_t)$, $s_{t+1} \sim P(\cdot|s_t, a_t)$. Letting $C(\tau)$ denote the discounted cost of a trajectory, we express the on-policy value function as $V^\pi(s) \doteq \mathbb{E}_{\tau\sim\pi}[C(\tau)|s_0 = s]$ and the on-policy action-value function as $Q^\pi(s,a) \doteq \mathbb{E}_{\tau\sim\pi}[C(\tau)|s_0 = s, a_0 = a]$. The advantage function is $A^\pi(s,a) \doteq Q^\pi(s,a) - V^\pi(s)$.

Also of interest is the discounted future state distribution, d^π, defined as $d^\pi(s) = (1 - \gamma) \sum_{t=0}^{\infty} \gamma^t P(s_t = s|\pi, \rho)$. It allows us to express the performance measure in the form of $J_c(\pi) = \mathbb{E}_{s\sim d^\pi} c(s,a)$

3-2 Stability

Among the various kinds of stability, we found that the following two definitions are particularly useful for the general class of learning control problems.

First, the so-called *mean square stability* (MSS) is generally used in the study of stochastic nonlinear systems[25]. In control theory, MSS implies that expectation over the norm of state converges to zero as time approaches infinity. For physical plants like drones and spacecrafts, MSS implies that the drones and spacecrafts will converge to the origin of the state space eventually, which is called *stabilization* in control theory. This is generalized to handle the tracking tasks for autonomous vehicles and field robots, where the norm of the position difference from the desired way point is considered instead.

Obviously, the classic definition of MSS is not directly applicable control tasks such as Atari games or Go. Thus We extend the MSS to the more general class of problems by replacing the measure of norm with a semi-definite cost function as following,

Definition 1. *Suppose $c_\pi(\cdot)$ is the cost function, $c_\pi : S \to \mathbb{R}_+$. The system is said to be mean square stable (MSS)* $\lim_{t\to\infty} \mathbb{E}_{s_t} c_\pi(s_t) = 0$ *holds for any initial condition s_0.*

The meaning of the definition above varies with the different choices of $c(\cdot)$. When chosen to be the norm $\|\cdot\|_2$, it still makes sense for systems involving physical dynamics. For the tasks like *Breakout* and *Pac-Man*, the cost could be chosen to be the remaining amount of bricks and *Pac-dots*. In these scenarios, MSS implies that the task converges towards the state where no bricks or dots remain.

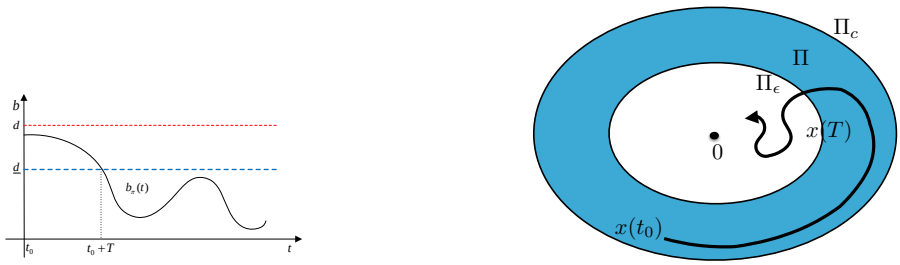
Similarly, for the scenarios where constraints exist, maintaining the safety cost under a certain thresholds rather than zero is also needed. Thus we introduce the following concept about *Ultimately Uniformly Boundness*,

Definition 2. *The system is said to be ultimately uniformly bounded (UUB) in mean square if there exists $\underline{d} > 0$ and for every $d \in (0, c)$, there exists $T = T(d, \underline{d})$ independent of t_0, such that $\mathbb{E}_{s_0} b_\pi(s_0) \leq d \implies \mathbb{E}_{s_t} b_\pi(s_t) \leq \underline{d}, \forall t \geq t_0 + T$.*

In the definition above , the term *uniform* indicates that the bound $\underline{d}$ does not depend on t_0. The term *ultimate* indicates that boundedness holds after the lapse of a certain time T. The constant d defines a neighborhood of the origin, independent of t_0, such that all trajectories starting in the neighborhood will remain bounded in time. If c can be chosen arbitrarily large then the UUB notion becomes global.

UUB suggests that the state converges to and stays inside the level set determined by the cost function. For tasks with the presence of constraints, UUB means the state will be driven into the so-called *safe set* where the constraints are satisfied. Take the *Pong* with Catastrophe Zone [2] as an example, the board will not crash into the obstacle if UUB holds. For the real-world applications like agriculture drone or autonomous vehicle, UUB guarantees that the machine will not leave the working area or road.

The concept of UUB and the conceptual idea on how Lyapunov analysis can be used to study UUB are illustrated Fig 3-1 below:



(a) UUB concept in Definition 2

(b) Lyapunov analysis in UUB

Figure 3-1: UUB Concept and Lyapunov Analysis. To demonstrate how Lyapunov analysis can be used to study UUB, consider a continuously differentiable positive definite function $L(x)$. Choose $0 < \epsilon < c$. Suppose that the sets $\Pi_\epsilon = \{L(x) \leq \epsilon\}$ and $\Pi_c = \{L(x) \leq c\}$ are compact. Let $\Pi = \{\epsilon \leq L(x) \leq c\} = \Pi_c - \Pi_\epsilon$ and suppose that it is known that the time derivative of $L(x(t))$ along the trajectories of the nonautonomous dynamical system is negative definite inside Π, that is $\dot{L}(x(t)) \leq -W(x(t)) < 0$, $\forall x \in \Pi, \forall t \geq t_0$ where $W(x(t))$ is a continuous positive definite function. Since $\dot{L}$ is negative in Π, a trajectory starting in Π must move in the direction of decreasing $L(x(t))$. In fact, it can be shown that in the set Π the trajectory behaves as if the origin was uniformly asymptotically stable. Consequently, the function $L(x(t))$ will continue decreasing until the trajectory enters the set Π_ϵ in finite time and stays there for all future time. Therefore, the solution of the dynamical system are UUB with the ultimate bound $b = \max_{x \in \Pi_\epsilon} \|x\|$

Chapter 4

Proposed Approach

In this charpter, we give the practical RL algorithms with stability guarantees given in Appendix. Based on the result in Appendix A-1, we formulate an Actor-Critic style learning algorithm, called the Lyapunov-based Actor Critic (LAC), to solve the general MDP problems. Based on the result in Section A-2, we combine with various popular off-policy and on-policy RL algorithms, to solve the CMDP problems.

Besides, to address the problem that some constraints are hard to define, we develop a novel method to model a constraint by conditional representation model and define the Wassertein constraint.

4-1 Off-policy algorithm

Off-policy algorithms are capable of learning from past experience, thus possesses higher sample efficiency compared with on-policy methods. DDPG [26] is a well known off-policy actor-critic method, which is capable of dealing a large class of continuous control tasks. A lately proposed actor-critic method SAC [27], based on the maximum entropy framework, outperformes DDPG and other policy gradient methods in a series of complex control tasks [27]. In our experiment, we adopt SAC as the benchmark for comparison and the baseline for developing safety guaranteed learning algorithm. Meanwhile, we also give a DDPG-based policy gradient method with safety guarantee in Algorithm 1.

First, based on the maximum entropy actor-critic framework, we use the Lyapunov function as the critic in the policy gradient formulation. In this algorithm, a critic Lyapunov function L_c is needed, which satisfies $L(s) = \mathbb{E}_{a\sim\pi} L_c(s,a)$. The objective function $J(\pi)$ is given as follow,

$$J(\pi) = \mathbb{E}_{(s,a,s',c)\sim\mathcal{D}} \left[\beta \log(\pi_\theta(f_\theta(\epsilon, s)|s)) + \lambda(L_c(s', f_\theta(\epsilon, s')) - L_c(s,a) + \alpha_3 c(s,a))\right] \tag{4-1}$$

where π_θ is parameterized by a neural network f_θ, ϵ is an input vector consisted of Gaussian noise. In the above objective, β and λ are positive variables which control the relative importance of policy entropy versus energy decreasing constraint. As in [27], the entropy of

policy is expected to remain above the target entropy $\mathcal{H}_t$. Both the values of β and λ are adjusted through gradient method, where the gradient of (4-1) is approximated by

$$\nabla_\theta J(\pi) = \nabla_\theta \beta \log(\pi_\theta(a|s)) + \nabla_a \beta \log(\pi_\theta(a|s)) \nabla_\theta f_\theta(\epsilon, s) + \lambda \nabla_{a'} L_c(s', a') \nabla_\theta f_\theta(\epsilon, s'). \tag{4-2}$$

For the CMDP problems, we combine our approach with the SAC through Lagrangian method. We call the combined algorithm Lyapunov-based soft actor critic (LSAC), of which the objective function for policy update is

$$\begin{aligned} J(\pi) = \ & \mathbb{E}_{(s,a,s')\sim\mathcal{D}} \left[\beta[\log(\pi_\theta(f_\theta(\epsilon, s)|s)) + \mathcal{H}_t] - Q(s, f_\theta(\epsilon, s))\right] \\ & + \mathbb{E}_{(s,a,s',b)\sim\mathcal{D}_e} \left[\lambda(L_c(s', f_\theta(\epsilon, s')) - L_c(s, a) + \alpha_3 b(s, a))\right] \end{aligned} \tag{4-3}$$

Since the constraint (A-9) in Theorem 3 on UUB is built upon the samples from the edge set Δ, an edge memory buffer $\mathcal{D}_e$ is needed. Following a similar derivation as (4-2), the gradient estimator is obtained. Due to space limitation, the update details of Q function and Lyapunov function, as well as β and λ, are referred to Algorithm 2 and Algorithm 3 , along with the pseudo-codes of LAC and LSAC.

Algorithm 1 Lyapunov-based Deep Deterministic Policy Gradient (LDDPG)

Initialize the edge set Δ, replay buffer R and edge replay buffer R_c, and the noise distribution $\epsilon \sim \mathcal{N}$, Lagrangian multiplier λ

Randomly initialize a critic network $Q(s,a)$, Lyapunov critic network $L_c(s,a)$, actor $\mu(s)$ with parameters ϕ_Q, ϕ_{L_c}, θ

Initialize the parameters of target networks with $\overline{\phi}_Q \leftarrow \phi_Q$, $\overline{\phi}_{L_c} \leftarrow \phi_{L_c}$, $\overline{\theta} \leftarrow \theta$

for each episode **do**

Sample s_0 according to ρ

for each time step **do**

Sample a_t from $\mu(s) + \epsilon$ and step forward

Observe s_{t+1}, r_t, b_t and store $(s_t, a_t, r_t, b_t, s_{t+1})$ in R

if $s_t \in \Delta$, store $(s_t, a_t, r_t, b_t, s_{t+1})$ in R_c

Sample minibatches of N and N_c transitions from R and R_c

Update Q by minimizing the TD-error $\delta = \frac{1}{N}\sum_i (r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1})) - Q(s_i, a_i))^2$

Update L_c by minimizing $\delta_L = \frac{1}{N}\sum_i (b_i + \gamma L_c'(s_{i+1}, \mu'(s_{i+1})) - L_c(s_i, a_i))^2$

Estimate policy gradient:

$$\nabla_\theta J \approx \nabla_\theta \Big(\frac{1}{N} \sum_i^N Q(s_i, \mu(s_i)) - L_\lambda \Big)$$

where

$$L_\lambda \doteq \frac{1}{N_c} \sum_j^{N_c} \lambda \left[L_c(s_{j+1}, \mu(s_{j+1})) - L_c(s_j, a_j) + \alpha_3 b_t \right]$$

Update actor and Lagrangian multiplier:

$$\begin{aligned} \theta &\leftarrow \theta + \alpha \nabla_\theta J \\ \lambda &\leftarrow \max\left(0, \lambda + \alpha_\lambda L_\lambda\right) \end{aligned}$$

Update the target networks:

$$\begin{aligned} \overline{\phi}_Q &\leftarrow \tau \phi_Q + (1-\tau)\overline{\phi}_Q \\ \overline{\phi}_{L_c} &\leftarrow \tau \phi_{L_c} + (1-\tau)\overline{\phi}_{L_c} \\ \overline{\theta} &\leftarrow \tau \theta + (1-\tau)\overline{\theta} \end{aligned}$$

end for

end for

4-1-1 Lyapunov-based Actor Critic

We use the following as the objective function for updating the Lyapunov critic,

$$J(L_c) = \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[\frac{1}{2}(L_c(s,a) - L_{target}(s,a))^2\right] \tag{4-4}$$

where L_{target} is the approximation target for L_c. For the case that use cost as Lyapunov function, $L_{target}(s,a) = c(s,a)$; for the case of value function as Lyapunov function,

$$L_{target}(s,a) = c(s,a) + \gamma L_c'(s', f(\epsilon, s')) \tag{4-5}$$

where L_c' is the target Lyapunov critic function as typically used in the actor critic methods [27, 26], which has the same structure with L_c but the parameter is updated through the exponentially moving average of weights of L_c controlled by a hyperparameter τ.

The value of temperature β and Lagrangian multiplier λ are automotive, adjusted by the policy gradient method optimizing the following objectives, respectively,

$$J(\beta) = \mathbb{E}_{(s,a)\sim\mathcal{D}} - \beta[\log(\pi_\theta a|s)) + \mathcal{H}_t] \tag{4-6}$$

$$J(\lambda) = \mathbb{E}_{(s,a)\sim\mathcal{D}} - \lambda[L_c(s', f_\theta(\epsilon, s')) - L_c(s,a) + \alpha_3 c(s,a)] \tag{4-7}$$

Algorithm 2 Lyapunov-based Actor Critic (LAC)

Initialize replay buffer R and the Lagrangian multiplier λ
Randomly initialize Lyapunov critic network $L_c(s,a)$, actor $\pi(a|s)$ with parameters ϕ_{L_c}, θ
Initialize the parameters of target networks with $\overline{\phi}_{L_c} \leftarrow \phi_{L_c}$, $\overline{\theta} \leftarrow \theta$
for each iteration **do**
  Sample s_0 according to ρ
  for each time step **do**
    Sample a_t from $\pi(s)$ and step forward
    Observe s_{t+1}, r_t, b_t and store $(s_t, a_t, r_t, b_t, s_{t+1})$ in R
    if $s_t \in \Delta$, store $(s_t, a_t, r_t, b_t, s_{t+1})$ in R_c
  end for
  for each update step **do**
    Sample minibatches of N and N_c transitions from R and R_c
    Estimate policy gradient:

$$\begin{aligned}\phi_{L_c} &\leftarrow \phi_{L_c} + \alpha_{\phi_{L_c}} \nabla_{\phi_{L_c}} J(L_c)\\ \theta &\leftarrow \theta + \alpha_\theta \nabla_\theta J(\pi)\\ \lambda &\leftarrow \max\left(0, \lambda + \alpha_\lambda L_\lambda\right)\end{aligned}$$

    Update the target networks:

$$\begin{aligned}\overline{\phi}_Q &\leftarrow \tau\phi_Q + (1-\tau)\overline{\phi}_Q\\ \overline{\phi}_{L_c} &\leftarrow \tau\phi_{L_c} + (1-\tau)\overline{\phi}_{L_c}\\ \overline{\theta} &\leftarrow \tau\theta + (1-\tau)\overline{\theta}\end{aligned}$$

  end for
end for

4-1-2 Lyapunov-based Soft Actor Critic

As in [27], the soft value function V is defined as

$$V(s) = \mathbb{E}_{a\sim\pi}(Q(s,a) - \beta log(\pi(a|s))) \tag{4-8}$$

which augments a standard value function with the entropy of policy at the given state. In practice, the Q-functions are updated by minimizing

$$J(Q) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}} \frac{1}{2}\left[r(s,a) + \gamma V(s') - Q(s,a)\right]^2 \tag{4-9}$$

In our implementation, two soft Q-functions $\{Q_1, Q_2\}$ are used and the target networks $\{Q_1', Q_2'\}$ are constructed accordingly. Following the objective function of actor (4-3), the gradient estimator in the double Q-function implementation is obtained as

$$\begin{aligned}\nabla_\theta J(\pi) =& \mathbb{E}_{\mathcal{D}}[\nabla_\theta \beta \log(\pi_\theta(a|s)) + \nabla_a(\beta\log(\pi_\theta(a|s)) - \min_i Q_i(s,a))\nabla_\theta f_\theta(\epsilon', s)] +\\ & \mathbb{E}_{\mathcal{D}_c}[\lambda \nabla_{a'} L_c(s', a')\nabla_\theta f_\theta(\epsilon, s')]\end{aligned} \tag{4-10}$$

The policy gradient is composed of two parts, the gradient estimated by the Q-function based on the samples from replay buffer $\mathcal{D}$, and that estimated by the Lyapunov critic based on the samples from the edge buffer $\mathcal{D}_c$. In a lot tasks, the violation of constraints only happens after the agent mastering certain level of skills, such as running forward in the HalfCheetah-Safe task. This implies that with a randomly initialized policy, the content of $\mathcal{D}_c$ hardly grows while $\mathcal{D}$ is ready for the updates. Thus, to overcome this inconvenience, we setup a initial objective function $\hat{J}(\pi)$ by setting the λ to zero until the $\mathcal{D}_c$ has stored enough transitions. In practice, we find this to be data-efficient and does not endanger the agent. Another point to be noted is that the policy gradient is estimated by the Q-function with lower value, which is found to be useful in migrating performance degradation caused by the bias in the value estimation [27]. The updates of Lyapunov critic L_c, temperature β and multiplier λ are the same as in LAC subsection 4-1-1.

Algorithm 3 Lyapunov-based Soft Actor Critic (LSAC)

Initialize the edge set Δ, replay buffer R and edge replay buffer R_c, and the Lagrangian multiplier λ
Randomly initialize a critic network $Q(s,a)$, Lyapunov critic network $L_c(s,a)$, actor $\pi(a|s)$ with parameters ϕ_Q, ϕ_{L_c}, θ
Initialize the parameters of target networks with $\overline{\phi}_Q \leftarrow \phi_Q$, $\overline{\phi}_{L_c} \leftarrow \phi_{L_c}$, $\overline{\theta} \leftarrow \theta$
for each iteration **do**
  Sample s_0 according to ρ
  for each time step **do**
    Sample a_t from $\pi(s)$ and step forward
    Observe s_{t+1}, r_t, b_t and store $(s_t, a_t, r_t, b_t, s_{t+1})$ in R
    if $s_t \in \Delta$, store $(s_t, a_t, r_t, b_t, s_{t+1})$ in R_c
  end for
  for each update step **do**
    Sample minibatches of N and N_c transitions from R and R_c

$$\phi_Q \leftarrow \phi_Q + \alpha_{\phi_Q} \nabla_{\phi_Q} J(Q)$$
$$\phi_{L_c} \leftarrow \phi_{L_c} + \alpha_{\phi_{L_c}} \nabla_{\phi_{L_c}} J(L_c)$$
$$\theta \leftarrow \theta + \alpha_\theta \nabla_\theta J(\pi)$$
$$\lambda \leftarrow \max\left(0, \lambda + \alpha_\lambda L_\lambda\right)$$
$$\beta \leftarrow \beta + \alpha_\beta \nabla_\beta J(\beta)$$

    Update the target networks:

$$\overline{\phi}_Q \leftarrow \tau\phi_Q + (1-\tau)\overline{\phi}_Q$$
$$\overline{\phi}_{L_c} \leftarrow \tau\phi_{L_c} + (1-\tau)\overline{\phi}_{L_c}$$
$$\overline{\theta} \leftarrow \tau\theta + (1-\tau)\overline{\theta}$$

  end for
end for

4-2 On-policy algorithm

A on-policy Lyapunov-based safe RL algorithms will be formulated, i.e., Lyapunov-based proximal policy optimization (LPPO).

4-2-1 Lyapunov-based Proximal Policy Optimization

In PPO, the clipped surrogate objective is designed as

$$J_{clip}(s,a) = \min(r(\theta)A^{\pi}(s,a), clip(r(\theta), 1-\delta, 1+\delta)A^{\pi}(s,a)) \tag{4-11}$$

Where $r(\theta) = \frac{\pi_{\theta}(a|s)}{\pi_{\theta_old}(a|s)}$, θ_{old} is the parameter of policy π of the last update. And where $\delta \in [0,1)$ is the clip range for the importance coefficient $r(\theta)$.

For the LPPO variant, following similar idea as in LSAC, we combine our approach with PPO [28] through Lagrangian method, forming a soft constrained objective function as follows

$$J(\pi_{\theta}) = \mathbb{E}_{\tau}\left[-J_{clip}(s,a) + \lambda(r(\theta)L(s') - L(s) + \alpha_3 b_{\pi}(s))\right] \tag{4-12}$$

As the LPPO is based on the result from 1, the safety constraint is implemented globally on the state space, ignoring the value of safety cost b. This could lead the policy update to be overly conservative. To mitigate the conservatism caused by global constraint, we use introduce an adaptive $alpha_3$, of which the value starts from a small positive number and gradually climbs to the designed value. This is reasonable since on general, the value of $alpha_3$ only determines whether the constrained problem is feasible and the robustness of validity of constraints in presence of estimation error, and the magnitude is not related to the stability guarantee. The update of α_3 follows the rule below:

$$\begin{aligned} &\textbf{if } \sum_{\tau} b(s,a) > 0, \alpha_3^{k+1} \leftarrow \min(1.5 * \alpha_3^k, \overline{\alpha}_3) \\ &\textbf{else}, \alpha_3^{k+1} \leftarrow \min(1.01 * \alpha_3^k, \overline{\alpha}_3) \end{aligned} \tag{4-13}$$

where $\overline{\alpha}_3$ is the designed upper bound. We found this technique useful in improving the performance while guaranteeing safety as using constant α_3 in LPPO.

It should be noted that this variant is based on the result from Theorem 1, i.e., suppressing the b_{π} to zero. Thus the system is guaranteed to converge to and stay inside a safe region $\{s \in \mathcal{S} | b_{\pi}(s) = 0\}$. Specifically, we use the clipped version of PPO, of which the pseudo-code and other details are referred to Algorithm 4.

Algorithm 4 Lyapunov-based Proximal Policy Optimization (LPPO)

Initialize Lagrangian multiplier λ
Randomly initialize a critic network $V(s)$, Lyapunov critic network $L(s)$, policy $\pi(a|s)$ with parameters ϕ_V, ϕ_L, θ
for each episode **do**
Collect a set of trajectories $\mathcal{D} = \tau_i$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
Compute return estimates

$$\hat{R}_t = r_t + \gamma V(s_{t+1})$$

Compute advantage estimates, $\hat{A}_t$ using GAE based on current value function V_ϕ
Update the policy by with gradient of the LPPO-clip objective (**??**):

$$\theta \leftarrow \theta + \alpha_\theta \nabla_\theta J(\pi)$$

Fit value function by regression on mean-squared error:

$$\phi_V \leftarrow \phi_V + \alpha_{\phi_V} \nabla_{\phi_V} \mathbb{E}_{\mathcal{D}} \frac{1}{2} (V_{\phi_V}(s) - \hat{R}_t)^2$$

Fit Lyapunpv function by regression on mean-squared error:

$$\phi_L \leftarrow \phi_L + \alpha_{\phi_L} \nabla_{\phi_L} \mathbb{E}_{\mathcal{D}} \frac{1}{2} (L_{\phi_L}(s) - L_{target})^2$$

Update Lagrangian multiplier:

$$\lambda \leftarrow \max(0, \lambda + \alpha_\lambda L_\lambda)$$

Update α_3 as Equation 4-13
end for

4-3 Lagrangian-based Safe Algorithm

We use two Lagrangian based methods as baseline in our experiments, namely the safe soft actor critic (SSAC) and safe proximal policy optimization (SPPO). Both of these approaches attempts to solve the following unconstrained optimization problem

$$\min_{\pi} \max_{\lambda \geq 0} \mathcal{L}(\pi, \lambda) = \mathbb{E}_{\tau \sim \pi}[\mathcal{C}(\tau) + \lambda(\mathcal{D}(\tau) - \overline{d})] \tag{4-14}$$

where $\mathcal{D}(\tau)$ denotes the discounted sum of safety cost over the trajectory τ and $\overline{d}$ is the safety threshold.

For the SSAC variant, the Lagragian function $\mathcal{L}(\pi, \lambda)$ in (4-14) is modified as the following,

$$\mathcal{L}(\pi, \lambda) = \mathbb{E}_{(s,a,s') \sim \mathcal{D}} \Big[\beta[\log(\pi_\theta(f_\theta(\epsilon, s)|s)) + \mathcal{H}_t] - Q(s, f_\theta(\epsilon, s)) + \lambda(Q_c(s, a) - \overline{d}) \Big] \tag{4-15}$$

where Q_c is the safety Q-function. By employing the same PG algorithm as in SAC, a safe policy is obtained. The temperature β and Lagrangian multiplier are updated in the same way as (4-6) and (4-7) in LAC.

For SPPO, the Lagragian function is as following,

$$\mathcal{L}(\pi, \lambda) = \mathbb{E}_\tau \left[J_{clip}(s, a) + \lambda(r(\theta) A_c(s, a) + \mathcal{D}(\tau) - \overline{d}) \right] \tag{4-16}$$

where A_c is the estimated safety advantage through the generalized advantage estimator (GAE) [29].

4-4 Model the constraint

However, in many complex tasks, the constraints are hard to define or difficult to design a sophisticated cost function, or even the constraints are dynamic. For example, a vehicle on roundabout, the constraint are dynamic and uncertain, it is hard to define a proper constraint or even we do not know what should be the constraint. Maybe we finally define the constraints are the distance between vehicles and velocity differences between others, we still need to tune the coefficients.

When humans driving a car, they do not know a specific constraint, but still perform well. Understanding the latent information of situation and awareness whether is safety or not is one of the key capabilities of humans which we heavily rely on to make decisions in daily life. A model that can accurately awareness current situation must internally represent the complex dynamics and covering enough latent information like constraints. Furthermore, such models can be inherently useful for reinforcement learning, for example, to allow an agent to decide how to interact with the world to bring about a desired outcome. Most importantly, the representation model, including the action-conditioned settings and dynamics uncertainty, is in fact not deterministic, and a deterministic model can lose many of the nuances. So, the model should be a distributional model.

To represent constraints by distribution, we propose a novel Conditional Representation Model based on conditional variantional auto-encoder (CVAE) for input representation learning. In other words, we model the distribution of the latent of the input, which contains the constraints information. The CVAE is a conditional directed graphical model whose input observations modulate the prior on Gaussian latent variables that generate the outputs. It is trained to maximize the conditional log-likelihood. And we use Wasserstein distance to measure the distance between constraints.

4-4-1 Conditional Representation Model

In Conditional Graphical Model, there are three types of variables in a conditional representation model (CRMs): input variables x, conditional variables c, and latent variables z. We use a same structure as VAE to train our model. The representation model encode the input x into latent z, and the reconstruct model decode the sample of the latent distribution to original. The difference is we assume $p(z)$ is a isotropic Gaussian distribution $N(c, 1)$. The CRMs are trained to maximize the conditional log-likelihood. Often the objective function is

intractable, and we apply the SGVB framework to train the model. The variational lower bound of the model is written as follows(complete derivation will be done soon):

$$logp_\theta(x) = -KL(q_\phi(z|x)||p_\theta(z|c)) + \mathbb{E}_{q_\phi}[logp_\theta(x|z)] \tag{4-17}$$

And the empirical lower bound is written as:

$$L(x;\theta,\phi) = -KL(q_\phi(z|x)||p_\theta(z|c)) + \frac{1}{L}\sum_{l=1}^{L} logp_\theta(x|z^{(l)}) \tag{4-18}$$

The difference between some CVAE is we do not use any other parameters during inference. The conditional variables only work on training stage. During inference time, our model could encode the input to distinguishable representations.

Basically, the CRM could represent different inputs into different distributions and we could sample from these priors to generate new situations.

4-4-2 Wasserstein Constraint

Wasserstein distances are metrics between probability distributions that are inspired by the problem of optimal transportation. Wasserstein distances can be used to derive weak convergence and convergence of moments, and can be easily bounded; they are well-adapted to quantify a natural notion of perturbation of a probability distribution.

The p-Wasserstein1 distance between probability measures μ and ν on $\mathbb{R}^d$ is defined as:

$$W_P(\mu,\nu) = inf_{X\sim\mu,Y\sim\nu}(\mathbb{E}||\mathbb{X}-\mathbb{Y}||')^{1/p} \tag{4-19}$$

As the constraints is represented by a Gaussian distribution, the current Wasserstein distance between the constraints is:

$$W_2^2(X,Y) = ||m_1 - m_2||^2 - tr[\Sigma_1 + \Sigma_2 - 2(\Sigma_1^{1/2}\Sigma_2\Sigma_1^{1/2})] \tag{4-20}$$

Chapter 5

Experiments and Results

In our experiments, We aim to answer the following questions:

Based on the stability definitions in Section 3-2, how does our policy with stability guarantee ensure the system to recover to normal status under external perturbations?

How does our approach perform compared with a baseline in the continuous control tasks in terms of convergence speed and performance measure in the presence of stability guarantee? What's the influence of using different Lyapunov candidates?

How does our approach perform compared with other safe learning algorithms in CMDP tasks?

Does the partially constrained method (LSAC) outperform the globally constrained method (LPPO) in terms of safety and performance measure? Does our approach maintain safety in presence of function approximation error?

We designed ten MDP and CMDP experiments that are easy to interpret and motivated by these questions. These experiments involved many kind of environment. The very classic control task like CartPole, and some reinforcement learning benchmark experiment like Mujoco and Atari, as well as some robotics control task demo like UAV and UGV demos. Most importantly, all the experiments are reproducible.

5-1 Experiment environment

All the experiments are in OpenAI Gym structure.

5-1-1 CartPole Balancing

Controlling a CartPole has been widely analyzed in reinforcement learning and control literature[30] [31][32][33]. So we design this experiment . In this experiment, we aim to balance a Cartpole in a given location and keep it vertical absolutely instead of repeatedly moving left and right to prevent falling down.

To achieve this, we build a new environment. We used the OpenAI Gym's CartPole's dynamic model but change its action space from discrete to continuous, which used to be discrete , only support -10N or 10N. Now the action is a horizontal force on cart, and the range is in [-20N, 20N]

Figure 5-1: CartPole Balancin

For this task, we designed the a new reward function according to current position and angle, higher reward means that the cart is closer to the target position and the pole stands more straightly. The $x_{\text{threshold}}$ and $\theta_{\text{threshold}}$ represents the boundary of position and angle respectively. The position range is [-10m, 10m], the $x_{\text{threshold}}$ is 10 and the $\theta_{\text{threshold}}$ is 40 deg. When the state achieves the threshold, an episode will be done.

For the MDP task, we want the Cart stay at middle of the slide way with the pole stand straightly. So the target position is 0 and the **Cost** is designed as :

$$c = (\frac{x}{10})^2 + 20(\frac{\theta}{\theta_{\text{threshold}}})^2 \tag{5-1}$$

For the CMDP task, we want the Cart stay at the right of the slide way with the pole stand straightly. So the we set the target position at 6m and the **Reward** is designed as :

$$\begin{aligned} r_1 &= \frac{x_{\text{threshold}}/10 - |x-6|}{x_{\text{threshold}}/10} \\ r_2 &= \frac{\theta_{\text{threshold}}/4 - |\theta|}{\theta_{\text{threshold}}/4} \\ r &= 20 * sign(r1) * r1^2 + sign(r2) * r2^2 \end{aligned} \tag{5-2}$$

And we set the constraint to be defined on the position limit. We constrain the position to be less than 4, and the **Safety Cost** is :

$$b = \frac{max(|x| - 0.8 * 4, 0)^2}{5} \tag{5-3}$$

For these experiments, the episodes are of length 2500, the maximum global steps is 6e5, maximum episodes is 1e6.

5-1-2 Point-Circle

Point-Circle is a well-know reinforcement learning CMDP task, which has been done in Constrained Policy Optimization [18] and Lyapunov-based Safe Policy Optimization for Continuous Control [19]. The agent moves a point mass by controlling the orientation and movement increment.

For this environment, we rewrite the rllab [1] and CPO [2] open source environment into Gym's Mujoco structure.

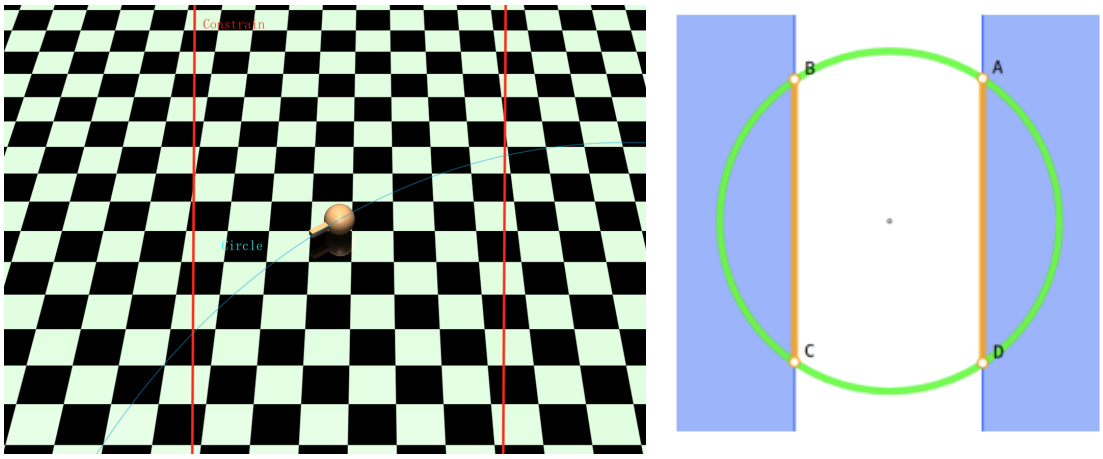


Figure 5-2: Point-Circle

And the task is moving the point mass counter-clockwise along a circle of radius 15m.

For CMDP task, we use CPO [18] default **Reward**, which is :

$$r = \frac{-y * v_x + x * v_y}{1 + |\sqrt{(x^2 + y^2)} - 15|} \tag{5-4}$$

And we set the constraint to be defined on the x-axis position limit. We constrain the x-axis position to be less than 3, and we designed the **Safety Cost** as :

$$b = max(|x| - 0.8 * 3, 0) \tag{5-5}$$

For this experiment, the episodes are of length 65, the maximum global steps is 1e6, maximum episodes is 1e6.

[1] https://github.com/rll/rllab
[2] https://github.com/jachiam/cpo

5-1-3 Ant-Safe

Ant is a classical reinforcement learning benchmark [34], the agent controls an ant (a 4-legged simulated robot) to run as fast as possible with lowest control cost and contact cost, which is a MDP task. And the Ant-Safe is a CMDP task, which has same goal but with constrains.

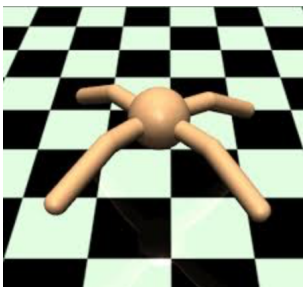


Figure 5-3: Ant-Safe

For CMDP task, we use we use CPO [18] default **Reward**, which is :

$$\begin{aligned} &reward_{forward} = v \\ &cost_{ctrl} = 0.5 * \sum a^2 \\ &cost_{contact} = 5e-4 * \sum clip(cfrc_{ext}, -1, 1)^2 \\ &reward_{survive} = 1 \\ &r = reward_{forward} - cost_{ctrl} - cost_{contact} + reward_{survive} \end{aligned} \tag{5-6}$$

where v is the speed of the ant's center of mass, a is the action. And we set the constraint to be defined on the speed limit. We constrain the speed to be less than 3, and the **Safety Cost** is :

$$b = max(|v| - 0.9 * 3, 0)^2 \tag{5-7}$$

For this experiment, the episodes are of length 200, the maximum global steps is 1e6, maximum episodes is 1e6.

5-1-4 HalfCheetah-Safe

HalfCheetah is a classical reinforcement learning benchmark [34], the agent controls a HalfCheetah (a 2-legged simulated robot) to run as fast as possible with lowest control cost, which is a MDP task. And the HalfCheetah-Safe is a CMDP task, which has same goal but with constrains.

For CMDP task, For CMDP task, we use we use CPO [18] default **Reward**, which is :

$$reward_{ctrl} = -0.1 \sum a^2 reward_{run} = vr = reward_{run} + reward_{ctrl} \tag{5-8}$$

where v is the speed of the HalfCheetah's center of mass, a is the action. And we set the constraint to be defined on the speed limit. We constrain the speed to be less than 3, , and the **Safety Cost** is :

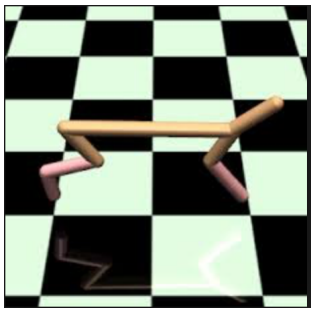


Figure 5-4: HalfCheetah-Safe

$$b = max(|v| - 0.9 * 3, 0)^2 \tag{5-9}$$

For this experiment, the episodes are of length 200, the maximum global steps is 1e7, maximum episodes is 1e6.

5-1-5 Pong-Safe

Pong is a classical reinforcement learning benchmark [34], which is an Atari 2600 game, the agent aims to get higher score. And Pong-Safe is our changed version from the standard Gym's Atari Task Pong-v0. In this environment, the observation is an RGB image of the screen, which is a data shaped in [210, 160, 3], and the action is the fixed discrete operation in the game.

And in this experiment, we defined that it's a catastrophe if the green paddle enters the Catastrophe Zone, see Fig 5-5 below:

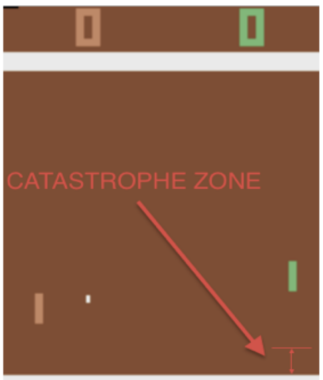


Figure 5-5: Catastrophe Zone [2]

To achieve this, we need to locate the green paddle's position. But the state is RGB image of the screen. So we use some simple computer vision technology to process the image and locate it (this step is only for obtain the constraint cost) : we first extract the grey-scale map from the RGB frame, and use a player paddle size filter to locate the player paddle. And we define the origin is the white line, and the Catastrophe Zone is y-axis position larger than 150.

The **Reward** is set as default. And we set the constraint to be defined on the green paddle y-axis position. We constrain the green paddle y-axis position to be less than 150, and the **Safety Cost** is :

$$b = max(y - 150, 0) \tag{5-10}$$

For this experiment, the episodes are of length 1e6, the maximum global steps is 2e7, maximum episodes is 1e6.

5-1-6 FetchReach-v1

FetchReach is a OpenAI Gym's robotics environment, which is an ingredient for robotics research. For this experiment, we use the default Gym's robotics environment. A goal position is randomly chosen in 3D space. The agent control Fetch's end effector to reach that goal as quickly as possible.

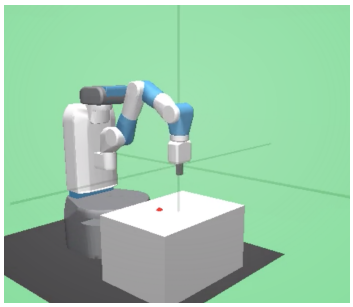


Figure 5-6: FetchReach

For MDP task, we use we use OpenAI's default **Cost**, which is :

$$c = d \tag{5-11}$$

where d is the distance between the target.

For these experiments, the episodes are of length 50, the maximum global steps is 3e5, maximum episodes is 1e5.

5-1-7 Car

Car environment is our changed version from a open source environment [3] into OpenAI Gym's structure. The task is controlling a car to drive on the road and avoid the wall according to sensor information.

Car has 5 sensors to obtain distance information. And each sensor provides the minimum distance between obstacles or the environment bound, and the intersections. The state is defined as : $s_t = [d, end_x, end_y]$, which is the the minimum distance between obstacle or the

[3] https://morvanzhou.github.io/tutorials/

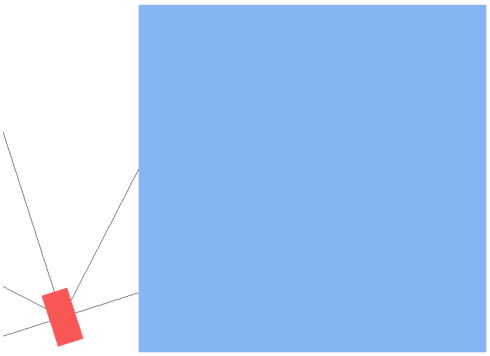


Figure 5-7: Car

environment bound, the possible intersection between obstacle and the possible intersection between environment bound. The car has constant velocity, and the agent only control the steering wheel's angle. The action range is [-1,1].

For MDP task, we hope it drive with out collision and with least actions. If collision, an episode will done, then the **Cost** is:

$$c = 1000 \tag{5-12}$$

And if not done, the **Cost** is :

$$c = a \tag{5-13}$$

where a is the actions.

For these experiments, the episodes are of length 600, the maximum global steps is 1e6, maximum episodes is 1e6.

5-1-8 Quadrotor

We transfer a open source Crazyflie simulator Matlab code [4] into OpenAI Gym's structure. The simulator has three main parts, the CrazyFlies physical parameters, the PD controller and the quadrotor equations of motion. The task is motion planning

We simplified the control problem in discretized time. The control proceeds as follows: The quadrotor simulator outputs the next state of the quadrotor given the force, torques and the current state. The control policy, implemented by a MLP, maps the observation of the current state to the desired step, and the desired state is the desired step added with the current state. Last, the PD controller converts the current state and desired state to force and torques, and the loop continues.

[4] https://github.com/yrlu/quadrotor

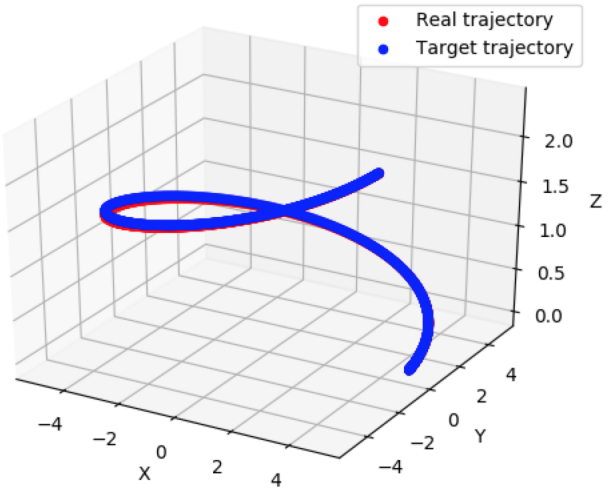


Figure 5-8: Quadrotor

The state describes the quadrotor's position, velocities, attitude ,angular velocities and target position, which is defined as $s_t = [x, y, z, \dot{x}, \dot{y}, \dot{z}, p, q, r, \dot{p}, \dot{q}, \dot{r}, x_{target}, y_{target}, z_{target}]$. The policy aims to plan the desired state, but as the state range is quite large, the desired state could not be too far from the current state. So, we set the policy output the desired step, which added with current state is desired state. And due to our goal is about the position, we only focus on the desired position and velocity and keep the same angle. So, the action represents the desired changes in position and velocity from current state, which is defined as $a_t = [\Delta x, \Delta y, \Delta z, \Delta \dot{x}, \Delta \dot{y}, \Delta \dot{z}]$. And an episode 'done' happened in the current position is too far from the target position. The target position is provided by a fine designed trajectory according to time.

For MDP task, the target position is 0. So the **Cost** is designed as :

$$c = ||d|| \tag{5-14}$$

where the d is the distance between the current position and target position.

For CMDP task, the **Reward** is designed as :

$$r = -||d|| + 1 \tag{5-15}$$

where the d is the distance between the current position and target position.

And we set the constraint to be defined on the z-axis position limit. We constrain the z-axis position to be less than 0.5, and the **Safety Cost** is :

$$b = 100 * max(|x| - 0.8 * 0.5, 0) \tag{5-16}$$

For these experiments, the episodes are of length 2000, the maximum global steps is 1e7, maximum episodes is 1e6.

5-1-9 CartPole stability against perturbations

This experiment is aiming to test the different policy's stability against perturbations. We use the same environment setting as the CMDP CartPole. To see whether the agent could recover to normal status from perturbations such as external forces and wind. We give the cart a large perturbation F (around 5 times of the input force) in the CartPole environment, and observe the behavior. The $\theta_{\text{threshold}}$ is 40 deg. If the pole's angle achieve the $\theta_{\text{threshold}}$ is dead, and we record the death rate during repeated experiment, the lower death rate the better the policy's stability is.

5-2 Hyperparameters

Here is the hyperparameters we used in all these experiments:

Hyperparameters	Point-Circle	Ant-Safe	HalfCheetah-Safe	Quadrotor	Pong-Safe
form of Lyapunov function	Cost	Cost	Cost	Cost	Cost
minibatch size	32	32	32	16	4
GAE parameter(λ)	0.95	0.95	0.95	0.95	0.95
ent.coef	0	0	0	0	0.01
learning rate	3e-4	3e-4	3e-4	1e-3	2.5e-4
discount(γ)	0.99	0.99	0.99	0.99	0.99
α_3	0.8*	0.2*	0.2*	0.2*	0.005

*α_3 is adaptive and initialized in 1e-9 and the upper bound is 0.2

Table 5-1: LPPO Hyperparameters used for CMDP experiments

Hyperparameters	Point-Circle	Ant-Safe	HalfCheetah-Safe	Quadrotor	Cartpole
Lyapunov function	Cost	Value	Value	Value/Cost	Value
Minibatch size	256	256	256	256	256
Actor learning rate	1e-4	1e-4	1e-4	1e-4	1e-4
Critic learning rate	3e-4	3e-4	3e-4	3e-4	3e-4
Lyapunove learning rate	3e-4	3e-4	3e-4	3e-4	3e-4
Target entropy	-2	-8	-6	-6	-1
Target smoothing coefficient(τ)	0.005	0.005	0.005	0.005	0.005
Discount(γ)	0.99	0.99	0.99	0.99	0.99
alpha3(fixed)	0.8	1	1	0.8	0.8

Table 5-2: LSAC Hyperparameters used for CMDP experiments

Hyperparameters	Quadrotor	Cartpole	FetchReach	Car
Lyapunov function	Value/Cost	Value/Cost	Value/Cost	Value/Cost
Minibatch size	256	256	256	256
Actor learning rate	1e-4	1e-4	1e-4	1e-4
Critic learning rate	3e-4	3e-4	3e-4	3e-4
Lyapunove learning rate	3e-4	3e-4	3e-4	3e-4
Target entropy	-6	-1	-4	-1
Soft replacement(τ)	0.005	0.005	0.005	0.005
Discount(γ)	0.99	0.99	0.99	0.99
α_3(fixed)	0.1/0.2	1/0.8	1/0.1	0.8/0.5

Table 5-3: LAC Hyperparameters used for MDP experiments

5-3 Network Structure

5-3-1 LPPO

For LPPO, we have three networks: the policy network, the value network and the Lyapunov network. For the policy network and the value network, we used the OpenAI baselines' [5] default structure setting. The Point-Circle, Ant-Safe, HalfCheetah-Safe and Quadrotor are all Mujoco type environment. For Mujoco type environments, the policy network is a fully-connected MLP with two hidden layers of 64 units, and tanh nonlinearities, outputting the mean of a Gaussian distribution with variable standard deviations. And the value network and Lyapunov network use a fully-connected MLP with two hidden layers of 64 units, and tanh nonlinearities, outputting V value and Lyapunov value.

The Pong-Safe is an Atari-type environment that uses an image as state input. So we use a CNN with three convolutional layers and two fully connected layers as policy network. The first convolutional layer has 32 8x8 size filters with stride 4. The second convolutional layer has 64 4x4 size filters with stride 2. The third convolutional layer has 64 3x3 size filters with stride 1. The first fully connected layer has 512 units. The second fully connected layer is the output layer. And each hidden layer is followed by a ReLU nonlinearity. The policy network's input is an 84x84x4 image produced by the preprocessing map. While the Lyapunov network and the value network shares the parameters between the policy network with additional one fully connected layer each.

5-3-2 LSAC and LAC

For LSAC and LAC, we have three networks as well: the policy network, the Q network and the Lyapunov network. For the policy network, we use a fully-connected MLP with two hidden layers of 256 units, and tanh nonlinearities, outputting the mean of a Gaussian distribution, with variable standard deviations with variable standard deviations. For the Q network and the Lyapunov network, we use a fully-connected MLP with two hidden layers of 256 units, and tanh nonlinearity respectively, outputting the Q value and the Lyapunov value.

[5] https://github.com/openai/baselines

5-4 Selection of Lyapunov function

The constraint Equation A-3 confines a rather broad range of parameterization for Lyapunov function. The sum of quadratic polynomials, e.g., $L(s) = s^T Qs$ where Q is a positive definite matrix, are extensively used in the control theory. Such Lyapunonv functions can be efficiently discovered by the semidefinite programming solvers and brings in limited conservatism for the control tasks where the cost are also of quadratic form. In [35], a neural network $\phi_\theta(\cdot)$ is designed to construct the Lyapunov function, $L(s) = \phi_\theta(s)^T \phi_\theta(s)$. As explored in [36] and [37], value function could be exploited as Lyapunonv function as well. Additionally, Lyapunov function could also be chosen to be the cost or the sum of cost over a limited time horizon, i.e., $L(s) = c_\pi(s)$ or $L(s) = \Sigma_t^{t+N} \mathbb{E} c_\pi(s_t)$ respectively. In principle, the value function evaluates the longest time horizon and thus produces better performance on general, but it is rather difficult to approximate, containing significant variance and bias. On the contrary, for the choice of cost function, though the approximation converges considerably fast, the agent may suffer from the short-sighted behaviour. In the experiments we will show the effect of these different choices and provide a principled way for Lyapunov design.

5-5 Selection of α_3

The hyperparameter α_3 dominates the speed of energy decreasing and takes a significant impact on the performance. The optimization may be infeasible for unreasonable large values; contrarily, with a too small α_3, the right hand-side of inequality could be submerged in the error of Lyapunov function, eventually failing in stabilizing the system. The range of α_3 varies with the choice of Lyapunov function. For the cost function choice, α_3 must be in the range of $[0, 1)$, since values larger than 1 will make the problem analytically infeasible.

5-6 Result

5-6-1 MDP Tasks

We compare our approach to SAC [27], a recent off-policy actor-critic algorithm, which aims to simultaneously maximize expected return and entropy. And itachieves state-of-the-art performance, and outperforms such DDPG [26], PPO [28] and twin delayed deep deterministic policy gradient algorithm (TD3) [38] on the continuous control benchmarks. So, we think SAC is a suitable and qualified baseline for comparison. To make fair comparison, we give every advantage to SAC, such as adaptive temperature and double Q functions, that reported to achieve the best performance. As for tuning, we use the default hyperparameter setting in [27]. We only did primary tuning to determine α_3 and clipped the value of Lagrangian multiplier between $[0, 1]$ to avoid divergence.

These results show the total average cost during training over 10 rollouts and the shaded areas standing for 1-sd confidence intervals. We include two versions of our LAC approach, the version that Lyapunov function approximates the value function and the one where Lyapunov function is to approximate cost.

CartPole

In MDP CartPole experiment, our algorithm LAC achieve the state-of-art result when the Lyapunov function approximates the value function. And the variance of LAC is generally less than SAC as in this task. While the Lyapunov function approximates the undiscounted sum of cost, the result is not as good as the other one.

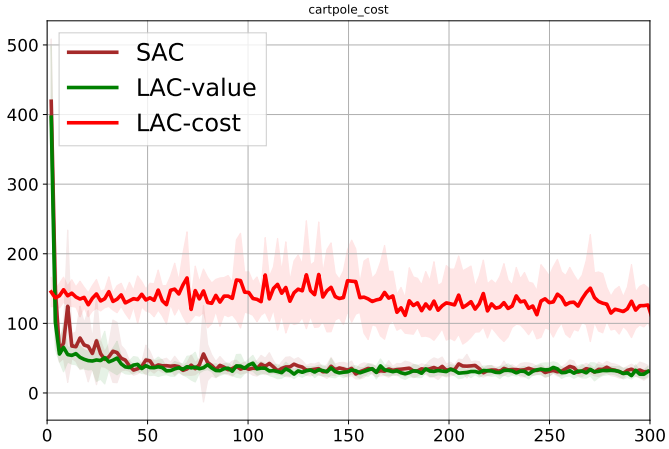


Figure 5-9: Cartpole Result

FetchReach

In FetchReach experiment, our algorithm LAC get better result than the state-of-art result, which converges quickier. And the variance of LAC is generally less than SAC as in this task. And the Lyapunov function approximates the value function or the undiscounted sum of cost both could get good result.

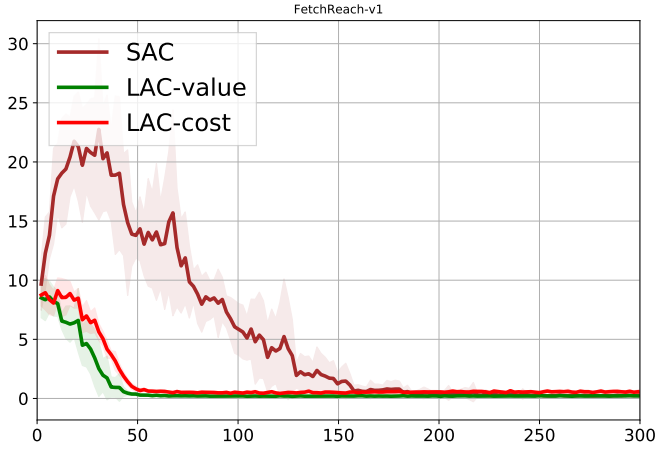


Figure 5-10: FetchReach Result

Quadrotor

In MDP Quadrotor experiment, our algorithm LAC achieve the state-of-art result. And the Lyapunov function approximates the value function or the undiscounted sum of cost both could get good result.

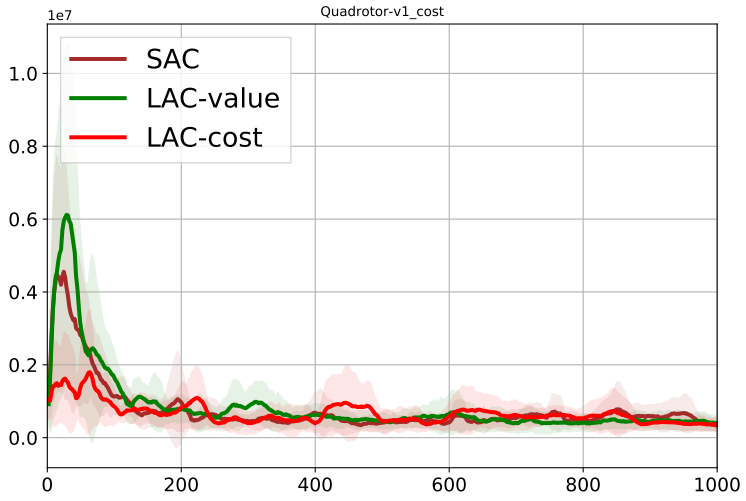


Figure 5-11: Quadrotor Result

Stability against perturbations

An inherent property of stability is to enable the system to recover to normal status from perturbations such as external forces and wind. To show this, we introduce a large external impulsive force F (around 5 times of the input force) in the Cartpole environment, and observe the performance difference between agents trained by LAC and SAC. As shown in the Table 1, the agent trained by LAC outperforms that trained by SAC by large in the presence of impulsive forces with different magnitudes.

F (Newton)	100	95	90	85	80
LAC	27.6%	5.8%	0.0%	0.0%	0.0%
SAC	62.8%	40.6%	25.2%	3.2%	1.2%

Table 5-4: Death rate of agents trained by LAC and SAC in the presence of sudden impulsive force F with different magnitudes. An agent is marked as dead when $\theta > \theta_{\text{threshold}}$ which means the pole will fall over. Both agents are evaluated over 500 random seeds in each setting.

5-6-2 CMDP Tasks

In this part, we evaluate the performance of the Lyapunov-based safe RL algorithms (LSAC, LPPO, LCPO) on the CMDP tasks and compare them with a few safe RL algorithms, including CPO [18], safe SAC (SSAC) and safe PPO (SPPO) [19] with optimized hyperparameters. SSAC and SPPO are safety constrained variants of the original algorithms through Lagrangian

relaxation procedure [39]. Details of the Lagrangian-based safe baselines are referred to 4-3. We also include SAC and PPO to show that the optimal behaviours are generally unsafe in our setting. In the Pong with Catastrophe Zone, only PPO and LPPO is implemented, since SAC is developed for the continuous control tasks and does not have a discrete action counterpart. We use the undiscounted sum of safety cost of episodes as the measure of safety. The goal is to suppress this measure to zero, i.e., zero violation of the state constraints.

Since the trust region methods (LCPO, CPO) require large batch sizes and only take a small step at each update, these methods take more global steps than the gradient-based methods to reach convergence and thus these two classes are compared separately.

Comparison with SSAC and SPPO As demonstrated in experiment results, though SSAC and SPPO could maintain state constraint satisfaction on some of the tasks (see Fig 5-17 and Fig 5-15), on the other tasks these methods may cause large overshoot (see Fig 5-19) or continuous swings (see Fig 5-23) in the safety cost, and even fail in finding safe policy completely (see Fig 5-19 and Fig 5-13). On the other hand, our methods(LSAC and LPPO) quickly converge to safe policies across all the tasks while maintaining reasonable return. In addition, LSAC maintains low safety cost (almost zero) throughout the training with low variance in all the continuous control tasks, even though all the policies are randomly initialized.

Partially constrained v.s. Globally constrained Compared with the globally constrained method (SSAC, LPPO), the partially constrained method (LSAC) obtains equal or less constraint safety cost (almost zero) across different tasks with less variance as shown in Fig 5-17, 5-19, 5-15, 5-23, 5-13 while maintaining higher or equal return. In CartPole experiment, even though LSAC converges to a safe policy, it still obtains higher return than SSAC.

Comparison with CPO As discussed in [17], the discounted-sum constrained methods suffer from tricky tuning for the safety threshold to handle the state constrained problems, thus we tested different threshold setting for CPO. In Fig 5-27, LCPO performs stably in terms of safety cost throughout the training, whereas the performance of CPO swings due to different setting of safety threshold. CPO either fails to find the feasible policy or reaches convergence after being unsafe for a long period (more than 500,000 steps). In Fig 5-25 , both methods converge to the safe policy, potentially due to the rather loose safety constraint. One more thing to note is that CPO requires additional cost shaping, by having a network evaluating the chance of constraint violation to achieve the best performance, while our approach doesn't need such techniques.

5-6-3 Ablation on Constraint

We compare the performance of LSAC in Cartpole-Safe with different sizes of safety set, see Fig 5-28 and Fig 5-29. We gradually strengthen the constraint and see how does LSAC trade off between safety and optimality. Specifically, we reduce the size of safety set $\{x|x \in [0, \overline{x}]\}$ by assigning $\overline{x}$ with $\{0, 1, 2, 3, 4\}$. As $\overline{x}$ approaches zero, the average return of LSAC also decreases while safety is maintained. However, when $\overline{x} = 0$ and only the origin is safe, the

agent fails to sustain the pole and dies almost immediately. This implies that LSAC may fail in the case that safety constraints are too strong.

CartPole

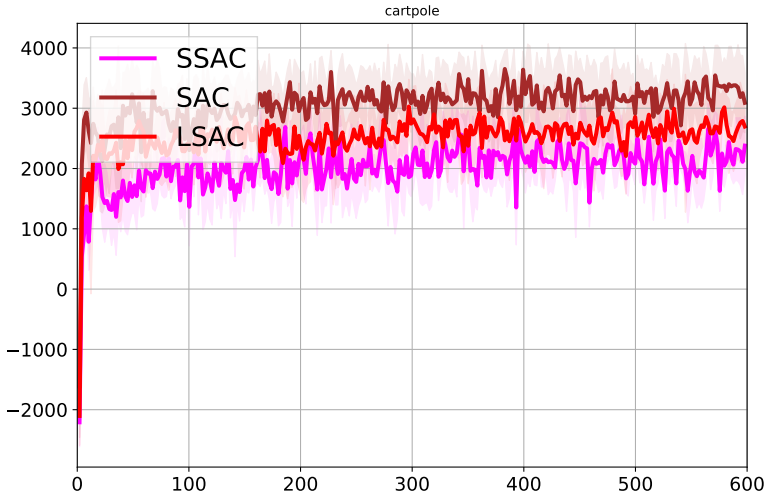


Figure 5-12: Cartpole return

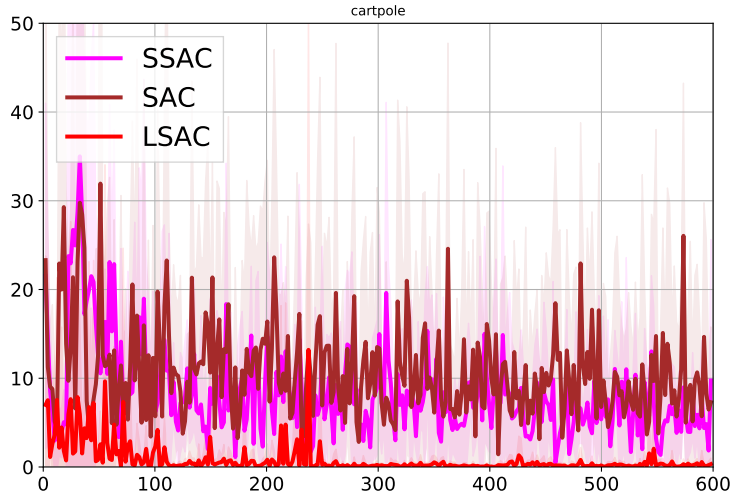


Figure 5-13: Cartpole safety cost

Point-Circle

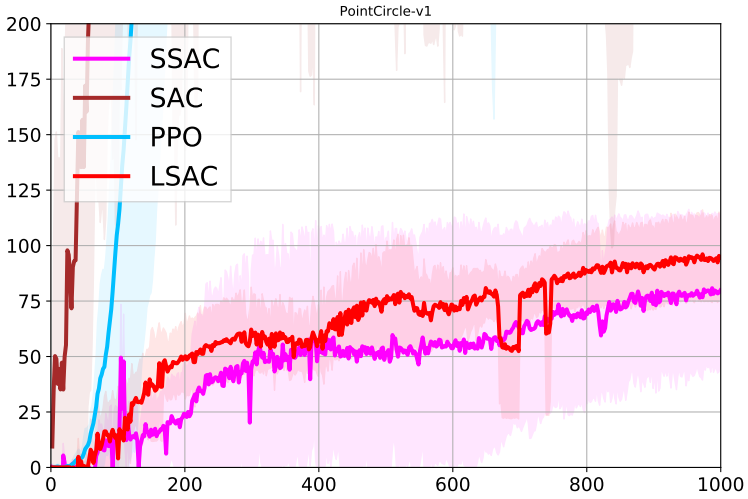


Figure 5-14: Point-Circle return

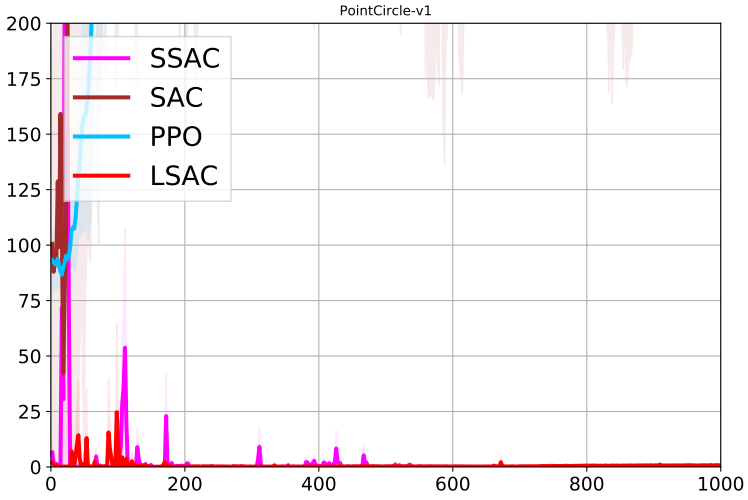


Figure 5-15: Point-Circle safety cost

Ant-Safe

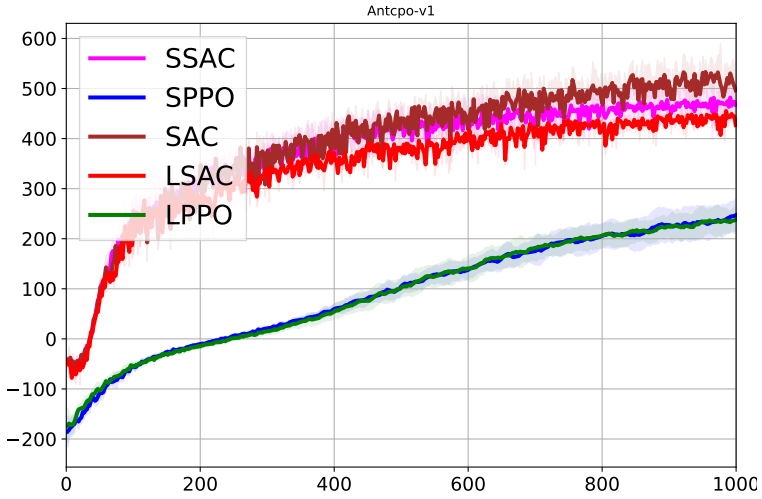


Figure 5-16: Ant-Safe return

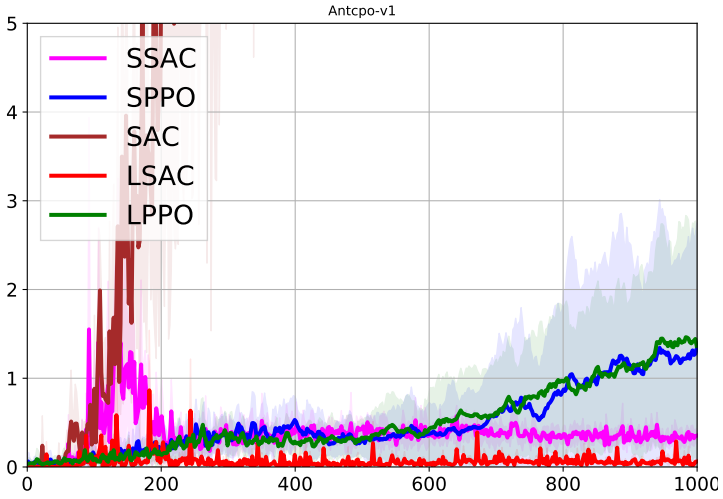


Figure 5-17: Ant-Safe safety cost

HalfCheetah-Safe

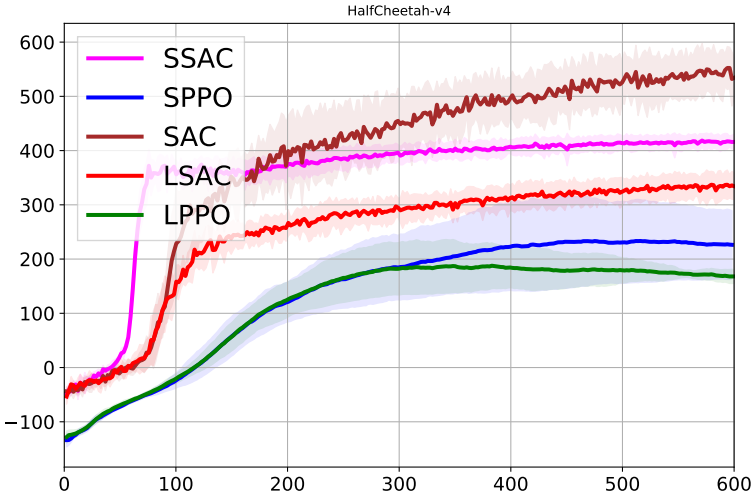


Figure 5-18: HalfCheetah-Safe return

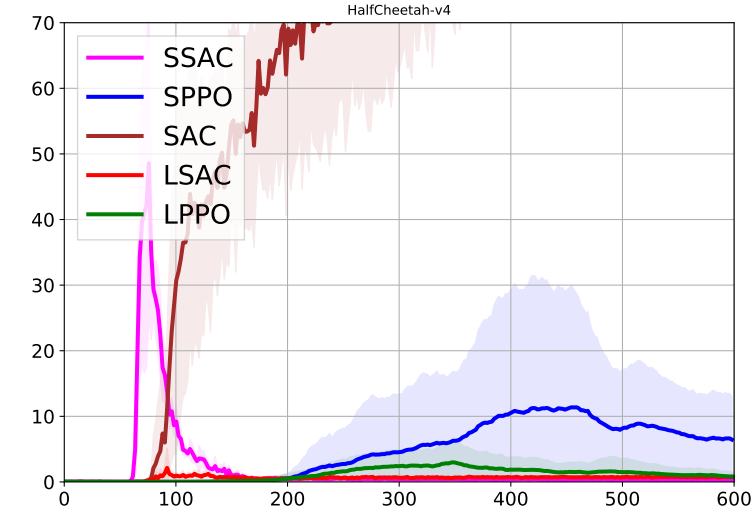


Figure 5-19: HalfCheetah-Safe safety cost

Pong-Safe

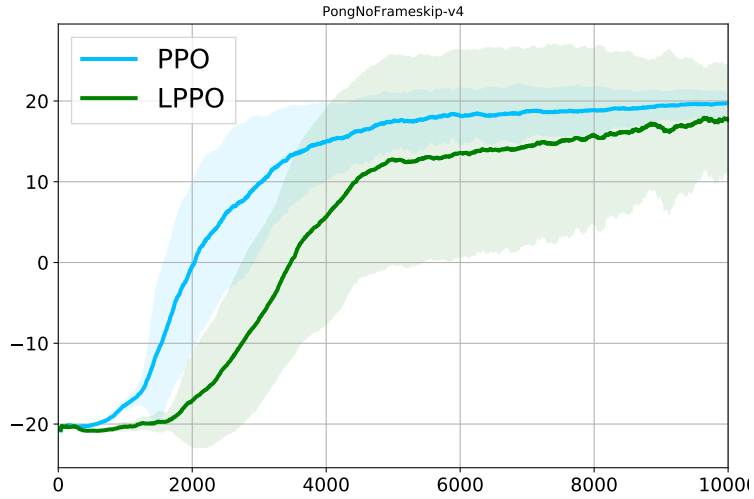


Figure 5-20: Pong-Safe return

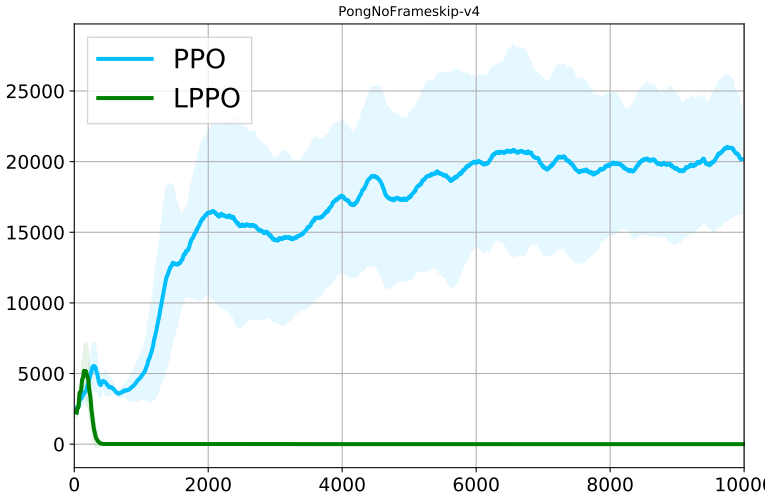


Figure 5-21: Pong-Safe safety cost

Quadrotor

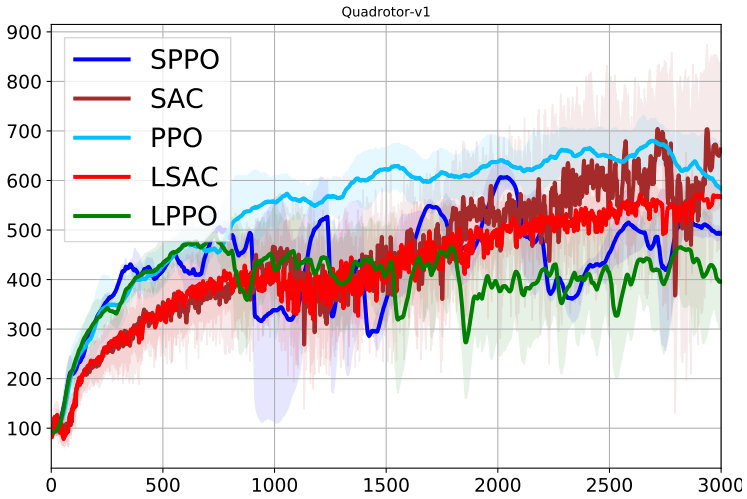


Figure 5-22: Quadrotor-Safe return

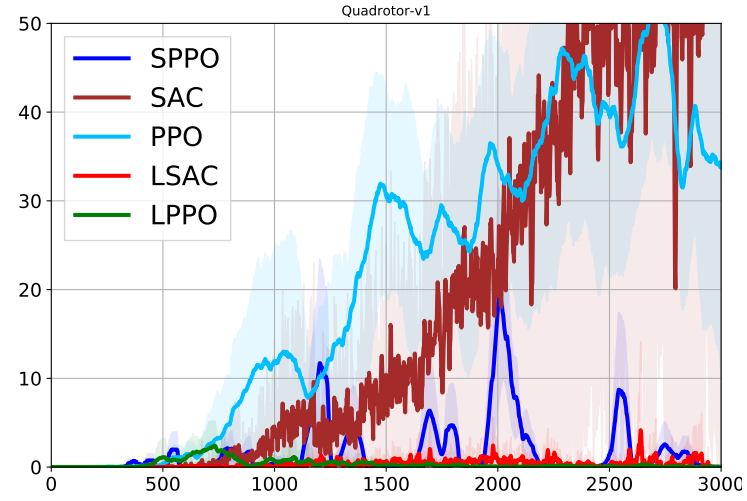


Figure 5-23: Quadrotor-Safe safety cost

Ant-Safe CPO

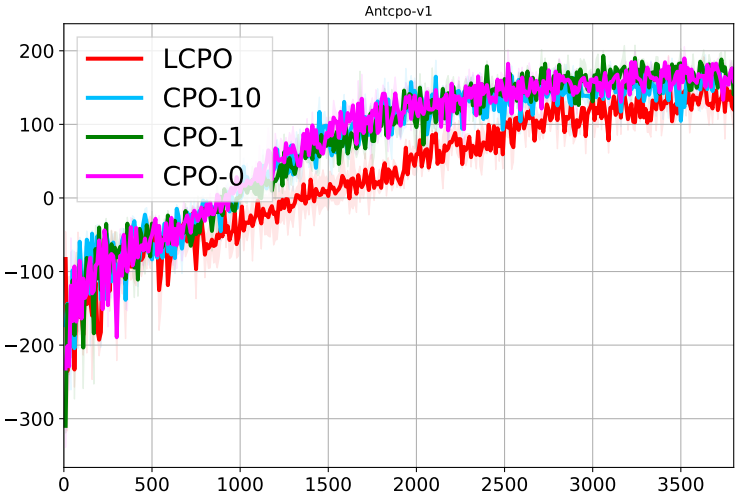


Figure 5-24: Ant-Safe CPO return

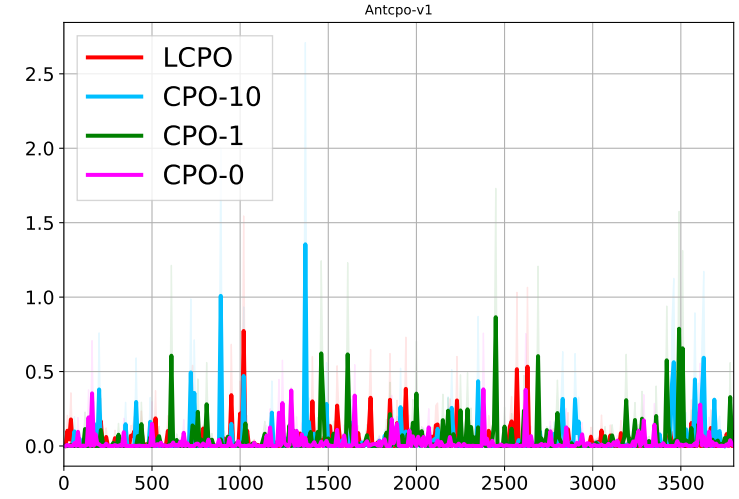


Figure 5-25: Ant-Safe CPO safety cost

HalfCheetah-Safe CPO

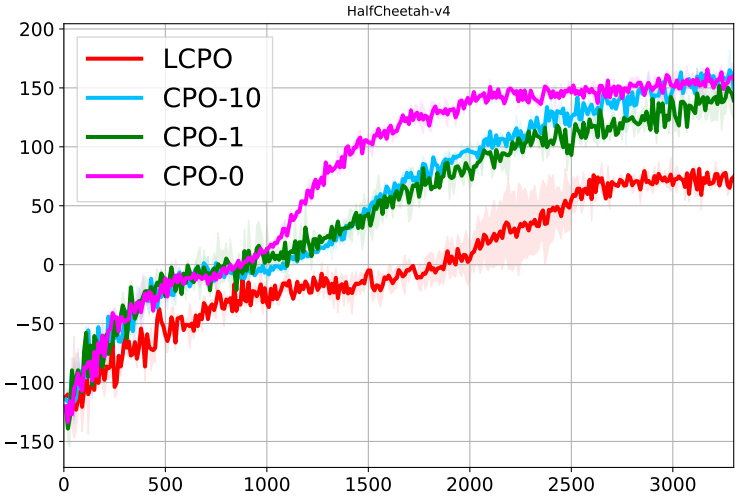


Figure 5-26: HalfCheetah-Safe CPO return

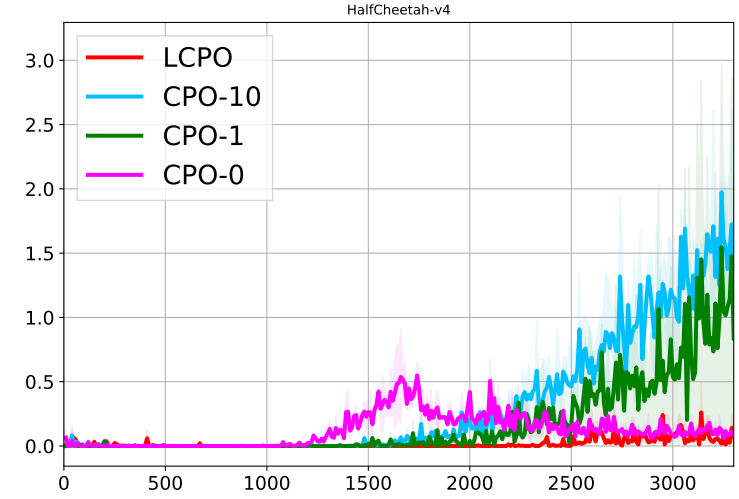


Figure 5-27: HalfCheetah-Safe CPO safety cost

CartPole with different Constrain

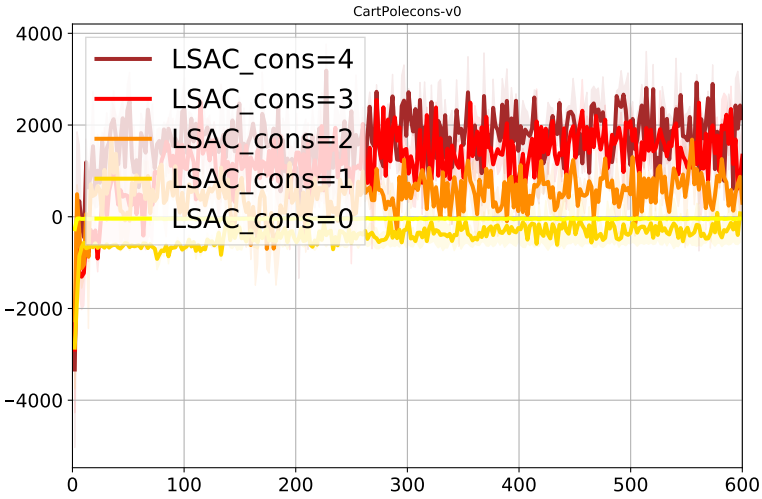


Figure 5-28: CartPole return

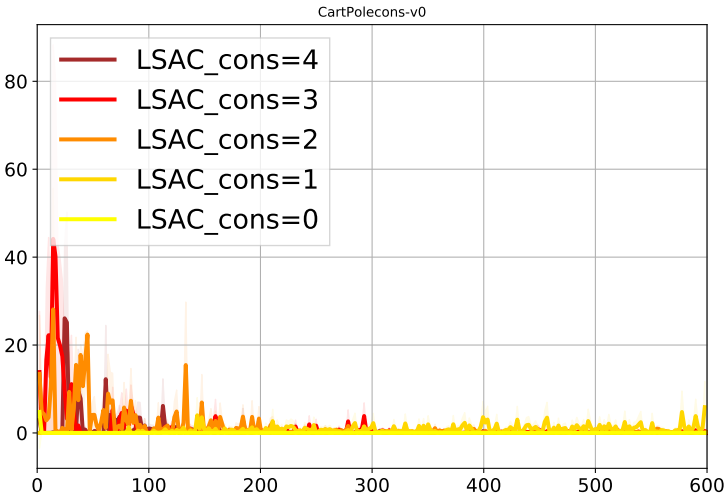


Figure 5-29: CartPole safety cost

Chapter 6

Conclusion

In this thesis, we proposed a novel model-free Lyapunov-based approach of analyzing the MSS and UUB stability of a learning control system and searching for the stability guaranteed policy. Based on these theoretical underpinnings, we developed various RL algorithms and exploit them in solving normal control tasks and those with state constraints. For MDP tasks, we developed the LAC algorithm to search stability guaranteed policy with maximum entropy. For safety constrained tasks, we proposed a novel locally constrained off-policy safe RL algorithm, LSAC, as well as two on-policy algorithms, LPPO and LCPO. We evaluated our algorithms on 9 MDP and CMDP benchmarks, including both continuous control and discrete tasks, such as robot locomotion, quadrotor motion planning and Atari game. Our works represent an initial step in combining RL and control theory for the purpose of applying learning agents in real-world. Besides, we proposed a conditional representation model to help agent learn a constraint.

Chapter 7

Byproduct

Policy Network Reduction in Deep Reinforcement Learning

Yuan Tian
TU Delft
y.tian-8@student.tudelft.nl

Wei Pan
TU Delft
wei.pan@tudelft.nl

1 Motivation

A simple controller is desired to be sought in robust and optimal control [3]. This is referred to as controller reduction. Typically, simple *linear* controllers are normally preferred over complex controllers in control system designs for some obvious reasons: they are easier to understand and computationally less demanding, they are also easier to implement and have higher reliability.

A challenging problem for modern control system is that system models and dynamics are not available. A compelling alternative is Reinforcement Learning, which requires minimal craftsmanship and promotes the natural evolution of a control policy. Recently, Deep Reinforcement Learning achieved state-of-the-art results in continuous and optimal control applications,by directly maximizing cumulative reward, where the controller policy can be modeled by DNN. However, it is known that real-world applications in robotics usually requires the response time to be less than *1ms*. However, a large deep neural network is in general incapable of achieving such real-time inference speed. In this work, we will consider the DNN based controller reduction problem without performance degradation.

2 Algorithm

A straightforward way is to compress the trained policy network can be done off-line as a supervised learning problem similar to DNN compression for computer vision tasks, *e.g.*, image classification. However, it does not work at all. Similar to the classic controller reduction, the new sparse controller needs to be re-derived/re-trained by optimizing the cost from beginning. Then we implement the reduction procedure into the typical policy gradient update in actor-critic algorithms, such as DDPG and PPO. We use CartPole model as our experiment environment in OpenAI Gym. Different from the default settings, we make the state and action space continuous. The control goal is to stabilize the CartPole in the middle and make the pendulum stand straightly.

In NN, the weight matrix from layer $\ell-1$ to layer ℓ, $\mathbf{W}^\ell = \left[(\mathbf{W}^\ell_{1,:})^\top,\ldots,(\mathbf{W}^\ell_{n^{\ell-1},:})^\top\right]^\top = \left[\mathbf{W}^\ell_{:,1},\ldots,\mathbf{W}^\ell_{:,n^\ell}\right]$ where $\mathbf{W}^\ell_{i,:}$ denote the i-th row of $\mathbf{W}^\ell$, $i=1,\ldots,n^{\ell-1}$; $\mathbf{W}^\ell_{:,j}$ denote the j-th column of $\mathbf{W}^\ell$, $j=1,\ldots,n^\ell$. We denote the parameter in policy network as $\theta^\mu=[\mathbf{W}]$.

Inspired by [2], our approach is to modify the update procedure for policy gradient. For example in DDPG [1], we have

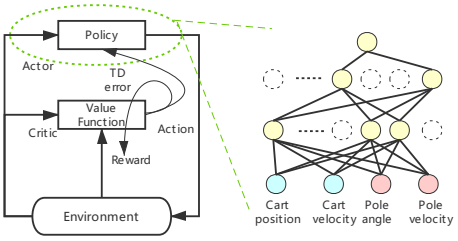


Figure 1: Controller reduction or policy compression in actor-critic algorithm

$$\nabla_{\theta^a} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i,a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i} + \nabla_{\theta^\mu} R$$

$$R = \lambda_1 \sum_{\ell=1}^{L}\sum_{j=1}^{n^\ell}\sqrt{\sum_{i=1}^{n^{\ell-1}}\left(W^\ell_{ij}\right)^2} + \lambda_2 \sum_{\ell=1}^{L}\sum_{i=1}^{n^{\ell-1}}\sqrt{\sum_{j=1}^{n^\ell}\left(W^\ell_{ij}\right)^2}$$

3 Result

For DDPG's actor network, we use a five layers structure $(4-256-256-256-128-1)$, and we only compressed the middle three layer. Almost without any performance degradation, the compression ratios (the number of zero weights / the number of weights) are $50.56\%, 46.17\%, 83.62\%$ respectively. A compressed weight matrix of can be found in Figure 2.

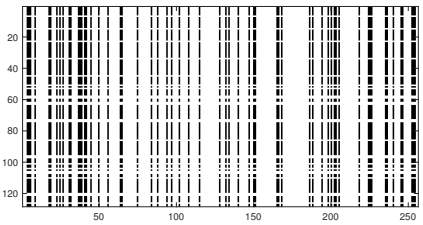


Figure 2: The weight matrix of the fourth layer after reduction

References

[1] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[2] Wei Pan, Hao Dong, and Yike Guo. Dropneuron: Simplifying the structure of deep neural networks. *arXiv preprint arXiv:1606.07326*, 2016.

[3] Kemin Zhou, John Comstock Doyle, Keith Glover, et al. *Robust and optimal control*, volume 40. Prentice hall New Jersey, 1996.

Appendix A

Appendix

A-1 Reinforcement Learning with MSS Guarantee

***Appendix A1,A2 and A3 are cited from another paper, and all the proofs are done by MINGHAO HAN, these parts are not my contribution. The paper is submitted to 33rd Conference on Neural Information Processing Systems (NeurIPS 2019).**

To achieve the "eventual" behaviour by Definition 1, the approach utilizes the Lyapunov function to construct the stability guarantee. This tool has long been used in the control theory for the purpose of stability analysis and controller design [40], but mostly exploited along with a known model, whether deterministic or probabilistic. The Lyapunov function is a class of continuously differentiable semi-definite functions $L : S \to \mathbb{R}_+$. The general idea of exploiting Lyapunov function is to ensure that the derivative of Lyapunov function along the state trajectory is semi-negative definite, so that the state goes in the direction of decreasing the value of Lyapunov function and eventually converges to the set or point where the value is zero.

Next, we show how Lyapunov function is engaged in stability analysis for stochastic systems following the proof of [41, Theorem 1].

Lemma 1. *The system is mean square stable if there exists a set of C^1 functions $L(s) : \mathbb{S} \to \mathbb{R}$ and positive constants α_1, α_2 and α_3, such that*

$$\alpha_1 c_\pi(s) \leq L(s) \leq \alpha_2 c_\pi(s), \tag{A-1}$$

$$\mathbb{E}_{s' \sim P_\pi} L(s') - L(s) \leq -\alpha_3 c_\pi(s), \forall s \in \mathbb{S} \tag{A-2}$$

Proof of above Lemma is given in Appendix A-3-1(The proof is cited from another paper). In the Lemma and the following, P_π refers to the closed-loop dynamics, i.e., $P_\pi(s'|s) \doteq \mathbb{E}_{a \sim \pi} P(s'|s, a)$. Though the mean square stability of the system has been proved, we should note that this is based on the condition that energy decreasing condition (A-2) holds everywhere in the state

space, which is a rather strong condition. In our model-free formulation, this requires the same amount of constraints as the number of states, which is impractical for the tasks in continuous state spaces. Thus the following sample-based stability theorem is proposed to guarantee closed-loop stability of the model-free learning approaches.

Theorem 1. *[MSS] The system is mean square stable if there exists a continuous differentiable function $L : S \to \mathbb{R}_+$ and positive constants α_1, α_2 and α_3, such that*

$$\alpha_1 c_\pi(s) \leq L(s) \leq \alpha_2 c_\pi(s) \tag{A-3}$$

$$\mathbb{E}_{s\sim\tau}(\mathbb{E}_{s'\sim P_\pi} L(s') - L(s)) \leq -\alpha_3 \mathbb{E}_{s\sim\tau} c_\pi(s) \tag{A-4}$$

where τ is the sampled trajectory with length of N.

Proof of Theorem 1 is given in Appendix A-3-2. The theorem above enforces a step-by-step constraint over the agent's trajectory. The policy satisfying the above theorem would drive the trajectory asymptotically to the null space of cost function, producing predictable behaviour of the agent.

One may notice that there are two hyperparameters to be determined, the selection of Lyapunov function and selection of the positive constant α_3. Various choices for Lyapunov function in the literature, such as quadratic polynomials, Lyapunov Neural Network and value function [35, 36, 37], which has a significant effect on the policy performance and we will discuss in Chapter 5. It is also important to select proper α_3, on which the discussion and we will discuss in Chapter 5 as well.

Though the stability guarantee is obtained in the above Theorem, it still remains unknown how does the agent perform in terms of the discounted total cost $J(\pi)$. The following Theorem will give an upper bound guarantee for the performance of stability guaranteed policy.

Theorem 2. *If there exists a continuous differentiable functional $L : S \to \mathbb{R}_+$ and positive constants $\alpha_1, \alpha_2, \alpha_3$, such that*

$$\alpha_1 c_\pi(s) \leq L(s) \leq \alpha_2 c_\pi(s) \tag{A-5}$$

$$\mathbb{E}_{s\sim d^\pi}\left(\mathbb{E}_{s'\sim P_\pi} L(s') - L(s)\right) \leq -\alpha_3 \mathbb{E}_{s\sim d^\pi} c_\pi(s) \tag{A-6}$$

Then the objective function J satisfies

$$J_c(\pi) \leq \frac{\alpha_2 (1-\gamma)}{\alpha_1 \gamma (1+\alpha_3-\gamma)} \mathbb{E}_{s_0\sim\rho} c_\pi(s_0). \tag{A-7}$$

Proof of Theorem 2 is given in Appendix A-3-3. As shown in (A-7), the value of the upper bound is related to the value of constants $\alpha_{1,2,3}$. Large α_3 leads the cost to decrease steeply, producing lower upper bound. On the other hand, though the values of $\alpha_{1,2}$ are not set by the designer, they are actually relevant to the parameterization of Lyapunov function. For example, in the case of quadratic polynomials, $\alpha_1 = \min(\text{eig}(Q))$ and $\alpha_2 = \max(\text{eig}(Q))$. Similarly, in the case of neural networks [35], $\alpha_{1,2}$ are the product of minimum and maximum eigenvalues of weight matrices of each layer, respectively. For the choice of approximating cost, it is obvious that $\alpha_{1,2} = 1$, while for the value function choice it may need optimization to determine the exact values.

Additionally, by optimizing the proposed upper bound over α_3 together with the previous constraints, an method of optimal control with stability guarantee is further obtained, where α_3 is automatically searched. Since this is not the main focus of this work, we leave it for future work.

A-2 Safe Reinforcement Learning with UUB Guarantee

To achieve the “eventual" behaviour by Definition 2, it will demonstrate how to exploit the Lyapunov-based approach to guarantee the *ultimately uniform boundness* of the system, and eventually achieve the constraint satisfaction in CMDP. First, the Theorem for UUB guarantee is given as follow.

Theorem 3. *[UUB] For the given safety constraint $\overline{d}$, let $\Omega = \left\{s \in S | b_\pi(s) < \overline{d}\right\}$. The system is ultimately uniformly bounded if there exists a set of C^1 functions $L(s) : S \to \mathbb{R}$ and positive constants α_1, α_2, α_3, such that $\forall s \in \Omega$*

$$\alpha_1 b_\pi(s) \leq L(s) \leq \alpha_2 b_\pi(s) \tag{A-8}$$

and for $s \in \Delta = \left\{s \in S | b_\pi(s) \geq \eta, \eta \in [0, \alpha_2^{-1}\alpha_1 \overline{d})\right\}$,

$$\mathbb{E}_{s\sim\tau}\left(\mathbb{E}_{s'\sim P_\pi} L(s') - L(s)\right) \leq -\alpha_3 \mathbb{E}_{s\sim\tau} b_\pi(s) \tag{A-9}$$

where τ is is the trajectory sampled from Δ of the length N. For trajectories starting inside the inner safe set $\mathcal{B} = \{s \in S | b_\pi(s) <= \alpha_2^{-1}\alpha_1 \overline{d}\}$, the state will not leave the safe set Ω, while the trajectories with initial states outside $\mathcal{B}$ will be attracted by $\mathcal{B}$ and stay inside Ω.

Proof of the above Theorem is referred to Appendix A-3-4. As guaranteed above, the trajectories will be kept in the safe set once it has entered it, and trajectories in the unsafe region will recover safety in finite time. It's also noteworthy that the safe set Ω is confined by the safety cost, rather than by setting a bound for the discounted sum of safety cost, which is hard for the designer to predict how many times the agent would violate the state constraint and thus difficult to implement [17].

Another point to be noted is that the energy decreasing constraint (A-9) is only applied on the edge set Δ. This indicates that the safety guarantee takes effect only in the "dangerous" state space, while the actions in the inner safe region $\mathcal{B}$ will not be interfered by the safety constraint. In the case of $\alpha_{1,2} = 1$, it's even possible that the intersection between edge set Δ and safe set Ω infinitely approach the empty set, which implies that the agent acts in the safe set with unconstrained freedom.

A-3 Proofs of Theorem and Lemma

A-3-1 Proof of Lemma 1

Proof. The condition (A-2) implies that the Lyapunov value between two consecutive instants is strictly decreasing, i.e.

$$\mathbb{E}_{P_\pi} L(s_{t+1}) \leq -\alpha_3 c_\pi(s_t) + L(s_t), \forall t \in [0, \infty)$$

Starting from the initial instant, iterate the above relation in for t steps and make expectation on both sides, one has

$$\mathbb{E}_{s_{t+1}} L(s_{t+1}) \leq -\alpha_3 \sum_{i=0}^{t} \mathbb{E}_{s_i} c_\pi(s_i) + \mathbb{E}_\rho L(s_0) \tag{A-10}$$

Combined with (A-1), it infers that

$$\sum_{i=0}^{t} \mathbb{E}_{s_i} c_\pi(s_i) \leq \frac{\alpha_2}{\alpha_3} \mathbb{E}_\rho c_\pi(s_0) - \frac{\alpha_1}{\alpha_3} \mathbb{E}_\rho c_\pi(s_{t+1}) \tag{A-11}$$

Take limitation on both sides, it follows that

$$\lim_{t\to\infty} \sum_{i=0}^{t} \mathbb{E}_{s_i} c_\pi(s_i) \leq \frac{\alpha_2}{\alpha_3} \mathbb{E}_\rho c_\pi(s_0)$$

Since the right side is a finite positive constant and $c_\pi(\cdot)$ is a semi-positive definite function, it holds that $\lim_{t\to\infty} \mathbb{E}_{s_t} c_\pi(s_t) = 0$, which infers mean square stability by Definition 1. □

A-3-2 Proof of Theorem 1

Proof. (A-4) can be written as

$$\begin{aligned} \sum_{i=t_0}^{t_0+N} \mathbb{E}_{s_i \sim P_\pi(s_i|s_{i-1})} (\mathbb{E}_{s_{i+1}\sim P_\pi} L(s_{i+1}) - L(s_i) + \alpha_3 c_\pi(s_i)) &\leq 0 \\ \mathbb{E}_{s_{t_0+N}} L(s_{t_0+N}) - \mathbb{E}_{s_{t_0}} L(s_{t_0}) &\leq -\alpha_3 \sum_{i=t_0}^{t_0+N} \mathbb{E}_{s_i} c_\pi(s_i) \end{aligned} \tag{A-12}$$

Since the trajectory is sampled randomly as the agent interacts with the environment, the starting instant t_0 can be viewed as equally distributed, which follows that (A-12) holds for $\forall t_0 \in [0, \infty)$. Iterate (A-12) from 0 to t gives that

$$\mathbb{E}_{s_{t+N}} L(s_{t+N}) - \sum_{i=0}^{N-1} \mathbb{E}_{s_i} L(s_i) \leq -\alpha_3 N \sum_{i=N}^{t} \mathbb{E}_{s_i} c_\pi(s_i) - \alpha_3 \sum_{i=0}^{N-1} i \mathbb{E}_{s_i} c_\pi(s_i)$$

Taking (A-4) into consideration, take the limitation on both sides

$$\lim_{t\to\infty} \mathbb{E}_{s_{t+N}} L(s_{t+N}) \leq \lim_{t\to\infty} -\alpha_3 N \sum_{i=N}^{t} \mathbb{E}_{s_i} c_\pi(s_i) - \sum_{i=0}^{N-1} \mathbb{E}_{s_i} (L(s_i) - i\alpha_3 c_\pi(s_i))$$

Given to (A-3), $0 \leq \alpha_1 c_\pi(s) \leq L(s) \leq \alpha_2 c_\pi(s)$, thus

$$\alpha_3 N \lim_{t\to\infty} \sum_{i=N}^{t} \mathbb{E}_{s_i} c_\pi(s_i) \leq \sum_{i=0}^{N-1} (\alpha_3 i - \alpha_2) \mathbb{E}_{s_i} c_\pi(s_i)$$

where the right hand side is a finite positive constant. Due to the semi-definity of cost function $c_\pi(s)$, it is inferred that $c_\pi(s) \to 0$ as $t \to \infty$, which infers mean square stability by Definition 1. □

A-3-3 Proof of Theorem 2

Proof. First, as explored in [18], let $p_\pi^t \in \mathbb{R}^{|S|}$ denote the vector with elements being $P(s_t = s|\pi, \rho)$, and let $P_\pi \in \mathbb{R}^{|S|} \times \mathbb{R}^{|S|}$ denote the transition matrix with components $P(s'|s,\pi) = \int_a P(s'|s,a)\,\pi(a|s)\,da$. Then $p_\pi^t = P_\pi p_\pi^{t-1} = P_\pi^{t-1}\rho$ and the vector form of d^π is obtained as follow:

$$\begin{aligned} d^\pi &= (1-\gamma)\sum_{t=0}^{\infty}(\gamma P_\pi)^t \rho \\ &= (1-\gamma)(I-\gamma P_\pi)^{-1}\rho \end{aligned}$$

Let $L \in \mathbb{R}^{|S|}$ be the vector composed of $L(s)$, then the expectation of $L(s)$ can be expressed in the form of innerproduct of two vectors:

$$\begin{aligned} \mathbb{E}_{s\sim d^\pi} L(s) &= \langle d^\pi, L\rangle \\ &= (1-\gamma)\left\langle (I-\gamma P_\pi)^{-1}\rho, L\right\rangle \end{aligned}$$

And

$$\begin{aligned} \mathbb{E}_{s\sim d^\pi}\left(\mathbb{E}_{s'\sim P_\pi} L(s') - L(s)\right) &= (1-\gamma)\left\langle (P_\pi - I)(I-\gamma P_\pi)^{-1}\rho, L\right\rangle \\ &= (1-\gamma)\left\langle [(1-\gamma)P_\pi + (\gamma P_\pi - I)](I-\gamma P_\pi)^{-1}\rho, L\right\rangle \\ &= (1-\gamma)\left((1-\gamma)\left\langle P_\pi (I-\gamma P_\pi)^{-1}\rho, L\right\rangle - \langle \rho, L\rangle\right) \\ &= (1-\gamma)\,\mathbb{E}_{s\sim d^\pi}\mathbb{E}_{s'\sim P_\pi} L(s') - (1-\gamma)\,\mathbb{E}_{s_0\sim\rho} L(s_0) \end{aligned}$$

With (A-6) holds, it follows that

$$\mathbb{E}_{s\sim d^\pi}\mathbb{E}_{s'\sim P_\pi} L(s') \le \mathbb{E}_{s_0\sim\rho} L(s_0) - \frac{\alpha_3}{(1-\gamma)}\mathbb{E}_{s\sim d^\pi} c_\pi(s) \tag{A-13}$$

Take a closer look at the left hand side of the inequality,

$$\begin{aligned} \mathbb{E}_{s\sim d^\pi}\mathbb{E}_{s'\sim P_\pi} L(s') &= (1-\gamma)\left\langle P_\pi \sum_{t=0}^{\infty}(\gamma P_\pi)^t \rho, L\right\rangle \\ &= \frac{(1-\gamma)}{\gamma}\left\langle \sum_{t=0}^{\infty}(\gamma P_\pi)^{t+1}\rho, L\right\rangle \\ &= \frac{(1-\gamma)}{\gamma}\left(\left\langle \sum_{t=0}^{\infty}(\gamma P_\pi)^t \rho, L\right\rangle - \langle \rho, L\rangle\right) \end{aligned}$$

Thus combining with (A-13) we got

$$\mathbb{E}_{s\sim d^\pi} L(s) \le \frac{1}{\gamma}\mathbb{E}_{s_0\sim\rho} L(s_0) - \frac{\alpha_3}{(1-\gamma)}\mathbb{E}_{s\sim d^\pi} c_\pi(s)$$

With (A-5) and the prior that $L(s)$ is positive def finite,

$$\begin{aligned} J &= \mathbb{E}_{s\sim d^\pi} c_\pi(s) \\ &\le \frac{1}{\alpha_1}\mathbb{E}_{s\sim d^\pi} L(s) \\ &\le \frac{1}{\alpha_1\gamma}\mathbb{E}_{s_0\sim\rho} L(s_0) - \frac{\alpha_3}{(1-\gamma)}J \\ &\le \frac{\alpha_2}{\alpha_1\gamma}\mathbb{E}_{s_0\sim\rho} c_\pi(s_0) - \frac{\alpha_3}{(1-\gamma)}J \end{aligned}$$

Thus moving the objective J to the left side, one gets

$$J \leq \frac{\alpha_2 (1-\gamma)}{\alpha_1 \gamma (1+\alpha_3-\gamma)} \mathbb{E}_{s_0 \sim \rho} c_\pi(s_0). \tag{A-14}$$

□

A-3-4 Proof of Theorem 3

Proof. Following similar procedure as in Theorem 1, the Lyapunov function at instant $t+N$ satisfies

$$\mathbb{E}_{s_{t+N}} L(s_{t+N}) \leq \mathbb{E}_{s_t} L(s_t) - \alpha_3 \sum_{i=t}^{t+N} \mathbb{E}_{s_i} b_\pi (s_i) \tag{A-15}$$

Since the trajectory is randomly sampled on the edge set Δ, (A-15) holds $\forall t \in [0, t_s - N)$ where t_s is the instant of termination or leaving the set Δ. Thus connect the consecutive trajectories along the time sequence, one has that

$$\mathbb{E}_{s_{t+N}} L(s_{t+N}) \leq \mathbb{E}_{s_0} L(s_0) - \alpha_3 \sum_{i=0}^{t+N} \mathbb{E}_{s_i} b_\pi (s_i) \tag{A-16}$$

Combined with (A-8), it follows that

$$\mathbb{E}_{s_{t+N}} b_\pi(s_{t+N}) \leq \alpha_1^{-1} \alpha_2 \mathbb{E}_{s_0} b_\pi(s_0) - \alpha_3 \sum_{i=0}^{t+N} \mathbb{E}_{s_i} b_\pi (s_i) \tag{A-17}$$

For trajectories starting from the inner safe set $\mathcal{B} = \{s \in S | b_\pi(s) <= \alpha_2^{-1} \alpha_1 \overline{d}\}$, following the above inequality,

$$\mathbb{E}_{s_{t+N}} b_\pi(s_{t+N}) < \alpha_1^{-1} \alpha_2 \mathbb{E}_{s_0} b_\pi(s_0) \leq \overline{d} \tag{A-18}$$

, i.e., the system will not leave the safe set Ω. For the case that a trajectory starting from the edge set Δ, taking limitation on both sides of (A-17),

$$\alpha_3 \lim_{t \to \infty} \sum_{i=0}^{t+N} \mathbb{E}_{s_i} b_\pi (s_i) \leq \alpha_1^{-1} \alpha_2 \mathbb{E}_{s_0} b_\pi(s_0) - \mathbb{E}_{s_{t+N}} b_\pi(s_{t+N}) \tag{A-19}$$

which implies that $b_\pi(s)$ converges to zero as t approaching infinity. Thus there exits a finite instant t_s that the trajectory enters the inner safe set $\mathcal{B}$. Combining both cases above, the system is ultimately uniformly bounded by Definition 2. □

A-4 Installation instruction

All the experiments in this thesis are running in OSX system, so this instruction is for OSX.

A-4-1 Conda environment

From the general python package sanity perspective, it is a good idea to use conda environments to make sure packages from different projects do not interfere with each other. And we choose python 3.6 as our experiment used version.

A-4-2 Mujoco

Some of the experiments use MuJoCo (multi-joint dynamics in contact) physics simulator, which is proprietary and requires binaries and a license (temporary 30-day license can be obtained from www.mujoco.org).

A-4-3 Installation

When you get the conda and Mujoco, run the following code in terminal to install all the necessary libraries and our code:

```
conda create -n test python=3.6
conda activate test
git clone https://github.com/RLControlTheoreticGuarantee/
    Guarantee_Learning_Control
pip install numpy==1.16.3
pip install tensorflow==1.13.1
pip install tensorflow-probability==0.6.0
pip install opencv-python
pip install cloudpickle
pip install gym
pip install gym[atari]
pip3 install -U 'mujoco-py==1.50.1.68'
pip install matplotlib
```

A-4-4 Example 1. LPPO with Atari Pong

For instance, to train a CNN network controlling Atari Pong using LPPO for 20M timesteps, run :

```
conda create -n test python=3.6
conda activate test
python run.py
```

The hyperparameters, the tasks and the learning algorithm can be changed via change the run.py.

A-4-5 Example 2. LSAC with continous CartPole

For instance, to train a MLP network controlling CartPole using LSAC for, run :

```
conda create -n test python=3.6
conda activate test
python main_for_sac.py
```

The hyperparameters, the tasks and the learning algorithm can be changed via change the variant.py.

Bibliography

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[2] W. Saunders, G. Sastry, A. Stuhlmueller, and O. Evans, "Trial without error: Towards safe reinforcement learning via human intervention," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 2067–2069, International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[3] Wikipedia contributors, "Optimal control — Wikipedia, the free encyclopedia," 2018. [Online; accessed 31-October-2018].

[4] B. Kiumarsi, K. G. Vamvoudakis, H. Modares, and F. L. Lewis, "Optimal and autonomous control using reinforcement learning: A survey," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 6, pp. 2042–2062, 2018.

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[6] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[7] N. Prasad, L.-F. Cheng, C. Chivers, M. Draugelis, and B. E. Engelhardt, "A reinforcement learning approach to weaning of mechanical ventilation in intensive care units," *arXiv preprint arXiv:1704.06300*, 2017.

[8] S. Nemati, M. M. Ghassemi, and G. D. Clifford, "Optimal medication dosing from suboptimal clinical examples: A deep reinforcement learning approach," in *Engineering in Medicine and Biology Society (EMBC), 2016 IEEE 38th Annual International Conference of the*, pp. 2978–2981, IEEE, 2016.

[9] OpenAI, "Openai five." https://blog.openai.com/openai-five/.

[10] N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, A. Eslami, M. Riedmiller, *et al.*, "Emergence of locomotion behaviours in rich environments," *arXiv preprint arXiv:1707.02286*, 2017.

[11] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 3389–3396, IEEE, 2017.

[12] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.

[13] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, p. eaau5872, 2019.

[14] D. Görges, "Relations between model predictive control and reinforcement learning," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 4920–4928, 2017.

[15] L. Buşoniu, T. de Bruin, D. Tolić, J. Kober, and I. Palunko, "Reinforcement learning for control: Performance, stability, and deep approximators," *Annual Reviews in Control*, 2018.

[16] E. Altman, *Constrained Markov decision processes*, vol. 7. CRC Press, 1999.

[17] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[18] J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained policy optimization," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 22–31, JMLR. org, 2017.

[19] Y. Chow, O. Nachum, A. Faust, M. Ghavamzadeh, and E. Duenez-Guzman, "Lyapunov-based safe policy optimization for continuous control," *arXiv preprint arXiv:1901.10031*, 2019.

[20] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[21] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, pp. 1057–1063, 2000.

[22] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[23] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1352–1361, JMLR. org, 2017.

[24] B. D. Ziebart, "Modeling purposeful adaptive behavior with the principle of maximum causal entropy," 2010.

[25] E.-K. Boukas and Z. Liu, "Robust stability and stabilizability of markov jump linear uncertain systems with mode-dependent time delays," *Journal of Optimization Theory and Applications*, vol. 109, no. 3, pp. 587–600, 2001.

[26] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[27] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[28] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[29] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[30] A. Stephenson, "Xx. on induced stability," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 15, no. 86, pp. 233–236, 1908.

[31] P. Donaldson, "Error decorrelation: a technique for matching a class of functions," in *Proceedings of the Third International Conference on Medical Electronics*, pp. 173–178, 1960.

[32] B. Widrow, "Pattern recognition and adaptive control," *IEEE Transactions on Applications and Industry*, vol. 83, no. 74, pp. 269–277, 1964.

[33] D. Michie and R. A. Chambers, "Boxes: An experiment in adaptive control," *Machine intelligence*, vol. 2, no. 2, pp. 137–152, 1968.

[34] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[35] S. M. Richards, F. Berkenkamp, and A. Krause, "The lyapunov neural network: Adaptive stability certification for safe learning of dynamic systems," *arXiv preprint arXiv:1808.00924*, 2018.

[36] Y. Chow, O. Nachum, E. Duenez-Guzman, and M. Ghavamzadeh, "A lyapunov-based approach to safe reinforcement learning," *arXiv preprint arXiv:1805.07708*, 2018.

[37] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause, "Safe model-based reinforcement learning with stability guarantees," in *Advances in neural information processing systems*, pp. 908–918, 2017.

[38] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," *arXiv preprint arXiv:1802.09477*, 2018.

[39] D. P. Bertsekas, "Nonlinear programming," *Journal of the Operational Research Society*, vol. 48, no. 3, pp. 334–334, 1997.

[40] E. Boukas and Z. Liu, "Robust stability and h/sub/spl infin//control of discrete-time jump linear systems with time-delay: an lmi approach," in *Decision and Control, 2000. Proceedings of the 39th IEEE Conference on*, vol. 2, pp. 1527–1532, IEEE, 2000.

[41] L. Shaikhet, "Necessary and sufficient conditions of asymptotic mean square stability for stochastic linear difference equations," *Applied Mathematics Letters*, vol. 10, no. 3, pp. 111–115, 1997.