Doctoral Dissertation

Entropy Regularization for Scalable, Safe and Robust Reinforcement Learning

Lingwei Zhu

Program of Information Science and Engineering
Graduate School of Science and Technology
Nara Institute of Science and Technology

Supervisor: Takamitsu Matsubara
Robot Learning Lab. (Division of Information Science)

Submitted on April 07, 2022

A Doctoral Dissertation
submitted to Graduate School of Science and Technology,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of Engineering

Lingwei Zhu

Thesis Committee:

Supervisor	Takamitsu Matsubara
	(Professor, Division of Information Science)
	Takahiro Wada
	(Professor, Division of Information Science)
	Yoshihisa Tsurumine
	(Assistant Professor, Division of Information Science)

Entropy Regularization for Scalable, Safe and Robust Reinforcement Learning *

Lingwei Zhu

Abstract

The last decade has been an era of Reinforcement Learning (RL). It has witnessed tremendous successes in lab robotics or games such as Go. Basically, RL describes how an agent should act in response to a given environment in order to maximize its feedback or minimize the penalty. Such mechanism has been verified to exist in animals and humans, and can be abstracted and described in a self-contained mathematical formulation. However, unlike animals that have the notion of efficiency and risk-awareness, the standard RL formulation offers only a way to reach the goal but not about how the goal is reached. As a consequence, an RL agent might take extraorbitantly long to reach the goal, and/or incur unacceptable cost alone its path. Furthermore, the classic RL algorithms are greedy, hence prone to errors and noises. Due to the above-mentioned issues, currently RL falls short of realistic applications compared to other branches of contemporal machine learning such as supervised learning that has applications permeated in our daily life. This thesis, being application-oriented, attempts to bridge the gap by providing potentially helpful reference for the recent resurgence of interest in pratically applicable RL such as autonomous industrial control via RL. Specifically, in this thesis I tackle three important problems: scalability, safety and robustness of RL by leveraging the information theoretic quantities known as entropy. Emerging from information theory, the appearance of entropies is now prevalent in almost everywhere in engineering.

Intuitively, entropy in RL measures the uncertainty of the decision maker. By taking entropies into account when formulating the conventional objectives,

*Doctoral Dissertation, Graduate School of Science and Technology, Nara Institute of Science and Technology, April 07, 2022.

provably more efficient algorithms can be attained. Inside nowadays successful applications of RL such as the aforementioned robotic tasks or games, one can always see the presence of them in various forms, serving different objectives such as smoothening the optimization landscape, promoting exploration or rendering the agent risk-resilient. In this thesis, we investigate two most prevalent entropies: Shannon entropy and relative entropy (also known as Kullback-Leibler divergence), and see how they can serve as a regularization for the return, such that the agent is enabled to learn to be scalable, safe and robust. Since Shannon entropy can be regarded as a special case of KL divergence, the main thread of the thesis is actually on leveraging KL divergence between two consecutive decision making policies as regularization to achieve the above-mentioned desirable properties, by restricting the aggressive updates in realistic setup.

Keywords:

Reinforcement Learning, Entropy Regularization, Safety-awareness, Plant-wide Control, Robotics

Acknowledgements

No man is an island. This famous saying by John Donne as I interpret it, has the meaning of we always have to take something from somebody else. Though the years of pursuing PhD has been long and lonely, it is by no means in those years I have been an island. Many people kindly offered their help in various forms to me, to whom I would like to extend my sincere gratitude. Though I have not been home for almost three years due to the difficult situations (and is ongoing!), I owe the biggest to my parents who have been continuously supported me regardless of circumstances.

I would like to extend my utmost gratitutde to my supervisor Professor Takamitsu Matsubara, under whom I really learned to strive. He has taught me many things not only important for academia but also for the life. "A person's background is often less important than his motivation" now becomes one of my favorite mottoes. Prof. Matsubara is always in a good shape and works tirelessly. His energy and power really affects me and makes me into the academic path. Besides that, I have taken many informal but greatly helpful lessons in discussion with him on how to better present the story and make clear the logic. As I see it now, five years under Prof. Matsubara has been really just a blink of an eye.

My thanks also go to Assoc. Prof. Yunduan Cui at SIAT. During my master and early years of PhD he had been a good informal teacher from whom I learned a lot. Being also an ex-student of Prof. Matsubara, he had been an exemplar in the Robot Learning Lab and shown great patience when guiding me when I was so ignorant.

My gratitude also goes to my dear friend and collegue Dr. Zheng Chen who graduated a little bit earlier ahead of me. We had (and are having!) endless discussions whenever we met. As an expert working in the domain of bio-informatics, he always presents a new and fresh aspect from the applied point of view to problems we are trying to tackle, while I standing on the other extreme, paying attention often to the more fundamental or theoretical side. Our cooperation in the end of my PhD journey has been fruitful, and I am convinced that this cooperation will continue alone with our friendship.

List of Publications

Journal paper

- **Lingwei Zhu**, Go Takami, Mizuo Kawahara, Hiroaki Kanokogi, Takamitsu Matsubara, "Alleviating Parameter-tuning Burden in Reinforcement Learning for Large-scale Process Control". Computers and Chemical Engineering (CCE), vol. 158, pp.107658, 2021.

- **Lingwei Zhu**, Yunduan Cui, Go Takami, Hiroaki Kanokogi, Takamitsu Matsubara, "Scalable reinforcement learning for plant-wide control of vinyl acetate monomer process", Control Engineering Practice (CEP), Vol. 97, pp.104331, 2020

- **Lingwei Zhu**, Takamitsu Matsubara, "Cautious Policy Programming: Exploiting KL Regularization in Monotonic Policy Improvement for Reinforcement Learning". Machine Learning, (under review), 2022

Conference paper

- **Lingwei Zhu**, Toshinori Kitamura, Takamitsu Matsubara, "Cautious Actor Critic", in Asian Conference on Machine Learning 2021, (ACML'21), pp. 220-235.

- **Lingwei Zhu**, Yunduan Cui, Takamitsu Matsubara, "Dynamic actor-advisor programming for scalable safe reinforcement learning", IEEE International Conference on Robotics and Automation 2020, (ICRA'20), pp. 10681-10687.

- Yunduan Cui, **Lingwei Zhu**[*], Morihiro Fujisaki, Hiroaki Kanokogi, Takamitsu, Matsubara, "Factorial kernel dynamic policy programming for vinyl acetate monomer plant model control", IEEE International Conference on Automation Science and Engineering (CASE), 2018, pp. 304-309

Contents

List of Figures

List of Tables

1 Introduction

1.1 Reinforcement Learning and Entropy

Reinforcement Learning (RL) has seen tremendous successes in the last decade, demontrating super-human level problem-solving abilities in playing video games [Mnih et al., 2015b], robotics [Andrychowicz et al., 2020], strategic games [OpenAI, 2018, Silver et al., 2017, Wurman et al., 2022], etc. The heart of RL lies in the simple mechanism that given a state the agent or decision maker (for example, a dog) decides which action to take to maximize the incoming reward (such as food) or to minimize penalty. Such simple mechanism that exists in our daily lives boils down to a set of mathematical equations expressing the relationship between state, action and reward, as shown in Figure 1.1: at time t, given a state

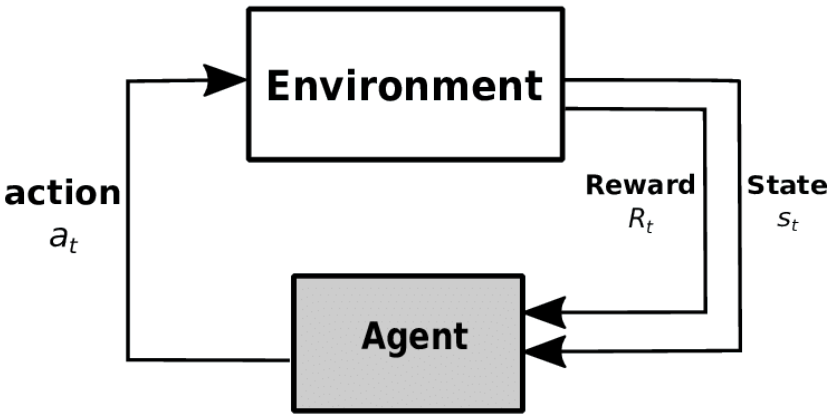


Figure 1.1: Reinforcement learning interaction loop.

s_t, the agent chooses an action a_t to interact with the environment in which it is

located, and then the environment outputs a reward signal R_t as a feedback to the act of the agent. While being seemingly simple, the above-mentioned astonishing successes as well as many other advances of RL are based on this simple but crucial notion [Sutton and Barto, 2018]. As a matter of fact, RL is intrinsically significantly more difficult than other contemporal branches of machine learning such as supervised learning [Kober et al., 2013]. In supervised learning, one has data and labels, the prediction on a data instance can be inspected to see if it matches the corresponding label. However, such labels or teaching signals are totally ambiguous in that it is unclear what does 'label' refer to in RL: an agent will have to infer important information for its decision solely from the reward function and past experience.

Another important point is that, RL is in nature *sequential*: the agent will have to make several decisions in a row. Starting from any given state, it decides an action, and then it is taken to the next state by that action (perhaps with some random factors in effect). In this regard, supervised learning can be somehow seen as a single-step case of RL (although not correct). This sequential nature is crucial for its power: since we know realistic interaction is seldom single-step, usually many steps are required for one to fulfill a goal. On the other hand, the extension from single step to sequential decision making brings significant complexity since what the agents has done now, will probably affect the very far future. It also implies that greedily selecting the most rewarding action in a given state might not result in the best overall return, since the actions will change the environment and potentially lead to very poor subsequent states, just like an athlete greedily consuming hamburgers for a moment's pleasure. In this case, the agent should care about the *long-term rewards* instead of being short-sighted. Now we have better understanding of the factor that distinguishes supervised learning from RL, which was also the reason why supervised learning is not a single-step instance of RL: it is usually assumed the labels in supervised learning are independently and identically distributed (iid). However, this assumption fails immediately in the RL context, which relies on Markov Decision Processes (MDPs) [Puterman, 1994] as the theoretical background. In short, we can understand the difficulty as the *future depends on the choice the agent makes now* i.e., the environment can be changed by the agent's actions. Therefore, the

data collected by the agent, or experiences such as $s_t, a_t, r_t, s_{t+1}, a_{t+1}, \ldots$, are necessarily correlated. The rich literature developed from supervised learning or statistics does not simply translate to methods applicable in an RL context.

Given the above-mentioned complexity of RL, how should the agent behave in order to maximize its long-term rewards over the course it interacts with the environment? In this thesis, we adopt the most 'classic' approach known as dynamic programming [Bellman, 2003, Bertsekas, 2005a, Bertsekas and Tsitsiklis, 1996] that evaluates the utility of each action separately and therefore learns an ordering between each action given arbitrary state [Rummery and Niranjan, 1994, Watkins and Dayan, 1992]. Once this ordering is learned, the agent will be able to pick the action with the largest utility in every state, which results in globally optimal policy, as a result of the famous Bellman's optimality principle [Bellman, 2003]: in order to optimally behave globally, the agent will have to behave optimally locally.

A crucial difference in dynamic programming methods lies in the availability of the model knowledge. In this thesis we are positioned in a model-free context which refers to that the agent does not have model knowledge of the environment such as what will happen if an action A is taken in state S. In this case, the agent will have to collect sufficient experiences and then distill the model knowledge from the experiences. This genre of dynamic programming is often known as approximate dynamic programming and is tightly connected with RL [Powell, 2007, Szepesvari, 2010] As it appears, model-free RL is inherently slower than those methods explicitly equipped with high-quality model knowledge, but it is still surprising powerful. In fact, most of the astonishing successes of RL by far were based on model-free RL methods.

The notion of maximizing long-term reward is essential but often insufficient for modern complicated RL scenarios. This is because solely maximizing reward will result in deterministic optimal policy [Puterman, 1994], which refers to the agent will always follow exactly same path to the goal. If the environment has been changed, e.g. by adding adversary effects, such deterministic strategy is brittle and might incur very poor performance even if the change to the environment is small. More technically, we can consider the utility function has been corrupted somehow with some noise. This implies the ordering between

actions might have also been corrupted as well. Greedily following this corrupted utility function by picking the action with the largest utility is no longer safe; actually sometimes very poor performance can be incurred by deterministically doing so. To tackle this problem, a very natural approach is to make action selection mechanism *stochastic*: if every action has probability being chosen, then it can be expected the problem be alleviated. While there exists plain stochastic methods such as the well-known ϵ-greedy, it cannot adjust the probability of each action according to some measure. The recent introduction of entropy regularization into RL [Todorov, 2006, Ziebart, 2010] provides a principled approach for obtaining stochastic action selection rule in which the probability of action being selected is proportional to its utility. More interestingly, as is often seen in the optimization literature [Beck, 2017, Boyd and Vandenberghe, 2004, Nesterov, 2018, Rockafellar, 1970], adding a convex regularizer often results in significantly accelerated convergence speed, even though this is not exactly the case in our dynamic programming setting which relies on contraction property of the Bellman operator [Kreyszig, 1991] instead of on gradients [Sutton et al., 1999] but it indeed has been observed the introduction of entropies such as the Shannon entropy, Kullback-Leibler (KL) divergence, or even the Tsallis entropy [Tsallis, 1988] improves the learning speed of the resultant algorithms. Those entropies bring probabilistic action selection rules which are well-defined probability distributions over actions. In fact, recent advances in RL often comprise one of them serving various purposes such as promoting exploration [Haarnoja et al., 2017], accelerating convergence speed [Haarnoja et al., 2018], robustifying the agent against errors and/or noises [Azar et al., 2012, Fox et al., 2016], to name a few.

This thesis is basically application-oriented, concerned with leveraging entropy regularization in model-free RL for achieving autonomous control on realistic problems. But we aim to attain superior control performance without the loss of theoretical rigor. Though RL has been mostly limited to simulation or lab robotics, the interest in extending it to more realistic problems such as industrial process control, recommendation systems and others has been increasing dramatically over the years. In this thesis, we aim to demonstrate the effectiveness of entropy-regularized RL mainly in the following three aspects with novel RL al-

gorithms suitable for each application: (1) Scalability; (2) safety; (3) robustness. We detail those three aspects in the following contribution section.

Thesis Statement: This thesis is concerned with proposing and exploiting novel methods of entropy-regularized model-free RL to realistic problems that are of crucial importance such as the autonomous control of industrial processes, robotics, etc. The power of the proposed methods is mainly displayed in the three aspects (1) scalability; (2) safety; (3) robustness. They respectively address the problem of (1) how to apply RL not just to game playing but also to complicated realistic problems; (2) how to safely apply RL to safety-critical problems such as industrial processes and (3) how to save the overhead of fine-tuning RL on realistic systems. This thesis contributes as a step towards fully autonomous control of the above-mentioned real-world systems by RL.

1.2 Thesis Contribution and Outline

The main contributions of this thesis lie in drawing close the theory of RL and practical applications. Compared with other branches of contemporal machine learning such as supervised learning, RL significantly falls short of realistic applications as a result of several dilemmas. In this thesis, I attempt to bridge the gap by proposing novel and useful theories based on entropy regularization and validate them on practical scenarios such as industrial plants or robotics. A commercial-level large-scale chemical process simulator for a Vinyl Acetate Monomer plant is frequently used as the testbed. This thesis mainly tackles the issues of scalability, safety and robustness in RL via the tool of entropy. I hope the thesis could play an important role in the coming resurgence of interest in pratical RL deployment and applications.

1. Scalability: we show that entropy-regularized model-free RL can be leveraged to realize fully autonomous control for a large-scale industrial process, with very cheap computational overhead and comparable performance to the state-of-the-art chemical knowledge based controllers (Chapter 3).

2. Safety: RL is an inherently trial-and-error process, the agent will have experience both good and bad experiences in order to learn to distinguish between actions leading to high return from those to poor situations. However in many realistic problems it is unacceptable to experience bad situation such as in the industrial processes. We propose a novel algorithm focusing on how to let the agent safely learn (Chapter 4).

3. Robustness: RL algorithms are notorious for the difficulty of fine-tuning hyperparameters and long running time. Even slightly varied hyperparameters could cause drastically degraded performance. In this chapter we resort to the classic but not straightforwardly applicable monotonic improvement literature and propose a novel algorithm by incorporating entropy regularization. The resulting algorithm is robust against hyperparameter changes and can achieve the same level of performance as prior methods but with significantly reduced tuning time (Chapter 5, 6).

2 Entropy

2.1 Introduction

In this thesis the entropies used are the Shannon entropy and relative entropy (known as Kullback-Leibler (KL) divergence). Emerging from the information theory [Cover and Thomas, 2006], those two entropies are probably the most widely used entropies in engineering. Basically, Shannon entropy measures how many bits are required to encode the information contained in the measured random variable, while KL divergence measures the expectation of the difference between two distributions in the logarithmic space [Kullback, 1959].

In RL, the Shannon entropy provides a measure of how stochastic a policy might be, while KL divergence measures how close two policies are. It can be intuitively conceived that if Shannon entropy of the policy is added to the objective function, the resultant solution policy is encouraged to be more stochastic. On the other hand, KL divergence added to the objective will make the learning process more 'conservative' since the maximum change for one policy update is somehow limited. In this thesis, we focus on value-based methods, which refers to learning a value function that is the conditional expectation for long-term reward given the initial state-action pair [Bertsekas and Tsitsiklis, 1996]. If the optimal value function could be learned, the optimal policy can be straightforwardly extracted by acting greedily with respect to this value function. Value-based methods are classic, enjoy strong theoretical guarantees. On the other hand, the downside is also apparent: value-based methods cannot handle continuous state-action spaces as compared to policy gradient methods that directly parametrize the policy and update it using modern auto-differentiation mechanisms. Nonetherless, we will

consider value-based methods with entropy regularization in this thesis, and defer analysis for the policy gradient methods to future work.

2.2 Value Iteration

Value iteration refers to learning the mapping between state-action pairs to a scalar value that is used to rank the desirability among all state-action pairs. Such scalar value is the conditional expectation expressed as follows:

$$Q_\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \middle| s_0 = s, a_0 = a\right]. \tag{2.1}$$

It is a classic result that if the agent only cares about maximizing the cumulative reward, the optimal policy is deterministic and stationary [Bertsekas and Tsitsiklis, 1996, Puterman, 1994]. Therefore, we can write the following recursion for iteratively obtaining the optimal value function and hence optimal policy:

$$\begin{cases} \pi_{k+1} = \mathbf{1}\{a = \arg\max Q_{\pi_k}(s, \cdot)\} \\ Q_{\pi_{k+1}} = r(s,a) + \gamma \mathbb{E}_{s' \sim p(\cdot|s,a)} \left[\mathbb{E}_{a' \sim \pi_{k+1}(\cdot|s')} \left[Q_{\pi_k}(s', a')\right]\right] \end{cases} \tag{2.2}$$

We can make the notations uncluttered by using abbreviation:

$$\begin{cases} \pi_{k+1} = \mathcal{G}(Q_k) := \arg\max_\pi \langle \pi, Q_k \rangle \\ Q_{k+1} = r + \gamma P^{\pi_k} Q_k \end{cases} \tag{2.3}$$

where $\gamma P^{\pi_k} Q_k = \gamma \left(\sum_{s'} P(s'|s,a) \sum_{a'} \pi_k(a'|s') Q_k(s', a')\right)$. The reason we used Q_k instead of Q_{π_k} is because exactly updating the value function is intractable since we often do not know the model P, and the state-action spaces are huge to be exhaustively searched. Modern RL such deep Q-network (DQN) [Mnih et al., 2015a] often uses a replay buffer to sample the state-action pairs, as well as approxmate the transition model. On the other hand, since the policy is greedy, it could easily be corrupted by various errors introduced by approximation [Fu et al., 2019]. This is because the greedy policies lie on the endpoints of the probability simplex and if the values are not separated large enough, perturbations could easily change the ordering the actions, leading to a completely different action being chosen.

2.3 Shannon Entropy

An intuitive solution to the above-mentioned issue of errors corrupting action ordering is to have a stochastic policy, which can be done via introducing the Shannon entropy since adding the Shannon entropy to the reward function forces the agent to maximize cumulative reward as well as the stochasticity of the policy. The optimal policy is this case is stochastic in the sense of being a Boltzmann softmax. Let us define the Shannon entropy using the shorthand notation $\mathcal{H}\left(\pi(\cdot|s)\right) := -\sum_a \pi(a|s)\ln\pi(a|s) = -\left\langle \pi, \ln\pi \right\rangle|_{s\in\mathcal{S}}$, we aim to learn the optimal value function:

$$V_*(s) = \max_\pi \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t\left(r(s_t,a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))\right)\middle| s_0 = s\right]. \tag{2.4}$$

That is, the optimal policy attempts to maximize not only the reward but also the entropy for of policy for every state. It can be shown by the Lagrangian multiplier method or Legendre-Fenchel transform [Boyd and Vandenberghe, 2004] the optimal policy takes the form of the Boltzmann softmax [Haarnoja et al., 2017, Ziebart, 2010]:

$$\pi_{k+1}(a|s) = \arg\max_\pi \left\langle \pi, Q_k + \alpha\mathcal{H}\left(\pi\right)\right\rangle =: \exp\left(\alpha^{-1}\left(Q_k - V_k\right)\right). \tag{2.5}$$

We can now modify the value iteration scheme Eq. (2.3) to a soft value iteration recursion as follows [Haarnoja et al., 2017, Song et al., 2019]:

$$\begin{cases} \pi_{k+1} = \mathcal{G}^{\alpha,0}(Q_k) := \arg\max_\pi \left\langle \pi, Q_k + \alpha\mathcal{H}\left(\pi\right)\right\rangle \\ Q_{k+1} = r + \gamma P^{\pi_k}\left(Q_k + \alpha\mathcal{H}\left(\pi_k\right)\right). \end{cases} \tag{2.6}$$

An immediate advantage is that the Boltzmann softmax policy assigns positive probability to every action since $\exp(\cdot) > 0$ so long as the action value does not take on $-\infty$. The stochasticity can be controlled by the scalar coefficient α. It is a classic result that

$$\lim_{\alpha\to 0}\exp\left(\alpha^{-1}\left(Q(s,a) - V(s)\right)\right) = \mathbf{1}\{a = \arg\max_a Q(s,a)\},$$

i.e. $\pi_{k+1} \to \mathcal{G}(Q_k)$ as $\alpha \to 0$.

2.4 KL Divergence

We can also consider the two-argument information theoretic regularizer KL divergence $D_{KL}(\pi||\bar{\pi}) := \langle \pi, \ln \pi - \ln \bar{\pi} \rangle$. Intuitively, maximizing Shannon entropy and reward will result in stochastic policy can be explained by KL divergence since the Shannon entropy is the KL divergence between the current policy and uniform distribution: $D_{KL}\left(\pi \middle\| \frac{1}{|\mathcal{A}|}\right) = -\langle \pi, \ln \pi - \ln |\mathcal{A}| \rangle = \mathcal{H}(\pi)$ up to a constant. Now we can express the recursion as:

$$\begin{cases} \pi_{k+1} = \mathcal{G}^{0,\beta}(Q_k) := \arg\max_\pi \langle \pi, Q_k - \beta D_{KL}(\pi||\pi_{\pi_k}) \rangle \\ Q_{k+1} = r + \gamma P^{\pi_k} \left(Q_k - \beta D_{KL}\left(\pi_{k+1} \,\|\, \pi_k\right)\right). \end{cases} \tag{2.7}$$

KL divergence plays the most important role in this thesis since it can effective prevent the errors from destroying learning. Furthermore, unlike the Shannon entropy whose effect is mostly empirical in value-based methods, KL divergence regularization provably reduces the error. Let us consider the following upper bound due to [Vieillard et al., 2020a] where reward is augmented by the KL divergence:

$$\|Q^* - Q^{\pi_{k+1}}\|_\infty \leq \frac{2}{1-\gamma} \left\| \frac{1}{k} \sum_{j=0}^{k} \epsilon_j \right\|_\infty + \frac{4}{1-\gamma} \frac{V_{max}}{k},$$

where ϵ_j are errors and $V_{max} = \frac{r_{\max}}{1-\gamma}$. By comparing it with the non-improvable approximate modified policy iteration (AMPI) bound where the reward is not augmented [Scherrer et al., 2015]:

$$\|Q^* - Q^{\pi_{k+1}}\|_\infty \leq \left((1-\gamma) \sum_{j=1}^{k} \|\epsilon_j\|_\infty \right) + \frac{2\gamma^{k+1}}{1-\gamma} V_{max},$$

we see that the error term for the KL regularization case is sup-over-sum. Under mild assumptions such as ϵ_j are iid distributed under the natural filtration [Azar et al., 2012], the summation over errors asymptotically cancels out. On the other hand, the error term for AMPI depends on the summation of *maximum* of every iteration, which is typically large. Further, the dependence of error on the horizon is linear $\frac{1}{1-\gamma}$ rather than quadratic, which is a significant improvement as typically $\gamma \approx 1$.

2.5 Miscellaneous

2.5.1 Putting Shanon entropy and KL divergence together

Recent works have investigated the possibility of putting the Shannon entropy and KL divergence together [Kozuno et al., 2019, Vieillard et al., 2020a,c]. In this case, the final optimal policy converges to the optimal Boltzmann softmax as in Section 2.3, rather than the ultimately deterministic policy in Section 2.4 (for the KL case the policies during learning are still stochastic).

$$\begin{cases} \pi_{k+1} = \mathcal{G}^{\alpha,\beta}(Q_k) := \arg\max_\pi \langle \pi, Q_k + \alpha\mathcal{H}(\pi) - \beta D_{KL}(\pi||\pi_{\pi_k})\rangle \\ Q_{k+1} = r + \gamma P^{\pi_k}\left(Q_k + \alpha\mathcal{H}(\pi_{k+1}) - \beta D_{KL}(\pi_{k+1} \,\|\, \pi_k)\right). \end{cases} \tag{2.8}$$

This formulation might find useful application in scenarios where stochastic policy is desired, but also conservativeness during learning is paramount.

2.5.2 Other entropies

There are other less popular entropies such as the generalizations of the Shannon entropy. For example, the Tsallis entropy generalizes Shannon entropy and plays an important role in statistical physics. Its expression is:

$$S_q(\pi) := \frac{1}{q-1}\left(1 - \langle \mathbf{1}, \pi^q\rangle\right), \; S_q(\pi) \to \mathcal{H}(\pi) \;\text{ as } q \to 1.$$

Here, $q \in \mathbb{R}$ is a scalar known as the entropic index controlling the properties of the Tsallis entropy [Lee et al., 2020, Tsallis, 1988, 2009]. When $q = 2$, we recover the sparsemax distribution [Lee et al., 2018, Martins and Astudillo, 2016] that assigns probability only to a small subset of actions. Another possible generalization of the Shannon entropy is the Rényi entropy [Zhang et al., 2021]

$$R_\tau(\pi) := \frac{1}{1-\tau}\ln\left(\langle \mathbf{1}, \pi^\tau\rangle\right) = \frac{1}{1-\tau}\ln\left(\sum_a \pi(a|s)^\tau\right).$$

$R_\tau(\pi)$ also tends to the Shannon entropy when $\tau \to 1$. Investigation of RL with those entropies is left to future work.

3 | Scalability

This Chapter focuses on the use of entropy-regularized RL algorithm in process control problems. We aim to realize autonomous control on a chemical process producing Vinyl Acetate Monomer. First we examine the problem in a simpler setting named *local control*, referred to as controlling the plant with only a part of the process is activated. We then study the more challenging *plant-wide* control setting all parts of the complicated chemical process is activated.

3.1 Introduction

Chemical plants consist of several processing units that cooperatively produce chemical product via complex interactions and their coordination is important for safe and profitable plant operations [Dotoli et al., 2015, Metzger and Polakow, 2011]. The conventional control strategies for chemical plants are primarily formed as a set of heuristics [Olsen et al., 2005, Zheng et al., 1999] and optimized in both steady and dynamic simulation [McAvoy, 1999]. In terms of the Model predictive control (MPC), another popular method in chemical process control [Cheng et al., 2008], it is a model-based method and suffers exorbitant online computational cost with large scale plant as well as the long prediction length [Lee and Wong, 2010]. A method that does not rely on both human engineer knowledge and model remains challenging in this field.

As a popular test problem in chemical plant design, optimization, and control, the Vinyl Acetate Monomer (VAM) plant model benchmark problem was originally proposed by [Luyben and Tyréus, 1998]. This problem is uniquely suited for researchers pursuing simulation, design, and control studies as it:

1. Has a realistically large flowsheet, containing several standard chemical unit operations with high level chemical interactions.

2. Includes typical industrial characteristics of recycle streams and energy integration.

This model has been further developed by [Chen et al., 2003, Machida et al., 2016], while several other studies on control system design have been conducted [Chen and McAvoy, 2003, Olsen et al., 2005, Seki et al., 2010]. Based on experienced engineers' intuition and judgment in eliminating and evaluating candidate architectures, these works mainly focus on classical controller design procedures that require analysis of the process dynamics, development of an abstract mathematical model, and derivation of a control law that meets certain design criteria.

As an integral part of contemporary machine learning, reinforcement learning [Sutton and Barto, 1998] agents search for optimal policies by interacting with their environment without any prior knowledge and express a remarkably broad range of control problems in a natural manner. Example applications include electrical power systems control [Ernst et al., 2005], dynamic power management [Liu et al., 2010], and robot control [Kober et al., 2013]. Such a bio-inspired approach is suitable for application towards learning the control policies for chemical process plants. However the application of reinforcement learning towards chemical process plants, which commonly features a large number of sensors, controllers parameters and requires high stability for safety, is a less explored area. Some previous works with reinforcement learning were conducted besides heuristic solutions. For model-free methods, [Hoskins and Himmelblau, 1992] applied neural networks and reinforcement learning to cool a reactor simulator via one valve. Applications in process control field range from electricity grid control to dynamic power management [Ernst et al., 2005, Hoskins and Himmelblau, 1992, Liu et al., 2010]. For model-based methods, MPC based dynamic programming was utilized to plant with one or two dimensional state and one dimensional action in [Lee and Wong, 2010]. However, to the best of the authors' knowledge there is no application of reinforcement learning towards the large scale problem like VAM plant model in the literature due to the curse of dimensionality in high dimensional state, the intractable computational cost with large action spaces and the difficulty in accurately modelling complex plant systems.

In this section, we focus on applying reinforcement learning to the local control problem in a large scale chemical plant control to optimize production while maintaining plant stability without any knowledge of the plant models. Since a typical chemical plant has a large number of sensors and actuators, its control problem could be formulated as a Markov decision process involving high dimensional state and large action space that might be difficult to solve by previous methods due to computational complexity and sample insufficiency. To overcome these issues, a new approach, Factorial Policy Dynamic Policy Programming (FKDPP) is proposed. Inspired by both Kernel Dynamic Policy Programming (KDPP) [Cui et al., 2017a] which outperforms other conventional reinforcement learning methods such as LSPI [Lagoudakis and Parr, 2003] in robot control tasks with high dimensional states and [Matsubara et al., 2014] that factorizes the high dimensional state Hidden Markov Model to reduce computational complexity, FKDPP enjoys a factorial policy model and factor-wise kernel-based smooth policy update by the regularization with the Kullback-Leibler divergence between the current and updated policies. Thus, the proposed approach is suited for learning near-optimal policies in large scale chemical plant models. The effectiveness of FKDPP is validated in the VAM plant model benchmark problem, with experimental results showing that the proposed method outperforms other methods in terms of VAM yield, quality and system stability.

3.2 Local Chemical Process Control

3.2.1 Local Control Definition

VAM Plant Description

The process flow diagram of the VAM plant model is shown in Fig. 3.1. It consists of eight parts:

Part 1 is for the input of raw materials: ethylene (C_2H_4), oxygen (O_2), and acetic acid (AcOH or CH_3COOH) are provided as fresh feed streams.

Part 2 converts raw materials into vinyl acetate (VAM, $CH_2 = CHOCOCH_3$ with $=$ denoting a double chemical bond) along with with water (H_2O) and carbon dioxide (CO_2) as byproducts in a reactor. The following reactions take

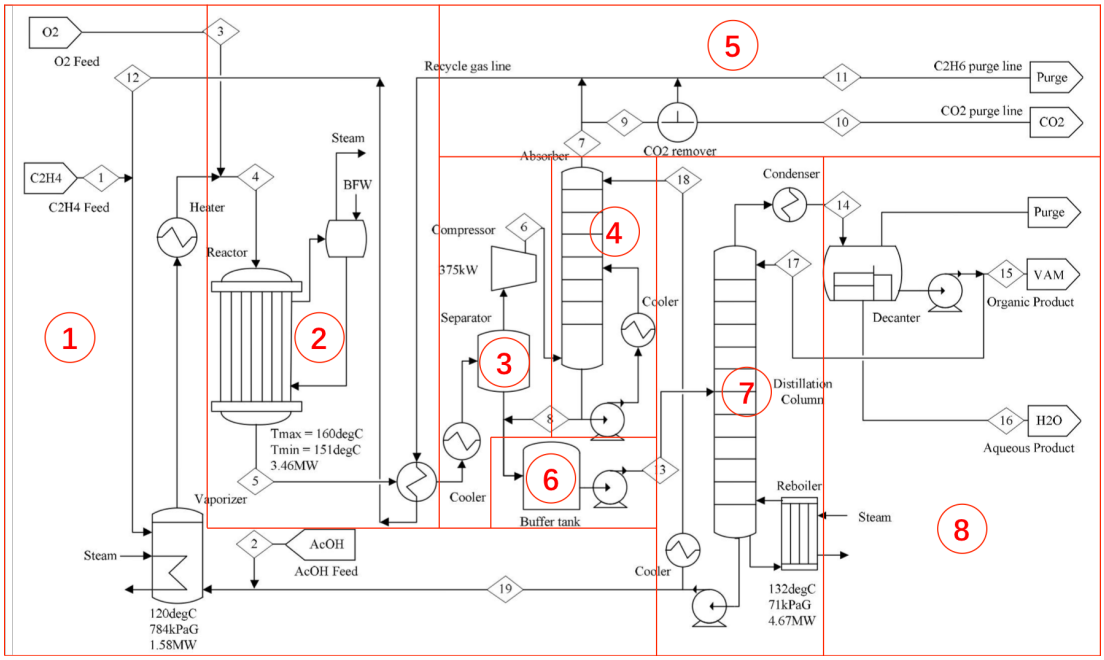


Figure 3.1: Process flow diagram of the VAM plant model [Machida et al., 2016].

place during this process:

$$C_2H_4 + CH_3COOH + \frac{1}{2}O_2 \longrightarrow CH_2{=}CHOCOCH_3 + H_2O,$$

$$C_2H_4 + 3\,O_2 \longrightarrow 2\,CO_2 + 2\,H_2O$$

Part 3 contains a cooler, a separator and a compressor. Since both reactions are highly exothermic, heat from the reaction is dissipated by boiler feed water (BFW) circulation. Steam is generated on the shell side of the reactor, while gas emitted from the reactor is processed this part where leftover acetic acid, water and vinyl acetate are condensed as liquid VAM crude while separated gases including unreacted ethylene, oxygen, carbon dioxide, inert ethane (C_2H_6), and a small amount of gaseous vinyl acetate is compressed for circulation.

Part 4 has an absorber to capture vinyl acetate gas via cold acetic acid from Parts 3, 7 and send it to Part 6. Other gases are fed into Part 5.

Part 5 is the gas purge system. It keeps the concentrations of CO_2 around $5 \sim 10$ mol% and C_2H_6 around 5 mol% in the gas recycle line.

Part 6 is the intermediate buffer tank to mix vinyl acetate and acetic acid with VAM crude condensed from Part 3.

Table 3.1: Observed variables and control parameters of investigated task.

Table 3.2: Observed variables of the local control task.

Name	Description	Control criteria
FI560.F	flow rate of VAM.	to be optimized as VAM yield
LI550.L	level (%) of the decanter tank.	$< 100\%$
s407.F	flow rate from the distillation column in Part 7 to the decanter tank.	> 0
QI553.Q	acetic acid density.	$< 150ppm$
QI560.Q	VAM density. (inversely proportional to VAM quality)	$< 500ppm$

Part 7 contains a distillation that separates the VAM crude and acetic acid from the intermediate buffer tank (Part 6). VAM-water mixture is then discharged from the bottom while the acetic acid is recycled to the absorber (Part 4) and raw material feed section (Part 1).

Part 8 contains a decanter where the production vinyl acetate is finally decanted in.

3.2.2 Tasks

Recall that we focus on local control which refers to observing and controlling the decanter tank (Part 8) and the distillation column (Part 7), with the overall objective of optimizing VAM yield and its quality while maintaining stability as the first step towards the optimal control of the whole VAM plant in Fig. 3.1 by reinforcement learning. This problem features a nine dimensional state spaces including observed variables and control parameters detailed in Tables 3.2 and 3.3, respectively. A finite discrete action set is defined as increasing/decreasing

Control unit	Description	Effect
Pressure controller 1	Maintain top pressure of the column with N_2 and gas purge	Acetic acid density rises/declines N_2 density rises/declines
Pressure controller 2	Maintain pressure of the 3rd stage of the distillation column with superheated steam	Column temperature rises/declines Affect VAM yield and quality Steam amount increase/decreases Steam generation is part of cost
Flow controller 2	Control decanter feed flow temperature with cooling water	Control Decanter feed flow Affect stability of the process
Temperature controller	Maintain temperature profile and product quality by controlling the reflux flow rate	Affect VAM yield and quality Affect stability of the process Control reverse flow rate

Table 3.3: Control units of the local control task.

each control parameter. The initial parameters of the plant are provided to start from equilibrium. The specific objective is optimizing the VAM production's quality and quantity while keeping the level of the decanter tank and the flow rate from the distillation column within safe limits. This is done by manipulating the decanter's reflux flow rate and feed flow temperature, the distillation column's steam flow rate and pressure.

3.2.3 Approach

Reinforcement Learning Basics

RL models problems as Markov Decision Processes (MDPs) expressed by a quintuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$. $\mathcal{S}$ denotes the state space; $\mathcal{A}$ denotes the action space, $\mathcal{T}^{a}_{ss'}$ denotes the transition from state s to $s^{'}$ with action a taken. $\mathcal{R} = r^{a}_{ss'}$ is the immediate reward under that transition. $\gamma \in (0, 1)$ is the discount factor.

Given a state s, the value function V_π following a specified policy π from that state is defined as the cumulative discounted reward:

$$V_\pi(s) = \mathrm{E}_{\pi,\mathcal{T}}\left[\sum_{t=0}^{\infty} \gamma^t r_{s_t} \,\middle|\, s_0 = s\right] \tag{3.1}$$

where s_t is the state at time step t and r_{s_t} is the reward at state s_t. $E_{\pi,\mathcal{T}}$ denotes the expectation with respect to π and transition probability $\mathcal{T}$.

RL methods search an optimal policy π^* to maximize (3.1), i.e., to satisfy the recursive Bellman equations:

$$V_{\pi^*}(s) = \max_{\pi} \sum_{\substack{a\in\mathcal{A} \\ s'\in\mathcal{S}}} \pi(a|s) \Big[\mathcal{T}^a_{ss'} \big(r^a_{ss'} + \gamma V_{\pi^*}(s') \big) \Big]. \tag{3.2}$$

It is also convenient to use Q_π to denote the value function of state-action pairs $(s, a) \in \mathcal{S} \times \mathcal{A}$ under the policy π:

$$Q_{\pi^*}(s,a) = \max_{\pi} \sum_{s'\in\mathcal{S}} \mathcal{T}^a_{ss'} \big(r^a_{ss'} + \gamma \sum_{a'\in\mathcal{A}} \pi(a'|s') Q_{\pi^*}(s', a') \big). \tag{3.3}$$

Dynamic Policy Programming

Dynamic Policy Programming (DPP) [Azar et al., 2012] is a value function based RL algorithm. Thanks to its theoretically proven convergence, sample efficiency and stability in learning, DPP has many extensions from complex robot arm manipulation [Cui et al., 2017a] to DRL based robot control [Tsurumine et al., 2019] and inverse RL [Uchibe, 2018]. DPP enforces the stability of policy update by adding a Kullback-Leibler (KL) divergence term between some baseline policy and the current policy that prevents the agent from taking aggressive steps:

$$KL\big(\pi(\cdot|s) \,\|\, \bar{\pi}(\cdot|s) \big) = \sum_{a\in\mathcal{A}} \pi(a|s) \log \left(\frac{\pi(a|s)}{\bar{\pi}(a|s)} \right). \tag{3.4}$$

The new optimal value function is obtained by incorporating Eq. (3.4) into (3.2):

$$V_{\pi^*}(s) = \max_{\pi} \sum_{\substack{a\in\mathcal{A} \\ s'\in\mathcal{S}}} \pi(a|s) \left[\mathcal{T}^a_{ss'} \big(r^a_{ss'} + \gamma V_{\pi^*}(s') \big) - \frac{1}{\eta} \log \left(\frac{\pi(a|s)}{\bar{\pi}(a|s)} \right) \right] \tag{3.5}$$

where the η is the temperature parameter that weights the effect of the KL term.

Following [Azar et al., 2011, Todorov, 2006], the new optimal value function and policy satisfy the recursive equations:

$$V^{\pi^*}_{\bar{\pi}}(s) = \frac{1}{\eta} \log \sum_{a\in\mathcal{A}} \bar{\pi}(a|s) \exp \left[\eta \sum_{s'\in\mathcal{S}} \mathcal{T}^a_{ss'} \big(r^a_{ss'} + \gamma V^{\pi^*}_{\bar{\pi}}(s') \big) \right], \tag{3.6}$$

$$\bar{\pi}^*(a|s) = \frac{\bar{\pi}(a|s)\exp\left[\eta \sum_{s' \in \mathcal{S}} \mathcal{T}^a_{ss'}\left(r^a_{ss'} + \gamma V^{\pi *}_{\pi}(s')\right)\right]}{\exp\left(\eta V^{\pi^*}_{\pi}(s)\right)}. \tag{3.7}$$

The optimal value function and policy need to be obtained iteratively. By repeatedly replacing the baseline policy in Eq.(3.6), the optimal value function can be asymptotically obtained. Defining an action preference function of the form in [Sutton, 1996] and denoting the iteration t:

$$\Psi_{t+1}(s,a) = \frac{1}{\eta}\log \bar{\pi}^t(a|s) + \sum_{s' \in \mathcal{S}} \mathcal{T}^a_{ss'}\left(r^a_{ss'} + \gamma V^t_{\bar{\pi}}(s')\right) \tag{3.8}$$

Dynamic Policy Programming Recursion

Dynamic Policy Programming (DPP) [Azar et al., 2012] solves Markov decision process (MDP) with smooth policy updates by employing the Kullback-Leibler divergence between current and new policies as a regularization term. According to [Azar et al., 2012], such a smooth policy update is beneficial when working with a limited number of samples. A MDP is defined by $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$. $\mathcal{S} = \{s_1, ..., s_n\}$ is a finite set of states, $\mathcal{A} = \{a_1, ..., a_m\}$ is a finite set of discrete actions. $\mathcal{T}^a_{ss'}$ represents the transition probability from s to s' under the a and $r^a_{ss'} = \mathcal{R}(s, s', a)$ is the corresponding reward, $\gamma \in (0, 1)$ is the discount factor. The policy $\pi(a|s)$ denotes the probability of taking the action a under the state s. The optimal value function with the regularization term that maximizes expected discounted total reward, while minimizing the difference between current policy π and baseline policy $\bar{\pi}$, follows a Bellman equation:

$$V^*_{\bar{\pi}}(s) = \max_{\pi} \sum_{\substack{a \in \mathcal{A} \\ s' \in \mathcal{S}}} \pi(a|s)\left[\mathcal{T}^a_{ss'}\left(r^a_{ss'} + \gamma V^*_{\bar{\pi}}(s')\right) - \frac{1}{\eta}\log\left(\frac{\pi(a|s)}{\bar{\pi}(a|s)}\right)\right]. \tag{3.9}$$

For solving the optimal value function, the action preferences [Sutton and Barto, 1998] for all state action pairs $(s, a) \in \mathcal{S} \times \mathcal{A}$ in the $(t+1)$-th iteration are defined according to [Azar et al., 2012]:

$$\Psi_{t+1}(s,a) = \frac{1}{\eta}\log \bar{\pi}^t(a|s) + \sum_{s' \in \mathcal{S}} \mathcal{T}^a_{ss'}\left(r^{\,a}_{ss'} + \gamma V^t_{\bar{\pi}}(s')\right). \tag{3.10}$$

Instead of the optimal value function, DPP learns optimal action preferences to determine the optimal control policy throughout the state-action space. The DPP recursion is calculated by:

$$\begin{aligned}\Psi_{t+1}(s,a) &= \mathcal{O}\Psi_t(s,a) \\ &= \Psi_t(s,a) - \mathcal{M}_\eta\Psi_t(s) + \sum_{s'\in\mathcal{S}} \mathcal{T}^a_{ss'}\left(r^{\ a}_{ss'} + \mathcal{M}_\eta\Psi_t(s')\right)\end{aligned} \tag{3.11}$$

where $\mathcal{M}_\eta\Psi_t(s) = \sum_{a\in\mathcal{A}} \frac{\exp\left(\eta\Psi_t(s,a)\right)\Psi_t(s,a)}{\sum_{a'\in\mathcal{A}}\exp\left(\eta\Psi_t(s,a')\right)}$ is the the Boltzmann soft-max operator.

DPP can be extended to large-scale (continuous) state spaces via function approximation, i.e., linear function approximation with basis functions. Here we define the n-th state-action pair from a set of N samples as $\boldsymbol{x}_n = [s_n, a_n]_{n=1:N}$, where $\boldsymbol{\phi}(\boldsymbol{x}_n)$ denotes the $m\times 1$ output vector of m basis functions, $[\varphi_1(\boldsymbol{x}_n), ..., \varphi_m(\boldsymbol{x}_n)]^{\mathrm{T}}$. The approximate action preferences in the t-th iteration follow $\hat{\Psi}_t(\boldsymbol{x}_n) = \boldsymbol{\phi}(\boldsymbol{x}_n)^{\mathrm{T}}\boldsymbol{\theta}_t$ where $\boldsymbol{\theta}_t$ is the corresponding $m\times 1$ weights vector. The empirical least-squares solution of minimizing the loss function $J(\boldsymbol{\theta}; \hat{\boldsymbol{\Psi}}_t) \triangleq \|\boldsymbol{\Phi}\boldsymbol{\theta} - \mathcal{O}\hat{\boldsymbol{\Psi}}_t\|_2^2$ is given by:

$$\boldsymbol{\theta}_{t+1} = [\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \sigma^2\mathbf{I}]^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\mathcal{O}\hat{\boldsymbol{\Psi}}_t \tag{3.12}$$

where σ is used to avoiding over-fitting due to the small number of samples. $\boldsymbol{\Phi} = \left[\boldsymbol{\phi}(\boldsymbol{x}_1), ..., \boldsymbol{\phi}(\boldsymbol{x}_N)\right]^{\mathrm{T}}$ and $\mathcal{O}\hat{\boldsymbol{\Psi}}_t$ is $N\times 1$ matrix with elements:

$$\mathcal{O}\hat{\Psi}_t(\boldsymbol{x}_n) \triangleq \hat{\Psi}_t(\boldsymbol{x}_n) + r^{\ a_n}_{s_n s'_n} + \gamma\mathcal{M}_\eta\hat{\Psi}_t(s'_n) - \mathcal{M}_\eta.\hat{\Psi}_t(s_n). \tag{3.13}$$

Kernel Dynamic Policy Programming

The main limitation of DPP in high dimensional systems is the intractable computational complexity as a result of an exponentially growing number of basis functions. Based on DPP, KDPP combines the kernel trick and smooth policy updates to learn tasks represented as high dimensional MDPs with both increased stability and significantly reduced computational complexity [Cui et al., 2017a]. Kernel ridge regression is applied to the least squares solution in Eq. (3.12). The weights vector is represented by dual variables $\boldsymbol{\alpha}_t = [\alpha_t^1, ..., \alpha_t^N]^{\mathrm{T}}$ as:

$$\boldsymbol{\theta}_t = \sum_{i=1}^{N} \alpha_t^i\boldsymbol{\phi}(\boldsymbol{x}_i) = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}_t, \tag{3.14}$$

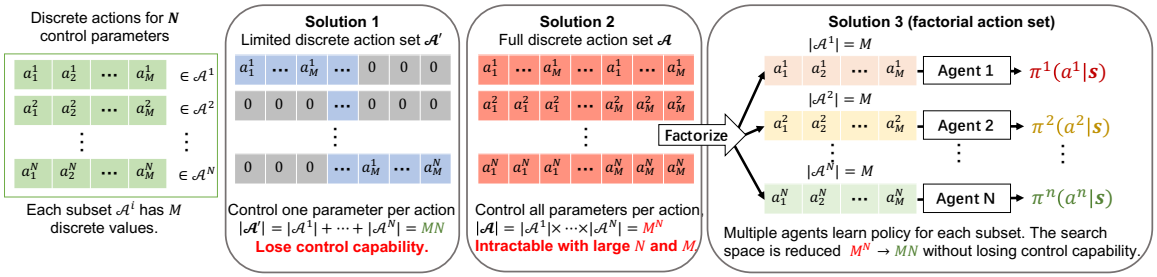


Figure 3.2: Handling huge discrete action set in RL by factorization.

and define the matrix of inner products as $\boldsymbol{K} := \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}$ that $[\boldsymbol{K}]_{ij} = \langle \boldsymbol{\phi}(\boldsymbol{x}_i), \boldsymbol{\phi}(\boldsymbol{x}_j) \rangle =: k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. The approximate action preferences therefore follow:

$$\hat{\Psi}_t(\boldsymbol{x}_n) = \boldsymbol{\phi}(\boldsymbol{x}_n)\boldsymbol{\theta}_t = \sum_{i=1}^{N} k(\boldsymbol{x}_n, \boldsymbol{x}_i)\alpha_t^i. \tag{3.15}$$

After translating Eq. (3.12) using the Woodbury identity:

$$[\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \sigma^2\mathbf{I}]^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\mathcal{O}\hat{\boldsymbol{\Psi}}_t = \boldsymbol{\Phi}^{\mathrm{T}}[\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} + \sigma^2\mathbf{I}]^{-1}\mathcal{O}\hat{\boldsymbol{\Psi}}_t. \tag{3.16}$$

The solution can also be represented by dual variables as:

$$\boldsymbol{\alpha}_{t+1} = [\boldsymbol{K} + \sigma^2\mathbf{I}]^{-1}\mathcal{O}\hat{\boldsymbol{\Psi}}_t \tag{3.17}$$

where $\mathcal{O}\hat{\boldsymbol{\Psi}}_t$ is calculated by plugging Eq. (3.15) into Eq. (3.13). According to [Cui et al., 2017a], a suitable kernel subset from all samples $\mathcal{D}_k = [\tilde{\boldsymbol{x}}_m]_{m=1:M}, M \ll N$ can be built via an online selection in order to reduce the computational complexity and therefore enables the learning in high dimensional state space.

Factorial KDPP

KDPP has shown its efficient learning in the control of a pneumatic muscle-driven robotic hand with a 32 dimensional state space [Cui et al., 2017a], while other conventional methods such as LSPI [Lagoudakis and Parr, 2003] could not. On the other hand, calculating $\mathcal{M}_\eta \Psi_t(s)$ in Eq. 3.11 through a huge discrete action set $\mathcal{A}$ is intractable. For example, define M discrete actions for one control parameter. In order to control N parameters, the entire set of all possible actions

Algorithm 1: Factorial KDPP

Require: number of iteration T, number of action dimensions N.

Initialize kernel subset $\mathcal{D}_n^{Kernel} = , \ n = 1, ..., N$.
for iteration $t = 0, 1, 2, ..., T$ **do**
  if $t == 0$ **then**
    Generate samples in $\mathcal{D}^t$ with random policies π_n^0.
  else
    Generate samples in t-th iteration $\mathcal{D}^t$ by setting π_n^t, $i = 1, ..., N$, as softmax exploration policies.
  end if
  for each dimension of actions $n = 1, 2, ..., N$ **do**
    Select samples from $\mathcal{D}^t$ to build kernel subset for the n-th dimension $\mathcal{D}_n^{Kernel}$.
    Update dual weights vector $\boldsymbol{\alpha}_n$ with samples $\mathcal{D}^i$, $i = 0, 1, ..., t$, and kernel subset $\mathcal{D}_n^{Kernel}$ following KDPP.
  end for
end for

becomes $|\mathcal{A}| = M^N$ (Solution 2 in Fig. 3.2) which is intractable with large M and N. As one solution to maintain a tractable computational complexity, the experiments in [Cui et al., 2017a] carefully coded actions to limit only one parameter's control available in each action (Solution 1 in Fig. 3.2). It therefore reduces the size of action set to MN. However, this trick clearly weakens control capability and is not suitable to tasks requiring the simultaneous control of several units (e.g., chemical plant control). To address this, Factorial Kernel Dynamic Policy Programming (FKDPP) is proposed to learn action space dimension by dimension separately under the KDPP framework following:

$$\pi(a|s) = \prod_{n=1}^{N} \pi^{(n)}(a^{(n)}|s). \tag{3.18}$$

FKDPP divides the discrete action set for N control parameters, $\mathcal{A}$, to N subsets $\mathcal{A}^n$ which only contains M discrete actions and leaves them to N KDPP agents respectively. The policy $\pi(a|s)$ then turns to N policies $\pi^{(n)}(a^{(n)}|s)$ as the Solution 3 in Fig. 3.2. With one agent searching M discrete values in subset, N

agents factorially search all M^N discrete actions in $\mathcal{A}$. It hugely decreases the computational complexity without losing control capability. Inheriting the regularization with the Kullback-Leibler divergence from KDPP, FKDPP features a factor-wise kernel-based smooth policy update that stabilizes the learnings among multiple agents since the over-large update of each agents' policy is avoided every iteration.

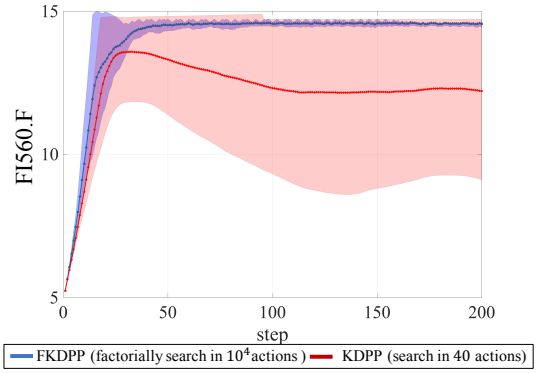
FKDPP adds a loop to learn each subset of actions separately according to Algorithm 1. Each subset is allocated to an agent, which is updated with corresponding samples generated through a soft-max exploration policy:

$$\pi^n_{\text{explore}}(a^n|s) = \frac{\exp(\eta_{\text{explore}}\hat{\Psi}^n_t(s,a^n))}{\sum_{a^{n'}\in\mathcal{A}^n}\exp(\eta_{\text{explore}}\hat{\Psi}^n_t(s,a^{n'}))}. \tag{3.19}$$

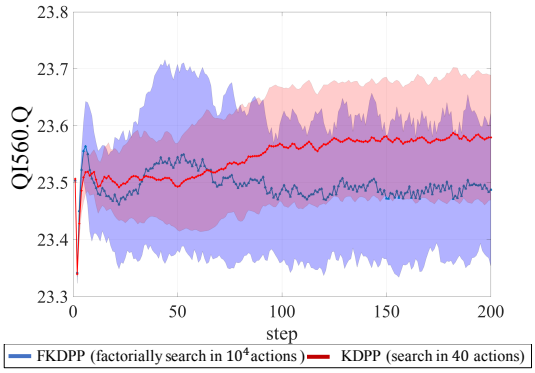
By doing this, tasks with huge discrete action set for multiple control parameters are divided into several sub-tasks with smaller action sets which are tractable for KDPP. Details of KDPP's kernel subset selection and weights update (lines lines 9 and 10 in Algorithm 1) are covered in [Cui et al., 2017a].

3.2.4 Experimental Results

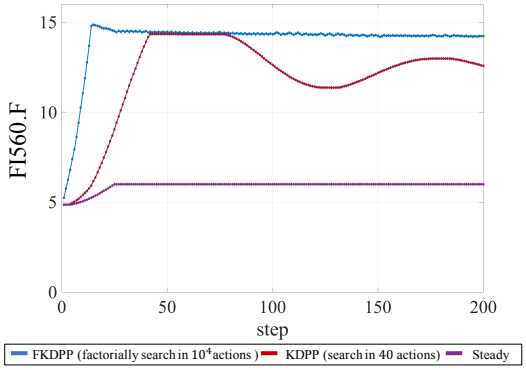
In this section FKDPP and KDPP are applied to the task introduced in Section 3.2.2. According to Table 3.2, the state space has nine dimensions including control state (FI560.F, LI550.L, s407.F, QI553.Q and QI560.Q) and control parameters (FC550.SVM, TC540.SVM, TC501.SVM and PC501.SVM). The discrete action set is defined as increasing/decreasing four control parameters by 10 discrete actions ($M = 10$) following: $a_{\text{FC550}} \in [-2, 2]$, $a_{\text{TC540}} \in [-4, 4]$, $a_{\text{TC501}} \in [-10, 10]$ and $a_{\text{PC540}} \in [-2, 2]$. The total number of actions is close to 10^4, resulting in an intractable computation in Eq. 3.11 for KDPP under Solution 2 in Fig. 3.2. Therefore Solution 1 is used in KDPP to consider only $M \times N = 40$ action combinations. For FKDPP, the action space with 10^4 actions is factorized by four agents. For each agent, there are $M = 10$ actions. The VAM plant simulation used in this experiment is detailed in [Machida et al., 2016], which is implemented on the commercial dynamic simulator, Visual Modeler developed by Omega Simulation Co., Ltd.. Each algorithm is trained over 30 iterations, with each iteration consisting of 200 steps. Each step simulates approx. 30 minutes due to lengthy



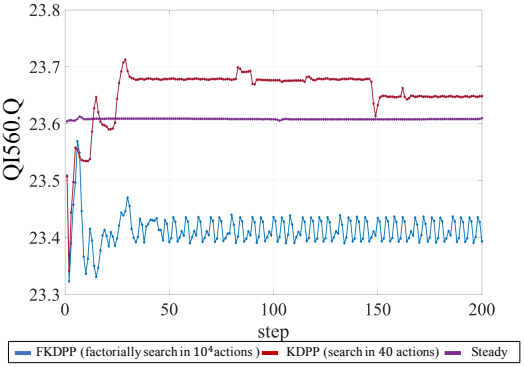
(a) production rate comparison



(b) quality comparison



(c) mean production rate comparison



(d) mean quality comparison

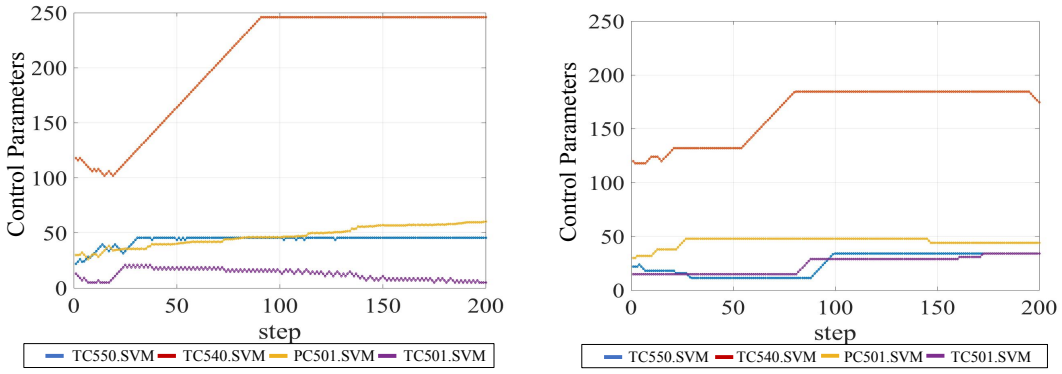
Figure 3.3: Learning results of VAM plant simulation.

chemical processes. The reward function used by both methods is defined as:

$$R = 30 \times \text{FI560.F} - 2 \times \text{LI550.L} - 5 \times \text{QI560.Q} - 20 \times \text{QI553.Q}. \tag{3.20}$$

It follows the strategy of giving a high reward when FI560.F is increased (VAM quantity up) or QI560.Q is decreased (VAM quality up). If any critical condition in Table 3.2 is violated, the reward is sharply decreased.

Figures 3.3(a) and 3.3(b) show the mean and variance of VAM yield (FI560.F) and quality (QI560.Q) for each 200 step period during 30 iterations of learning. FKDPP quickly converged to a good policy which maximized production (Fig.



(a) FKDPP action trajectories

(b) KDPP action trajectories

3.3(a)), while KDPP resulted with a lower mean and larger variance due to system instability (e.g., depleting the tank or causing other catastrophic failure scenarios). FKDPP also has a lower mean of QI560.Q during learning compared with KDPP, as shown in Fig. 3.3(b).

After 30 learning iterations, the policies of FKDPP and KDPP are tested. According to Figs. 3.3(c) and 3.3(d), FKDPP outperformed KDPP in both VAM quantity and quality. The sequences of control parameters are shown in Figs. 3.4(a) and 3.4(b), where FKDPP facilitates exploration with multiple control parameters at each step. On the other hand, KDPP only operates a single control parameter at each step which is insufficient in terms of exploration, especially when each step lasts for 30 minutes. In Fig. 3.4(b), KDPP has to use more iterations to operate each dimension of the action, resulting in worse performance. These results show that under identical learning conditions, the proposed FKDPP achieves superior performance compared with methods in which the policy is not factorized.

In terms of computational complexity for this task with its nine dimensional state and action set for four control parameters, FKDPP took an average $0.0017s$ per step to search over 10^4 actions while KDPP took $0.0014s$ to consider 10×4 actions. Because FKDPP reduces the computational complexity by limiting the number of action combinations (Eq. 3.11), computation time for FKDPP with 10 actions each is negligibly slower in comparison to running a single agent with 40 actions in the case of KDPP.

3.3 Plant-wide Control

In the industrial process control community, while there was early attempts on leveraging RL, few studies exist on solving large-scale chemical plant control problems. Section 3.2 can also be seen as part of the literature that focuses only *local* control problems without touching upon the more important *plant-wide* control. The main reasons are intractable computation and memory cost incurred by an extremely large number of samples, subsequent derivation for feasible policies, and the poor scalability of traditional RL algorithms, e.g., Q-learning. Such algorithms easily incur the curse of dimensionality [Bellman, 2003] which refers to the explosion of the number of samples needed as the dimensionality increases.

This dilemma can be seen from looking back into the literature: Syafiie et al. [Syafiie et al., 2007] leveraged Q-learning in the classical pH neutralization control task on a laboratory plant. In other process control problems aside from chemical plants, Ernest et al. [Liu et al., 2010] used the enhanced Q-learning algorithm to derive a policy that achieves better power-performance tradeoff on both synthetic and small-sized real workloads. Harp et al. [Harp et al., 2000] embedded RL algorithm in a simulator for electricity management to learn a profitable electricity pricing policy. While these methods achieve successes in low dimensional simulations or small-scaled real-world experiments, Q-learning [Watkins and Dayan, 1992] has been well-known that it does not scale well into high dimensional problems, and is typically expensive in terms of both learning time and computational resources for high dimensional systems. Deep RL (DRL) techniques on process control [Kubosawa et al., 2018, Spielberg et al., 2017] such as Deep Q-Network (DQN), usually have sample complexity several magnitudes larger than traditional RL. This prohibitive sample complexity excludes them from consideration for plant-wide process control and may be the reason that Kubosawa et al. [Kubosawa et al., 2018] has conducted the experiment only with some isolated components of the VAM manufacturing process from malfunctioning, without considering the full context of VAM process control problems, e.g., improving the production rate or product quality.

In this section, we focus on plant-wide control of the VAM process. Since the problem is intractable for conventional algorithms, including FKDPP proposed

in the next section, we propose a more scalable algorithm: Factorial Fast-food Dynamic Policy Programming (FFDPP), for realizing autonomous control on large-scale chemical plants. Similar with FKDPP, FFDPP is also based on Dynamic Policy Programming [Azar et al., 2012] which considers KL regularization [Cui et al., 2017a,b, Tsurumine et al., 2019] and factorial policies [Cui et al., 2018, Matsubara et al., 2014]. However, a crucial difference lies in that we exploit a more advanced kernel function approximation scheme known as Fast-food approximation [Le et al., 2013]. Factorial policy representation is coupled with factor-wise policy update to further reduce computational cost. Intuitively, Fast-food kernel approximation samples in the frequency domain to approximate feature maps [Le et al., 2013] instead of sampling directly in the original space to alleviate the curse of dimensionality.

3.3.1 Fast-food Approximation

Before introducing our solution to the plant-wide control problem, we need to recap on the concept of Fast-food, which was proposed by Le et al. [2013] to approximate kernel expansion with only $\mathcal{O}(n \log d)$ time and $\mathcal{O}(n)$ storage, where n denotes the number of samples, d denotes the dimensionality. In our case, the problem is n continues to grow, which renders computing the kernel matrix in FKDPP intractable since inverting the kernel matrix takes $\mathcal{O}(n^3)$ computations. We aim to approximate the kernel matrix $\boldsymbol{K}$ such that its inversion can be computed in a straightforward manner. Let us first introduce some basics associated with kernels.

Mercer's Theorem *Let $\mathcal{X}$ be the input space. Any kernel k: $\mathcal{X} \times \mathcal{X} \to \mathbb{R}$ satisfying $\int k(x, x^{'}) f(x) f(x^{'}) dx dx^{'} \geq 0$ for all measurable functions f can be expanded into the following summation:*

$$k(x, x^{'}) = \sum_{j} \lambda_j \phi_j(x) \phi_j(x^{'}) \tag{3.21}$$

where $\lambda_j > 0$ and ϕ_j are orthonormal.

Mercer's theorem states that any positive definite kernel can be computed as a summation of weighted inner products of feature vectors. This in turn implies that lifted datapoints can be computed as $K(x, y) = k(x, y) = \langle \phi_j(x), \phi_j(y) \rangle$,

where $x, y \in \mathbb{R}^d$ are the input datapoints, $\phi_j : \mathbb{R}^d \to \mathbb{R}^q$ defines the lifting, where $q \gg d$.

Rahimi proposed that, instead of relying on the implicit lifting defined by the kernel trick, it might be better to explicitly map the data to a low-dimensional Euclidean distance space using a randomized feature map $z : \mathbb{R}^d \to \mathbb{R}^D$, such that the inner product between a pair of lifted datapoints approximates their kernel counterpart:

$$k(x, y) = \langle \phi(x), \phi(y) \rangle \approx z(x)^T z(y) \tag{3.22}$$

where unlike the high dimensional mapping ϕ, z is low-dimensional. The low-dimensionality of z provides us with the benefit that can be used directly with the transformed input to approximate the desired nonlinear kernel machine. Specifically, to obtain the desired kernel, an important theorem from harmonic analysis called *Bochner's theorem* is used, which further complements the Mercer theorem in the sense that a kernel is valid only if its Fourier transform is a valid probability measure, i.e., a proper probability distribution.

Bochner's Theorem *A continuous kernel $k(x, y) = k(x - y)$ is positive definite if and only if $k(\delta)$ is the Fourier transform of a non-negative measure, i.e.,*

$$k(x - y) = \int p(w) e^{jw^T(x-y)} dw = \mathbb{E}_w \left[\varphi_w(x) \varphi_w(y) \right] \tag{3.23}$$

where $\varphi_w(x) = e^{jw^T x}$. This states that, if w is drawn from the distribution $p(w)$, then we have an unbiased estimate of original kernel $k(x, y) = k(x - y)$. Note that by writing so we mean that the kernel should be shift-invariant, as is the case of common kernel functions including RBF, Cauchy and Laplacian, etc.

Two more steps to obtain the desired kernel function. The first is that, RBF and the corresponding Gaussian distribution are real, hence the complex components in Eq.(3.23) can be replaced by cosine function $z_w(x) = \sqrt{2} \cos(w^T x + b)$ and $p(w)$ set to be Gaussian. Second, to manipulate the *width* σ^2 of the desired feature map, one simply samples from the Gaussian with zero mean and variance σ^2. The approach of RKS is summarized below:

input Width σ^2, n, d

Sample entries of $Z \in \mathbb{R}^{n \times d}$ i.i.d. from $\mathcal{N}(0, \sigma^{-2})$

for all x,

Compute the feature map using Eq.(3.23)

$\phi_j = \frac{1}{\sqrt{n}} \exp\left(i[Zx]_j\right)$

end for

The RKS approximation is a probabilistic approximation that converges with high probability and at the rate of independent empirical averages [Rahimi and Recht, 2008]. However, one shortcoming is that one needs to compute Zx for every observation, leading to $O(nd)$ operations and storage for each input x. In the application considered in this thesis, input comes as a stream of observations resulted from interacting with the environment. This stream of observations can be arbitrarily many if one wants to collect as many samples as possible. Hence it is desirable to further reduce the cost of computation and storage.

Le et al. proposed Fast-food approximation [Le et al., 2013] to improve the computation time of RKS from $O(nd)$ to $O(n \log d)$ and storage from $O(nd)$ to $O(n)$. The improvement is crucial for applications with very big number of features, e.g., image processing. Specifically, fast-food achieves the improvement by considering a matrix formed from multiplications of several diagonal simple matrices instead of Z. The fast-food matrix is denoted as V in the following equation:

$$V = \frac{1}{\sigma\sqrt{d}} SHG\Pi HB \tag{3.24}$$

where S, G, B are diagonal matrices. More specifically, $G_{ii} \sim \mathcal{N}(0,1), B_{ii} \in \{\pm 1\}$, the scaling matrix S can be adjusted to produce different kinds of kernels desired. For the RBF case, it follows Chi-distribution, namely $S_{ii} \sim r^{d-1} e^{-\frac{r^2}{2}}$. $\Pi \in \{0,1\}$ is a permutation matrix, H is the Walsh-Hadamard matrix of the form:

$$H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad \text{and} \quad H_{2d} = \begin{bmatrix} H_d & H_d \\ H_d & -H_d \end{bmatrix}$$

Matrices S, G, B are once computed and stored. The Walsh-Hadamard matrix H can be obtained via the fast Walsh-Hadamard transform. The detail of gener-

ating fast-food matrix is provided below, in which the important properties are explained.

Every row of V is a Gaussian variable

This statement consists of two facts:

(1) The rows of matrix V excluding S are i.i.d. Gaussian variables.

(2) Based on (1), multiplying with the scaling matrix S, rows of the fast-food matrix S are Gaussian, but no longer independent.

This first fact is because every entry of V excluding S takes the form of $[HG\Pi HB]_{ij} = B_{jj}H_i^T G\Pi H_j$, which is a sum of zero-mean independent Gaussian variables. According to the fact that adding d independent variables each of $\mathcal{N}(0,1)$ yields a Gaussian variable of $\mathcal{N}(0,d)$, the variance of each entry is d. The binary matrix B ensures that different entries in a row $[HG\Pi HB]_i$ have zero correlation, hence guaranteeing that they are i.i.d Gaussian variables.

The second fact is because of that, scaling rows of $HG\Pi HB$ into a unit-ball using entries of $S_{ii} = s_i||G||_{Frob}^{-\frac{1}{2}}$, $s_i \sim (2\pi)^{-\frac{d}{2}} A_{d-1}^{-1} r^{d-1} e^{-\frac{r^2}{2}}$ serves the purpose of making Gaussian random vectors distributed uniformly on the unit-ball [Blum et al., 2020]. It provides information on sampling, since knowing the radius of the ball one knows that which regions are more likely to be concentrated with distributed points, hence the independence no longer holds.

Rows of $HG\Pi HB$ have the same length

This is because the length can be written as $l^2 = [HG\Pi HB(HG\Pi HB)^T]_{ii} = [HG^2H]_j d = \sum_i H_{ij}^2 G_{ii}^2 d = ||G||_{Frob}^2 d$.

Fast-food preserves the advantages of RKS, while improves the scalability in the sense that instead of storing Z computing Zx with complexity $O(nd)$ for every observation encountered, now the complexity of computing Vx drops to $O(n \log d)$ and storing V drops to $O(n)$. This is because S, G, B are diagonal matrices that cost only $3n$ storage. The Hadamard matrix H is implicitly represented using the recursive formula. These two facts show that storage of V is $O(n)$. To demonstrate that the computation costs $O(n \log d)$, one notes that the computation relies on the Hadamard transform which costs $O(n \log d)$ and evaluation of n basis functions which costs $O(n)$, hence the cost in total is $O(n \log d)$. We produce fast-food feature maps using the follow procedure:

Algorithm 2: Computing Fast-food feature maps

Require: σ, n, d

Ensure: Feature map ϕ_j

compute Fast-food matrix V using Eq.(3.24)

for all x **do**

Compute the feature map by $\phi_j = \frac{1}{\sqrt{n}} \exp\left(i[Vx]_j\right)$

end for

In Section 3.2 Factorial Kernel Dynamic Policy Programming (FKDPP) was designed by selecting a subset of most informative samples and then compute the approximated kernel matrices from the subset and whole samples. However, when the dimensionality increases, the size of the subset also increases drastically, quickly rendering computation intractable. On the other hand, Fast-food solves this problem by tranferring the data into the frequency domain and then conducts sampling. It is mathematically demonstrated that the generated feature map $\hat{\Phi}$ recovers the exact Φ with exponentially decreasing error with increasing sampling frequency [Rahimi and Recht, 2008]. This is especially suitable for computing in high dimensional problems. By leveraging Fast-food, the DPP-based value function and policy are now expressed as:

$$\hat{V}^t_{\bar{\pi}}(s) = \frac{1}{\eta} \log \sum_{a \in \mathcal{A}} \exp\left(\eta \hat{\Psi}_t(s,a)\right), \qquad \hat{\pi}^t(a|s) = \frac{\exp\left(\eta \hat{\Psi}_t(s,a)\right)}{\sum_{a' \in \mathcal{A}} \exp\left(\eta \hat{\Psi}_t(s,a')\right)} \tag{3.25}$$

where $\hat{\Psi}_t = \hat{\Phi}\theta, \hat{\Psi}_t = [\phi_1, \ldots, \phi_j, \ldots]$ is now computed by random feature maps following Alg. 2. Since we have factorial policies, we maintain several weight vectors $\theta^{(m)}$ for each $\hat{\Psi}^{(m)}_t$ and hence $\pi^{(m)}$, where the superscript m denotes the m-th factorized policy. We list the pseudo-code of FFDPP in Alg. 3.

3.3.2 Experimental Settings

To prove the efficacy of the proposed algorithm, ten independent experiments are repeated and averaged to collect statistical evidence. Each experiment consists

Algorithm 3: Factorial Fast-food Dynamic Policy Programming

Input: $M, \tau, \gamma, \eta, \sigma, T, I$
Output: weight vectors $\theta^{(m)}, m = 1, \ldots, M$

Divide $\mathcal{A}$ into $\{\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(\mathcal{M})}\}$
Random exploration with $\pi_{random} = [\pi^{(1)}_{random}, \ldots, \pi^{(M)}_{random}]^T$
Compute Fast-food matrix V using τ
for $i = 1, \ldots, I$ **do**
  for $t = 1, \ldots, T$ **do**
    measure state s_t
    for $m = 1, \ldots, M$ **do**
      compute $\phi^{(m)}(x)$ using Fast-food
      compute $\hat{\Psi}^{(m)}_t$ following (3.8)
      sample action $a^{(m)}_t$ from $\pi^{(m)}$
      $x^{(m)} = [s_t, a^{(m)}_t]^T$
      collect $x^{(m)}$ in $\mathcal{D}^{(m)}_i$
    end
    apply action $a_t = [a^{(1)}_t, \ldots, a^{(M)}_t]$
  end
  for $m = 1, \ldots, M$ **do**
    compute $\hat{\Phi}^{(m)}_t$ using $\mathcal{D}^{(m)}_i$
    $\hat{\theta}^{(m)}_{t+1} = [\hat{\Phi_t}^{(m)^T} \hat{\Phi_t}^{(m)} + \sigma^2 I]^{-1} \hat{\Phi_t}^{(m)^T} \mathcal{O} \hat{\Psi}^{(m)}_t$
  end
end

Observation	Description	Control criteria
Production rate sensor	Flow rate of VAM	To be optimized as VAM yield, part of the $profit$
Level sensor	Level(%) of the decanter	$< 100\%$
Quality sensor 1	Acetic acid density	< 100 ppm
Quality sensor 2	VAM density (inversely proportional to the quality)	< 150 ppm
Temperature sensor 1 - 8	Temperature inside the distillation column	To be optimized as cost, part of the profit
Gross profit	Profit of VAM product (JPY/h)	to be optimized as gross profit (part of the profit)

Table 3.4: Observation units of the investigated task.

of 70 iterations and each iteration has 1000 steps. Each time step in simulation corresponds to around five minutes in the real time. Hence 1000 steps is approximately two days of real time plant running. The performance of the proposed

Control unit	Description	Effect
Flow controller 1	Control superheated stream flow rate for the reboiler	Column temperature rises/declines Affect VAM yield and quality Steam amount increases/decreases Steam generation is part of cost
Pressure controller 1	Maintain top pressure of the column with N_2 and gas purge	Acetic acid density rises/declines N_2 density rises/declines
Pressure controller 2	Maintain pressure of the 3rd stage of the distillation column with superheated steam	Column temperature rises/declines Affect VAM yield and quality Steam amount increase/decreases Steam generation is part of cost
Flow controller 2	Control decanter feed flow temperature with cooling water	Control Decanter feed flow Affect stability of the process
Temperature controller	Maintain temperature profile and product quality by controlling the reflux flow rate	Affect VAM yield and quality Affect stability of the process Control reverse flow rate

Table 3.5: Control units of the investigated task.

algorithm is evaluated in terms of plant-wide stability, profit and computational resources needed in the period of simulated two days.

For each of the M control units, corresponding action set is constructed by uniformly choosing N values from the interval $[-0.01, 0.01]$. The number of N is chosen empirically to achieve balance between high resolution of discrete actions and computational tractability. Here $N = 10$ to realize high resolution. Note that without the factorial policy setup, the size of the entire action set to be considered is $M^N = 5^{10} \gg 2^{20}$, which is intractable for common value function based RL approaches to efficiently explore.

Aside from maximizing accumulated profit as defined in Eq. (3.20), maintaining process stability is also a learning goal. Stability is defined by violations of the safety ranges in Table 3.5. When a violation occurs, a corresponding penalty is added to the reward. The experiment is conducted on a PC with processor i7-8700k, 3.70GHz and 32GB memory.

The meta parameters of the experiment are summarized in Table 4.1. Parameter τ controls the balance of Fast-food approximation accuracy and learning speed and needs to be specified by the user. Here $\tau = 100$ and interested readers are referred to [Rahimi and Recht, 2008] for technical details.

3.3.3 Experiment results

Performance of FFDPP

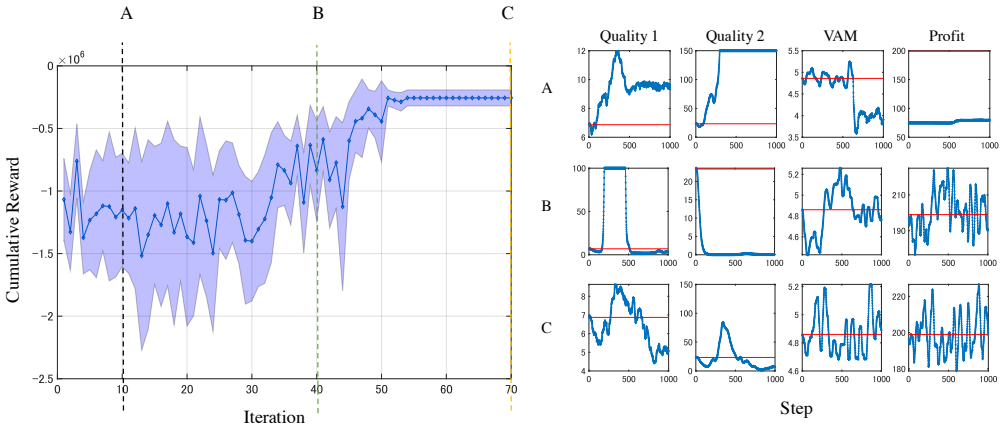


Figure 3.4: Black dashed line marked by A show the performance at 10th iteration; green dashed line B the performance at 40th iteration; yellow line C the performance at 70th iteration. The y-axis for righthand subplots are: Quality 1-(ppm), Quality 2-(ppm), VAM (Production rate)-(t/h), Profit (Gross profit)-(10^3 yen/h).

The learning performance evolving is shown in the left part of Figure 3.4, averaged over ten independent learning results for statistical evidence. The blue curve shows the mean reward, while the transparent blue area depicts variance. The y-axis is the accumulated reward given in Eq. (3.20) summed over 1000 steps of every iteration. On the right are subplots illustrating the performance evolving through iteration: after 10, 40 and 70 iterations. It can be seen from the figure that after trial-and-error learning and without any prior knowledge about the model, the learned FFDPP policy successfully learns a control policy to maximize the reward, solving the process control problem of 13-dimension state space and 5-dimension action space in 70 iterations within 70,000 samples.

To visualize the correspondence between states and learning progress, t-SNE [van der Maaten, 2014] is applied to compress the 13 dimensional state vectors into two dimensions to draw the scatter plot in Figure 3.5. By trial-and-error

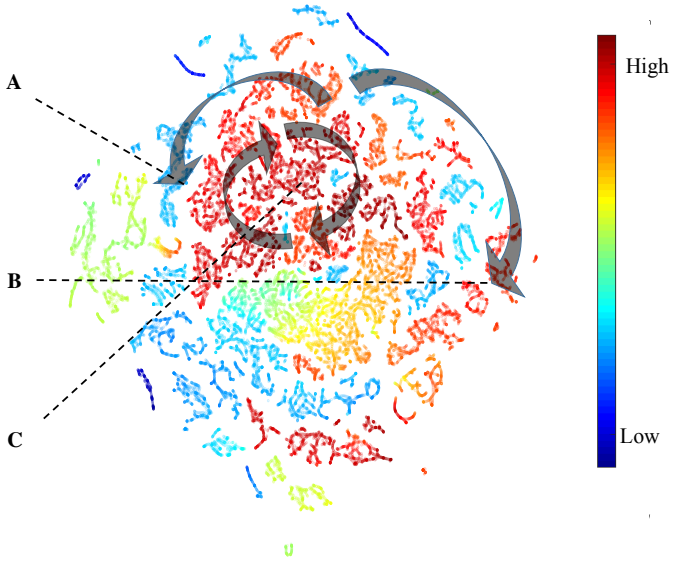
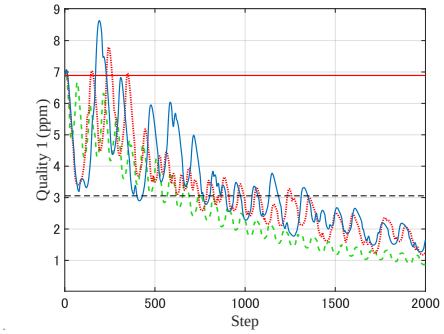


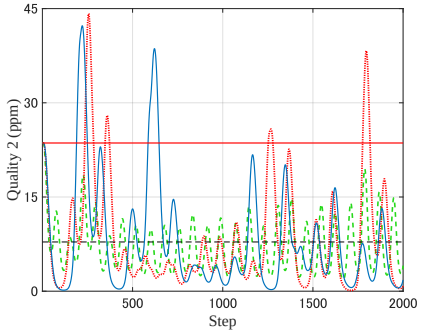
Figure 3.5: The t-SNE plot shows a trajectory of the agent in one experiment. Undesirable states, i.e., overflow of Quality sensor 1 or 2, low production rate, etc., are marked with blue, desirable states are marked in red. Markers A,B,C refer to the performance plots in Figure 3.4.

learning the agent explores state space and finds a path that cycles around the optimal region indicated by deep red. Intermediate iterations might take the agent through some desirable states, but cannot control it to be within the optimal regions. Likewise, optimal states might be reached by some policies, but undesirable states will be experienced along the paths. Optimal policy is derived as circling regularly within the optimal region.

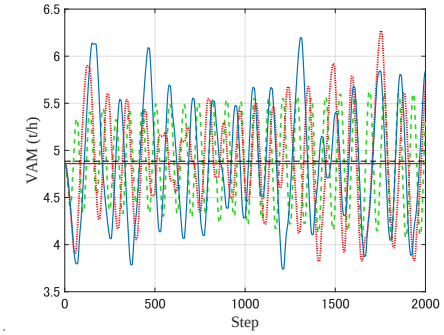
Performance of learned policies is further examined in Figure 3.6. To testify to the robustness and stability of FFDPP, independent experiments are performed. The rollout is conducted with length of 2000 steps, 100% more than the length of



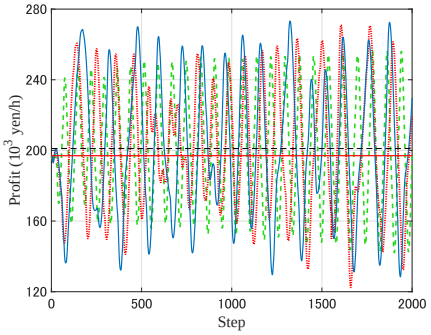
(a) Readings of the Quality sensor 1 under the three learned policies. Reading is inversely proportional to product quality.



(b) Readings of the Quality sensor 2 under the three learned policies. Reading is inversely proportional to product quality.



(c) VAM production rate under the three learned policies.



(d) Profit per time unit under the three learned policies.

Figure 3.6: Learned results of FFDPP on the plant-wide control task.

learning phase. Comparison is made between performance of the learned policies and of the model-based controller proposed in [Machida et al., 2016, Seki et al., 2010]. Policies are shown here in blue solid line, red dot line and green dashed line. The red solid straight lines indicate performance under the model-based controller which have minor oscillation, while the three curves show adaptations made by the learned policies. In each subfigure, the black dashed line indicates the mean value of the three curves averaged over 2000 steps. Figure 3.6(a) and 3.6(b) show the readings of two product quality measurers. Their readings are inversely proportional to the product quality. The product quality controlled by the learned policy is better on average than that of the model-based controller in

the simulated period. Figures 3.6(c) and 3.6(d) show the VAM production rate and profit per time unit, respectively. As the result of adapting to the process dynamics, the curves are changing, hence mean values are needed to be compared with performance of the model-based controller in terms of VAM produced and profit made. It is clear that without any model knowledge, in the simulated period corresponding to around 5 real days, the FFDPP agent succesfully learns a policy that yields comparative performance to the state-of-the-art model-based controller.

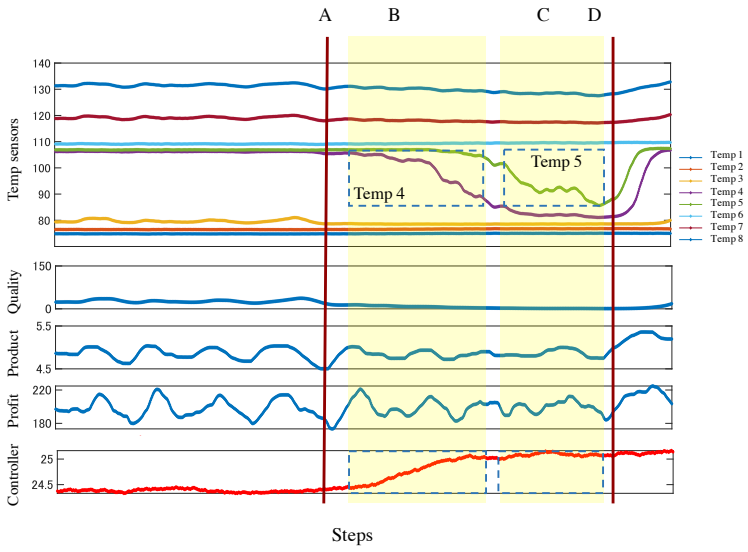


Figure 3.7: The analysis of controlled behavior using the learned FFDPP policy. The agent changed behavior from A, followed by B and C. After time step D the agent managed to restore the normal temperature of the distillation column.

Comparison to Other RL Methods

The computational mitigation brought by the factorial framework and Fast-food kernel approximation is examined in Table 3.7 to serve as the ablation test, which compares the computational resources that DPP-based algorithms require. It is worth mentioning that in the original DPP paper [Azar et al., 2012], DPP has been compared with the classic model-free algorithm Q-learning and demon-

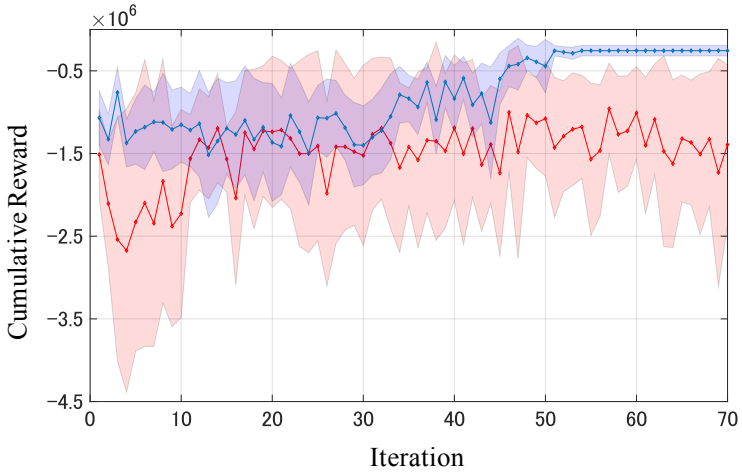


Figure 3.8: Comparison between unfactorized FDPP and FFDPP. FFDPP performance is marked in blue, unfactorized FDPP is marked in red.

strated superior performance. Hence in this comparison only DPP-based algorithms are considered. The configurations for KDPP and FKDPP respectively follow the definition in [Cui et al., 2017a] and [Cui et al., 2018], with the exception the sample threshold is set to $\eta = 0.9$, which is equivalent to discarding 90% of samples and retaining the rest 10% most informative ones. For ablation test, unfactorized Fast-food DPP is also evaluated (see Section 3.3.3 for details). All parameters of the compared algorithms were tuned to yield best performance empirically.

In the described experiment, *Success* is defined as after 70 iterations' learning, the learned policy does not violate any constraint and perform comparable to the model-based controller (with cumulative reward approximately equal or better than 1.97×10^5 units). An indicator *Error* is also defined as the situation where the computing software reports error due to intractable computational time or running out of memory resulted from the large amount of high dimensional samples. The second column of Table 3.7 shows the maximum number of samples that each algorithm can handle before triggering the *Error* indicator. The computation time, policy evaluation time are evaluated with the maximum acceptable number of samples. For factorial algorithms, time costs are recorded

Table 3.6: Meta parameters of the FFDPP process control experiment.

Parameter	Description (Number of)	Value
T	steps	1000
I	iterations	70
M	factorial agents	5
N	discrete actions per agent	10
τ	sampling basis functions	100

for one agent, hence the total time cost is that cost multiply by the number of agents.

It is worth noting that the large acceptable number of samples of FFDPP not only relies on Fast-food, but also on the factorial policy for decomposing the one-time evaluation of huge number of action preferences to several sequential evaluations, as will be verified in the next section.

Algorithm	Acceptable number of samples	Maximum computation time (s)	Maximum policy evaluation time (s)	Success	Error
DPP	5000	552.5	0.0152	No	Yes
KDPP ($\eta = 0.9$)	20000	586.8	0.0015	No	Yes
FKDPP ($\eta = 0.9$)	30000	178.5×5	0.0014×5	No	Yes
FDPP (Unfactorized)	>70000	285.9	0.1405	No	No
FFDPP	>70000	31.9×5	0.0044×5	Yes	No

Table 3.7: Performance comparison between DPP, KDPP, FKDPP, Fast-food DPP and FFDPP on the investigated task. Acceptable number of samples indicates maximum capacity of each algorithm before triggering the *Error* indicator. Maximum computation time and policy evaluation time are evaluated for the maximum acceptable number of samples.

Unfactorized Fast-food DPP

In this section the efficacy of factorial policy is empirically verified. Unfactorized Fast-food DPP (UFDPP) is run on the same task to give empirical evidence. As

shown in Figure 3.8, the failure of UFDPP could be due to ineffective exploration, insufficient samples or large number of action preferences to learn. The learning curve is averaged over ten independent experiments.

To conclude, RL on plant-wide process control dictates a large amount of samples for building robust controller. The bottleneck of KDPP and FKDPP lies in their failure to process such huge number of samples. On the other hand, the ability to efficiently search in high dimensional actions space is also indispensable even equipped with the capability of processing that large number of samples. In this experiment, Fast-food approximation is exploited to tackle the first problem, factorial policy is leveraged to handle the latter problem.

3.4 Discussion and Conclusions

In this chapter we investigated how KL divergence as a regularization could help greatly improve the performance as well as sample efficiency, even with simple linear function approximation. Intuitively, KL divergence between two consecutive policies helped reducing the aggressiveness of updates which often resulted in brittle value functions especially in high dimensional spaces. It is worth mentioning that our use of KL regularization was in the form of DPP, which is a value iteration method rather than the conventional policy iteration style. This special formulation also contributed greatly to the sample efficiency of the proposed method, since the policy loop was removed from the algorithm.

It is worth noting that the proposed method in this chapter concerned only how to efficiently realize autonomous control by using RL, without considering realistic factors such as safety. Real-world control problems often necessitate safe control which refers to minimized risk or constraint violation during learning, which will be the topic of the next chapter. Another important research direction is to apply the proposed method to some small-scale real-world plant to demonstrate the effectiveness of FFDPP continue to hold.

4 | Safety

4.1 Introduction

While RL has achieved tremendous success in games [Silver et al., 2017] or robot manipulation [Andrychowicz et al., 2020, Levine et al., 2018], unlike supervised learning which has already been massively applied to real-world applications, RL deployments remain largely in game playing and laboratories. One major reason is that there lacks a constraint satisfying mechanism that guarantees constraint satisfaction during RL agent's trial-and-error exploration, where constraint is used to model scenarios that dictates safety. Safe reinforcement learning algorithms that can simultaneously minimize the total cost and the risk of constraint violation are crucial. The paradox of ensuring good learning performance and avoiding constraint violation of safe RL can be likened to that of exploration-exploitation tradeoff. An RL agent obtain knowledge from environment by experiencing both constraint satisfaction and violation, nonetheless in real-world applications it is desired to minimize or even completely eliminate constraint violations.

In this chapter, we aim to propose a scalable and efficient safe reinforcement learning algorithm applicable to high dimensional systems such as robots. The safe RL algorithm should be capable of simultaneously satisfying constraint while minimizing cost leveraging limited number of samples. Existing methods, however, has more or less failed in doing so. Lyapunov-based approaches, originally proposed by [Perkins and Barto, 2002], has been gaining interests recently as the Lyapunov property is suitable for modeling safe learning problem: starting from a point inside the attraction region, the agent is forced to stay in the region during the learning course [Berkenkamp et al., 2017, Chow et al., 2018]. This concept

is also known as the Lyapunov stability and is naturally suitable for safe RL. However, despite the theoretical attractiveness, their applications are limited to low dimensional systems that are hard to transfer to robot control since it is still unclear how to efficiently find a Lyapunov function in a systematic manner. More seriously, it is shown in [Banijamali et al., 2019, Moldovan and Abbeel, 2012] that optimizing over a restricted class of policies is in general an NP-hard problem.

Another candidate of safe RL is to model the safety via a natural extension of MDP maintaining a 'budget' or 'constraint' term, which results in Constrained Markov Decision Processes (CMDPs) [Altman, 1999]. Geibel et al. adopt CMDP to compute optimal deterministic policy [Geibel and Wysotzki, 2005]. Chow et al. [Chow et al., 2018] propose to solve CMDP using Lyapunov functions constructed by adding an auxiliary constraint to the original value function. However, though with a known model CMDPs can be solved using linear programming methods, extending to situations of unknown model or large state and action spaces is non-trivial. Reducing constraint violation is easier by constrast, as keeping a memory for states of constraint violation is one popular method [Garcia and Fernandez, 2012, Lipton et al., 2016]. Nonetheless, it brings memory storage burden which might be a heavy burden if the state and action spaces are large and continuous.

One of the most scalable memory-free methods is to maintain an additional *advisor* policy aside from the decision maker or *actor*. The advisor learns from experiences to prevent actor from causing constraint violation. In [Eysenbach et al., 2018], forward and backward policies are trained as a special case of actor-advisor setting to reduce the number of hard reset, an extreme form of constraint violation in robot experiments. The advisor requires training an ensemble of advice functions Q_{advice}, whose estimation in high dimensional spaces may be difficult and limited to the number of samples. Advisor 'advises' actor or triggers the reset when $Q_{advice} > Q_{decision}$, which may result in very slow learning as opposed to the sample efficiency of using KL divergence in high dimensional systems [Cui et al., 2017a, Tsurumine et al., 2019]. Moreover, since the above scheme trains actor and advisor *independently*, which renders actor and advisor focus on different regions of the state space and hence requires extra samples for individual training.

Motivated by the above-mentioned observation, we propose to address the

safe learning problem by *intertwining* actor and advisor in policy updates and introduce it into the KL-regularized RL, specifically, Dynamic Policy Programming (DPP) framework that we extensively leveraged in Chapter 3. However, unlike in Chapter 3 (and many RL algorithms), our proposed method leverage one set of state-action samples and *two set of rewards* for efficiently controlling the agent to safely learn: one as the conventional task reward the agent attempts to maximize, the other one as the safety indicator suggesting which part of the state-action spaces is dangerous. Dynamic Actor-Advisor Programming (DAAP) intertwiningly trains two policies respectively for optimizing the cost and minimizing constraint violation leveraging smoothness of policy update. The intertwining update scheme uses Kullback-Leibler (KL) divergence update to smoothly shift the policy to consider more about cost reduction while preserving constraint satisfaction. DAAP inherits the scalability from well-established DPP-based algorithms as [Cui et al., 2017a, 2018, Tsurumine et al., 2019]. To produce safer exploration during learning, the actor and advisor are mixed using the idea of conservative policy update proposed by [Kakade and Langford, 2002]. Constraint satisfaction, scalability and sample-efficiency of DAAP are demonstrated through its applications to simulated arm control tasks with performance comparisons to baselines.

4.2 Dynamic Actor-Advisor Programming

Since we have introduced the general formulation of DPP in Chapter 3, we skip it here. We address safe RL by introducing an additional constraint function into the DPP framework. Script d is reserved for constraint policy (advisor) and value function, as c denotes the counterparts for cost (actor). For simplicity, we consider the problem of $\{0, 1\}$ constraint violation (CV) coding. Another cost r_d is associated with each state-action pair such that $r_d : \mathcal{S} \times \mathcal{A} \to \{0, 1\}$. For every state-action pair there is only one fixed constraint. Now every state-action pair has two fixed 'costs': r_c and r_d. The expected reward in Eq. (3.2) now is computed with respect to both the probability of yielding a sequence of r_c and a sequence of r_d.

One might propose to use DPP by simply mixing r_c and r_d for the DPP update

rule Eq. (3.11) such that:

$$V^*_{\bar{\pi}}(s) = \min_{\pi} \sum_{\substack{a\in\mathcal{A}\\ s'\in\mathcal{S}}} \pi(a|s)\left[\mathcal{T}^a_{ss'}\left((r^a_{ss'})_c + (r^a_{ss'})_d + \gamma V^*_{\bar{\pi}}(s')\right) - \frac{1}{\eta}\log\left(\frac{\pi(a|s)}{\bar{\pi}(a|s)}\right)\right]. \tag{4.1}$$

This style incurs loss on individual information carried by r_c and r_d and might induce poor balance between cost and constraint.

Motivated by the need to separately consider cost and constraint yet to closely relate them to achieve sample-efficiency, our proposed method intertwines the actor and advisor to learn a near-optimal balance between safer exploration and better goal reaching.

By substituting the advisor and actor counterparts to the baseline policy in Eq. (3.10) and (3.11), we propose to intertwiningly update two policies in the following fashion:

$$V^c_{\bar{\pi}^d}(s) = \sum_{\substack{a\in\mathcal{A}\\ s'\in\mathcal{S}}} \pi^c(a|s)\left[\mathcal{T}^a_{ss'}\left(\left(r^a_{ss'}\right)_c + \gamma V^c_{\bar{\pi}^d}(s')\right) - \frac{1}{\eta_c}\log\left(\frac{\pi^c(a|s)}{\bar{\pi}^d(a|s)}\right)\right], \tag{4.2}$$

$$V^d_{\bar{\pi}^c}(s) = \sum_{\substack{a\in\mathcal{A}\\ s'\in\mathcal{S}}} \pi^d(a|s)\left[\mathcal{T}^a_{ss'}\left(\left(r^a_{ss'}\right)_d + \gamma V^d_{\bar{\pi}^c}(s')\right) - \frac{1}{\eta_d}\log\left(\frac{\pi^d(a|s)}{\bar{\pi}^c(a|s)}\right)\right], \tag{4.3}$$

in which we alternately use actor preference as the baseline for next step advisor preference update and vice versa. η_c, η_d denotes KL regularization coefficients for the actor and advisor, respectively. In practice it is desirable to have η_c, η_d at different time scales to promote learning depending whichever of reward maximization and constraint violation minimization is more important. Since we care more about constraint observation, we set $\eta_d > \eta_c$ in experiments. Specifically, $\eta_c = 0.01$ and $\eta_d = 0.1$.

By initializing $\pi^c = \pi^d := \pi_{explore}$, the two policies are tightly unified by KL divergence. Following Eq. (3.10), two action preference functions Ψ^c, Ψ^d are introduced to leverage update rule of DPP and can be approximated by linear or nonlinear function approximation. We choose linear function approximation (LFA) since DPP with LFA is well-studied in robotics domain [Cui et al., 2017a, Tsurumine et al., 2019]. For a given state s, Ψ is approximated as: $\Psi(s) =$

$\boldsymbol{\phi}^T(s)\boldsymbol{\theta}$, where $\boldsymbol{\phi}$ is a basis function and $\boldsymbol{\theta}$ is the weight parameter. One specifies an optimal policy by finding an optimal $\boldsymbol{\theta}^*$, usually given by the solution of the least-square problem: $||\mathcal{O}\Psi - \Phi\boldsymbol{\theta}||^2, \Phi = [\boldsymbol{\phi}(s_1), \ldots, \boldsymbol{\phi}(s_N)]^T$. θ may vary in size depending on the approximation method used, e.g. FKDPP or FFDPP as introduced in Chapter 3.

4.2.1 Scaling Constraint Violation

We propose to scale up the $\{0, 1\}$ encoding of r_d as it is taken expectation with respect to r_c with unknown magnitude. If the magnitudes of cost and constraint vary drastically, huge number of samples are needed for convergence. Motivated by [Bušić and Meyn, 2018] that scaling reward is equivalent to solving a family of MDPs, in our approach r_d is scaled up in accordance to the magnitude of r_c with interpretation that different values of scaling factor correspond to different levels of conservativeness.

4.2.2 Conservative Exploration

Intertwining π^c and π^d renders actor considers advisor and vice versa. However, optimal balance between safe exploration and goal reaching requires more than individual roles played by π^c, π^d. Drawing inspiration from [Kakade and Langford, 2002], we propose the exploration policy:

$$\pi_{explore} = (1 - \beta)\pi^c + \beta\pi^d. \tag{4.4}$$

In [Kakade and Langford, 2002], π^d is the assumed given greedy policy chooser. We interpret greedy as considering only safety and ignore goal reaching. Coefficients β is reweighting the importance of cost and constraint to strengthen exploration and convergence. In practice, one can gradually increase β to shift the exploration policy towards greedy constraint satisfaction. The resultant policy takes both cost minimization and constraint satisfaction into account. We formalize Dynamic Actor-Advisor Programming in Alg. 4. N, τ, γ, σ are the parameters from original DPP. β is the new parameter introduced in the DAAP framework. At every time step t, safe action is generated by Eq. (4.4), cost r_c and constraint violation r_d are stored in the buffers. At the end of every iteration i, we update the action preferences Ψ^c, Ψ^d following Eqs. (4.2) and (4.3), respectively.

Algorithm 4: Dynamic Actor-Advisor Programming

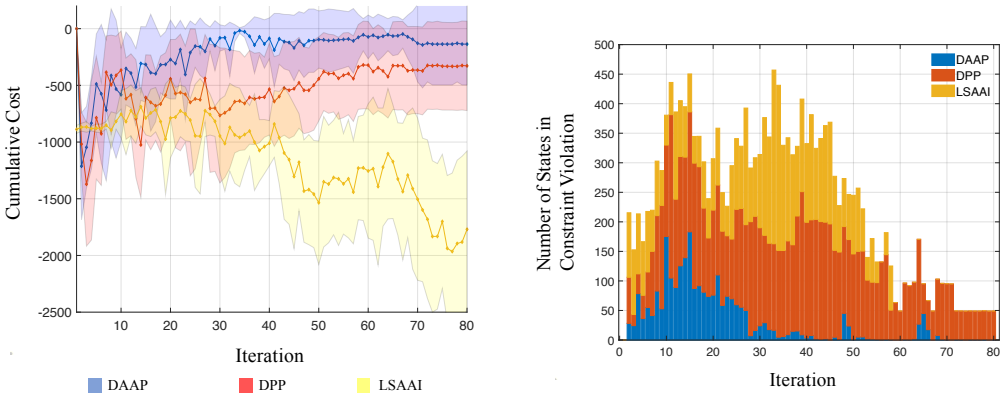
Input: $\gamma, \eta, T, I, \sigma, \beta$

Output: weight vectors θ^c, θ^d

Initialize policy $\Psi^c_{explore} = \Psi^d_{explore} := \Psi_{random}$,
    Buffer $D_i = \{\}$, $\mathcal{R}^c_i = \{\}$, $\mathcal{R}^d_i = \{\}$
for $i = 1, \ldots, I$ **do**
    for $t = 1, \ldots, T$ **do**
        compute $V^c_{\bar{\pi}^d}, V^d_{\bar{\pi}^c}(s)$ by Eqs. (4.2) (4.3)
        compute $\hat{\Psi}^c_t, \hat{\Psi}^c_t$ following standard DPP Eq. (3.10)
        sample a_t following Eq. (4.4)
        collect $x_t = [s_t, a_t]$ in $\mathcal{D}_i$
        collect r^c_t in $\mathcal{R}^c_i$
        collect r^d_t in $\mathcal{R}^d_i$
    end
    compute $\mathcal{O}\hat{\Psi}^c_t$ using $\mathcal{D}_i$, $\mathcal{R}^c_i$
    compute $\mathcal{O}\hat{\Psi}^d_t$ using $\mathcal{D}_i$, $\mathcal{R}^d_i$
    update θ using least square solution:
    $\theta^c_{t+1} = [{\Phi_t}^T \Phi_t + \sigma^2 I]^{-1} {\Phi_t}^T \mathcal{O}\hat{\Psi}^c_t$
    $\theta^d_{t+1} = [{\Phi_t}^T \Phi_t + \sigma^2 I]^{-1} {\Phi_t}^T \mathcal{O}\hat{\Psi}^d_t$
    update $\beta \to 0$
end

4.3 Experimental Results

In this section DAAP is examined in two simulations: a 5-DOF reaching task with obstacle and a robot arm peg-in-hole assembly task with movable peg. We call the method DPP that simply mixes cost and constraint shown in Eq. (4.1); the method Least Square Actor-Advisor Iteration (LSAAI) that adopts independent actor and advisor using Eq. (3.2) but without KL divergence [Lagoudakis and Parr, 2003]. Performance of DAAP on two tasks is compared with DPP and LSAAI to show respectively the importance of KL divergence and intertwining actor and advisor. All three methods use their kernel version [Cui et al., 2017a,



(a) Comparison of cost during 80 iterations' learning, plotted with mean and variance.

(b) Comparison of number of states in constraint violation during 80 iterations' learning, plotted with mean.

Figure 4.1: Comparisons on cost and constraint violation between DAAP, LSAAI and DPP, averaged over ten independent experiments.

Xu et al., 2007] for efficient calculation.

4.3.1 5-DOF Reaching Task

A 5-DOF reaching task with an obstacle blocking some possible ways to the goal is inspected. N-DOF reaching task is a good testbed for RL algorithms [Cui et al., 2017a] and obstacle avoidance is an open problem that is difficult to solve efficiently.

Experimental Setting

The continuous state space of the 5-DOF reaching task is $[\theta_1, \theta_2, \ldots, \theta_5]^T, \theta_i \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ rads that represent joints of the arm. Each joint has discrete actions of increasing/decreasing 0.0175 or 0.0875 rad angle, or stay unchanged. Hence the discrete action space is of size 5×21. Each segment connecting two joints has length $\frac{1}{5}$. The goal is set at $[X_{Target}, Y_{Target}] = [0.68, 0]$ while the obstacle square is fixed at $[0.475, 0.6]$ with length 0.2. The experiment consists of 80 iterations and each iteration comprises 500 steps. All iterations begin with the same point

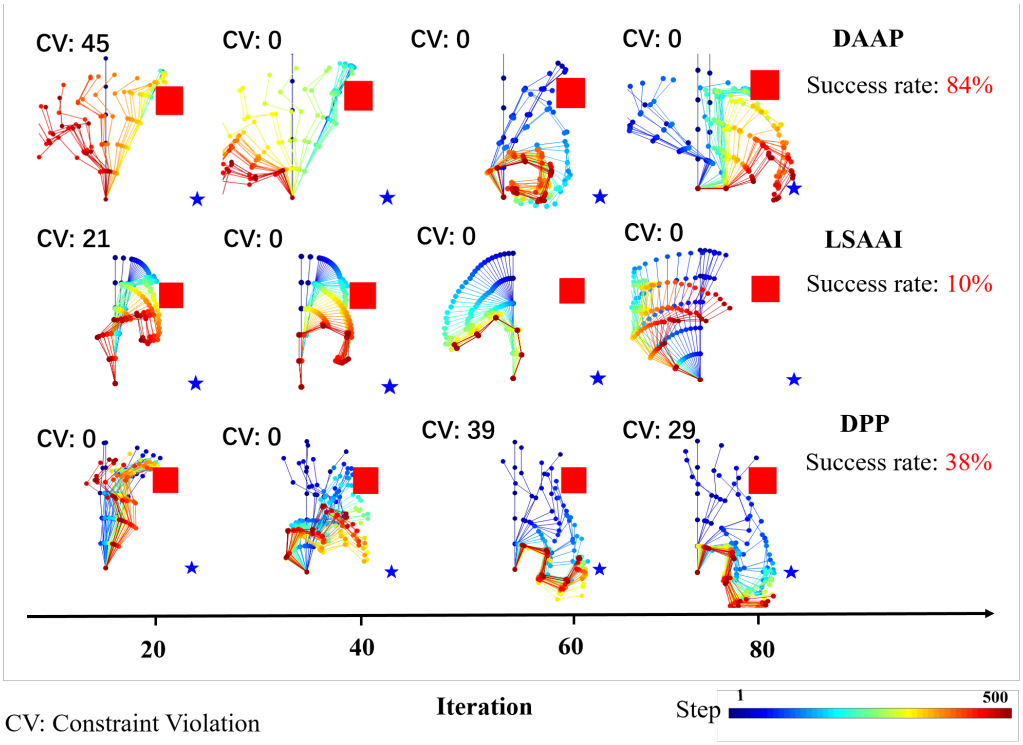


Figure 4.2: Performance comparison between DAAP, LSAAI and DPP. Success rates are obtained from 100 independent experiments. DAAP learns to avoid danger after one iteration of zero CV. LSAAI only learns to stay away from danger zone but not the goal reaching. Information loss of cost and CV renders DPP touch danger zone even after iterations of zero CV.

with all segments being vertical. Reward is set to be proportional to the distance of the end effector to the goal:

$$r_c = -1000 \times \Big((X - X_{Target})^2 + (Y - Y_{Target})^2 \Big), \tag{4.5}$$

$$r_d = \begin{cases} -1, & \text{if the obtacle is hit} \\ 0, & \text{otherwise} \end{cases} \tag{4.6}$$

Table 4.1: Meta parameters of the DAAP used for both experiments.

Parameter	Description (Number of)	Value
T	steps	500
I	iterations	80
η	DPP parameter	0.01
γ	DPP parameter	0.95
β	Mixture policy parameter	0.7
σ	Parameter preventing singularity	0.01

Results

The comparison in cost and constraint violation of DAAP, DPP and LSAAI during the 80 iterations' learning is shown in Fig. 4.1(a) and 4.1(b), respectively. Learning curves in Fig. 4.1(a) are plotted with mean and variance of ten independent experiments. It can be seen the proposed method outperforms LSAAI and DPP in both cost minimization and constraint satisfaction. Mixing cost and constraint violation in a single policy corresponds to heuristic that adds penalty term when undesired situations happen. The loss of individual information carried by the r_c and r_d incurs the poor performance. On the other hand, though LSAAI learns complete constraint satisfaction, it fails to minimize the cost.

Fig. 4.2 shows the evolution of learning. DAAP learns a path that bypass the danger and leads to the goal. KL-divergence between policy updates plays a key role in remembering the path. On the opposite, LSAAI does not share this property, the agent moves back and forth between constraint violation and satisfaction. The limited number of samples further deteriorates learning to success rate of around 10% due to random exploration. DPP performs better than LSAAI as a heuristic approach. However the information loss disables the agent from learning constraint violation distribution and further improving the success rate.

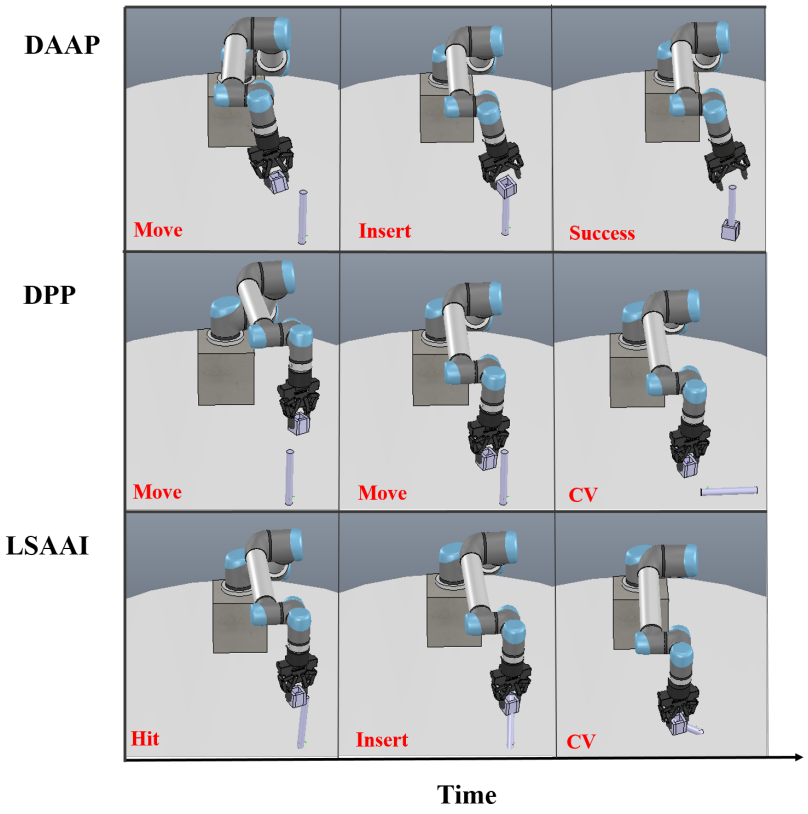


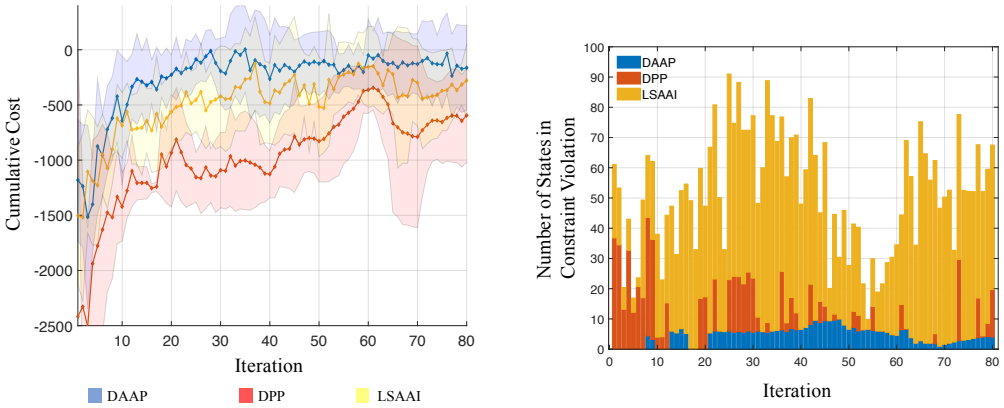
Figure 4.3: Best-case performance comparison on the assembly task by three algorithms. DAAP successfully learns to release the hole at a proper position. The blend of CV and cost renders DPP hit the peg which is near the goal. LSAAI forcefully inserts the hole and induces more CV as shown in Fig. 4.4(b).

4.3.2 Assembly Task

In this section the proposed algorithm is examined on a peg-in-hole assembly task using UR5 robot arm in simulation against LSAAI and DPP.

Experimental Setting

UR5 robot arm has six DOF. Analogously, the continuous state space is formed by $[\theta_1, \theta_2, \ldots, \theta_6]^T, \theta_i \in [-\pi, \pi]$ rads representing joints of the arm. Each joint has same discrete actions as in 5-DOF reaching task, hence the size of action space is 6×25. The robot arm needs to pick up the hole and reach an appro-



(a) Comparison of cost during 80 iterations' learning, plotted with mean and variance.

(b) Comparison of number of states in constraint violation during 80 iterations' learning, plotted with mean.

Figure 4.4: Comparisons on cost and constraint violation between DAAP, LSAAI and DPP, averaged over ten independent experiments.

priate position to release it to the bottom of the cylinder(peg). The cylinder is not fixed and can be knocked down by the robot arm or incorrect placement of the hole. Besides resetting the experiment immediately, fallover of the cylinder will assign the rest of state-action pairs and r_c, r_d the same values as that of the hitting state. All iterations begin with the same point after the gripper picking up the hole. Reward is the distance to the target joint configuration $[\theta_1^{Target}, \ldots, \theta_6^{Target}] = [-1.458, 0.925, 0.901, -0.281, -1.570, 1.683]$, likewise in Eq. (4.5), the reward function is:

$$r_c = -1000 \times \left(\sum_{i=1}^{6} (\theta_i - \theta_i^{Target})^2 \right) \tag{4.7}$$

$$r_d = \begin{cases} -1, & \text{if the cylinder falls over} \\ 0, & \text{otherwise} \end{cases} \tag{4.8}$$

When the gripper reaches the point to release the hole, an impetus of 500 is added to the reward. Except that every iteration comprises 200 steps, meta parameters are same with the 5-DOF reaching task and is summarized in Table 4.1.

This task is challenging as the cylinder is movable, and radius of the hole is

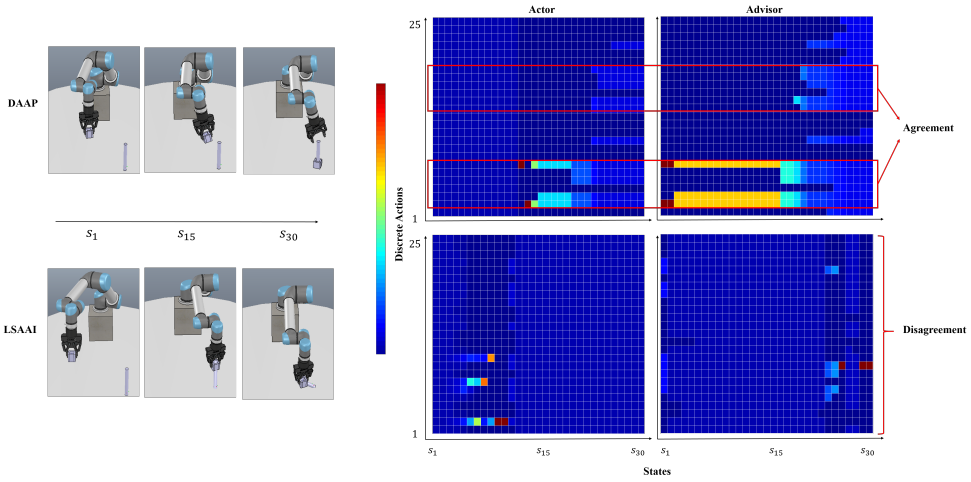


Figure 4.5: Policy comparison between DAAP and LSAAI in the assembly task over one rollout. Horizontal axis of colormaps indicates states to the goal. Vertical axis is the 25 discrete actions. Colors indicate the probability of that action being chosen. DAAP achieves sufficient estimation of actor, advisor and their agreement marked by similar colors in similar states. Sample-inefficiency of LSAAI incurs insufficient estimation and disagreement of actor and advisor.

only slightly larger than that of the cylinder. Performance should be evaluated in terms of both cost and constraint satisfaction rather than just one of them.

Results

Best-case performance of three algorithms is depicted in Fig. 4.3. Learning curves and the number of states in constraint violation shown in Fig. 4.4(a) and 4.4(b) are averaged over ten independent experiments. From Fig. 4.3, DAAP successfully learns to reach a safe position to release the hole and ensure the hole reaches the bottom of the cylinder. It can be seen from Fig. 4.4(b) that the number of states in constraint violation is significantly decreased compared to the DPP and LSAAI. Since the danger and goal are close, DPP learns a poor balance between constraint satisfaction and goal reaching due to the information loss caused by mixing r_c and r_d: instead of placing the hole at appropriate positions, the agent holds the hole at somewhere near the peg without releasing it, testifying

to the importance of separately considering cost and constraint. Even constraint violation happens sparsely in the later stage of learning, DPP performs the worst in Fig. 4.4(a). On the other hand, LSAAI also learns a policy to reach the desired joint configuration but forcefully inserts the hole, which causes the cylinder to fall over and induce constraint violation, which is due to the absense of smoothness in policy updates of LSAAI.

Intertwining actor and advisor in KL divergence of DPP update is crucial for exploiting limited number of samples to reach an agreement in decision making, as can be see in Fig. 4.5. Horizontal axis shows the states to goal and vertical axis correspond to 25 discrete actions. Colors indicate the probability of an action being chosen in different states. DAAP achieves effective estimation of actor, advisor and their agreement marked by similar color in every column with limited number of samples. The success of hole placing and reduction in constraint violation is owing to this agreement. On the other hand, LSAAI trains independent actor and advisor that leads to great disagreement and subsequent knockdown of the peg or incorrect placement of the hole especially when the agent is close to destination. Further, the problem is compounded by the sample-inefficiency of LSPI that has uniform policy in many columns shown in Fig. 4.5.

4.4 Discussion and Conclusion

In this chapter we proposed a novel KL-regularized formulation that concerned about maximizing the task reward as well as minimizing constraint violation. Aside from the KL-regularized actor that tackles the main task as usual, an additional advisor was introduced to handle the constraint such as the robot hand hitting the obstacle or sensor reading crossing over some prespecified. The reason for considering KL-regularized advisor is that (1) it generalizes the greedy advisor and provides an additional degree of freedom for making the advisor more flexible; (2) a greedy advisor caring only about minimizing the constraint violation can actually lead the actor astray into achieving low reward also, as was shown in Figure 4.2. The formulation of DAAP is general and can acommodate a wide range of tasks where safety or constraint is of concern, as long as the constraint can be readily captured as a scalar value (0 or 1 here).

It is worth noting that, the proposed method DAAP is meant only as a prototype rather than a complete, off-the-shelf algorithm. Many possible extensions exist. For example, using the binary constraint violation variable actually introduced a sparse-reward problem, which is known to be challenging in practice, since the advisor had zero value at most of the states. We could apply some transform to the constraint violation to turn it into a continuous variable, or re-define it using multiple categorical values such as $\{0, 1, 2, \dots\}$ to avoid the sparse-reward issue.

Another important observation is that the proposed method, as well as an entire catogory of safe RL methods to which it belongs to, can only learn to minimize the constraint violation when it has experienced violation. This is undesirable in practice since such violation may be prohibitive or unacceptable. Possible solutions to this issue are to employ the CMDP formulation which assigns violation budgets for every state so in critical states constraint violation might be ruled out. Another solution is to use the proposed method DAAP in simulation to learn a safe policy, but concatenate a sim2real downstream training procedure to transfer the learned policy to real-world systems.

5 | Robustness: policy oscillation

Reminder: The topic of *robustness* is divided into two chapters according to their respective different definitions: the first part, being more theoretical and concerned with robustness against policy oscillation which is a long unsolved problem in RL, is presented in this chapter. The second chapter, aiming to solve practical problems, focuses on the robustness againist configurations which refer to hypeparameters, MDP design, etc. to achieve better learning efficiency since RL is notorious for being unstable and sensitive to the above-mentioned ingredients of an algorithm. We present it in Chapter 6.

5.1 Introduction

Aside from safety in RL, another crucial reason for the limitation of real-world RL deployment is the lack of guarantee that the performance of RL policies will improve monotonically: they often oscillate during policy updates, which is unacceptable for some applications such as in recommender systems, chemical processes, self-driving, etc. For example, it is expected that alone with the collection of user data, the system should be increasingly accurate on capturing the items the user is most interested in. In such cases, oscillation in performance might result in the system recommending items the customer does not have interest. As

such, deploying such policies that have oscillated performance without examining their reliability might bring severe consequences in real-world scenarios.

Dynamic programming (DP) [Bertsekas, 2005b] offers a well-studied framework under which strict policy improvement is possible: with a known state transition model, reward function, and exact computation, monotonic improvement is ensured and convergence is guaranteed within a finite number of iterations [Ye, 2011]. However, in practice an accurate model of the environment is rarely available. In situations where either model knowledge is absent or the DP value functions cannot be explicitly computed, approximate DP and corresponding RL methods are to be considered. However, approximation introduces unavoidable update and Monte-Carlo sampling errors, and possibly restricts the policy space in which the policy is updated, leading to the *policy oscillation* phenomenon [Bertsekas, 2011, Wagner, 2011], whereby the updated policy performs worse than pre-update policies during intermediate stages of learning. Inferior updated policies resulting from policy oscillation could pose a physical threat to real-world RL applications. Further, as value-based methods are widely employed in the state-of-the-art RL algorithms [Haarnoja et al., 2018], addressing the problem of policy oscillation becomes important in its own right.

Previous studies [Kakade and Langford, 2002, Pirotta et al., 2013b] attempt to address this issue by optimizing lower bounds of policy improvement: the classic conservative policy iteration (CPI) [Kakade and Langford, 2002] algorithm states that, if the new policy is linearly interpolated by the greedy policy and the baseline policy, non-negative lower bound on the policy improvement can be defined. Since this lower bound is a negative quadratic function in the interpolation coefficient, one can solve for the maximizing coefficient to obtain maximum improvement at every update. CPI opened the door of monotonic improvement algorithms and the concept of linear interpolation can be regarded as performing regularization in the stochastic policy space to reduce greediness. Such regularization is theoretically sound as it has been proved to converge to global optimum [Neu et al., 2017, Scherrer and Geist, 2014]. For the last two decades, CPI has inspired many studies on ensuring monotonic policy improvement. However, those studies (including CPI itself) are mostly theory-oriented and hardly applicable to practical scenarios, in that maximizing the lowerbound requires solving several state-action-space-

wise maximization problems, e.g. estimating the maximum distance between two arbitrary policies. One significant factor causing the complexity might be its excessive generality [Kakade and Langford, 2002, Pirotta et al., 2013b]; these bounds do not focus on any particular class of value-based RL algorithms, and hence without further assumptions the problem cannot be simplified.

Another recent trend of developing algorithms robust to the oscillation is by introducing regularizers into the reward function. For example, by maximizing reward as well as Shannon entropy of policy [Ziebart, 2010], the optimal policy becomes a multi-modal Boltzmann softmax distribution which avoids putting unit probability mass on the greedy but potentially sub-optimal actions corrupted by noise or error, significantly enhancing the robustness since optimal actions always have nonzero probabilities of being chosen. On the other hand, the introduction of Kullback-Leibler (KL) divergence [Todorov, 2006] has recently been identified to yield policies that average over all past value functions and errors, which enjoys state-of-the-art error dependency theoretically [Vieillard et al., 2020a]. Though entropy-regularized algorithms have superior finite-time bounds and enjoy strong empirical performance, they do not guarantee to reduce policy oscillation since degradation during learning can still persist [Nachum et al., 2018].

It is hence natural to raise the question of whether the practically intractable lowerbounds from the monotonic improvement literature can benefit from entropy regularization if we restrict ourselves to the entropy-regularized policy class. By noticing that the policy interpolation and entropy regularization actually perform regularization in different aspects, i.e. in the stochastic policy space and reward function, we answer this question by affirmative. We show focusing on the class of entropy-regularizede policies significantly simplifies the problem as a very recent result indicates a sequence of entropy-regularized policies has bounded KL divergence [Kozuno et al., 2019]. This result sheds light on approximating the intractable lowerbounds from the monotonic improvement algorithms since many quantities are related to the maximum distance between two arbitrary policies.

In this chapter, we aim to tackle the policy oscillation problem by ensuring monotonic improvement via optimizing a more tractable lowerbound. This novel entropy regularization aware lower bound of policy improvement depends only the expected policy advantage function. We call the resultant algorithm *cau-*

tious policy programming (CPP). CPP leverages this lower bound as a criterion for adjusting the degree of a policy update for alleviating policy oscillation. By introducing heuristic designs suitable for nonlinear approximators, CPP can be extended to working with deep networks. The extensions are compared with the state-of-the-art algorithm [Vieillard et al., 2020b] on monotonic policy improvement. We demonstrate that our approach can trade off performance and stability in both didactic classic control problems and challenging Atari games.

The contribution of this chapter can be succinctly summarized as follows:

- we develop an easy-to-use lowerbound for ensuring monotonic policy improvement in RL.
- we propose a novel scalable algorithm CPP which optimizes the lowerbound.
- CPP is validated to reduce policy oscillation on high-dimensional problems which are intractable for prior methods.

5.2 Policy Oscillation and entropy regularization

The policy oscillation phenomenon, also termed *overshooting* by [Wagner, 2011] and referred to as degraded performance of updated policies, frequently arises in approximate policy iteration algorithms [Bertsekas, 2011] and can occur even under asymptotically converged value functions [Wagner, 2011]. It has been shown that aggressive updates with sampling and update errors, together with restricted policy spaces, are the main reasons for policy oscillation [Pirotta et al., 2013b]. In modern applications of RL, policy oscillation becomes an important issue when learning with deep networks when various sources of errors have to been taken in to account. It has been investigated by [Fu et al., 2019, Fujimoto et al., 2018] that those errors are the main cause for typical oscillating performance with deep RL implementations.

To attenuate policy oscillation, the seminal algorithm conservative policy iteration (CPI) [Kakade and Langford, 2002] propose to perform regularization in the stochastic policy space, whereby the greedily updated policy is interpolated with the current policy to achieve less aggressive updates. CPI has inspired numerous conservative algorithms that enjoy strong theoretical guarantees [Abbasi-Yadkori

et al., 2016, Metelli et al., 2018, Pirotta et al., 2013a,b] to improve upon CPI by proposing new lower bounds for policy improvement. However, since their focus is on general Markov decision processes (MDPs), deriving practical algorithms based on the lower bounds is nontrivial and the proposed lower bounds are mostly of theoretical value. Indeed, as admitted by the authors of [Papini et al., 2020] that a large gap between theory and practice exists, as manifested by the their experimental results that even for a simple Cartpole environment, state-of-the-art algorithm failed to deliver attenuated oscillation and convergence speed comparable with heuristic optimization scheme such as Adam [Kingma and Ba, 2015]. This might explain why *adaptive coefficients* must be introduced in [Vieillard et al., 2020b] to extend CPI to be compatible with deep neural networks. To remove this limitation, our focus on entropy-regularized MDPs allows for a straightforward algorithm based on a novel, significantly simplified lower bound.

Another line of research toward alleviating policy oscillation is to incorporate regularization as a penalty into the reward function, leading to the recently booming literature on entropy-regularized MDPs [Azar et al., 2012, Fox et al., 2016, Haarnoja et al., 2017, Kozuno et al., 2019, Mei et al., 2019, Vieillard et al., 2020a]. Instead of interpolating greedy policies, the reward is augmented with entropy of the policy, such as Shannon entropy for more diverse behavior and smooth optimization landscape [Ahmed et al., 2019], or Kullback-Leibler (KL) divergence for enforcing policy similarity between policy updates and hence achieving superior sample efficiency [Uchibe, 2018, Uchibe and Doya, 2021]. The Shannon entropy renders the optimal policy of the regularized MDP stochastic and multi-modal and hence robust against errors and noises in contrast to the deterministic policy that puts all probability mass on a single action [Haarnoja et al., 2018]. On the other hand, augmenting with KL divergence shapes the optimal policy an average of all past value functions, which is significantly more robust than a single point estimate. Compared to the CPI-based algorithms, entropy-regularized algorithms do not have guarantee on per-update improvement. But they have demonstrated state-of-the-art empirical successes on a wide range of challenging tasks [Cui et al., 2017a, Tsurumine et al., 2019, Zhu et al., 2020, 2022]. To the best of the authors' knowledge, unifying those two regularization schemes has not

been considered in published literature before.

It is worth noting that, inspired by [Kakade and Langford, 2002], the concept of monotonic improvement has been exploited also in policy search scenarios [Akrour et al., 2018, Mei et al., 2020, Papini et al., 2020, Schulman et al., 2015, Shani et al., 2019]. However, there is a large gap between theory and practice in those policy gradient methods. On one hand, though [Schulman et al., 2015, 2017] demonstrated good empirical performance, their relaxed trust region is often too optimistic and easily corrupted by noises and errors that arise frequently in the deep RL setting: as pointed out by [Engstrom et al., 2019], the trust region technique itself alone fails to explain the efficiency of the algorithms and lots of code-level tricks are necessary. On the other hand, exactly following the guidance of monotonic improving gradient does not lead to tempered oscillation and better performance even for simple problems [Papini et al., 2017, 2020]. Another shortcoming of policy gradient methods is they focus on local optimal policy with strong dependency on initial parameters. On the other hand, we focus on value-based RL that searches for global optimal policies.

5.3 Preliminary

This chapter requires additional notations besides that in Section 3.2.3. Let us consider the reward $r^a_{ss'}$ as bounded in the interval $[-1, 1]$. $\gamma \in (0, 1)$ is the discount factor. For simplicity, we consider the infinite horizon discounted setting with a fixed starting state s_0. Recall that a policy π is a probability distribution over actions given some state. We also define the stationary state distribution induced by π as $d^\pi(s) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t\mathcal{T}(s_t = s|s_0, \pi)$.

5.3.1 Lower Bounds on Policy Improvement

To frame the monotonic improvement problem, we introduce the following lemma that formally defines the criterion of policy improvement of some policy π' over π:

Lemma 5.3.1. *[Kakade and Langford, 2002] For any stationary policies π' and π, the following equation holds:*

$$\begin{aligned} &\Delta J^{\pi'}_{\pi,d^{\pi'}} := J^{\pi'}_d - J^{\pi}_d = \sum_s d^{\pi'}(s) \sum_a \pi'(a|s) A_\pi(s,a), \\ &\textit{where } J^{\pi'}_d := \mathbb{E}_{s_0,a_0,\ldots}\left[(1-\gamma)\sum_{t=0}^{\infty}\gamma^t r_t\right] = \sum_s d^{\pi'}(s)\sum_a \pi'(a|s) r^a_{ss'}, \end{aligned} \tag{5.1}$$

J is the discounted cumulative reward, and $A_\pi(s,a) := \quad Q_\pi(s,a) - V_\pi(s)$ is the advantage function. Though Lemma 5.3.1 relates policy improvement to the expected advantage function, pursuing policy improvement by directly exploiting Lemma 5.3.1 is intractable as it requires comparing π' and π point-wise for infinitely many new policies. Many existing works [Kakade and Langford, 2002, Pirotta et al., 2013b, Schulman et al., 2015] instead focus on finding a π' such that the right-hand side of Eq. (5.1) is lower bounded. To alleviate policy oscillation brought by the greedily updated policy $\tilde{\pi}$, [Kakade and Langford, 2002] proposes adopting *partial update*:

$$\pi' = \zeta\tilde{\pi} + (1-\zeta)\pi. \tag{5.2}$$

Eq. (5.2) corresponds to performing regularization in the stochastic policy space by interpolating the greedy policy and the current policy to achieve conservative updates.

The concept of linearly interpolating policies has inspired many algorithms that enjoy strong theoretical guarantees [Akrour et al., 2018, Metelli et al., 2018, Pirotta et al., 2013b]. However, those algorithms are mostly of theoretical value and have only been applied to small problems due to intractable optimization or estimation when the state-action space is high-dimensional/continuous. Indeed, as admitted by the authors of [Papini et al., 2020], there is a large gap between theory and practice when using algorithms based on policy regularization Eq. (5.2): even on a simple CartPole problem, a state-of-the-art algorithm fail to compete with heuristic optimization technique. Like our proposal in this paper, a very recent work [Vieillard et al., 2020b] attempts to bridge this gap by proposing heuristic coefficient design for learning with deep networks. We discuss the relationship between it and the CPP in Section 5.4.6.

In the next section, we detail the derivation of the proposed lower bound by exploiting entropy regularization. This novel lower bound allows us to significantly simplify the intractable optimization and estimation in prior work and provide a scalable implementation.

5.4 Proposed Method

This section features the proposed novel lower bound on which we base a novel algorithm for ensuring monotonic policy improvement.

5.4.1 Entropy-regularized RL

In the following discussion, we provide a general formulation for entropy-regularized algorithms [Azar et al., 2012, Haarnoja et al., 2018, Kozuno et al., 2019]. At iteration K, the entropy of current policy π_K and the Kullback-Leibler (KL) divergence between π_K and some baseline policy $\bar{\pi}$ are added to the value function:

$$\begin{aligned} V_{\bar{\pi}}^{\pi_K}(s) &:= \sum_{\substack{a\in\mathcal{A}\\ s'\in\mathcal{S}}} \pi(a|s)\left[\mathcal{T}_{ss'}^{a}\left(r_{ss'}^{a}+\gamma V_{\bar{\pi}}^{\pi_K}(s')\right)-\mathcal{I}_{\bar{\pi}}^{\pi_K}\right], \\ \mathcal{I}_{\bar{\pi}}^{\pi_K} &= -\tau\log\pi_K(a|s)-\sigma\log\frac{\pi_K(a|s)}{\bar{\pi}(a|s)}, \end{aligned} \tag{5.3}$$

where τ controls the weight of the entropy bonus and σ weights the effect of KL regularization. The baseline policy $\bar{\pi}$ is often taken as the policy from previous iteration π_{K-1}. Based on [Nachum et al., 2017, 2018], we know the state value function $V_{\bar{\pi}}^{\pi_K}$ defined in Eq. (5.3) and state-action value function $Q_{\bar{\pi}}^{\pi_K}$ also satisfy the Bellman recursion:

$$Q_{\bar{\pi}}^{\pi_K}(s,a) := r_{ss'}^{a}+\gamma\sum_{s'}\mathcal{T}_{ss'}^{a}V_{\bar{\pi}}^{\pi_K}(s').$$

For notational convenience, in the remainder of this paper, we use the following definition:

$$\alpha := \frac{\tau}{\tau+\sigma}, \quad \beta := \frac{1}{\tau+\sigma}. \tag{5.4}$$

An intuitive explanation to Eq. (5.3) is that the entropy term endows the optimal policy with multi-modal policy behavior [Haarnoja et al., 2017] by placing nonzero probability mass on every action candidate, hence is robust against error and noise in function approximation that can easily corrupt the conventional deterministic optimal policy [Puterman, 1994]. On the other hand, KL divergence provides smooth policy updates by limiting the size of the update step [Azar et al., 2012, Kozuno et al., 2019, Schulman et al., 2015]. Indeed, it has been recently shown that augmenting the reward with KL divergence renders the optimal policy an exponential smoothing of all past value functions [Vieillard et al., 2020a]. Limiting the update step plays a crucial role in the recent successful algorithms since it prevents the aggressive updates that could easily be corrupted by errors [Fu et al., 2019, Fujimoto et al., 2018]. It is worth noting that when the optimal policy is attained, the KL regularization term becomes zero. Hence in Eq. (5.3), the optimal policy maximizes the cumulative reward while keeping the entropy high.

5.4.2 Entropy-regularization-aware Lower Bound

Recall in Eq. (5.2) performing regularization in the stochastic policy space for the greedily updated policy $\tilde{\pi}$ requires preparing a reference policy π. This policy could be from expert knowledge or previous policies. The resultant π', has guaranteed monotonic improvement which we formulate as the following lemma:

Lemma 5.4.1 ([Pirotta et al., 2013b])**.** *Provided that policy π' is generated by partial update Eq. (5.2), ζ is chosen properly, and $A^{\tilde{\pi}}_{\pi,d^\pi} \geq 0$, then the following policy improvement is guaranteed:*

$$
\begin{aligned}
\Delta J^{\pi'}_{\pi,d^{\pi'}} &\geq \frac{\left((1-\gamma)A^{\tilde{\pi}}_{\pi,d^\pi}\right)^2}{2\gamma\delta\Delta A^{\tilde{\pi}}_{\pi}}, \\
\textit{with } \zeta &= \min\left(1, \zeta^*\right), \\
\textit{where } \zeta^* &= \frac{(1-\gamma)^2 A^{\tilde{\pi}}_{\pi,d^\pi}}{\gamma\delta\Delta A^{\tilde{\pi}}_{\pi}}, \\
\delta &= \max_s \left| \sum_{a\in\mathcal{A}} \left(\tilde{\pi}(a|s) - \pi(a|s)\right) \right|, \\
\Delta A^{\tilde{\pi}}_{\pi} &= \max_{s,s'} |A^{\tilde{\pi}}_{\pi}(s) - A^{\tilde{\pi}}_{\pi}(s')|,
\end{aligned}
\tag{5.5}
$$

where $A^{\tilde{\pi}}_{\pi,d^\pi} := \sum_s d^\pi(s) \sum_a \left(\tilde{\pi}(a|s) - \pi(a|s)\right) Q_\pi(s,a)$.

Proof. The proof follows the similar derivation in the classic CPI [Kakade and Langford, 2002] and similar results appeared many times in e.g. [Metelli et al., 2018, Pirotta et al., 2013b]. We also show that the role of ζ and $(1-\zeta)$ in Eq. (5.2) can be exchanged by solving a similar problem. To begin, we leverage Theorem 3.5 of [Pirotta et al., 2013b] that:

$$\Delta J^{\pi'}_{\pi,d^{\pi'}} \geq A^{\pi'}_{\pi,d^\pi} - \frac{\gamma \Delta A^{\pi'}_\pi}{2(1-\gamma)^2} \max_s \left| \sum_{a \in \mathcal{A}} \left(\pi'(a|s) - \pi(a|s)\right) \right|. \tag{5.6}$$

Substituting in $\pi' = \zeta\tilde{\pi} + (1-\zeta)\pi$, we have:

$$\begin{aligned} A^{\pi'}_{\pi,d^\pi} &= \sum_s d^\pi(s) \sum_a \pi'(a|s) A_\pi(s,a) \\ &= \sum_s d^\pi(s) \sum_a \left(\zeta\tilde{\pi}(a|s) + (1-\zeta)\pi(a|s)\right) A_\pi(s,a) \\ &= \zeta \sum_s d^\pi(s) \sum_a \tilde{\pi}(a|s) A_\pi(s,a) = \zeta A^{\tilde{\pi}}_{\pi,d^\pi}, \end{aligned} \tag{5.7}$$

$$\begin{aligned} \Delta A^{\pi'}_\pi &= \max_{s,s'} |A^{\pi'}_\pi(s) - A^{\pi'}_\pi(s')| \\ &= \max_{s,s'} |\zeta A^{\tilde{\pi}}_\pi(s) - \zeta A^{\tilde{\pi}}_\pi(s')|, \\ \delta &= \max_s \left| \sum_{a \in \mathcal{A}} \left(\pi'(a|s) - \pi(a|s)\right) \right|, \\ &= \max_s \left| \sum_{a \in \mathcal{A}} \left(\zeta\tilde{\pi}(a|s) - \zeta\pi(a|s)\right) \right|. \end{aligned} \tag{5.8}$$

Hence, Eq. (5.6) is transformed into:

$$\Delta J^{\pi'}_{\pi,d^{\pi'}} \geq \zeta A^{\tilde{\pi}}_{\pi,d^\pi} - \frac{\gamma\zeta^2 \Delta A^{\tilde{\pi}}_\pi}{2(1-\gamma)^2} \max_s \left| \sum_{a \in \mathcal{A}} \left(\tilde{\pi}(a|s) - \pi(a|s)\right) \right|. \tag{5.9}$$

The right hand side (r.h.s.) is a quadratic function in ζ and has its maximum at

$$\zeta^* = \frac{(1-\gamma)^2 A^{\tilde{\pi}}_{\pi,d^\pi}}{\gamma \Delta A^{\tilde{\pi}}_\pi \max_s \left| \sum_{a \in \mathcal{A}} \left(\tilde{\pi}(a|s) - \pi(a|s)\right) \right|}. \tag{5.10}$$

By substituting ζ^* back to Eq. (5.9), we obtain:

$$\Delta J^{\pi'}_{\pi,d^{\pi'}} \geq \frac{\big((1-\gamma)A^{\tilde{\pi}}_{\pi,d^{\pi}}\big)^2}{2\gamma\delta\Delta A^{\tilde{\pi}}_{\pi}}. \tag{5.11}$$

When $\zeta^* > 1$, we clip it using $\min(1, \zeta^*)$.

Note that, if we exchange the roles of ζ and $(1-\zeta)$, the coefficients in Eq. (5.7) should be $(1-\zeta)$. Equation (5.9) would become a quadratic function in $(1-\zeta)$; hence the r.h.s. of Eq. (5.11) would be the maximum of $(1-\zeta^*)$. This concludes the proof.

□

The interpolated policy π' optimizes the bound and the policy improvement is a negative quadratic function in ζ. However, this optimization problem is highly non-trivial as δ and $\Delta A^{\tilde{\pi}}_{\pi}$ require searching the entire state-action space. This challenge explains why CPI-inspired methods have only been applied to small problems with low-dimensional state-action spaces [Metelli et al., 2018, Papini et al., 2020, Pirotta et al., 2013b]. When the expert knowledge is not available, we can simply choose previous policies. Specifically, at any iteration K, we want to leverage the monotonic policy improvement given policy π_K. We propose constructing a new monotonically improving policy as:

$$\tilde{\pi}_{K+1} = \zeta\pi_{K+1} + (1-\zeta)\pi_K. \tag{5.12}$$

It is now clear by comparing Eq. (5.2) with Eq. (5.12) that our proposal takes $\pi', \tilde{\pi}, \pi$ as $\tilde{\pi}_{K+1}, \pi_{K+1}, \pi_K$, respectively. It is worth noting that π_{K+1} is the updated policy that has not been deployed.

However, the intractable quantities δ and $\Delta A^{\tilde{\pi}}_{\pi}$ in Lemma 5.4.1 are still an obstacle to deriving a scalable algorithm. Specifically, by writing the component $A^{\tilde{\pi}}_{\pi}(s)$ of $\Delta A^{\tilde{\pi}}_{\pi}$ as

$$A^{\tilde{\pi}}_{\pi}(s) = \sum_a \big(\tilde{\pi}(a|s) - \pi(a|s)\big)Q_\pi(s,a),$$

we see that both δ and $\Delta A^{\tilde{\pi}}_{\pi}$ require accurately estimating the total variation between two policies. This could be difficult without enforcing constraints such as gradual change of policies. Fortunately, by noticing that the consecutive entropy-regularized policies π_{K+1}, π_K have bounded total variation, we can leverage the boundedness to bypass the intracatable estimation.

Lemma 5.4.2 ([Kozuno et al., 2019]). *For any policies π_K and π_{K+1} generated by taking the maximizer of Eq. (5.3), the following bound holds for their maximum total variation:*

$$\max_s D_{TV}\left(\pi_{K+1}(\cdot|s) \,\|\, \pi_K(\cdot|s)\right) \leq \min\left\{\sqrt{1-e^{-4B_K-2C_K}}, \sqrt{8B_K+4C_K}\right\},$$
$$\text{where } B_K = \frac{1-\gamma^K}{1-\gamma}\epsilon\beta, \quad C_K = \beta r_{max}\sum_{k=0}^{K-1}\alpha^k\gamma^{K-k-1}, \tag{5.13}$$

K denotes the current iteration index and $0 \leq k \leq K-1$ is the loop index. ϵ is the uniform upper bound of error.

Proof. By the Fenchel conjugacy of the Shannon entropy and KL divergence [Boyd and Vandenberghe, 2004], it is clear that the maximizing policies for the regularized MDP are Boltzmann softmax [Geist et al., 2019] as shown in Section 5.4.4. The relationship between Boltzmann softmax policies has recently been actively investigated [Asadi and Littman, 2017, Azar et al., 2012]. We leverage the very recent result [Kozuno et al., 2019, Propsition 3], which states that:

$$\max_s D_{KL}\left(\pi_{K+1}(\cdot|s) \,\|\, \pi_K(\cdot|s)\right) \leq 4B_K + 2C_K,$$
$$\text{where } B_K = \frac{1-\gamma^K}{1-\gamma}\epsilon\beta,\ C_K = \beta r_{max}\sum_{k=0}^{K-1}\alpha^k\gamma^{K-k-1}, \tag{5.14}$$

where ϵ is the uniform upper bound of errors.

While Pinsker's inequality $D_{TV}(p\|q) \leq \sqrt{2D_{KL}(p\|q)}$, where p, q are distributions can be used to directly exploit Eq. (5.14), there is a gap between the total variation and KL divergence since $D_{TV} \leq 1$ and D_{KL} is potentially unbounded. Leveraging Pinsker's inequality on Eq. (5.14) and then on Eqs. (5.7,5.8) will result in large errors when $D_{KL} \geq \frac{\sqrt{2}}{2}$.

To tackle this problem, we introduce the following bound due to [Bretagnolle and Huber, 1978] that has more benign behavior[1]:

$$D_{TV}(p\|q) \leq \sqrt{1-e^{-D_{KL}(p\|q)}}. \tag{5.15}$$

[1]Eq. (5.15) appears in other places in different forms such as in [Sason and Verdú, 2016, Eq. (4)]). It is worth mentioning they are the same in essence and differ only in notations.

A similar bound appears also in [Tsybakov, 2008] but is a slightly looser. More relevant inequalities of such kind can be found in [Sason and Verdú, 2016]. Both [Bretagnolle and Huber, 1978] and [Tsybakov, 2008] feature the component $e^{-D_{KL}(p\|q)}$ that ensures the total variation bound is well-defined: the upperbound $\sqrt{1-e^{-D_{KL}(p\|q)}}$ is guaranteed to be no large than 1. Hence we can combine Eq. (5.15) with Eq. (5.14) by taking the maximization on both sides, yielding the following relationship:

$$\begin{aligned}\max_s D_{TV}(\pi_{K+1}(\cdot|s)\|\pi_K(\cdot|s)) &\leq \sqrt{1-e^{-\max_s D_{KL}(\pi_{K+1}(\cdot|s)\|\pi_K(\cdot|s)}} \\ &\leq \sqrt{1-e^{-4B_K-2C_K}}.\end{aligned} \tag{5.16}$$

Now by applying Pinsker's inequality on Eq. (5.14), we have the following relationship:

$$\max_s D_{TV}\left(\pi_{K+1}(\cdot|s)\,\|\,\pi_K(\cdot|s)\right) \leq \sqrt{8B_K+4C_K}, \tag{5.17}$$

taking the minimum of Eqs. (5.16, 5.17) yields the promised result.

□

Lemma 5.4.2 states that, entropy-regularized policies have bounded total variation (and hence bounded KL divergence by Pinsker's and Kozuno's inequality [Kozuno et al., 2019]). This bound allows us to bypass the intractable estimation in Lemma 5.4.1 and approximate $\tilde{\pi}_{K+1}$ that optimizes the lowerbound. We formally state this result in the Theorem 5.4.3 below.

For convenience, we assume there is no error, i.e. $B_K = 0$. Setting $B_K = 0$ is only for the ease of notation of our latter derivation. Our results still hold by simply replacing all appearance of C_K to $B_K + C_K$. On the other hand, in implementation it requires a sensible choice of upper bound of error which is typically difficult especially for high dimensional problems and with nonlinear function approximators. Fortunately, by the virtue of KL regularization in Eq. (5.3), it has been shown in [Azar et al., 2012, Vieillard et al., 2020d] that if the sequence of errors is a martingale difference under the natural filtration, then the summation of errors asymptotically cancels out. Hence it might be safe to simply set $B_K = 0$ if we assume the martingale difference condition.

Theorem 5.4.3. *Provided that partial update Eq. (5.12) is adopted,* $A^{\pi_{K+1}}_{\pi_K, d^{\pi_K}} \geq 0$, *and* ζ *is chosen properly as specified below, then any maximizer policy of Eq. (5.3) guarantees the following improvement that depends only on* α, β, γ *and* $A^{\pi_{K+1}}_{\pi_K, d^{\pi_K}}$ *after any policy update:*

$$\begin{aligned}
\Delta J^{\tilde{\pi}_{K+1}}_{\pi_K, d^{\tilde{\pi}_{K+1}}} &\geq \frac{\big(1-\gamma\big)^3 (A^{\pi_{K+1}}_{\pi_K, d^{\pi_K}})^2}{4\gamma} \max\left\{ \frac{1}{1-e^{-2C_K}} \ , \ \frac{1}{4C_K} \right\}, \\
\textit{with } \zeta &= \min\big(1, \zeta^*\big), \quad C_K = \beta \sum_{k=0}^{K-1} \alpha^k \gamma^{K-k-1}, \\
\textit{where } \zeta^* &= \frac{(1-\gamma)^3 A^{\pi_{K+1}}_{\pi_K, d^{\pi_K}}}{2\gamma} \max\left\{ \frac{1}{1-e^{-2C_K}}, \frac{1}{4C_K} \right\},
\end{aligned} \tag{5.18}$$

α, β *are defined in Eq. (5.4) and*

$$A^{\pi_{K+1}}_{\pi_K, d^{\pi_K}} := \sum_s d^{\pi_K}(s)\, A^{\pi_{K+1}}_{\pi_K}(s), \tag{5.19}$$

$$A^{\pi_{K+1}}_{\pi_K}(s) := \sum_a \big(\pi_{K+1}(a|s) - \pi_K(a|s)\big) Q_{\pi_K}(s,a) \tag{5.20}$$

are the expected policy advantage, and the policy advantage function, respectively.

Proof. The proof follows similarly to the proof of Lemma 2 and hence [Pirotta et al., 2013b]. We prove Theorem 5.4.3 by noticing the following inequalities hold for δ and $\Delta A^{\tilde{\pi}}_{\pi}$ of Eq. (5.5), respectively:

$$\begin{aligned}
\Delta A^{\tilde{\pi}}_{\pi} &= \max_{s,s'} |A^{\tilde{\pi}}_{\pi}(s) - A^{\tilde{\pi}}_{\pi}(s')| \\
&\leq 2 \max_s |A^{\tilde{\pi}}_{\pi}(s)| = 2 \max_s \Big| \sum_a \tilde{\pi}(a|s) \big(Q_\pi(s,a) - V_\pi(s)\big) \Big| \\
&= 2 \max_s \Big| \sum_a \big(\tilde{\pi}(a|s) Q_\pi(s,a) - \pi(a|s) Q_\pi(s,a)\big) \Big| \\
&\leq 2 \max_s \sum_a \Big| \big(\tilde{\pi}(a|s) - \pi(a|s)\big) Q_\pi(s,a) \Big| \\
&\leq 2 \big|\big|Q_\pi\big|\big|_\infty \max_s \sum_a \big|\tilde{\pi}(a|s) - \pi(a|s)\big| \\
&\leq 2\sqrt{2} V_{max} \max_s \sqrt{D_{KL}\big(\tilde{\pi}(\cdot|s) || \pi(\cdot|s)\big)},
\end{aligned} \tag{5.21}$$

where $V_{max} := \frac{1}{1-\gamma} r_{max}$ is the maximum possible value function. Since we assume reward is upper bounded by 1, $V_{max} = \frac{1}{1-\gamma}$. The second inequality makes use of the triangle inequality:

$$\delta \leq \max_s \sum_{a \in \mathcal{A}} \left| \left(\tilde{\pi}(a|s) - \pi(a|s) \right) \right|, \tag{5.22}$$

and the third inequality makes use of Hölder's inequality $\frac{1}{p} + \frac{1}{q} = 1$, with p set to 1 and q set to ∞. The last inequality is because of Pinsker's inequality:

$$\max_s \sum_{a \in \mathcal{A}} \left| \tilde{\pi}(a|s) - \pi(a|s) \right| \leq \max_s \sqrt{2 D_{KL}(\tilde{\pi}(\cdot|s) \| \pi(\cdot|s))}, \tag{5.23}$$

and the fact that $\|Q_\pi\|_\infty \leq V_{max} = \frac{1}{1-\gamma}$.

Following [Pirotta et al., 2013b], by incorporating Eqs. (5.22, 5.23) and Eqs. (5.16, 5.17) into $\Delta J^{\pi'}_{\pi, d^{\pi'}} \geq \frac{\left((1-\gamma) A^{\tilde{\pi}}_{\pi, d^\pi}\right)^2}{2\gamma \delta \Delta A^{\tilde{\pi}}_{\pi}}$, we have:

$$\begin{aligned}
\Delta J^{\pi'}_{\pi, d^{\pi'}} &\geq \frac{\left((1-\gamma) A^{\tilde{\pi}}_{\pi, d^\pi}\right)^2}{2\gamma \delta \Delta A^{\tilde{\pi}}_{\pi}} \\
&\geq \frac{\left((1-\gamma) A^{\tilde{\pi}}_{\pi, d^\pi}\right)^2}{2\gamma} \underbrace{\frac{1}{\max_s D_{TV}}}_{\delta,\ \text{Eq.(A.15)}} \underbrace{\frac{1}{2 V_{max} \max_s D_{TV}}}_{\Delta A^{\tilde{\pi}}_{\pi},\ \text{Eq.(A.13)}} \\
&= \frac{(1-\gamma)^3 (A^{\tilde{\pi}}_{\pi, d^\pi})^2}{2\gamma} \frac{1}{2 \max_s D^2_{TV}} \\
&\overset{\text{(A.9)}}{\geq} \frac{(1-\gamma)^3 (A^{\tilde{\pi}}_{\pi, d^\pi})^2}{4\gamma} \frac{1}{4 C_K}, \\
\text{or} \quad &\overset{\text{(A.11)}}{\geq} \frac{(1-\gamma)^3 (A^{\tilde{\pi}}_{\pi, d^\pi})^2}{4\gamma} \frac{1}{1 - e^{-2C_K}},
\end{aligned} \tag{5.24}$$

by taking the maximum of the two possible outcomes, the result becomes:

$$\Delta J^{\pi'}_{\pi, d^{\pi'}} \geq \frac{(1-\gamma)^3}{4\gamma} \cdot (A^{\tilde{\pi}}_{\pi, d^\pi})^2 \cdot \max \left\{ \frac{1}{1 - e^{-2C_K}}, \frac{1}{4 C_K} \right\}.$$

The way of choosing ζ is same as Eq. (5.11) solving the equation that is negative quadratic in ζ.

□

While theoretically we need to compare $1 - e^{-2C_K}$ and $4C_K$ when computing ζ^*, in implementation the exponential function e^{-2C_K} might be sometimes close to 1 and hence causing numerical instability. Hence in the rest of the paper we shall stick to using the constant C_K rather than the exponential function.

In the lower bound Eq. (5.18), only $A^{\pi_{K+1}}_{\pi_K, d^{\pi_K}}$ needs to be estimated. It is worth noting that $\forall s, A^{\pi_{K+1}}_{\pi_K}(s) \geq 0$ is a straightforward criterion that is naturally satisfied by the greedy policy improvement of policy iteration when computation is exact. To handle the negative case caused by error or approximate computations, we can simply stack more samples to reduce the variance, as will be detailed in Sec. 5.4.7.

5.4.3 The CPP Policy Iteration

We now detail the structure of our proposed algorithm based on Theorem 5.4.3. Specifically, value update, policy update, and stationary distribution estimation are introduced, followed by discussion on a subtlety in practice and two possible solutions.

Following [Scherrer et al., 2015], CPP can be written in the following succinct policy iteration style:

$$\text{CPP} = \begin{cases} \pi_{K+1} \leftarrow \mathcal{G} Q^{\pi_K}_{\tilde{\pi}} \\ Q_{\pi_{K+1}} \leftarrow (T_{\pi_{K+1}})^m Q_{\pi_K} \\ \zeta = \min\left\{ (4C_K)^{-1} C_\gamma A^{\pi_{K+1}}_{\pi_K, d^{\pi_K}} \ , \ 1 \right\} \\ \tilde{\pi}_{K+1} \leftarrow \zeta \pi_{K+1} + (1-\zeta)\pi_K, \end{cases} \tag{5.25}$$

where $C_\gamma := \frac{(1-\gamma)^3}{2\gamma}$ is the horizon constant. Note that for numerical stability we stick to using $(4C_K)^{-1}$ as the entropy-bounding constant rather than using $\frac{1}{1-e^{-2C_K}}$.

Like CPI, CPP can obtain *global optimal policy* rather than just achieving monotonic improvement (which might still converge to a local optimum) by the argument of [Scherrer and Geist, 2014]. The first step corresponds to the greedy step of policy iteration, the second step policy estimation step, third step computing interpolation coefficient ζ and the last step interpolating the policy.

5.4.4 Policy Improvement and Policy Evaluation

The first two steps are standard update and estimation steps of policy iteration algorithms [Sutton and Barto, 2018]. The subscript of $Q_{\bar{\pi}}^{\pi_K}$ indicates it is entropy-regularized as introduced in Eq. (5.3).

The policy improvement step consists of evaluating $\mathcal{G}Q_{\bar{\pi}}^{\pi_K}$, which is the greedy operator acting on $Q_{\bar{\pi}}^{\pi_K}$. By the Fenchel conjugacy of Shannon entropy and KL divergence, $\mathcal{G}Q_{\bar{\pi}}^{\pi_K}$ has a closed-form solution [Beck, 2017, Kozuno et al., 2019]:

$$\mathcal{G}Q_{\bar{\pi}}^{\pi_K}(a|s) = \frac{\bar{\pi}(a|s)^\alpha \exp\left(\beta Q_{\bar{\pi}}^{\pi_K}(s,a)\right)}{\sum_b \bar{\pi}(b|s)^\alpha \exp\left(\beta Q_{\bar{\pi}}^{\pi_K}(s,b)\right)},$$

where α, β were defined in Eq. (5.4).

The policy evaluation step estimates the value of current policy π_{K+1} by repeatedly applying the Bellman operator $T_{\pi_{K+1}}$:

$$\begin{aligned}(T_{\pi_{K+1}})^m Q_{\pi_K} &:= \underbrace{T_{\pi_{K+1}} \dots T_{\pi_{K+1}}}_{m \text{ times}} Q_{\pi_K}, \\ T_{\pi_{K+1}} Q_{\pi_K} &:= r_{ss'}^a + \gamma \sum_{s'} \mathcal{T}_{ss'}^a \sum_a \pi_{K+1}(a|s') Q_{\bar{\pi}}^{\pi_K}(s',a).\end{aligned} \tag{5.26}$$

Note that $m = 1, \infty$ correspond to the value iteration and policy iteration, respectively [Bertsekas and Tsitsiklis, 1996]. Other interger-valued $m \in [2, \infty)$ correspond to the approximate modified policy iteration [Scherrer et al., 2015].

Now in order to estimate $A_{\pi_K, d^{\pi_K}}^{\pi_{K+1}}$ in Theorem 5.4.3, both $A_{\pi_K}^{\pi_{K+1}}$ and d^{π_K} need to be estimated from samples. Estimating $A_{\pi_K}^{\pi_{K+1}}(s)$ is straightforward by its definition in Eq. (6.3c). We can first compute $Q_{\pi_K}(s,a) - V_{\pi_K}(s)$, $\forall s, a$ for the current policy, and then update the policy to obtain $\pi_{K+1}(a|s)$. On the other hand, sampling with respect to d^{π_K} results in an on-policy algorithm, which is expensive. We provide both on- and off-policy implementations of CPP in the following sections, but in principle off-policy learning algorithms can be applied to estimate d^{π_K} by exploiting techniques such as importance sampling (IS) ratio [Precup, 2000].

5.4.5 Leveraging Policy Interpolation

Computing ζ in Eq. (5.18) involves the horizon constant $C_\gamma := \frac{(1-\gamma)^3}{2\gamma}$ and policy difference bound constant C_K. The horizon constant is effective in DP scenarios

where the total number of timesteps is typically small, but might not be suitable for learning with deep networks that feature large number of timesteps: a vanishingly small C_γ will significantly hinder learning, hence it should be removed in deep RL implementations. We detail this consideration in Section 5.4.7.

The updated policy π_{K+1} in Eq. (5.25) cannot be directly deployed since it has not been verified to improve upon π_K. We interpolate between π_{K+1} and π_K with coefficient ζ such that the resultant policy $\tilde{\pi}_{K+1}$ by finding the maximizer of a negative quadratic function in ζ. The maximizer ζ^* optimizes the lowerbound $\Delta J^{\tilde{\pi}_{K+1}}_{\pi_K, d^{\tilde{\pi}_{K+1}}}$. Here, ζ is optimally tuned and dynamically changing in every update. It reflects the *cautiousness* against policy oscillation, i.e., how much we trust the updated policy π_{K+1}. Generally, at the early stage of learning, ζ should be small in order to explore conservatively.

However, a major concern is that Lemma 5.4.2 holds only for Boltzmann policies, while the interpolated policies are generally no longer Boltzmann. In practice, we have two options for handling this problem:

1. we use the interpolated policy only for collecting samples (i.e. behavior policy) but not for computing next policy;

2. we perform an additional projection step to project the interpolated policy back to the Boltzmann class as the next policy.

The first solution might be suitable for relatively simple problems where the safe exploration is required: the behavior policy is conservative in exploring when $\zeta \approx 0$. But learning can still proceed even with such small ζ. Hence this scheme suits problems where interaction with the environment is crucial but progress is desired. On the other hand, the second scheme is more natural since the off-policyness caused by the mismatch between the behavior and learning policy might be compounded by high dimensionality. The increased mismatch might be perturbing to performance. In the following section, we introduce CPP using linear function approximation for the first scheme and deep CPP for the second scheme.

For the second scheme, manipulating the interpolated policy is inconvenient since we will have to remember all previous weights and more importantly, the theoretical properties of Boltzmann policies do not hold any longer. To solve

this issue, heuristically an information projection step is performed for every interpolated policy to obtain a Boltzmann policy. In practice, this policy is found by solving $\min_\pi D_{KL}(\pi \| \zeta \bar{\pi}_{K+1} + (1-\zeta)\pi_K)$. Though the information projection step can only approximately guarantee that the CVI bound continues to apply since the replay buffer capacity is finite, it has been commonly used in practice [Haarnoja et al., 2018, Vieillard et al., 2020b]. In our implementation of deep CPP, the projection problem is solved efficiently using autodifferentiation (Line 7 of Algorithm 6).

5.4.6 Approximate Interpolation Coefficient

The lowerbound of policy improvement depends on $A^{\pi_{K+1}}_{\pi_K, d^{\pi_K}}$. Though it is general difficult to compute exactly, very recently [Vieillard et al., 2020b] propose to estimate it using batch samples. We hence define several quantities following [Vieillard et al., 2020b]: let B_t denote a batch randomly sampled from the replay buffer B and define $\hat{A}_K(s) := \max_a Q(s,a) - V(s)$ as an estimate of $A^{\tilde{\pi}}_{\pi}(s)$, $\hat{\mathbb{A}}_K := \mathbb{E}_{s\sim B}[\hat{A}_K(s)]$ as an estimate of $A^{\tilde{\pi}}_{\pi, d^\pi}$, and $\hat{A}_{K,\min} := \min_{s\sim B} \hat{A}_K(s)$ as the minimum of the batch. The reward term $\frac{r_{max}}{1-\gamma}$ in CPI can be approximated by batch maximum $\hat{Q}_K := \max_{s,a,\cdots \in B} Q_K(s,a)$ which itself approximates the maximum norm $\|Q_{\pi_K}\|_\infty$. When we use linear function approximation with on-policy buffer B_K, we simply change the minibatch B in the above notations to the on-policy buffer B_K.

Given the notations defined above, we can compare the existing interpolation coefficients as the following:

- **CPI**: the classic CPI algorithm proposes to use the coefficient:

$$\zeta_{\text{CPI}} = \frac{(1-\gamma)\hat{\mathbb{A}}_K}{4r_{max}}, \tag{5.27}$$

  where r_{max} is the largest possible reward. When the knowledge of the largest reward is not available, approximation based on batches or buffer will have to be employed.

- **Exact SPI**: SPI proposes to extend CPI by using the following coefficient:

$$\zeta_{\text{E-SPI}} = \frac{(1-\gamma)^2\hat{\mathbb{A}}_K}{\gamma\delta\Delta A^{\pi_{K+1}}_{\pi_K}}, \tag{5.28}$$

where $\delta, \Delta A_{\pi_K}^{\pi_{K+1}}$ were specified in Lemma 5.4.1. When $\delta, \Delta A_{\pi_K}^{\pi_{K+1}}$ cannot be exactly computed, sample-based approximation will have to employed.

- **Approximate SPI**: as suggested by [Pirotta et al., 2013b, Remark 1], approximate ζ can be derived if we naïvely leverage $\delta \Delta A_{\pi_K}^{\pi_{K+1}} < \frac{4}{1-\gamma}$:

$$\zeta_{\text{A-SPI}} = \frac{(1-\gamma)^3 \hat{\mathbb{A}}_K}{4\gamma}. \tag{5.29}$$

- **Linear CPP**: if policies are entropy-regularized as indicated in Eq. (5.3), we can upper bound $\delta \Delta A_{\pi_K}^{\pi_{K+1}}$ by using Lemma 5.4.2:

$$\zeta_{\text{CPP}} = \frac{(1-\gamma)^3 \hat{\mathbb{A}}_K}{8\gamma C_K}. \tag{5.30}$$

  By the definition of C_K in Eq. (5.13), ζ_{CPP} can take on a wider range of values than $\zeta_{\text{A-SPI}}$.

- **Deep CPI**: for better working with deep networks, the following adaptive coefficient was proposed in deep CPI (DCPI) [Vieillard et al., 2020b]:

$$\zeta_{\text{DCPI}} = \hat{\zeta}_0 \frac{m_K}{M_K}, \quad \begin{cases} m_K = \rho_1 m_{K-1} + (1-\rho_1)\hat{\mathbb{A}}_K \\ M_K = \max(\rho_2 M_{K-1}, \hat{Q}_K), \end{cases} \tag{5.31}$$

  where $\rho_1, \rho_2 \in (0,1)$ are learning rates, and $\hat{\zeta}_0 = \frac{1}{4}$ same with CPI [Kakade and Langford, 2002].

- **Deep CPP**: we follow the DCPI coefficient design for making ζ_{CPP} suitable for deep RL. Specifically, we modify DCPP by defining $\hat{\zeta}_0 = \frac{1}{C_K}$:

$$\zeta_{\text{DCPP}} = \texttt{clip}\left\{\frac{1}{C_K}\frac{m_K}{M_K},\ 0,\ 1\right\}, \tag{5.32}$$

  where m_K, M_K are same as Eq. (5.31).

Based on Eqs. (5.30), (5.32), we detail the linear and deep implementations of CPP in the next section.

Algorithm 5: Linear Cautious Policy Programming

Require: α, β, γ CPP parameters, I the total number of iterations, T the number of steps for each iteration

1: Initialize $\theta, \tilde{\pi}_0$ at random, empty on-policy buffer $B_K = \{\}$;
2: **while** $K \leq I$ **do**
3: **while** $t \leq T$ **do**
4: Interact with the environment using policy π_ϵ;
5: Collect a transition tuple (s, a, r, s') into buffer $\mathcal{B}$;
6: Interact using policy $\tilde{\pi}_{K-1}$;
7: Collect $(s_t^K, a_t^K, r_t^K, s_{t+1}^K)$ into buffer B_K ;
8: **if** K mod $C == 0$ **then**
9: Compute basis matrix Φ_K using B_K;
 update θ by Eq. (5.33);
10: Compute $\hat{\zeta}_0 = \frac{1}{C_K}$ and $\hat{\zeta} = \hat{\zeta}_0 \frac{m_K}{M_K}$ using Eq. (5.31);
11: Empty on-policy buffer B_K;
12: **end if**
13: **end while**
14: **end while**

5.4.7 Approximate CPP Implementation

We introduce the linear implementation of CPP following [Azar et al., 2012, Lagoudakis and Parr, 2003] and deep CPP inspired by [Vieillard et al., 2020b] in Algs. 5 and 6, respectively. It is worth noting that in linear CPP we assume the interpolated policy $\tilde{\pi}$ is used only for collecting samples (line 5 of Alg. 5) hence no projection is necessary as it does not interfere with computing next policy.

- **Linear CPP**. We adopt linear function approximation (LFA) to approximate the Q-function by $Q(s, a) = \phi(s, a)^T \theta$, where $\phi(x) = [\varphi_1(x), \ldots, \varphi_M(x)]^T, x = [s, a]^T$, $\varphi(x)$ is the basis function and θ corresponds to the weight vector. One typical choice of basis function is the radial basis function:

$$\varphi_i(x) = \exp\big(-\frac{\|x - c_i\|^2}{\sigma^2}\big),$$

where c_i is the center and σ is the width. We construct basis matrix $\Phi =$

Algorithm 6: Deep Cautious Policy Programming

Require: α, β, γ CPP parameters, T the total number of steps, F the interaction period, C the update period

1: Initialize θ at random, set $\theta^- = \theta$, $K = 0$ and buffer B to be empty;
2: **while** $K \leq T$ **do**
3: Interact with the environment using policy π_ϵ;
4: Collect a transition tuple (s, a, r, s') into buffer $\mathcal{B}$;
5: **if** K mod $F == 0$ **then**
6: Sample a minibatch B_t and compute $\mathcal{L}_{value}$ and $\mathcal{L}_{policy}$ using Eqs. (5.34), (5.35);
7: Do one step of gradient descent on the loss $\mathcal{L}_{train} = \mathcal{L}_{value} + \mathcal{L}_{policy}$; compute $\hat{A}_K, \hat{\mathbb{A}}_K$ and moving average m_K, M_K using Eq. (5.31);
8: **end if**
9: **if** K mod $C == 0$ **then**
10: $\theta^- \leftarrow \theta$;
11: Compute $\hat{\zeta}_0 = \frac{1}{C_K}$ and $\zeta_{\text{CPP}} = \hat{\zeta}_0 \frac{m_K}{M_K}$ using Eq. (5.31);
12: **end if**
13: **end while**

$[\phi_1(x_1), \ldots, \phi_M(x_N)] \in \mathbb{R}^{T \times M}$, where T is the number of timesteps. Specifically, at K-th iteration, we maintain an on-policy buffer B_K. For every timestep $t \in [1, T]$, we collect $(s_t^K, a_t^K, r_t^K, s_{t+1}^K)$ into the buffer and compute the basis matrix at the end of every iteration.

To obtain the best-fit θ_{K+1} for the $K+1$-th iteration, we solve the least-squares problem $\|T_{\pi_{K+1}} Q_{\pi_K} - \Phi\theta_K\|^2$:

$$\theta_{K+1} = \left(\Phi^T\Phi + \varepsilon I\right)^{-1} \Phi^T T_{\pi_{K+1}} Q_{\pi_K}, \tag{5.33}$$

where ε is a small constant preventing singular matrix inversion and $T_{\pi_{K+1}} Q_{\pi_K}$ is the empirical Bellman operator defined by

$$T_{\pi_{K+1}} Q_{\pi_K}(s_t^K, a_t^K) := r(s_t^K, a_t^K) + \gamma \sum_a \pi_{K+1}(a|s_{t+1}^K) Q_K(s_{t+1}^K, a).$$

Since the buffer is on-policy, we empty it at the end of every iteration (line 11).

- **Deep CPP.** Though CPP is an on-policy algorithm, by following [Vieillard et al., 2020b] off-policy data can also be leveraged with the hope that random sampling from the replay buffer covers areas likely to be visited by the policy in the long term. Off-policy learning greatly expands CPP's coverage, since on-policy algorithms require expensively large number of samples to converge, while off-policy algorithms are more competitive in terms of sample complexity in deep RL scenarios.

We implement CPP based on the DQN architecture, where the Q-function is parameterized as Q_θ, where θ denotes the weights of an online network, as can be seen from Line 2. Line 3 begins the learning loop. For every step we interact with the environment using policy π_ϵ, where ϵ denotes the epsilon-greedy policy threshold. As a result, a tuple of experience is collected to the buffer.

Line 6 of Alg. 6 begins the update loop. We sample a minibatch from the buffer and compute the loss $\mathcal{L}_{value}, \mathcal{L}_{policy}$ defined in Eqs. (5.34), (5.35), respectively. Since our implementation is based on DQN, we do not include additional policy network as done in [Vieillard et al., 2020b]. Instead, we denote the policy as π_θ to indicate that the policy is a function of Q_θ as shown in Eq. (5.36). The base policy is hence denoted by π_θ^- to indicate it is computed by the target network of θ^-. We define the regression target as:

$$\hat{Q}(s_t, a_t, r_t, s_{t+1}) = r_t + \gamma \sum_{a \in \mathcal{A}} \pi_\theta(a|s_{t+1}) \big(Q(s_{t+1}, a) + \tau \log \pi_\theta(a|s_{t+1}) + \sigma \log \frac{\pi_\theta(a|s_{t+1})}{\pi_\theta^-(a|s_{t+1})} \big).$$

Hence, the loss for θ is defined by:

$$\mathcal{L}_{value}(\theta) = \mathbb{E}_{(s_t, a_t, \dots) \sim B} \left[\left(Q_\theta(s_t, a_t) - \hat{Q}(s_t, a_t, r_t, s_{t+1}) \right)^2 \right]. \tag{5.34}$$

It should be noted that the interpolated policy cannot be directly used as it is generally no longer Boltzmann. To tackle this problem, we further incorporate the following minimization problem to project the interpolated policy back to the Boltzmann policy class:

$$\mathcal{L}_{policy}(\theta) = \mathbb{E}_{(s_t, a_t, \dots) \sim B} \left[D_{KL} \left(\pi_\theta(a_t|s_t) \;\middle\|\; \zeta \mathcal{G} Q_\theta + (1 - \zeta) \pi_\theta^-(a_t|s_t) \right) \right], \tag{5.35}$$

where $\mathcal{G}Q_\theta$ takes the maximizer of the action value function. The reason why we can express the policy π and $\mathcal{G}Q_\theta$ with the subscript θ is because the policy is a

function of action value function, which has a closed-form solution (see [Kozuno et al., 2019] for details):

$$\mathcal{G}Q_\theta(a|s) = \frac{\pi_\theta^-(a|s)^\alpha \exp\left(\beta Q_\theta(s,a)\right)}{\sum_{a' \in \mathcal{A}} \pi_\theta^-(a'|s)^\alpha \exp\left(\beta Q_\theta(s,a')\right)}, \tag{5.36}$$

which by simple induction can be written completely in terms of Q_θ as $\mathcal{G}Q_\theta(a|s) \propto \exp\left(\sum_{j=0} Q_{\theta_j}(s,a)\right)$ [Vieillard et al., 2020a]. Line 8 performs one step of gradient descent on the the compound loss and line 9 computes the approximate expected advantage function for computing ζ.

There is one subtlety in that the definition of K is unclear in the deep RL context: there is no clear notion of *iteration*. If we naïvely define K as the the number of steps or the number of updates, then by definition C_K in Eq. (5.32) could quickly converge to 0 or explode, rendering CPP losing the ability of controlling update. Hence in our implementation, we increment K by one every time we update the target network (every C steps), which results in a suitable magnitude of K.

5.4.8 Experimental Results

The proposed CPP algorithm can be applied to a variety of entropy-regularized algorithms. In this section, we utilize conservative value iteration (CVI) as the base algorithm in [Kozuno et al., 2019] for our experiments. In our implementation, for the $K+1$-th update, the baseline policy $\bar{\pi}$ in Eq. (5.3) is π_K.

For didactic purposes, we first examine all algorithms (specified below) in a safety gridworld and the classic control problem pendulum swing-up. The tabular gridworld allows for exact computation to inspect the effect of algorithms. On the other hand, pendulum swing-up leverages linear function approximation detailed in Alg. 5. We then apply the algorithms on a set of Atari games to demonstrate the effectiveness of our proposed method. It is worth noting that even state-of-the-art monotonic improving methods failed in complicated Atari games [Papini et al., 2020]. The gridworld, pendulum swing-up and Atari games manifest the growth of complexity and allow for comparison on how the algorithms trade off stability and scalability.

For the gridworld and pendulum experiments, we compare Linear CPP using coefficient Eq. (5.30) against *safe policy iteration* (SPI) [Pirotta et al., 2013b]

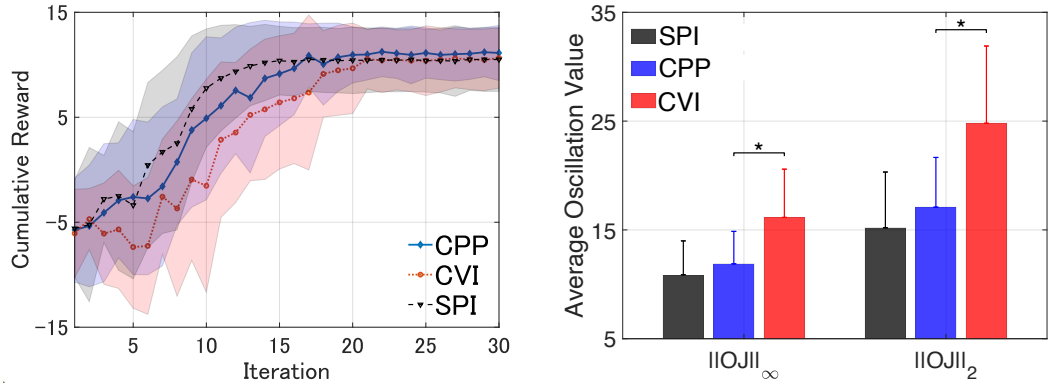
which is the closest to our work. We employ Exact-SPI (E-SPI) coefficient in Eq. (5.28) on the gridworld since in small state spaces where the quantities $\delta, \Delta A_{\pi_K}^{\pi_{K+1}}$ can be accurately estimated. As a result, SPI performance should upper bound that of CPP since CPP was derived by further loosening on SPI. For problems with larger state-action spaces, SPI performance may become poor as a result of insufficient samples for estimating those quantities, hence Approximate-SPI (A-SPI) Eq. (5.29) should be used. However, leveraging A-SPI coefficient often results in vanishingly small ζ values.

For Atari games, we compare Deep CPP leveraging Eq. (5.32) against on- and off-policy state-of-the-art algorithms, see Section 5.4.8 for a detailed list. Specifically, we implement deep CPP using off-policy data to show it is capable of leveraging off-policy samples, hence greatly expanding its coverage since on-policy algorithms typically have expensive sample requirement.

Gridworld with Danger

Experimental Setup. The agent in the 5×5 grid world starts from a fixed position at the upper left corner and can move to any of its neighboring states with success probability p or to a random different direction with probability $1 - p$. Its objective is to travel to a fixed destination located at the lower right corner and receives a $+1$ reward upon arrival. Stepping into two danger grids located at the center of the gridworld incurs a cost of -1. Every step costs -0.1. We maintain tables for value functions to inspect the case when there is no approximation error. Parameters are tuned to yield empirically best performance. For testing the *sample efficiency*, every iteration terminates after 20 steps or upon reaching the goal, and only 30 iterations are allowed for training. For statistical significance, the results are averaged over 100 independent trials.

Results. Figure 5.1(a) shows the performance of SPI, CPP, and CVI, respectively. Recall that SPI used the *exact* coefficient Eq. (5.28). The black, blue, and red lines indicate their respective cumulative reward (y-axis) along the number of iterations (x-axis). The shaded area shows ± 1 standard deviation. CVI learned policies that visited danger regions more often and result in delayed convergence compared to CPP. Figure 5.1(b) compares the average policy oscillation defined in Eq. (5.37).



(a) (b)

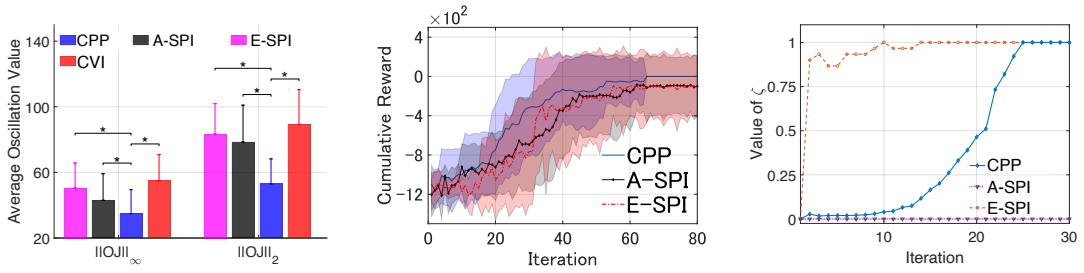
Figure 5.1: Comparison between SPI, CPP, and CVI on the safety grid world. The black line shows the mean SPI cumulative reward, the blue line CPP, and the red line CVI in Figure 5.1(a), with the shaded area indicating ± 1 standard deviation. Figure 5.1(b) compares the respective policy oscillation value defined in Eq. (5.37).

The slightly worse oscillation value of CPP than SPI with $\zeta_{\text{E-SPI}}$ is expected as CPP exploited a lower bound that is looser than that of SPI. However, as will be shown in the following examples when both linear and nonlinear function approximation are adopted, SPI failed to learn meaningful behaviors due to the inability to accurately estimate the complicated lower bound.

Pendulum Swing Up

Since the state space is continuous in the pendulum swing up, E-SPI can no longer expect to accurately estimate $\delta \Delta A_{\pi_K}^{\pi_{K+1}}$, so we employ A-SPI in Eq. (5.29) and compare both E-SPI and A-SPI against Linear CPP Eq. (5.30).

Experimental Setup. A pendulum of length 1.5 meters has a ball of mass 1kg at its end starting from the fixed initial state $[0, -\pi]$. The pendulum attempts to reach the goal $[0, \pi]$ and stay there for as long as possible. The state space is two-dimensional $s = [\theta, \dot{\theta}]$, where θ denotes the vertical angle and $\dot{\theta}$ the angular velocity. Action is one-dimensional torque $[-2, 0, 2]$ applied to the pendulum. The reward is the negative addition of two quadratic functions quadratic in angle



(a) policy oscillation value. (b) cumulative reward. (c) ζ values.

Figure 5.2: Comparison of SPI, CPP, and CVI on the pendulum swing up task. 5.2(a) illustrates the policy oscillation value defined in Eq. (5.37). 5.2(b) shows the cumulative reward with ± 1 standard deviation. 5.2(c) shows the ζ values.

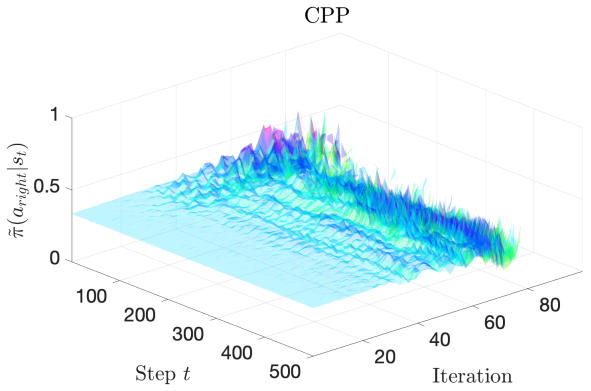
and angular velocity, respectively:

$$R = -\frac{1}{z}(a\theta^2 - b\dot{\theta}^2),$$

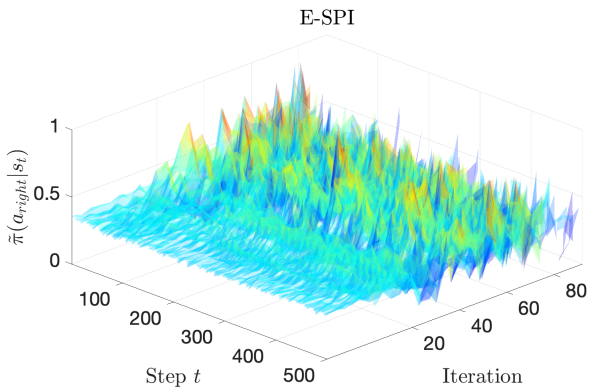
where $\frac{1}{z}$ normalizes the rewards and a large b penalizes high angular velocity. We set $z = 10, a = 1, b = 0.01$. To demonstrate that the proposed algorithm can approximately ensure monotonic improvement even with a small number of samples, we allow 80 iterations of learning; each iteration comprises 500 steps. For statistical evidence, all figures show results averaged over 100 independent experiments.

Results. We compare CPP with CVI and both E-SPI and A-SPI in Figure 5.2. In this simple setup, all algorithms showed similar trend. But CPP managed to converge to the optimal solution in all seeds, as can be seen from the variance plot. On the other hand, both SPI versions exhibited lower mean scores and large variance, which indicate that for many seeds they failed to learn the optimal policy. In Figure 5.2(a), both E-SPI and CVI exhibited wild oscillations, resulting in large average oscillaton values, in which the oscillation criterion is defined as:

$$\begin{aligned} &\forall K, \text{ s.t. } R_{K+1} - R_K < 0, \\ &\|\mathcal{O}J\|_\infty = \max_K |R_{K+1} - R_K|, \\ &\|\mathcal{O}J\|_2 = \sqrt{\Big(\sum_K (R_{K+1} - R_K)^2\Big)}, \end{aligned} \tag{5.37}$$



(a)



(b)

Figure 5.3: CPP and E-SPI policies $\tilde{\pi}(a_{right}|s_t)$ (z-axis).

where R_{K+1} refers to the cumulative reward at the $K+1$-th iteration. It is worth noting that the difference $R_{K+1} - R_K$ is obtained by $\tilde{\pi}_{K+1}, \tilde{\pi}_K$, which is the lower bound of that by $\tilde{\pi}_{K+1}, \pi_K$. Intuitively, $\|\mathcal{O}J\|_\infty$ and $\|\mathcal{O}J\|_2$ measure *maximum* and *average* oscillation in cumulative reward. The stars between CPP and CVI represent statistical significance at level $p = 0.05$.

The reason for SPI's drastic behavior can be observed in Figure 5.2(c) (truncated to 30 iterations for better view); in E-SPI, insufficient samples led to very large ζ. The aggressive choice of ζ led to a large oscillation value. On the other hand, A-SPI went to the other extreme of producing vanishingly small ζ due to the loose choice of ζ for ensuring improvement of $\Delta J^{\pi'}_{\pi,d^{\pi'}} \geq \frac{(1-\gamma)^3 (A^{\tilde{\pi}}_{\pi})^2}{8\gamma}$, as can

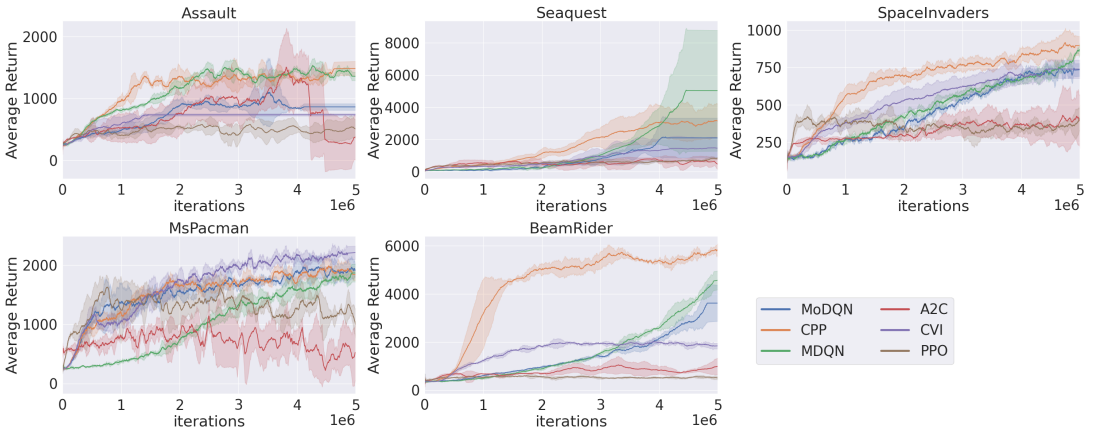


Figure 5.4: Comparison on Atari games averaged over 3 random seeds. CPP, MoDQN, MDQN and CVI are implemented as variants of DQN and hence are off-policy. PPO and A2C are on-policy. Correspondence between algorithms and colors is shown in the lower right corner. Overall, CPP achieved the best balance between final scores, learning speed and oscillation values.

be seen from the almost horizontal lines in the same figure; A-SPI had average value $\Delta J^{\tilde{\pi}_{K+1}}_{\pi_K, d^{\tilde{\pi}_{K+1}}} = 2.39 \times 10^{-9}$ and $\zeta = 1.69 \times 10^{-6}$. CPP converged with much lower oscillation thanks to the smooth growth of the ζ values; CPP was cautious in the beginning ($\zeta \approx 0$) and gradually became confident in the updates when it was close to the optimal policy ($\zeta \approx 1$).

However, it might happen that ζ values are large but probability changes are actually small and vice versa. To certify CPP did not produce such pathological mixture policy and indeed cautiously learned, we plot in Figure 5.3 the interpolated policies of CPP and E-SPI yielding action probability of the pendulum swinging right $\tilde{\pi}(a_{right}|s_t)$. The probability change is plotted in z-axis, timesteps $t = 1, \ldots, 500$ of all iterations are drawn on x, y axes. For both cases, $\tilde{\pi}(a_{right}|s) \approx 0.33$ which is uniform at the beginning of learning. However, E-SPI policy $\tilde{\pi}(a_{right}|s)$ gradually peaked from around 10th iteration, which led to very aggressive behavior policy. Such aggressive behavior was consistent with the overly large ζ values shown in Figure 5.2(c). On the other hand, CPP policy $\tilde{\pi}(a_{right}|s)$ was more tempered and showed a gradual change conforming to its ζ change. The probability plots together with ζ values in Figure 5.2(c) indicate

that the CPP interpolation was indeed effective in producing non-trivial diverse mixture policies.

Atari Games

Experimental Setup. We applied the algorithms to a set of challenging Atari games: `MsPacmann`, `SpaceInvaders`, `Beamrider`, `Assault` and `Seaquest` [Bellemare et al., 2013] using the adaptive ζ introduced in Eq. (5.32). We compare deep CPP with both on- and off-policy algorithms to demonstrate that CPP is capable of achieving superior balance between learning speed and oscillation values.

For on-policy algorithms, we include the celebrated proximal policy gradient (PPO) [Schulman et al., 2017], a representative trust-region method. We also compare with Advantage Actor-Critic (A2C) [Mnih et al., 2016] which is a standard on-policy actor-critic algorithm: our intention is to confirm the expensive sample requirement of on-policy algorithms typically render them underperformant when the number of timesteps is not sufficiently large.

For the off-policy algorithms, we decide to include several state-of-the-art DQN variants: Munchausen DQN (MDQN) [Vieillard et al., 2020c] features the implicit KL regularization brought by the Munchausen log-policy term: it was shown that MDQN was the only non-distributional RL method outperforming distributional ones. We also include another state-of-the-art variant: Momentum DQN (MoDQN) [Vieillard et al., 2020d] that avoids estimating the intractable base policy in KL-regularized RL by constructing momentum. MoDQN has been shown to obtain superior performance on a wide range of Atari games. Finally, as an ablation study, we are interested in the case $\zeta = 1$, which translates to conservative value iteration (CVI) [Kozuno et al., 2019] based on the framework Eq. (5.3). CVI has not seen deep RL implementation to the best of our knowledge. Hence a performant deep CVI implementation is of independent interest.

All algorithms are implemented using library Stable Baselines 3 [Raffin et al., 2021], and tuned using the library Optuna [Akiba et al., 2019]. Further, all on- and off-policy algorithms share the same network architectures for their group (i.e. MDQN and CPP share the same architecture and PPO and A2C share another same architecture) for fair comparison. The experiments are evaluated

Criterion	Algorithm	Assault	Seaquest	SpaceInvaders	MsPacman	BeamRider
$\|\mathcal{O}J\|_2$	CPP	151	622	89	249	460
	MDQN	129	2149	77	202	220
	MoDQN	162	813	91	288	718
	CVI	77	449	83	292	220
	PPO	74	68	72	280	74
	A2C	218	98	48	395	87
$\|\mathcal{O}J\|_\infty$	CPP	59	561	42	26	292
	MDQN	51	2141	16	52	149
	MoDQN	111	716	36	124	665
	CVI	6	361	51	98	105
	PPO	16	9	7	36	33
	A2C	52	15	8	249	34

Table 5.1: The oscillation values of algorithms measured in $\|\mathcal{O}J\|_2$ and $\|\mathcal{O}J\|_\infty$ defined by Eq. (5.37). CPP achieved the best balance between final score, learning speed and oscillation values. Note that CPP was implemented to leverage off-policy data. Algorithms of small oscillation values, such as PPO, failed to compete with CPP in terms of final scores and convergence speed.

over 3 random seeds. We expect that on simple tasks PPO and A2C might be stable due to the on-policy nature, but too slow to learn meaningful behaviors. However, PPO is known to take drastic updates and heavily needs code-level optimization to correct the drasticity [Engstrom et al., 2019]. On the other hand, for complicated tasks, too drastic policy updates might be corrupted by noises and errors, leading to divergent learning. By contrast, CPP should balance between learning speed and oscillation value, leading to gradual but smooth improvement.

Results: Final Scores. As is visible from Figure 5.4, Deep CPP achieved either the first or second place in terms of final scores on all environments, with the only competitive algorithm being MDQN which is the state-of-the-art DQN variant, and occasionally CVI which is the case of $\zeta = 1$. However, MDQN suffered from numerical stability on the environment `Seaquest` as can be seen from the flat line at the end of learning.

CVI performed well on the simple environment `MsPacman`, which can be interpreted as that learning on simple environments is not likely to oscillate, and

hence the policy regularization imposed by ζ is not really necessary, setting $\zeta = 1$ is the best approach for obtaining high return. However, in general it is better to have adjustable update: on the environment `BeamRider` the benefit of adjusting the degree of updates was significant: CPP learning curve quickly rised at the beginning of learning, showing a significant large gap with all other algorithms. Further, while CVI occasionally performed well, it suffered also from numerical stability: on the environment `Assault`, CVI and MoDQN achieved around 1000 final scores but ran into numerical issues as visible from the end of learning. This problem has been pointed out in [Vieillard et al., 2020d].

On the other hand, on all environments on-policy algorithms A2C and PPO failed to learn meaningful behaviors. On some environment such as Assault A2C showed divergent learning behavior at around 4×10^6 and PPO did not learn meaningful behavior until the end. This observation suggests that the sample complexity of on-policy algorithms is high and generally not favorable compared to off-policy algorithms.

Results: Oscillation. The averaged oscillation values of all algorithms are listed in Table 5.1. While MDQN showed competitive performance against CPP, it exhibited wild oscillation on the difficult environment `Seaquest` [Fortunato et al., 2018] and finally ran into numerical issue as indicated by the flatline near the end. The oscillation value reached to around 2100. Since MDQN is the state-of-the-art regularized value iteration algorithm featuring implicit regularization, this result illustrates that on difficult environments, only reward regularization might not be sufficient to maintain stable learning. On the other hand, CPP achieved a balance between stable learning and small oscillation, with oscillation value around 600, attaining final score slightly lower than MDQN and higher than MoDQN and CVI.

The oscillation values and final scores should be combined together for evaluating how algorithms perform. CVI, MoDQN sometimes showed similar performance to CPP, but in general the final scores are lower than CPP, with higher oscillation values. On the other hand, MDQN showed competitive final scores, but sometimes it exhibited wild oscillation and ran into numerical issues, implying that on some environments where low oscillation is desired, CPP might be more desirable than MDQN. On-policy algorithms even showed low oscillation

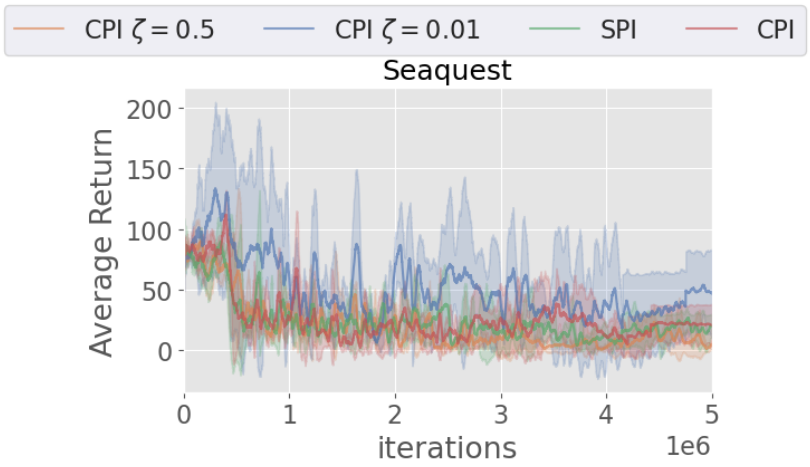


Figure 5.5: Learning curves of DCPI on `Seaquest` with four coefficient designs. All designs achieved the final score of 50, while CPP achieved around 3000 in Figure 5.4.

values, but their final scores are considered unacceptable.

Ablation Study

We are interested in comparing the performance of DCPI with CPP to see the role played by ζ_{DCPP}. It is also enlightening by inspecting the result of fixing ζ as a constant value. In this subsection, we perform ablation study by comparing the the following four designs:

- DCPI with fixed $\zeta = 0.01$: this is to inspect the result of constantly low interpolation coefficient.
- DCPI with fixed $\zeta = 0.5$: this is to examine the performance of equally weighting all policies.
- CPI: this uses the coefficient from Eq. (5.27).
- SPI: this uses the DCPI architecture, but we compute $\zeta_{\text{A-SPI}}$ by using Eq. (5.29).

We examine those four designs on the challenging environment `Seaquest`. Other experimental settings are held same with Sec. 5.4.8.

As can be seen from Figure 5.5, all designs showed a similar trend of converging to some sub-optimal policy. The final scores were around 50, which was significantly lower than CPP in Figure 5.4. This result is not surprising since for $\zeta = 0.01$, almost no update was performed. For $\zeta = 0.5$, the algorithm weights contribution of all policies equally without caring about their quality. On this environment, $\zeta_{\text{A-SPI}}$ is vanishingly small similar with that shown in Figure (5.2(c)). Lastly, for CPI the number of learning steps is not sufficient for learning meaningful behavior.

5.5 Discussion and Conclusion

This chapter developed a novel robust RL algorithm applicable to high dimensional problems with the help of Shannon entropy and KL divergence. Specifically, the intractable lower bound from prior work was simplified by exploiting the upper bound introduced by KL-regularized policy sequence to yield a slightly looser but readily computable new bound. By computing interpolation coefficient according to this new lower bound, we demonstrated that better tradeoff between performance of policy oscillation could be attained.

Though the proposed method improved upon the prior MI methods, it did not achieve state-of-the-art performance compared to other coefficient-free methods such as soft actor-critic (SAC) or Munchausen DQN. This suggests that the interpolation coefficient, despite being theoretically attractive, might hamper learning in practice due to the presence of a variety of errors which possibly shift the optimum of the lower bound and hence cause the mismatch between the coefficient and the bound. From another perspective, the inferior performance of MI methods can be viewed from the fact that the lower bound is not subject to any optimizaion, but is determined once the new policy is obtained. We could in principle, add another optimization procedure on top of the existing MI algorithm to explicitly maximize the lower bound and hence further improve the performance. This interesting direction is left to future work.

6 | Robustness: configuration

Reminder: This chapter presents entropy regularization for *robustness* which is defined as the insensitivity against configurations such as algorithmic hyperparameters or MDP design. The plant-wide chemical process control problem introduced in Chapter 3 is used as the testbed. The method leveraged is the monotonic improvement algorithm we developed in Chapter 5.

6.1 Introduction

As modern industrial process controllers necessitate high quality models and remedial model re-identification upon performance degradation that typically lasts for weeks [Kano and Ogawa, 2009], model-free RL is a potentially powerful tool for replacing the above-mentioned laborious procedures by adaptively learning control policies in a limited period of time [Spielberg et al., 2019]. However, the application of RL algorithms to such realistic time-limited problems is hindered by the need for repeated human-agent interactions such as searching for performant parameters. Brute-force search is only suitable for small-sized problems with few number of parameters. In practice, human-agent interaction accounts for the most time-consuming part of modeling [Winder, 2020].

It is not a surprise that conventional RL algorithms fail to deliver on the promise within a short, limited time for realistic applications: convergence results of RL typically state guarantees under the assumption that state-action pairs be visited infinitely often and the Markov Decision Process (MDP) parameters such

as state-action space or reward function perfectly characterize the underlying problem [Bertsekas and Tsitsiklis, 1996, Sutton and Barto, 2018]. This is at odds with requirements of realistic applications, e.g. fixed training time and partial knowledge of the model [Kano and Nakagawa, 2008, Spielberg et al., 2017]. In practice, a large portion of time is spent on identifying suitable parameters that include

1. MDP parameters such as state, action, reward function (an MDP is formally defined in Section 5.3).

2. Algorithmic parameters that characterizes the performance with a fixed learning horizon.

Identifying a performant set of parameters might be difficult as RL algorithms are sensitive to parameters: slight varying a parameter might cause huge difference in the learning performance [Henderson et al., 2017]. Generally, extensive human-agent interaction for finding performant parameters is required and it accounts for most of the time spent for an RL algorithm to work. This time-consuming interaction might be the bottleneck or even problematic since for complicated algorithms it takes long time to inspect the utility of one set of parameters [Yoo et al., 2021]. Those parameters determine the solvability of the modelled MDP and the quality of learned controller [Bertsekas and Tsitsiklis, 1996, Puterman, 1994]. The relationship between the final performance and parameters is complicated and needs extensive empirical validation. Therefore, given limited time, an algorithm that is robust to both MDP and algorithmic parameters is naturally desired, since the agent is promised to achieve adequate performance for a given task for a range of parameters rather than exhibiting good performance only for some specific choices, greatly saving human-agent interaction time.

The consideration above motivates us to introduce monotonic policy improvement algorithms into process control problems. Monotonic improvement (MI) refers to that one can ensure nonnegative improvement in terms of return of the new policy over the current one. By placing the problem under the monotonic improvement framework, it is expected that improving policies can still be obtained even with underperforming parameters and hence robustness against parameters could be achieved. In this paper we focus on a very recent algorithm of mono-

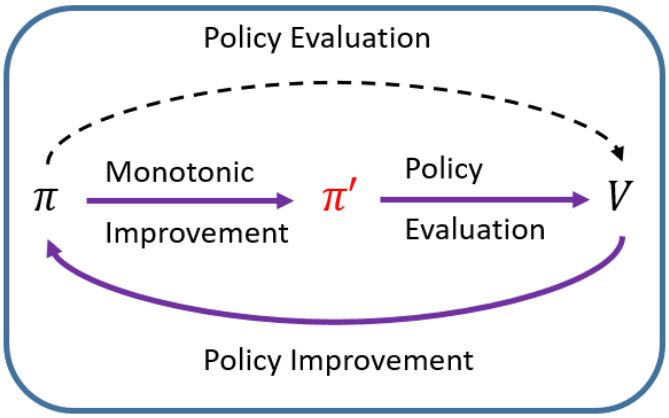
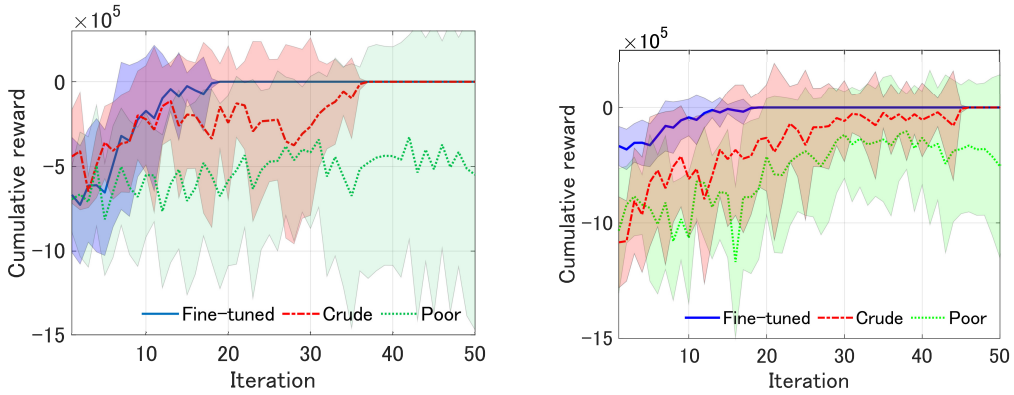


Figure 6.1: The illustration of monotonic improvement (MI) mechanism. The outer circle between policy π and value function V indicates standard policy iteration loop [Sutton and Barto, 2018]. In MI the policy evaluation step (dashed curve) is modified to be the MI step in the middle that leads to the monotonic improving policy π' and its evaluation.

tonic improvement algorithm family: cautious policy programming (CPP) [Zhu and Matsubara, 2020] that combines the theoretical guarantees of monotonic improvement and the scalability of entropy-regularized methods. CPP lays down the foundation for our proposal on enhancing robustness against parameters on large-scale process control problems. Specifically, the concept of monotonic improvement can be visualized as Fig. 6.1. The outer loop between a policy π and value function V corresponds to standard policy iteration that iteratively evaluates/improves the policy [Sutton and Barto, 2018]. In the monotonic improvement scheme, the evaluation step (upper dashed curve) is replaced with the MI step that leads to the monotonic improving policy π' and its evaluation.

Original CPP algorithm might not be applicable for large-scale process control problems that require simultaneous control of multiple PID controllers which constitute high dimensional action space. This might render any algorithm suffer from the curse of dimensionality [Bellman, 2003] if the high-dimensional state-action space issue is not properly handled. Inspired by the recent success on leveraging RL on process control [Zhu et al., 2020], we introduce the key ingredients for efficiently solving control problems with large state-action spaces to the



(a) Curves show the cumulative rewards of FFDPP [Zhu et al., 2020] with different action space design.

(b) Curves show the cumulative rewards of FFDPP with different combinations of algorithmic parameters.

Figure 6.2: Evaluation of FFDPP on the local control task introduced in Chapter 3 for different action space designs and parameters.

cautious learning mechanism: random features and factorial policy. We apply the resultant algorithm: factorial cautious policy programming (FCPP) as a specific instantiation of the proposed framework on the classic simulated Vinyl Acetate Monomer (VAM) manufacturing process. VAM process has realistic complexity and is representative of chemical processes [Luyben and Tyréus, 1998, Machida et al., 2016, Seki et al., 2010]. To emphasize the contribution of the cautious learning mechanism, we apply the algorithm Factorial Fastfood Dynamic Policy Programming (FFDPP) of [Zhu et al., 2020] with different parameters under a fixed horizon and plot the results in Fig. 6.2. It is clear that even slightly altering the parameters will significantly degrade the performance of FFDPP. On the other hand, by incorporating the cautious learning mechanism, FCPP is both scalable since it adopts factorial policy and random feature approximation, and robust to parameters, thanks to the introduction of cautious learning, as will be shown by extensive results.

The contribution of this chapter consists in introducing the concept of cautious learning into process control problems to reduce human-agent interaction time. The central idea is to ensure monotonically increasing improvement even with underperforming parameters such as MDP or algorithmic parameters. Though

the notion of monotonic improvement exists for two decades, it has seen few applications and has never been applied to practical domains such as process control (see Chapter 5 for survey in the RL side). The second contribution lies in that we proposed a novel algorithm as a specific instantiation of the cautious learning: FCPP, by augmenting the cautious mechanism with specific components such as factorial policy and random features that have been proved suitable for process control problems. We believe the robustness of FCPP to parameters renders it valuable for realistic RL applications where time is precious and the human-agent interaction phase needs to be shortened.

6.2 Robust RL and Process Control

6.2.1 RL for Process Control

RL for process control has a long history [Badgwell et al., 2018, Nian et al., 2020] and to the best of the authors' knowledge, the first attempt was made by Hoskins and Himmelblau [Hoskins and Himmelblau, 1992] that applied Q-learning to controlling a continuous-stirred-tank-reactor. Since then, there has been modestly growing interests in optimizing process performance using model-free RL algorithms, with applications ranging from electricity grid control [Ernst et al., 2005], dynamic power management [Lee and Wong, 2010, Liu et al., 2010] to laboratory plant control [Syafiie et al., 2007]. RL has also been studied by [Harp et al., 2000] to learn a profitable electricity pricing policy for electricity management. The main algorithm adopted in these works was tabular Q-learning [Watkins and Dayan, 1992], which is known to be slow to converge and not scalable.

Recently, with the success of deep RL (DRL), there has been a resurgence of interest in applying DRL methods to process control [Hubbs et al., 2020, Spielberg et al., 2017]. Kubosawa et al. [2018] used DQN to recover a malfunctioning component of the VAM process. Yoo et al. [2021] leveraged deep deterministic policy gradient (DDPG) for the optimal control of batch processes. However, DRL is notorious for its sample-inefficiency (typically requires more than 10^8 samples) and difficulty of parameter tuning [Henderson et al., 2017], due to run-

ning sensitive and expensive stochastic gradient descent on hundreds of thousands of parameters. For example, DDPG has been reported in [Haarnoja et al., 2018, Henderson et al., 2017] as highly sensitive to algorithmic parameters. This sample inefficiency is not limited to the above-mentioned off-policy deep RL methods: popular on-policy deep policy gradient methods that can also approximately ensure monotonic improvement such as Trust Region Policy Optimization (TRPO) [Schulman et al., 2015] and its improved version Proximal Policy Optimization (PPO) [Schulman et al., 2017] have even worse sample complexity due to their on-policy nature: all samples can be used only once and then discarded.

To take realistic factors such as learning horizon (sample complexity), robustness, expressiveness into consideration, linear function approximation (LFA) might be a preferable choice than DRL methods. Unlike deep RL that is often short of theoretical guarantees, the combination of LFA and dynamic programming methods has been well-studied over decades [Bertsekas, 2011, Bertsekas and Tsitsiklis, 1996], where strong theoretical underpinnings such as convergence rate or sample complexity are readily available [Antos et al., 2008, Lazaric et al., 2012]. Policies obtained via LFA can be sufficiently powerful for performing difficult tasks, such as playing the game of Tetris [Scherrer et al., 2015], achieving state-of-the-art performance on challenging continuous control problems [Mania et al., 2018], or realizing plant-wide control on the classic VAM process [Zhu et al., 2020]. Further, LFA methods tend to be relatively insensitive to algorithmic parameters [Mania et al., 2018] compared to deep RL methods.

In realistic process control applications, it is unlikely that excessive resources such as time are allowed for training the agent as assumed in theoretical studies [Bertsekas and Tsitsiklis, 1996, Sutton and Barto, 2018]. For example, RL is expected to replace the arduous human system re-identification process that typically costs 2 - 4 weeks [Spielberg et al., 2019], which means the allowed training time is at maximum 4 weeks. However, learning meaningful controllers within such short time is difficult: it is typical that trial-and-error human-agent interaction such as grid search is necessary for finding a performant set of parameters, including both MDP parameters such as state-action space design and algorithmic parameters. In the ideal case, optimal setup and parameters could be found by the virtue of unlimited resources. However, when the budget is limited, there

is an exploitation-exploration trade-off of whether to exploit the current configuration or to continue explore. The trade-off subsequently determines the quality of the learned controller. The above-mentioned scenario can be intuitively cast as that the agent is only allowed to learn for a fixed horizon and fixed number of iterations, and hence a trade-off of exploitation-exploration on parameter and setup must be wisely made.

6.2.2 Cautiousness in RL

Cautiousness in RL is actively studied under different names such as safe RL [García and Fernández, 2015] or monotonic policy improvement [Kakade, 2003, Pirotta et al., 2013b] with different focuses. As pointed out by [Nian et al., 2020], instability (lack of cautiousness) is one of the major shortcomings of RL for process control problems. Indeed, since RL learns from both good and bad experiences from trial-and-error interactions, in process control scenarios trial-and-error learning might incur unacceptable loss such as low quality product or even damaging the process components and causing emergency shutdown. This shortcoming is general to RL as the concept of danger or instability is not taken into account, which partly accounts for the reason why RL falls short of real-world applications as compared to supervised learning.

To tackle the cautiousness problem, one straightforward criterion is to enforce a safety constraint or budget that the agent is allowed the spend, this idea naturally leads to the constrained MDP (CMDP) setting [Achiam et al., 2017, Altman, 1999, Berkenkamp et al., 2017, Chow et al., 2018]. While CMDPs can be solved as shown by [Dogru et al., 2021, Tessler et al., 2019], those approaches are typically combined with deep learning implementations and hence the computational resources required are also high [Chow et al., 2018], in this paper we focus on low-cost linear function approximation. On the other hand, pursuing monotonic improvement requires less stringent assumptions and the goal is to approximately ensure the updated policy does not result in lower rewards than the pre-update one. However, these methods have been mostly evaluated in simple problems with at most two- or three-dimensional state space and one-dimensional action space due to the intense computational requirements on accurately estimating quantities such as total variation between two policies [Kakade, 2003, Pirotta

et al., 2013a,b]. In general, there is a large gap between theory and practice in monotonic improvement literature as admitted by the authors of [Papini et al., 2020]: even for a simple CartPole task, state-of-the-art monotonic improvement algorithm failed to compete with heuristic optimization technique such as Adam [Kingma and Ba, 2015]. In this paper, we also bridge this gap by applying monotonic improvement algorithm FCPP on large-scale control problems.

6.3 Factorial Cautious Policy Programming

The name *cautiousness* is stemmed from our use of scalable entropy-regularized policies within the monotonic improvement framework. It is worth noting that the linear mixture of policies performs a regularization in the stochastic space. This is combined with the entropy regularization detailed in Chapter 5, which was introduced as a regularization with the reward function. Hence, the two regularizations are from different aspects and can coexist.

Specifically, suppose the agent is endowed with a redundantly large state-action space, naturally it needs to see more state-action samples to build up the probability densities for meaningful control strategy. To effectively build up the policy with fixed number of samples (learning horizon), one might want to 'reinforce' the knowledge of experienced state-action pairs. This can be achieved by the fact that KL regularization averages over all past action values:

$$\begin{aligned}\pi_{k+1}(a|s) &\propto \pi_k(a|s)\exp\left(Q_k(s,a)\right) \propto \pi_{k-1}(a|s)\exp\left(Q_k(s,a)+Q_{k-1}(s,a)\right)\\ &\propto \cdots \propto \exp\left(\sum_{j=1}^{k} Q_j(s,a)\right).\end{aligned} \tag{6.1}$$

However, Eq. (6.1) weights the contribution from all state-action value functions equally, which is undesirable when some value functions have poor estimates. Linear interpolation introduced in Eq. (5.12) can come to rescue since it has an interpretation of *the momentum* of adaptively weighting the contribution of previous solution, which is also known as the Frank-Wolfe method [Vieillard et al., 2019], [Beck, 2017, Chapter 13]:

$$\tilde{\pi}_{k+1}(a|s) = \zeta_{k+1}\pi_{k+1}(a|s) + (1-\zeta_{k+1})\tilde{\pi}_k(a|s),$$

where we use $\tilde{\pi}_{k+1}$ to indicate that the new policy is a linear mixture of the updated policy π_{k+1} (output of function approximator) and the current policy $\tilde{\pi}_k$, which itself is the outcome of interpolation from last round. The subscript of ζ indicates that it should change dynamically along with the policy. We stick to this notation in the rest of the paper.

It is clear by induction that we can write the interpolated policy as:

$$\begin{aligned}\tilde{\pi}_{k+1} &= \zeta_{k+1}\pi_{k+1} + (1-\zeta_{k+1})\tilde{\pi}_k \\ &= \zeta_{k+1}\pi_{k+1} + (1-\zeta_{k+1})\zeta_k\pi_k + (1-\zeta_{k+1})(1-\zeta_k)\tilde{\pi}_{k-1} \\ &= \dots \\ &= \sum_{j=0}^{k} \zeta_{k-j+1}\pi_{k-j+1}\prod_{i=1}^{j}(1-\zeta_{k-i+2}).\end{aligned} \tag{6.2}$$

Within the entropy-regularized policy class, the linear interpolation provides an additional degree of freedom for weighting information from previous value functions. This is desirable as the value functions in early stages of learning tend to be inaccurate. Combining the above-mentioned ideas, we recall the following theorem from Chapter 5 for designing a cautious agent against.

Theorem 6.3.1 (Chapter 5)**.** *Provided that [1] The policy is interpolated as* $\tilde{\pi}_{k+1}=\zeta_{k+1}\pi_{k+1}+(1-\zeta_{k+1})\pi_k$*; [2]* $A^{\pi_{k+1}}_{\pi_k,d^{\pi_k}} \geq 0$*; [3]* ζ *is chosen properly. Then the policy improvement* $\Delta J^{\tilde{\pi}_{k+1}}_{\pi_k,d^{\tilde{\pi}_{k+1}}}$ *is guaranteed:*

$$\Delta J^{\tilde{\pi}_{k+1}}_{\pi_k,d^{\tilde{\pi}_{k+1}}} \geq \frac{(1-\gamma)^3(A^{\pi_{k+1}}_{\pi_k,d^{\pi_k}})^2}{16\gamma C_K}, \tag{6.3a}$$

$$\textit{with } \zeta_{k+1} = \min(1, \frac{(1-\gamma)^3 A^{\pi_{k+1}}_{\pi_k,d^{\pi_k}}}{2\gamma C_K}), \tag{6.3b}$$

$$A^{\pi_{k+1}}_{\pi_k,d^{\pi_k}} = \mathbb{E}^{d^{\pi_k}}\left[\sum_a \pi_{k+1}(a|s)A^{\pi_k}(s,a)\right], \tag{6.3c}$$

$$C_K = \beta\sum_{j=0}^{K-1}\alpha^k\gamma^{K-k-1}, \tag{6.3d}$$

where $\Delta J^{\tilde{\pi}_{k+1}}_{\pi_k,d^{\tilde{\pi}_{k+1}}}$ *is the improvement of* $\tilde{\pi}_{k+1}$ *over* π_k*, and* π_{k+1},π_k *are consecutive optimal entropy-regularized policies. If* $A^{\pi_{k+1}}_{\pi_k,d^{\pi_k}} < 0$*, it indicates the update is*

corrupted by noise or error, hence we implement a simple rejection method to retrain the policy.

It is worth noting that, unlike monotonic improvement methods introduced in Sec. 5.4.2, there is no term like D_{TV} that is difficult to estimate. The intuition behind is that the entropy-regularized policies have bounded distance with each other by the merit of KL divergence. It has been shown by [Kozuno et al., 2019, Proposition 3] that the maximum distance between consecutive policies is proportional to the maximum reward. Hence, all we need to estimate within the cautious learning framework is the term $A^{\pi_{k+1}}_{\pi_k, d^{\pi_k}}$.

When the state-action spaces are high dimensional, the term $A^{\pi_{k+1}}_{\pi_k, d^{\pi_k}}$ might be difficult to accurately estimate with limited number of samples. The factorial policy can be used to decompose high-dimensional state-action value functions to low-dimensional ones learned in a multi-agent framework. Hence, we propose to specify $\zeta^{(m)}$ for every factorial policy $\pi^{(m)}$, and every $\zeta^{(m)}$ is computed by solving Eq. (6.3b) independently:

$$\tilde{\pi}^{(m)}_{k+1} = \zeta^{(m)}_{k+1}\pi^{(m)}_{k+1} + (1-\zeta^{(m)}_{k+1})\pi^{(m)}_{k}, \tag{6.4}$$

$$\zeta^{(m)}_{k+1} = \min\left(1,\ (2\gamma C_K)^{-1}(1-\gamma)^3 A^{\pi^{(m)}_{k+1}}_{\pi^{(m)}_k, d^{\pi^{(m)}_k}}\right). \tag{6.5}$$

This strategy allows us to speed up estimation of $A^{\pi_{k+1}}_{\pi_k, d^{\pi_k}}$ and subsequently achieve cautious learning.

Same with Chapter 3, we leverage Fast-food approximation for computing the feature maps. Recall that the use of trigonometric functions and G reduce computation time to $\mathcal{O}(n \log d)$ and storage to $\mathcal{O}(n)$ [Le et al., 2013], where n denotes the number of samples and d the dimension of state and action $[s, a]$. We use $\hat{A}^{\pi^{(m)}}(\cdot) = \theta^T \hat{\boldsymbol{\phi}}(\cdot)$ to denote the m-th agent's advantage vector approximated by random features. The random feature map has a tunable parameter l that controls the sampling frequency and hence accuracy of the approximation. Increasing l results in exponentially more accurate approximation. We follow [Zhu et al., 2020] for setting l.

Denote by $\hat{\Phi}^{(m)}$ the feature matrix for the m-th sub-policy having $\hat{\boldsymbol{\phi}}^{(m)T}$ as rows and following Eq. (5.33), the factorial weight vectors can now be computed

Algorithm 7: Factorial Cautious Policy Programming

Input: $\alpha, \beta, \gamma, \kappa, K, T, M$

for $m = 1, \ldots, M$ **do**
  # `Initialization`
  initialize buffer $D^{(m)}$ and policy $\pi_0^{(m)}$;
end
for $k = 1, \ldots, K$ **do**
  # `Collecting samples`
  for $t = 1, \ldots, T$ **do**
    measure state s_t;
    for $m = 1, \ldots, M$ **do**
      sample a_t from $\pi^{(m)}(\cdot|s_t)$;
      observe s_{t+1}, r_t;
      collect (s_t, a_t, r_t, s_{t+1}) in $D_{1:k}^{(m)}$;
    end
  end
  # `Multi-agent update`
  for $m = 1, \ldots, M$ **do**
    compute $\hat{\Phi}^{(m)}$, $\hat{\boldsymbol{A}}^{\pi}$ from $D_{1:k}^{(m)}$;
    compute $\theta_{k+1}^{(m)}$ using $\hat{\Phi}^{(m)}$ and Eq. (6.6);
    compute $\widehat{A}^{\pi_{k+1}^{(m)}}_{\pi_k^{(m)}, d^{\pi_k^{(m)}}}$ using Eqs. (6.6), (6.7) ;
    compute $\zeta_{k+1}^{(m)}$ using Eq. (6.5);
    output policy $\tilde{\pi}_{k+1}^{(m)}$ using Eq. (6.4);
  end
end

as:

$$\theta^{(m)*} = (\hat{\Phi}^{(m)T}\hat{\Phi}^{(m)} + \kappa^2 I)^{-1}\hat{\Phi}^{(m)T}\hat{\boldsymbol{A}}^{\pi^{(m)}}, \tag{6.6}$$

where $(\cdot)^{(m)}$ denotes components for the m-th sub-policy.

6.3.1 Summary of the Algorithm

While in general accurately computing the stationary distribution as in Eq. (6.3c) is non-trivial [Wen et al., 2020], we only need an approximate quantity for comput-

ing the expected advantage. Hence, we compute the surrogate objective $\hat{A}^{\pi_{k+1}}_{\pi_k, d^{\pi_k}}$:

$$\hat{A}^{\pi_{k+1}}_{\pi_k, d^{\pi_k}} := \hat{\mathbb{E}}_{(s,a)\sim D_{1:k}} \left[\sum_a \pi_{k+1}(a|s) A^{\pi_k}(s,a) \right], \tag{6.7}$$

where $\hat{\mathbb{E}}_{(s,a)\sim D_{1:k}}$ denotes empirical expectation with respect to the data buffer $D_{1:k}$. This combined with the approximation Eq. (6.6) yields $\widehat{A}^{\pi^{(m)}_{k+1}}_{\pi^{(m)}_k, d^{\pi^{(m)}_k}}$ for every m, as shown in Line 19 of Algorithm 7. Note that π_{k+1} is evaluated by Eq. (6.1). The proposed algorithm is listed in Algorithm 7. FFDPP, which will be used as the baseline for comparison in the experiments, differs with FCPP only in the cautious learning framework: instead of computing ζ and expected advantage function, FFDPP only evaluates factorial Q function and the corresponding policy.

6.4 Experimental Results

We examine FCPP on the tasks introduced in Chapter 3. To validate the effectiveness of FCPP given limited learning horizon, we examine the algorithm with a wide range of algorithmic parameters α, β and a variety of MDP parameter $\mathcal{A}$ to inspect their respective insensitivity and final performance (recall from Chapter 5 that popular policy/value iteration algorithms such as Q-learning are covered by varying α, β).

The MDP parameter $\mathcal{A}$ worth a few comments: we focus on redundantly large action since insufficient actions render the corresponding optimal controller impotent of learning meaningful control strategy and is considered as a degenerate case. Efficient exploration in redundantly large action space is crucial and has not been widely studied before compared to state space design, whose study falls into the category of partially observable MDPs where rich literature exists [Krishnamurthy, 2016]. On the other hand, reward function completely specifies the goal and it necessitates prior knowledge on the objective to optimize.

To deal with redundant action space design, a few definitions are in order. In manufacturing processes the control variates are often continuous-valued, hence we focus on discretization of continuous action spaces. We denote the nonnegative value *range* as b and *resolution* as n and collect them into a tuple (b, n), which

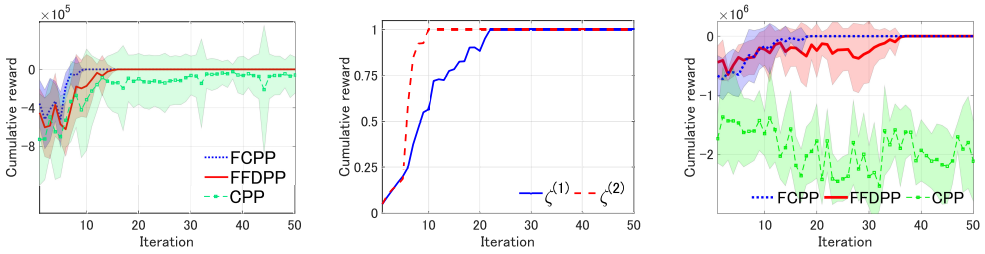
we call *redundant action design* or just *action design*, to refer to that we take n equidistant points from the interval $[-b, b]$. (b, n) is of primary interest when comparing the performance of FCPP and FFDPP. In general, different b induces different MDPs [Puterman, 1994], and larger n increases the sample complexity of the algorithm. For the ease of reading, we make use of the following definitions on the action design, which will be referred extensively in the experimental section:

- *fine-tuned*: refers to the action design is $(0.01, 5)$, same with FFDPP [Zhu et al., 2020].
- *crude*: refers to the action design $(15 \times 0.01, 2 \times 5)$.
- *poor*: refers to the action design is $(50 \times 0.01, 2 \times 5)$.
- *plant-wide crude*: refers to $(5 \times 0.01, 2 \times 5)$.

To investigate the utility of the proposed method, we pose the following questions regarding both action design and algorithmic parameters:

1. with algorithmic parameters α, β held fixed and relatively accurate action design, does FCPP perform similarly with FFDPP? This question is answered in Secs. 6.4.1, 6.4.2.
2. with algorithmic parameters α, β held fixed, is FCPP efficient given redundantly large action design (b, n) and limited learning horizon, i.e. we use large values of b, n rather than the fine-tuned values in [Cui et al., 2018, Zhu et al., 2020]? This question is addressed in Secs. 6.4.1, 6.4.2.
3. with action design held fixed, is FCPP efficient for a large range of parameters α, β given limited learning horizon? This question is addressed in Secs. 6.4.3, 6.4.4.

The answer to the first question ensures that given fine-tuned algorithmic parameters and action design, FCPP indeed performs similarly to FFDPP, hence achieving sample-efficient control. On the other hand, the second and third questions examine the utility of FCPP's robustness against one aspect of parameters given the other fixed. We answer the three questions in both local and plant-wide control tasks.



(a) Learning curves with fine-tuned action design.

(b) ζ values of FCPP with the fine-tuned action design.

(c) Learning curves with *crude* action design.

(d) ζ values of FCPP with *crude* action design.

(e) Learning curves with *poor* action design.

(f) ζ values of FCPP with *poor* action design.

Figure 6.3: FCPP, CPP and FFDPP on the local control task.

For the local control task, one experiment consists of 50 iterations each comprising 500 steps. For the plant-wide control task, one experiment consists of 70 iterations each comprising 1000 steps. All results are averaged over 10 independent experiments for statistical evidence.

6.4.1 Local Control - Action Design

We applied FCPP, CPP and FFDPP to the local control problem with linear function approximation and random features. The 5-dimensional state space is formed by [`VAMSensor`, `LevelSensor`, `FlowSensor`, `QualitySensor1-2`]^T, the 2-dimensional action space is formed by [`TempController`, `FlowController`]^T. The `VAMSensor` measures the production rate, `QualitySensors` measure the quality of VAM product. Other sensors are related to the process stability. The two controllers control the production rate and quality by adjusting the temperature and steam flow inside the process. The action design used in [Cui et al., 2018]

was *fine-tuned*.

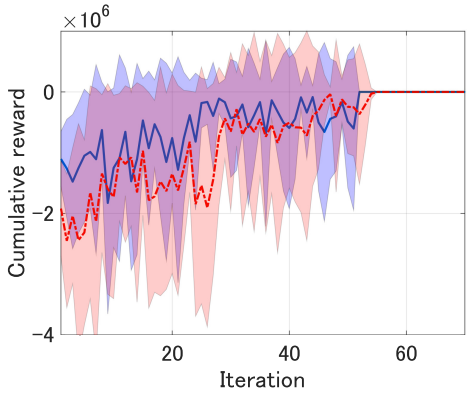
To examine FCPP with fine-tuned design as in the first question we posed, we ran FCPP, FFDPP and CPP with the fine-tuned action design following [Zhu et al., 2020]. The result is shown in Fig. 6.3(a). FCPP and FFDPP did not differ much and all quickly converged to the optimum thanks to the factorial policy. On the other hand, CPP curve did not fully converge to zero, indicating some independent trials failed to converge even with fine-tuned action design.

To see how the algorithms behave with a variety of action designs as in the second question we posed, we used the same parameters but different action designs. Figs. (6.3(c)), (6.3(e)) show the performance and Figs. (6.3(d)), (6.3(f)) show the ζ values of FCPP and FFDPP with *crude* and *poor* action designs, respectively. It is clear that the performance of FFDPP quickly deteriorates with the growth of range b. While FFDPP can still converge with *crude* action design, the sample complexity increased and hence it converged at around 40-th iteration. When the action design was *poor*, FFDPP failed to learn any meaningful behaviors. Lastly, the disastrous performance of CPP in Figs. (6.3(c)), (6.3(e)) was due to the overly large action space (2^{10} as described in Chapter 3, which caused emergency shutdown of the plant, leading to the lowest cumulative rewards in the two cases.

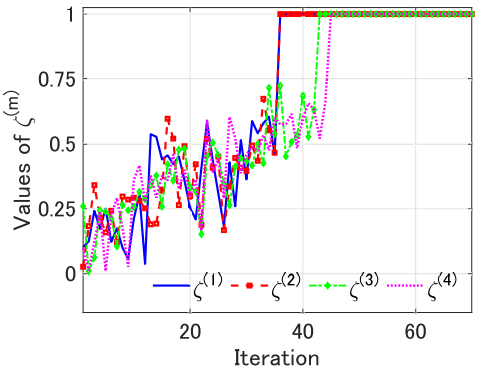
On the other hand, increasing both range and resolution did not harm the performance of FCPP: in both cases FCPP almost converged with almost the same rate. This is due to the cautious learning nature of FCPP. To be specific, at k-th iteration, FCPP maintains a linear class of policies spanned by π_{k+1}, π_k (condition [1] in Theorem (6.3.1)), with the next policy chosen by solving ζ, whose values are shown in Figs. (6.3(d)), (6.3(f)). The values of ζ matched the corresponding learning curves, which demonstrates that the robustness was successfully achieved by FCPP. By contrast, FFDPP only has access to a point estimate π_{k+1}, hence cannot effectively reuse information.

6.4.2 Plant-wide Control - Action Design

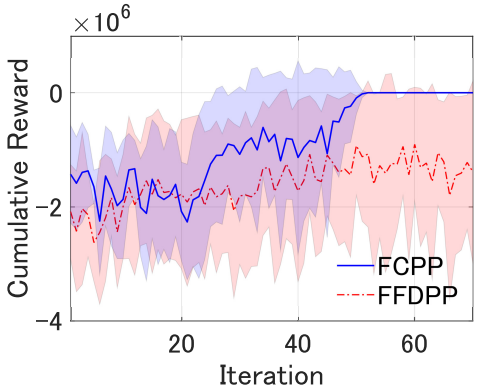
This section serves to verify that the results of the previous section continue to hold in the plant-wide control setting. We examine both algorithms on the challenging plant-wide control task which comprises a 13-dimensional state space



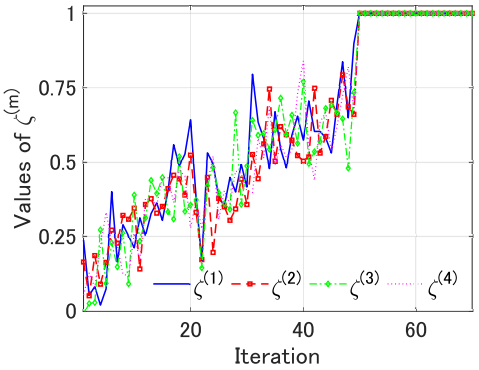
(a) Learning curves with fine-tuned action design.



(b) ζ values with fine-tuned action design.



(c) Learning curves with *plant-wide crude* action design.



(d) ζ values with *plant-wide crude* action design.

Figure 6.4: FCPP and FFDPP on the plant-wide control task.

[VAMSensor, LevelSensor, QualitySensor1-2, TempSensors1-8, Profit]^T, and 4-dimensional action space [FlowController, FlowController2, PressureController, TempController]^T. Beside the sensors and controllers introduced in the local control task, TempSensors measure the temperature inside the distillation column and are related to the *cost*; Profit directly measures the profit of the current product rate and quality in unit (yen/h). FlowController2 controls the reflux flow to the distillation column and PressureController allows for controlling the pressure of the process.

Fig. 6.4(a) shows the performance of FCPP and FFDPP with the fine-tuned action design. In this challenging task, FCPP was able to slightly outperform FFDPP due to the guarantee in Eq. (6.3a).

However, as shown in Fig. 6.4(c) for *plant-wide crude* action design, since the task is intrinsically very sensitive, FFDPP failed to learn meaningful behaviors with the *plant-wide crude* action design. On the other hand, FCPP still converged robustly at the latest around 50-th iteration. Figs. 6.4(b), 6.4(d) show the values of $\zeta^{(m)}, m = 1, \ldots, 4$ in both tasks. All ζ values have the same trend of gradually converging to the optimum $\zeta^{(m)} = 1$, demonstrating the success of factorized policies on decomposing high-dimensional value functions and accurately computing factorized values, as supported by the convergence in Fig. 6.4(c).

6.4.3 Local Control - Parameters α, β

In this section we examine the utility of FCPP against FFDPP for a wide range of algorithmic parameters α, β, which are the weighting coefficients introduced in Eq. (5.4). Intuitively, α determines the optimum of the MDP, which is equivalent to the final reward. β controls the update step, and is indirectly related to the process stability.

Since $\alpha \in [0, 1]$, we take 5 equidistant points from $[0, 1]$ for α. We also set the upper and lower bound for β as $10, 10^{-4}$, respectively. Recall that β appears in the form e^{β} and hence the upper and lower bound cover a wide enough range. We take 6 equidistant points for β from the interval $[10^{-4}, 10]$ and hence there are a total of 30 different groups of (α, β) values. For each of the group, we evaluate 5 independent trials of FCPP and FFDPP on the local control task and average their final cumulative reward, plotted in Fig. 6.5. With our hardware

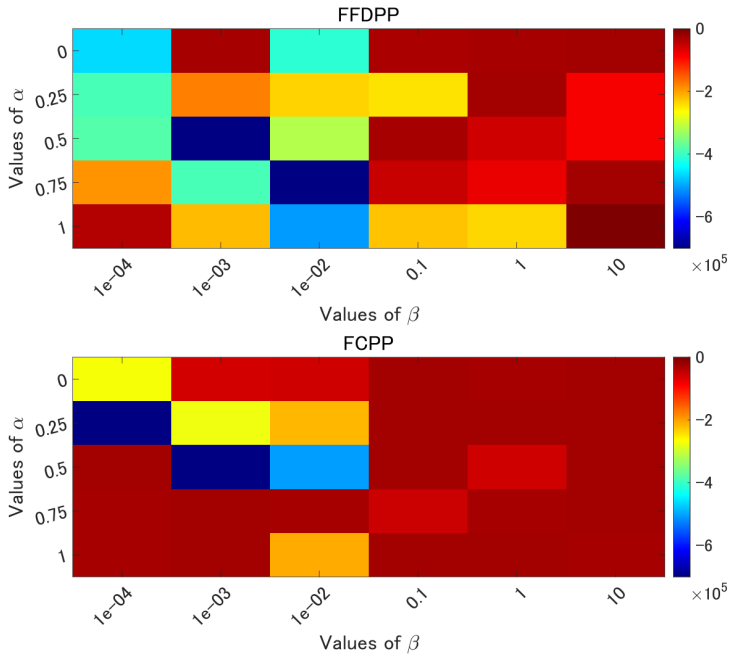


Figure 6.5: Comparison between FCPP and FFDPP on the localcontrol task for a wide range of algorithmic parameters α, β. Slots are experiments with α, β values on the axes and the color indicates the corresponding cumulative reward for the final iteration. All slots are averaged over 5 independent experiments. Evaluating the total of 300 experiments costed around 800 hours in reality.

specification, the total of $30 \times 5 \times 2 = 300$ experiments took around 800 hours in reality (since red slots correspond to fast convergence).

It is obvious from the above part of Fig. 6.5 that FFDPP is not robust towards different algorithmic parameters α, β: its highest rewards were achieved at the two extremes $(\alpha = 0, \beta = 10)$ and $(\alpha = 1, \beta = 10)$. For $\beta = 10^{-4} \sim 10^{-2}$, almost no configuration can provide a reasonable final performance. Even for larger β, careful tuning of α is still required to avoid the relatively low reward regions such as $(\alpha = 1, \beta = 0.1)$ or $(\alpha = 0.5, \beta = 10)$. On the other hand, the plot for FCPP demonstrated a much smoother trend: the regions of $\beta = 10^{-4} \sim 10^{-2}$ has high reward regions for $\alpha = 0.75 \sim 1$. From $\beta = 0.1 \sim 10$ almost all slots indicated high final rewards. This is achieved by effectively reusing previous information of the FCPP mechanism, demonstraing the importance of introducing

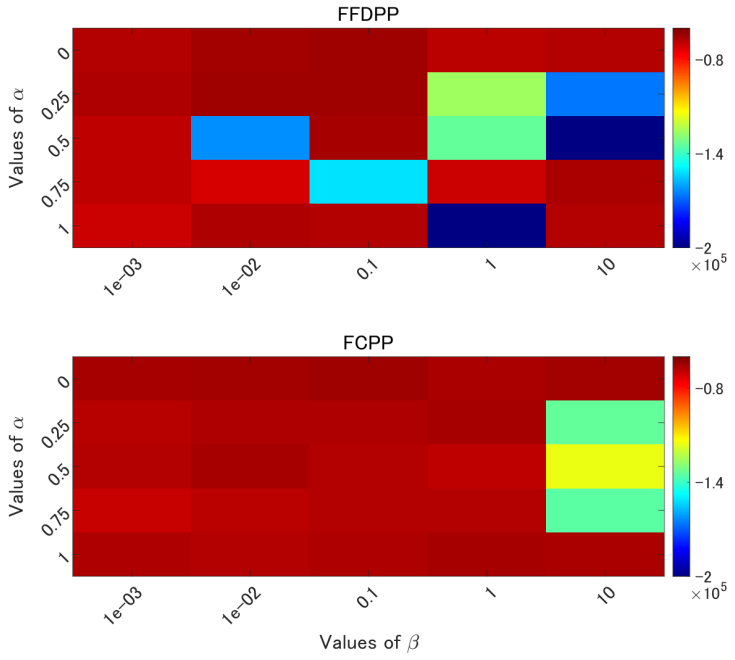


Figure 6.6: Comparison between FCPP and FFDPP on the plant-wde control task for a wide range of algorithmic parameters α, β. Slots are experiments with α, β values on the axes and the color indicates the corresponding cumulative reward for the final iteration. All slots are averaged over 3 independent experiments. Evaluating the total of 150 experiments costed around 1500 hours in reality.

the cautiousness into RL for process control problems.

6.4.4 Plant-wide Control - Parameters α, β

This section extends the comparison of algorithmic parameters in Sec. 6.4.3 to the plant-wide control task. We take 5 equidistant points from $[0, 1]$ for the value of α and 5 equidistant points from $[10^{-3}, 10]$ for the value of β. Due to the difficulty for running more experiments with longer learning horizon and more iterations, we evaluate every (α, β) pair for 3 independent experiments. The total number of $25 \times 2 = 150$ experiments costed around 1500 hours in reality (since red slots correspond to fast convergence).

It is clear from Fig. 6.6 that the same trend of local control for FFDPP

and FCPP continues to hold in the plant-wide case: for FFDPP the optimal parameter set lies within the region $[\alpha = 10^{-2} \sim 0.1, \beta = 0 \sim 0.25]$. However, slightly changing α to 0.5 or β to 1 causes the final reward to drop significantly, showing FCPP is highly sensitive to parameter choice. Within the region $\alpha \geq 0.25, \beta \geq 0.1$, slots with poor final performance are dominant.

On the other hand, FCPP exhibits much smoother final performance for slots $\beta \leq 1$ and all α thanks to the longer learning horizon and more iterations. The reason why $\beta = 10$ exhibits low final performance is possibly due to the numerical issue associated with exponential function e^{β}: repeated application of the Softmax operator can easily cause numerical instability. This suggests that in practice large values of β should be avoided or some tricks such as subtracting a constant from the exponential function be employed.

6.5 Discussion and Conclusion

Monotonic improvement and reinforcing experience using policy interpolation is not novel and has been well studied since the seminal conservative policy iteration algorithm [Kakade and Langford, 2002]. While CPI has inspired many extensions that have provably guarantees on the monotonic improvement, those algorithms are theory-oriented and no scalable implementation is available to the best of authors' knowledge. On the other hand, strengthening experience using the exp-over-sum in Eq. (6.1) (called follow-the-leader by [McMahan, 2017, Shalev-Shwartz, 2012]) exploits entropy regularization has demonstrated superior performance on a wide variety of tasks. However, it does not guarantee the agent can achieve any improvement over a given limited period of time.

The *cautious* learning mechanism introduced in this paper combined the above-mentioned two approaches to simultaneously ensure monotonic improvement and scalablity in high dimensional spaces, which bridged the gap between theory and practice in the monotonic improvement literature and provided guarantees for the entropy-regularized algorithms. The resultant algorithm was examined on the representative VAM process control problem, showing that given limited resources such as learning horizon, meaningful controllers can still be learned.

Specifically, from Figs. 6.5 and 6.6 it is obvious that FCPP outperformed FFDPP in terms of hyperparameter insensitivity: in both local and plant-wide control tasks FCPP had many available choices of performant parameters, and we believe that is because the cautious learning framework was in effect. By performing double regularization in both the reward function and policy space, FCPP was able to fully utilize past information and hence achieved good performance for a variety of parameters. This stood in contrast with FFDPP, which only performed one layer of regularization for the reward function and hence the ability of reusing information was limited. While in some parts there was oscilloation in the FCPP curves (performance getting low in some iterations), we believe that was due to the approximation error as a result of linear function approximation. Such approximation error is unavoidable and will sometimes caused performance oscillation.

The discretization of action space seems to contribute significantly to the fast and robust convergence of FCPP. From Fig. 6.3(c) we see that FCPP can still convergen with *crude* action design, indicating the valid range for action discretization is quite large and there might still be room for imporvement. However, we would like to leave it to the future work.

We think FCPP provides a promising first step towards applying RL algorithms to real-world process control problems, where a proper problem setup as well as suitable algorithmic parameters should be found with minimum efforts of trial-and-error adjust. We believe FCPP is a general framework applicable to various types of systems. However, a certain adaptation might be necessary for application on a different system: since we assumed the state-action space is representative of the plant, expert knowledge for selecting such relevant units might be indispensable. Further, the reward definition may vary depending on the characteristics of the system, e.g. there might not be an equivalent of `StabilityReadings` as we had [Hubbs et al., 2020]. Another possibility is that the target system has high dimensional action space but the dimensions are strongly correlated and hence the factorial assumption Eq. (6) fail. However, currently we are unaware of any such systems, and the assumption has been verified to work well with a variety of systems such as in [Matsubara et al., 2014, Tang and Agrawal, 2020, Zhu et al., 2020].

7 | Conclusion

This thesis discussed the use of entropy regularization in reinforcement learning. The contributions were mainly twofold: first, we formulated the use of Shannon entropy and KL regularization in the use of value-based algorithms for practical applications. Specifically, KL divergence played an important role in the regularization by constraining the distance between consecutive updated policies to not be overly large. This is especially important since in realistic applications the samples are often scarce and expensive. Ensuring the policies staying close is helpful in obtaining coheren trajectories of updated policies, and hence the quality, informativeness of the samples. Also, Shannon entropy can also be seen as a special case of KL divergence by enforcing the closedness to the uniform distribution. Arising from such constraint is the stochasticity of the optimal policies: they are more robust to errors, more efficient in exploring the state-action spaces compared to prior deterministic greedy optimal policies. Second, we derived useful algorithms for important applications, with examples featuring scalability, safety and robustness. Those applications included controlling industrial processes and safely manipulating robots, which provided important reference to today's RL deployment. We proposed several extensions such as factorial policies and actor-advisor framework. The factorial policies are suitable for handling the multi-dimensional action spaces of control problems by assuming the overall policy can be factorized into conditionally independent sub-policies. On our chemical process control problem, such assumption seemed to hold and yielded superior control performance. On the other hand, the actor-advisor framework effectively captured the notion of safety in minimizing the risk while simultaneously maximizing task reward. We can use the following list to summarize how

we leveraged KL divergence:

1. Scalability: multi-agent + KL regularization $D_{KL}\left(\pi_{k+1}^{(n)} \,\|\, \pi_{k}^{(n)}\right)$, where (n) denotes the index of the n-th agent.

2. Safety: actor-advisor, $D_{KL}\left(\pi^{act} \,\|\, \pi^{adv}\right), D_{KL}\left(\pi^{adv} \,\|\, \pi^{act}\right)$, where those two regularizers were added to the actor and advisor, respectively.

3. Robustness: relaxed bounded improvement by $D_{KL}(\pi_{k+1}||\pi_k) \leq C$, where C is some constant.

8 | Discussion

There are many potential directions for future work. First, there are still a lot to explore within the value-based methods by exploiting other entropies such as the Tsallis entropy and Rényi entropy that generalize the Shannon entropy as we mentioned in Section 2.5.2. The benefits are more flexible modeling choices: since the Shannon entropy only corresponds to a specific choice of the entropic index, it is possible for specific application, the best entropic index varies and does not lie close to the Shannon case. Another appeal is that the Shannon entropy induces Boltzmann softmax policy that assigns nonzero probability to every action. This is especially undesirable in some applications where some actions are dangerous. Very recently, there has seen literature discussion using the general Tsallis entropy in place of the Shannon entropy to induce policies that assign probabilities to only a subset of actions. Rényi entropy was used to explore the reward-free setting. By using those two entropies, one can formulate more general regularization known as the Tsallis KL divergence and Rényi divergence that generalize the KL divergence. It is hence interesting to explore the properties come with those regularizations.

Another direction is on the deployment: it is possible to go further by applying the algorithms we developed in this thesis to real-world problems? Those applications might require integration of the all the techniques such as scalability, safety and robustness for practical use. For those applications with strict requirement for safety, we might have to consider entropy-regulaized constrained Markov decision processes rather than the standard ones considered in this thesis. Similarly, more formal requirement of robustness might also bring the techniques developed here to the robust MDP context.

A | Cautious Actor-Critic

Reminder: This chapter provides a solution for extending the theoretically sound value-based methods to the more scalable and popular actor-critic setting. It can be seen as an extension of Chapter 5 that focuses on robustness in the finite action setting to continuous action spaces.

A.1 Introduction

Actor-critic (AC) methods of reinforcement learning (RL) have been gaining increasing interests recently due to their scalability to large-scale problems: they can learn with both on-policy or/and off-policy samples and handle continuous action spaces [Lillicrap et al., 2016, Schulman et al., 2015]; both in model-free or model-based setting [Haarnoja et al., 2018, Hafner et al., 2020]. Recently in the model-free setting there has seen a booming in off-policy AC methods [Fakoor et al., 2020, Haarnoja et al., 2018, Wang et al., 2017]. However, while these methods are sample-efficient in exploiting off-policy samples for continuous control, it is those samples that often bring oscillating performance during learning as a side-effect due to distribution mismatch. The oscillating performance of off-policy learning and persisting errors in the AC setting [Fujimoto et al., 2018] call for algorithms that can conservatively learn to better suit the stability-critical applications.

The performance oscillation and degradation problems have been widely discussed in the approximate dynamic programming (ADP) literature [Bertsekas, 2011, Wagner, 2011] that has motivated efficient learning algorithms against var-

ious sources of error. The seminal work of [Kakade and Langford, 2002] propose a principled approach to tackle performance degradation by leveraging policy interpolation which is *conservative* in that it reduces greediness of the updated policy. However, though it enjoys strong theoretical guarantees, its drawbacks limit its use in the AC setting: (1) it is difficult to obtain a reliable reference policy in high-dimensional continuous state-action spaces; (2) the interpolation is often regarded inconvenient to use and it is unclear how to design the interpolation in continuous action spaces. In practice, two popular variants [Schulman et al., 2015, 2017] that sidestep the interpolation and directly approximate the updated policy are more often used in the AC setting. On the other hand, the recently booming entropy-regularized ADP literature [Azar et al., 2012, Fox et al., 2016, Kozuno et al., 2019, Vieillard et al., 2020a] also features *conservative learning* [Kozuno et al., 2019] as they average over past value functions [Vieillard et al., 2020a]. Though these methods do not explicitly address the performance oscillation problem, they have been empirically verified to be error-tolerant and yield state-of-the-art performance on a wide range of tasks. Extending this conservative learning to AC has been studied [Haarnoja et al., 2018, Nachum et al., 2018]. However, the resulted *conservativeness* exists only in the critic: In challenging tasks, the performance degradation and oscillation still occur.

This paper aims to tackle the performance oscillation problem of off-policy AC by proposing a novel algorithm: *cautious actor critic* (CAC), where the naming cautious comes from the *doubly conservative* nature as we combine a conservative actor leveraging the concept of conservative policy iteration (CPI) [Kakade and Langford, 2002] with a conservative critic exploiting the entropy-regularization of conservative value iteration (CVI) [Kozuno et al., 2019]. The key observation is that the entropy-regularized critic can find error-tolerant reference policies and simplifies the unwieldy interpolated actor update while still ensures robust policy improvement. CAC leverages automatically adjusted interpolation to reflect the faith during learning: when performance oscillation is likely to happen, CAC behaves cautiously to rely more on validated previous policy rather than on the new policy. Our novel interpolation design is inspired by a very recent study from the ADP literature [Vieillard et al., 2020b] but improved for the continuous AC setting.

A.2 Related Work

It has been noticed that various sources of error such as approximation error in AC algorithms [Fu et al., 2019, Fujimoto et al., 2018] are the cause of performance degradation and oscillation during learning. In this section, we briefly survey some related works (partially) tackling this problem and outline our contributions.

Robust AC. Algorithms learning from off-policy data are sample-efficient but are also at the risk of divergence. To solve the divergence problem, an approach is to incorporate the importance sampling (IS) ratio [Precup et al., 2001]. However, the resultant algorithms typically have large variance as the IS ratio is the product of many potentially unbounded terms. Munos et al. [2016] proposed to clip the IS ratio and proved the resulting algorithm Retrace (λ) can attain the globally optimal policy. Retrace (λ) has motivated recent successful AC methods [Fakoor et al., 2020, Wang et al., 2017] that exploit both on- and off-policy samples for better stability while retaining sample-efficiency. The robustness comes from that any off-policy samples can be used without causing divergence and wild variance thanks to the clipping. However, one still has to trade off the learning speed with learning stability by the user-defined clipping threshold. If we favor more learning stability, the agent might fail to learn meaningful behaviors.

Entropy-regularized AC. The recently booming entropy-regularized ADP literature has established that by augmenting the reward with Shannon entropy, the optimal policy is multi-modal and hence robust against adversarial settings [Ahmed et al., 2019, Haarnoja et al., 2018, Nachum et al., 2017]. Another popular candidate entropy is the relative entropy or Kullback-Leibler (KL) divergence that renders the optimal policy an average of all past value functions [Azar et al., 2012, Fox et al., 2016, Kozuno et al., 2019, Vieillard et al., 2020a], which is more conservative and robust under mild assumptions such as the sequence of errors is martingale difference under the natural filtration [Azar et al., 2012]. These methods have been extended to the AC setting including state-of-the-art [Haarnoja et al., 2018] that exploits the Shannon entropy and [Nachum et al., 2018] that leverages the KL divergence. Those methods demonstrate strong empirical performance on a wide range of tasks. However, it should be noted that the conservativeness brought by the entropy-regularized reward augmentation exists only

in the critic. Since the average is prone to outliers, performance degradation can still happen if the value function estimates at some iterations are poor.

Conservative Policy Iteration. Tackling the performance oscillation problem has been widely discussed in the ADP literature [Bertsekas, 2011, Wagner, 2011], of which the seminal CPI algorithm [Kakade and Langford, 2002] has inspired many conservative learning schemes with strong theoretical guarantees for per-update improvement [Abbasi-Yadkori et al., 2016, Pirotta et al., 2013b]. However, CPI has seen limited applications to the AC setting due to two main drawbacks: (1) it assumes a *good* reference policy that is typically difficult to obtain in high-dimensional continuous state-action spaces; (2) the interpolation coefficient that interpolates the reference policy and current policy depends on the horizon of learning, which is typically short in ADP scenarios. In the AC setting featuring long learning horizon, this coefficient becomes vanishingly small and hence significantly hinders learning. A very recent work extended CPI to learning with deep networks and has demonstrated good performance on Atari games [Vieillard et al., 2020b]. However, it is limited to discrete action spaces while our method mainly focuses on continuous action spaces and can be easily adapted to discrete action setting. The above-mentioned drawbacks render CPI generally perceived as unwieldy [Schulman et al., 2015].

Trust-region Methods. Motivated by the above-mentioned drawbacks of CPI, two popular variants trust region policy optimization (TRPO) [Schulman et al., 2015] and its improved version proximal policy optimization (PPO) [Schulman et al., 2017] sidestep the interpolation and directly approximate the resultant conservative policy. TRPO and PPO are welcomed choices for learning from scratch when the reference policy is unavailable or unreliable, but they also ignore this knowledge when we have a good reference policy at our disposal. Further, TRPO and PPO require on-policy samples which are expensive since all samples can be used only once and then discarded.

A.3 Actor Critic methods

In this section we briefly introduce recent actor-critic algorithms and discuss their pros and cons and shed light on our proposal in Section A.4.

Trust Region Methods

TRPO exploits the policy improvement lemma of [Kakade and Langford, 2002] for ensuring approximately monotonic policy improvement. However, unlike in [Kakade and Langford, 2002] that at k-th iteration the policy is updated as $\pi_{k+1} = \zeta\pi' + (1-\zeta)\pi_k$, where π' is the greedy policy; TRPO constructs an algorithm that directly computes π_{k+1} without resorting to π'. Specifically, TRPO has the following update rule:

$$\begin{aligned} & J_{\pi_k}^{\text{TRPO}}(\pi) := \arg\max_{\pi} \mathbb{E}_{\pi, d^{\pi_k}}\left[A^{\pi_k}\right], \\ & \text{subject to} \quad C_\gamma \, \Delta_\pi \, D_{KL}^{\max}(\pi_k \| \pi) \leq \delta, \\ & \text{with} \quad \Delta_\pi = \max_{s,a} |A^\pi(s,a)|, \end{aligned} \tag{A.1}$$

where C_γ is a horizon-dependent constant, D_{KL} is the KL divergence and δ is the trust region parameter.

As computing $J_{\pi_k}^{\text{TRPO}}(\pi)$ requires sampling according to the stationary distribution d^{π_k}, it is inherently an on-policy algorithm, which is not sample-efficient as the samples can only be used only once and discarded.

Off-policy Maximum Entropy Actor-Critic

As state-of-the-art model-free off-policy AC algorithm, soft actor-critic (SAC) [Haarnoja et al., 2018] maximizes not only task reward but also the Shannon entropy of policy. The entropy term in the reward function renders the optimal policy multi-modal as opposed to deterministic policies of algorithms that solely maximize task reward, which is beneficial due to the multi-modality [Haarnoja et al., 2017] and has demonstrated superior sample-efficiency due to more effective exploration of the state-action spaces. Writing in the ADP manner, SAC has the following update rule (we drop the state-action pair for the Q function for simplicity):

$$\begin{aligned} & \begin{cases} \pi_{k+1} \leftarrow \arg\max_\pi \mathbb{E}_\pi \left[Q_{\mathcal{H}}^{\pi_k}(s,a) + \kappa \mathcal{H}\left(\pi(\cdot|s_t)\right)\right] \\ Q_{\mathcal{H}}^{\pi_{k+1}} \leftarrow r(s,a) + \gamma \left(\mathbf{P} V_{\mathcal{H}}^{\pi_k}\right)(s,a) \end{cases} \\ & \text{where } V_{\mathcal{H}}^{\pi}(s) = \sum_{t \geq 0} \gamma^t \mathbb{E}_\pi \left[r(s_t, a_t) + \kappa \mathcal{H}\left(\pi(\cdot|s_t)\right) \middle| s_0 = s \right]. \end{aligned} \tag{A.2}$$

$\mathcal{H}(\pi) := -\sum_a \pi(a|s) \log \pi(a|s)$ denotes the Shannon entropy of policy π, κ denotes the weighting coefficient and $V^\pi_\mathcal{H}$ denotes the soft value function when regularized with the Shannon entropy. SAC performs one step look-ahead for updating the actor, where states are randomly sampled from a replay buffer, and then actions are generated by feeding the states into the policy network [Haarnoja et al., 2018]. As such, SAC does not need an IS ratio, but it has been demonstrated that SAC often oscillates wildly in performance.

A.4 Cautious Actor Critic

In this section we present CAC, an off-policy actor-critic method capable of learning conservatively against performance oscillation and degradation.

A.4.1 CAC Algorithm

For the ease of understanding, we write CAC in the following approximate policy iteration style [Vieillard et al., 2020a]. Specifically, the first step corresponds to the policy (actor) improvement and the last step corresponds to the interpolation:

$$\text{CAC} \begin{cases} \pi_{k+1} \leftarrow \arg\max_\pi \mathbb{E}_\pi \left[Q^{\pi_k}_\mathcal{I}(s,a) + \mathcal{I}^\pi_{\pi_k}(s) \right] \\ Q^{\pi_{k+1}}_\mathcal{I} \leftarrow r(s,a) + \gamma \left(\mathbf{P} V^{\pi_{k+1}}_\mathcal{I} \right)(s,a) \\ \zeta \leftarrow \left(\tilde{\Delta}^{\pi_{k+1}}_{\pi_k} \right)^{-1} \left(\mathbb{E}_{\pi_{k+1}, \mathcal{B}_K} \left[A^{\pi_k}(s,a) \right] \right) \\ \tilde{\pi}_{k+1} \leftarrow \zeta \pi_{k+1} + (1-\zeta) \pi_k \end{cases} \tag{A.3}$$

$$\text{with } V^{\pi_{k+1}}_\mathcal{I}(s) = \sum_{t \geq 0} \gamma^t \mathbb{E}_\pi \left[r(s_t, a_t) + \mathcal{I}^{\pi_{k+1}}_{\pi_k}(s_t) \,\middle|\, s_0 = s \right],$$

$$\mathcal{I}^{\pi_{k+1}}_{\pi_k}(s) = \mathbb{E}_{\pi_{k+1}} \left[-\kappa \log \pi_{k+1}(a|s) - \tau \log \frac{\pi_{k+1}(a|s)}{\pi_k(a|s)} \right],$$

where ζ is the interpolation coefficient computed by ζ^* in Eq. (A.7), $\mathcal{B}_K$ denotes the on-policy replay buffer, with K indicating the number of steps up to now. κ, τ denote the Shannon entropy and KL divergence regularization coefficient, respectively. $Q_\mathcal{I}, V_\mathcal{I}$ (and hence $A_\mathcal{I}$) denote the entropy-regularized value functions. For uncluttered notations, in the rest of the paper we drop the subscript $\mathcal{I}$. Except the computation of ζ, all other steps are computed using samples from

the off-policy replay buffer $\mathcal{B}$. For later convenience, we define $\alpha := \frac{\kappa}{\kappa+\tau}$ and $\beta := \frac{1}{\kappa+\tau}$. Note our use of both on- and off-policy replay buffers renders CAC similar in spirit to [Fakoor et al., 2020, Gu et al., 2017, Wang et al., 2017].

We first compute the greedy policy π_{k+1}. Due to the Fenchel conjugacy [Geist et al., 2019], when $\mathcal{I}_{\pi_k}^{\pi_{k+1}}$ is included in the arg max, the maximizer policy can be analytically derived as $\pi_{k+1}(a|s) \propto \pi_k^{\alpha}(a|s) \exp\left(\beta Q_{\pi_k}(a|s)\right)$ [Kozuno et al., 2019]. Then the entropy-regularized action value function $Q^{\pi_{k+1}}$ is evaluated. In the third step, we use the on-policy replay buffer $\mathcal{B}_K$ for computing ζ as described in Eq. (A.7). Finally, the optimal policy in the sense of guaranteeing policy improvement is obtained by interpolating π_{k+1} with π_k. We present the following theorem of CAC convergence in the tabular case.

Theorem A.4.1. *Repeated application of CAC Eq. (A.3) on any initial policy π will make it converges to the entropy regularized optimal policy $\pi^*(a|s) = \frac{\exp\left(\frac{1}{\kappa}Q^*(s,a)\right)}{\int_{a\in\mathcal{A}}\exp\left(\frac{1}{\kappa}Q^*(s,a)\right)}$.*

From Theorem 1 we see the optimal policy and corresponding optimum of the MDP is biased by the choice of κ. If we gradually decay the value of κ then we recover the optimum of the non-regularized MDP [Vieillard et al., 2020a]. In the following sections, we describe in detail the CAC actor and critic, as well as the derivation of actor gradient expression and practical interpolation coefficient ζ design.

Conservative Actor

As discussed in Section A.3, at k-th iteration TRPO directly constructs a new policy π by maximizing $J_{\pi_k}^{\text{TRPO}}(\pi)$. This is useful if the agent learns from scratch, but it discards the *reference policy* when available. On the other hand, we follow the exact form of [Kakade and Langford, 2002] by taking the information of reference policy into account, where we choose π_{k+1} to be the reference policy π':

$$\tilde{\pi}_{k+1} = \zeta\pi_{k+1} + (1-\zeta)\pi_k. \tag{A.4}$$

Our objective function $J_{\pi_k,\pi_{k+1}}^{\text{CAC}}(\pi)$ explicitly features the knowledge of the reference policy [Pirotta et al., 2013b]. Specifically, the objective can be lower-bounded

as:

$$
\begin{aligned}
J^{\mathrm{CAC}}_{\pi_k,\pi_{k+1}}(\pi) &:= \mathbb{E}_{\pi_{k+1},d^{\pi_{k+1}}}\left[A^{\pi_k}(s,a)\right] \\
&\geq C'_\gamma\ (v\,\tilde{\Delta}^{\pi_{k+1}}_{\pi_k})^{-1}\left(\mathbb{E}_{\pi_{k+1},d^{\pi_k}}\left[A^{\pi_k}(s,a)\right]\right)^2, \\
\text{given } \zeta^* &= 2\,C'_\gamma(v\,\tilde{\Delta}^{\pi_{k+1}}_{\pi_k})^{-1}\left(\mathbb{E}_{\pi_{k+1},d^{\pi_k}}\left[A^{\pi_k}(s,a)\right]\right), \\
\tilde{\Delta}^{\pi_{k+1}}_{\pi_k} &= \max_{s,s'}\left|A^{\pi_{k+1}}_{\pi_k}(s) - A^{\pi_{k+1}}_{\pi_k}(s')\right|, \\
v &= \max_s D_{TV}\left(\pi_{k+1}(\cdot|s)\|\pi_k(\cdot|s)\right),
\end{aligned}
\tag{A.5}
$$

where D_{TV} denotes the total variation. v, $\tilde{\Delta}^{\pi_{k+1}}_{\pi_k}$ and the expectation wrt π_{k+1}, d^{π_k} require estimation. C'_γ absorbs the horizon-dependent constants. Hence, when optimizing the lower bound of $J^{\mathrm{CAC}}_{\pi_k,\pi'}(\pi)$, we can achieve guaranteed improvement.

In the existing literature [Abbasi-Yadkori et al., 2016, Kakade and Langford, 2002, Pirotta et al., 2013b], the difficulties of extending Eq. (A.5) to large-scale problems are: (1) preparing a reliable reference policy in high dimensional continuous state-action spaces is difficult; (2) it is hard to accurately estimate v, the maximum total variation between two policies without enforcing a gradual change of policies, which is absent in these works. On the other hand, naively using $v \leq 2$ as suggested by [Pirotta et al., 2013b] often yields vanishingly small ζ, which significantly hinders learning. (3) the horizon-dependent constant C'_γ developed in the classic ADP literature is not suitable for learning with deep networks that feature long horizon of learning. As will be demonstrated in the following sections, we tackle the first and second problems by leveraging entropy-regularized critic, and the third problem via a novel design of ζ inspired by a very recent work for discrete action problems [Vieillard et al., 2020b].

Conservative Critic

In Eq. (A.5) we see in order to yield a meaningful interpolation coefficient ζ one is required to accurately estimate the maximum total derivation v, which is intractable in high dimensional continuous action spaces. However, by introducing an entropy-regularized critic, we can leverage the following theorem to avoid estimating v:

Theorem A.4.2. *[Kozuno et al., 2019, Proposition 3] For any two consecutive entropy-regularized policies π_k, π_{k+1} generated by Eq. (A.3), the following bound*

for their maximum total deviation holds:

$$\max_s D_{TV}\left(\pi_{k+1}(\cdot|s)\|\pi_k(\cdot|s)\right) \leq \sqrt{4B_K + 2C_K},$$
$$\text{where } B_k := \frac{1-\gamma^k}{1-\gamma}\epsilon\beta, \quad C_k := r_{max}\beta\sum_{j=0}^{k} k-1\alpha^j\gamma^{k-j-1}, \tag{A.6}$$

ϵ is the uniform upper bound of errors.

Recall from Section A.4.1 that $\alpha := \frac{\kappa}{\kappa+\tau}, \beta := \frac{1}{\kappa+\tau}$. Specifically, it has been proved in [Kozuno et al., 2019] that this bound is non-improvable, i.e. there exists an MDP such that the inequality becomes equality.

By leveraging an entropy-regularized critic, the objective in Eq. (A.5) becomes:

$$J^{\text{CAC}}_{\pi_k,\pi_{k+1}}(\pi) \geq C'_\gamma C_k \left(\tilde{\Delta}^{\pi_{k+1}}_{\pi_k}\right)^{-1}\left(\mathbb{E}_{\pi_{k+1},d^{\pi_k}}\left[A^{\pi_k}(s,a)\right]\right)^2,$$
$$\text{given } \zeta^* = 2\,C'_\gamma C_k(\tilde{\Delta}^{\pi_{k+1}}_{\pi_k})^{-1}\left(\mathbb{E}_{\pi_{k+1},d^{\pi_k}}\left[A^{\pi_k}(s,a)\right]\right), \tag{A.7}$$

where C'_γ absorbs horizon-dependent constants and C_k is from Theorem A.4.2. Estimating the optimal ζ^* now requires estimating the expectation wrt π_{k+1}, d^{π_k} and $\tilde{\Delta}^{\pi_{k+1}}_{\pi_k}$ which have been studied by [Vieillard et al., 2020b].

The KL divergence also manifests its importance for generating reasonable reference policies even for high dimensional or continuous action problems. Consider the following upper bound due to [Vieillard et al., 2020a] where reward is augmented by the KL divergence:

$$\|Q^* - Q^{\pi_{k+1}}\|_\infty \leq \frac{2}{1-\gamma}\left\|\frac{1}{k}\sum_{j=0}^{k}\epsilon_j\right\|_\infty + \frac{4}{1-\gamma}\frac{V_{max}}{k},$$

where ϵ_j are errors and $V_{max} = \frac{r_{\max}}{1-\gamma}$. By comparing it with the non-improvable approximate modified policy iteration (AMPI) bound where the reward is not augmented [Scherrer et al., 2015]:

$$\|Q^* - Q^{\pi_{k+1}}\|_\infty \leq \left((1-\gamma)\sum_{j=1}^{k}\|\epsilon_j\|_\infty\right) + \frac{2\gamma^{k+1}}{1-\gamma}V_{max},$$

we see that the error term for the KL regularization case is sup-over-sum. Under mild assumptions such as ϵ_j are iid distributed under the natural filtration [Azar

et al., 2012], the summation over errors asymptotically cancels out. On the other hand, the error term for AMPI depends on the summation of *maximum* of every iteration, which is typically large. Further, the dependence of error on the horizon is linear $\frac{1}{1-\gamma}$ rather than quadratic, which is a significant improvement as typically $\gamma \approx 1$.

Network Optimization Perspective

Given the above ADP-style characterization for both the actor and the critic, we now examine Eq. (A.3) from the optimization perspective. Suppose the critic is parametrized by a network with parameters θ and the actor by a network with parameters ϕ. CAC updates the network weights θ, ϕ by solving the following minimization problems:

$$y = r + \gamma \left(\mathbb{E}_{a \sim \pi_\phi} \left[Q_{\bar{\theta}}(s', a) \right] + \mathcal{I}^{\pi_\phi}_{\pi_{\bar{\phi}}}(s') \right), \tag{A.8a}$$

$$\theta \leftarrow \arg\min \mathbb{E}_{\mathcal{B}} \left[\left(Q_\theta(s, a) - y \right)^2 \right], \tag{A.8b}$$

$$\phi \leftarrow \arg\min \mathbb{E}_{\mathcal{B}} \left[D_{KL} \left(\pi_\phi \| (1-\zeta)\pi_{\bar{\phi}} + \zeta \mathcal{G}_{\pi_{\bar{\phi}}} Q_\theta \right) \right], \tag{A.8c}$$

where the update of ϕ corresponds to solving an *information projection* problem. This is because policies π_{k+1}, π_k are Boltzmann softmax [Geist et al., 2019] but their summation is generally not Boltzmann, which might results in loss of desirable properties. By the following theorem, in the ideal case we can find a Boltzmann policy π_ϕ that perfectly represents the interpolation by solving the information projection problem.

Theorem A.4.3. *[Ziebart, 2010, Theorem 2.8] Let $\pi^{(1)}$, $\pi^{(2)}$, $\ldots, \pi^{(n)}$ be an arbitrary sequence of policies and $\zeta_1, \ldots \zeta_n$ be a sequence of numbers such that $\zeta_i \geq 0, \forall i$, $\sum_{i=1}^n \zeta_i = 1$. Then the policy π' defined by:*

$$\pi'(a|s) := \frac{\sum_{i=1}^n \zeta_i \, P(\mathcal{S} = s, \mathcal{A} = a | \pi^{(i)})}{\sum_{i=1}^n \zeta_i \, P(\mathcal{S} = s | \pi^{(i)})} \tag{A.9}$$

has same expected number of state-action occurrences when the denominator is nonzero.

In implementation as the states are sampled from the replay buffer, there is an error term in this information projection step. Taking the above information

projection into account, we elaborate upon gradient expression of the actor via the following proposition:

Proposition A.4.4. *Let the actor network be parametrized by weights ϕ and critic by θ. Define $\mathcal{G}Q_{\bar{\phi},\theta}$ as the greedy policy with respect to the CAC critic. The subscript $\bar{\phi}$ comes from the baseline policy introduced by KL divergence. Then the gradient of the actor update can be expressed as:*

$$\begin{aligned} &\nabla_\phi \mathbb{E}_{\substack{s\sim\mathcal{B} \\ a\sim\pi_\phi}} \left[D_{\bar{\phi}}^{\phi} - \frac{\beta}{1+\mathcal{X}} Q_\theta(s,a) \right], \\ &\textit{where } D_{\bar{\phi}}^{\phi} = \log \pi_\phi \left(a \mid s\right) - \frac{\alpha+\mathcal{X}}{1+\mathcal{X}} \log \pi_{\bar{\phi}}(a;s) \\ &\mathcal{X} = \frac{1-\zeta}{\zeta} \cdot \frac{\pi_{\bar{\phi}}(a \mid s)}{\mathcal{G}Q_{\bar{\phi},\theta}(a \mid s)}. \end{aligned} \tag{A.10}$$

This gradient expression is similar to SAC [Haarnoja et al., 2018] which is off-policy since states s are sampled from the off-policy replay buffer $\mathcal{B}$. However, CAC has the term $\log \pi_{\bar{\phi}}(a;s)$ from the KL regularization. The term $\mathcal{X}$ in both the Q_θ and $\log \pi_{\bar{\phi}}(a;s)$ involves ζ that encodes the information for guiding the gradient to *cautiously* learn.

A.4.2 Design of Interpolation Coefficient

One of the main difficulties to extending CPI to learning with deep networks is that ζ becomes vanishingly small due to the typically long horizon in the AC setting. To tackle this problem, [Vieillard et al., 2020b] propose to heuristically design ζ to be a non-trivial value, which features the consideration of *moving averages*.

Recall that in Eq. (A.7), ζ is computed by a function of the form:

$$\zeta = C'_\gamma C_k \frac{\mathbb{E}_{\pi_{k+1}, d^{\pi_k}} \left[A^{\pi_k}(s,a) \right]}{\tilde{\Delta}_{\pi_k}^{\pi_{k+1}}},$$

where C'_γ absorbs horizon-dependent constants and C_k is defined in Eq. (A.6). We propose to remove C'_γ since it tends to zero as the horizon increases. Recall also that $\tilde{\Delta}_{\pi_k}^{\pi_{k+1}}$ is the maximum difference of the expected advantage function defined in Eq. (A.5). We propose the following novel ζ design:

$$\zeta^{\text{CAC}} = \text{clip}\left(\frac{\tilde{\mathbb{A}}}{|\tilde{\mathbb{A}}^{\text{MaxDiff}}|}, 0, \ 1 \right), \tag{A.11}$$

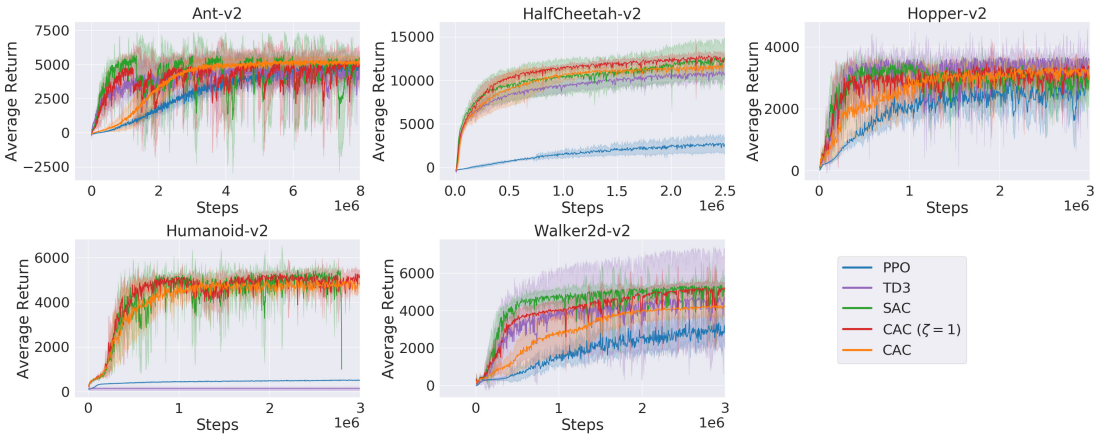


Figure A.1: Training curves on the continuous control benchmark problems. The solid curves show the mean and shaded regions the standard deviation over the five independent trials. In all tasks CAC achieved comparable performance but significantly stabilized learning.

where by following the moving average concept of [Vieillard et al., 2020b], we update $\tilde{\mathbb{A}}$ and $\tilde{\mathbb{A}}^{\text{MaxDiff}}$ as:

$$\begin{aligned} M &= \mathbb{E}_{\substack{s\sim\mathcal{B}_K \\ a\sim\mathcal{GQ}^{\pi_k}}} \left[A^{\pi_k}(s,a)\right] \\ \tilde{\mathbb{A}} &\leftarrow \begin{cases} c, & \text{if } M \leq 0 \\ (1-\nu_A)\,\tilde{\mathbb{A}} + \nu_A M, & \text{else} \end{cases} \\ \tilde{\mathbb{A}}^{\text{MaxDiff}} &\leftarrow (1-\nu_{A^{\text{MaxDiff}}})\,\tilde{\mathbb{A}}^{\text{MaxDiff}} + \nu_{A^{\text{MaxDiff}}}\, M. \end{aligned} \tag{A.12}$$

Here, M is the current estimate of $\mathbb{E}_{\pi_{k+1},d^{\pi_k}}\left[A^{\pi_k}(s,a)\right]$. We propose to set an if-else judgement here, as $M < 0$ indicates the updated policy has worse performance than the current policy, we let $\tilde{\mathbb{A}}$ be a negative value c, hence enforcing $\zeta = 0$. This information is incorporated into $\tilde{\mathbb{A}}$ by exponential moving average with the previous estimates. $\tilde{\mathbb{A}}^{\text{MaxDiff}}$ attempts to approximate the maximum difference $\tilde{\Delta}_{\pi_k}^{\pi_{k+1}}$. $\mathcal{B}_K$ is a FIFO replay buffer storing K on-policy samples, and $\nu_A, \nu_{A^{\text{MaxDiff}}} \in [0,1]$ are the hyperparameters controlling the average. Following [Vieillard et al., 2020b], it is beneficial to have $\nu_{A^{\text{MaxDiff}}} \leq \nu_A$ for smooth learning.

Computing ζ in Eq. (A.11) using the moving average in Eq. (A.12) is in spirit similar to [Vieillard et al., 2020b]. However, they focus on general stationary

Table A.1: The performance oscillation values of all algorithms for all environments. The bold numbers indicate the smallest performance oscillation values. $\times$ indicates the algorithm failed to learn meaningful behaviors. CAC recorded the smallest performance oscillation values for all the environments. PPO is the only on-policy algorithm in the comparison.

	$\|\mathcal{O}J\|_\infty$					$\|\mathcal{O}J\|_2$				
	PPO	**TD3**	**SAC**	**CAC** ($\zeta = 1$)	**CAC**	**PPO**	**TD3**	**SAC**	**CAC** ($\zeta = 1$)	**CAC**
Ant	1979	4979	7793	7160	**1811**	359	510	642	591	**297**
HalfCheetah	$\times$	2337	3717	4200	**1870**	$\times$	331	425	397	**286**
Hopper	**1598**	3515	2598	2944	1944	318	609	454	394	**279**
Humanoid	$\times$	$\times$	4115	3092	**2199**	$\times$	$\times$	645	436	**313**
Walker2d	1673	3729	4577	4310	**1345**	330	461	499	334	**183**

policies. As $\tilde{\mathbb{A}}$ is an off-policy estimate of the on-policy term $\mathbb{E}_{\pi_{k+1}, d^{\pi_k}}\left[A^{\pi_k}(s,a)\right]$, it might corrupt the improvement guarantee. On the other hand, we focus on entropy-regularized policies which allow one to bound the performance loss of leveraging off-policy estimate $\tilde{\mathbb{A}}$ [Zhu and Matsubara, 2020, Theorem 3].

A.5 Experiments

As CAC combines concepts from ADP literature such as KL regularization and conservative learning that have not seen applications in AC, it is interesting to examine the combination against existing AC methods in challenging tasks. We choose a set of high dimensional continuous control tasks from the OpenAI gym benchmark suite [Brockman et al., 2016].

For comparison, we compare CAC with twin delayed deep deterministic policy gradient (TD3) [Fujimoto et al., 2018] that comprehensively surveys the factors causing poor performance of actor-critic methods, to examine the cautious learning mechanism. As CAC is based on the CPI that has also inspired TRPO and PPO, we compare it with PPO which is improved over TRPO [Schulman et al., 2017]. As PPO does not involve computing ζ, we include the curves when $\zeta = 1$.

We also compare with SAC [Haarnoja et al., 2018] which has similar architecture.

To better illustrate and quantify the *stability* during learning, we follow [Zhu and Matsubara, 2020] to define the measure of performance oscillation:

$$\begin{aligned} &\forall k, \text{ such that } R_{k+1} - R_k < 0 \\ &\|\mathcal{O}J\|_\infty = \max_k |R_{k+1} - R_k|, \\ &\|\mathcal{O}J\|_2 = \sqrt{\tfrac{1}{N}\sum_{k=1}^{N} (R_{k+1} - R_k)^2}, \end{aligned} \tag{A.13}$$

where N is the steps of the learning and R_k refers to the cumulative reward reported at k-th evaluation. Intuitively, $\|\mathcal{O}J\|_\infty$ and $\|\mathcal{O}J\|_2$ measure the maximum and average degradation during learning, respectively.

A.5.1 Comparative Evaluation

We run all algorithms with the same set of hyperparameters listed in Section **??** of the Appendix. All figures are plotted with statistics from 10 different random seeds, with each performing 10 evaluation rollouts every 5000 environment steps.

Figure A.1 shows the learning curves of the algorithms. CAC achieved comparable performance with other AC algorithms while significantly stabilized learning curves. PPO's learning speed was the slowest among all algorithms on all environments, due to the on-policy nature of PPO which is sample-inefficient. Other methods were able to leverage off-policy samples to quickly learn meaningful behaviors. However, the fast learning came at a cost: except the relatively simple `HalfCheetah-v2`, on all environments these off-policy algorithms oscillated wildly, especially on the challenging `Humanoid-v2` where both PPO and TD3 failed to learn any meaningful behaviors, and the performance of SAC, CAC with $\zeta = 1$ degraded frequently. On the other hand, CAC traded off a little bit slower learning for stability, exhibiting smooth curves. Indeed, the convergence rate of CAC is $O\left(e^{-(1-\gamma)\sum_{j=1}^{k}\zeta_j}\right)$ [Vieillard et al., 2020b], which emphasizes stability more as $\zeta \to 0$.

The comparison on stability of the algorithms can be seen from the Table A.1 that summarized the values of $\|\mathcal{O}J\|_\infty$ and $\|\mathcal{O}J\|_2$ for all algorithms. It provided empirical support as CAC showed least oscillation during learning. This is in contrast to other off-policy algorithms oscillated wildly during learning. Since CAC with $\zeta = 1$ still showed huge oscillation, it can be concluded that the

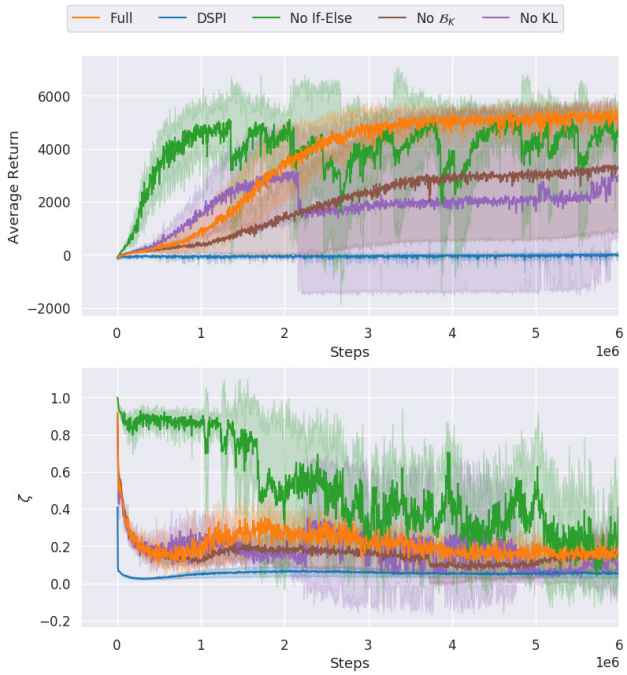


Figure A.2: **Top:** The learning curves of derived methods of CAC on Ant-v2. **Bottom:** The value of the mixture coefficient ζ.

mixture coefficient introduced in CAC is effective in preventing significant policy degradation.

A.5.2 Ablation study on mixture coefficient

In this section we conduct an ablation test to study the effectiveness of CAC as well as the proposed ζ design in Section A.4.2. We compare the following setup:

1. **Full.** This is CAC with the proposed ζ^{CAC} in Eq. (A.12). The curve is same as in Figure A.1.

2. **No KL.** We remove KL regularization from Eq. (A.3). This corresponds to SAC with the cautious actor.

3. **DSPI.** This corresponds to *Deep Safe Policy Iteration* [Zhu and Matsubara, 2020] that uses the ζ suggested by [Vieillard et al., 2020b].

4. **No $\mathcal{B}_K$.** We replace the on-policy replay buffer $\mathcal{B}_K$ with off-policy $\mathcal{B}$ as suggested by [Vieillard et al., 2020b].

5. **No If-Else.** This corresponds to removing the `if` term in Eq. (A.12) and learns with only the `else` condition.

As is obvious from Figure A.2, while removing the `if-else` judgement accelerated learning, it ignored the warning from $M < 0$ that the updated policy was poorer. The consequent curve oscillated drastically as the result of aggressive ζ in the bottom image. Removing KL regularization induced learning curve similar with **Full** in the beginning, but the performance degraded significantly since the middle stage and failed to recover. This is probably due to the policies were corrupted by error. Using off-policy replay buffer $\mathcal{B}$ demonstrated stable learning. This is expected as the agent was forced to learn cautiously by the CAC mechanism. On the other hand, the learning was slow as off-policy samples were not as informative as on-policy ones, as we observed small ζ values in the bottom figure.

It is most interesting to examine the DSPI case where ζ was set according to the suggestion of [Vieillard et al., 2020b]. Though this scheme works well in Atari games, the resulting algorithm failed to learn any meaningful behaviors in the challenging control tasks with continuous action spaces. This is because they estimate with the entire off-policy replay buffer $\mathcal{B}$ which tends to produce very large estimate of $\tilde{\Delta}_{\pi_k}^{\pi_{k+1}}$, leading to vanishingly small ζ^{DSPI} and subsequent poor performance.

A.6 Conclusion

We have presented CAC, a novel actor-critic algorithm by introducing several concepts from the approximate dynamic programming literature. The cautiousness in CAC consists in the doubly conservativeness: the actor follows conservative policy iteration [Kakade and Langford, 2002] that ensures monotonic improvement and the critic exploits conservative value iteration [Kozuno et al., 2019, Vieillard et al., 2020a] that has been shown to yield state-of-the-art guarantees in ADP literature. Our key observation was by introducing an entropy-regularized

critic the unwieldy interpolated actor update can be simplified significantly while still ensuring robust policy improvement. CAC performed comparable to the state-of-the-art AC methods while significantly stabilized learning on the benchmark control problems with high dimensional continuous state-action spaces. An interesting future direction is to incorporate other entropy for different purposes. For example, α-divergence could be used to achieve sparse optimal policies.

References

Y. Abbasi-Yadkori, P. L. Bartlett, and S. J. Wright. A fast and reliable policy improvement algorithm. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 1338–1346, 2016.

J. Achiam, D. Held, A. Tamar, and P. Abbeel. Constrained policy optimization. In *Proceedings of the 34th International Conference on Machine Learning (ICML) - Volume 70*, ICML'17, page 22–31, 2017.

Z. Ahmed, N. Le Roux, M. Norouzi, and D. Schuurmans. Understanding the impact of entropy on policy optimization. In *Proceedings of 36th International Conference on Machine Learning*, volume 97, pages 151–160, 2019.

T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, KDD '19, page 2623–2631. Association for Computing Machinery, 2019.

R. Akrour, A. Abdolmaleki, H. Abdulsamad, J. Peters, and G. Neumann. Model-free trajectory-based policy optimization with monotonic improvement. *Journal of Machine Learning Research*, 19(14):1–25, 2018.

E. Altman. *Constrained Markov decision processes*. 1999.

O. M. Andrychowicz, B. Baker, M. Chociej, and Others. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

A. Antos, C. Szepesvári, and R. Munos. Learning near-optimal policies with bellman-residual minimization based fitted policy iteration and a single sample path. *Mach. Learn.*, 71(1):89–129, Apr. 2008.

K. Asadi and M. L. Littman. An alternative softmax operator for reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, volume 70 of *Proceedings of Machine Learning Research*, pages 243–252, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

M. G. Azar, V. Gómez, and H. J. Kappen. Dynamic policy programming with function approximation. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 119–127, 2011.

M. G. Azar, V. Gómez, and H. J. Kappen. Dynamic policy programming. *Journal of Machine Learning Research*, 13(1):3207–3245, 2012.

T. A. Badgwell, J. H. Lee, and K.-H. Liu. Reinforcement learning – overview of recent progress and implications for process control. In *13th International Symposium on Process Systems Engineering (PSE)*, Computer and Chemical Engineering, pages 71 – 85. 2018.

E. Banijamali, Y. Abbasi-Yadkori, M. Ghavamzadeh, and N. Vlassis. Optimizing over a restricted policy class in mdps. In *Proceedings of Machine Learning Research*, volume 89 of *Proceedings of Machine Learning Research*, pages 3042–3050. PMLR, 2019.

A. Beck. *First-Order Methods in Optimization*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2017.

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47(1):253–279, 2013. ISSN 1076-9757.

R. E. Bellman. *Dynamic Programming*. Dover Publications, Inc., New York, NY, USA, 2003. ISBN 0486428095.

F. Berkenkamp, M. Turchetta, A. P. Schoellig, and A. Krause. Safe model-based reinforcement learning with stability guarantees. In *Proc. of Neural Information Processing Systems (NIPS)*, pages 908–919, 2017. URL https://arxiv.org/abs/1705.08551.

D. Bertsekas. Approximate policy iteration: A survey and some new methods. *Journal of Control Theory and Applications*, 9:310–335, 08 2011.

D. P. Bertsekas. *Dynamic Programming and Optimal Control*, volume I. Athena Scientific, Belmont, MA, USA, 3rd edition, 2005a.

D. P. Bertsekas. *Dynamic Programming and Optimal Control*. 2005b. ISBN 1886529264.

D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1st edition, 1996. ISBN 1886529108.

A. Blum, J. Hopcroft, and R. Kannan. *Foundations of Data Science*. Cambridge University Press, 2020.

S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, USA, 2004.

J. Bretagnolle and C. Huber. Estimation des densités : risque minimax. *Séminaire de probabilités de Strasbourg*, 12:342–363, 1978.

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

A. Bušić and S. Meyn. Action-constrained Markov Decision Processes with Kullback-Leibler cost. In *Proceedings of the 31st Conference On Learning Theory*, volume 75 of *Proceedings of Machine Learning Research*, pages 1431–1444, 2018.

R. Chen and T. McAvoy. Plantwide control system design: Methodology and application to a vinyl acetate process. *Industrial and Engineering Chemistry Research*, 42(20): 4753–4771, 2003.

R. Chen, K. Dave, T. J. McAvoy, and M. Luyben. A nonlinear dynamic model of a vinyl acetate process. *Industrial and Engineering Chemistry Research*, 42(20):4478–4487, 2003.

R. Cheng, J. F. Forbes, and W. San Yip. Dantzig–Wolfe decomposition and plant-wide MPC coordination. *Computers and Chemical Engineering*, 32(7):1507–1522, 2008.

Y. Chow, N. Ofir, E. Duenez-guzman, and M. Ghavamzadeh. A Lyapunov-based Approach to Safe Reinforcement Learning. In *Annual Conference on Neural Information Processing Systems (NIPS)*, pages 1–10, 2018.

T. M. Cover and J. A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, USA, 2006.

Y. Cui, T. Matsubara, and K. Sugimoto. Kernel Dynamic Policy Programming: Applicable Reinforcement Learning to Robot Systems with High Dimensional States. *Neural networks*, 94:13–23, 2017a.

Y. Cui, T. Matsubara, and K. Sugimoto. Pneumatic artificial muscle-driven robot control using local update reinforcement learning. *Advanced Robotics*, pages 1–16, 2017b.

Y. Cui, L. Zhu, M. Fujisaki, H. Kanokogi, and T. Matsubara. Factorial kernel dynamic policy programming for vinyl acetate monomer plant model control. In *IEEE International Conference on Automation Science and Engineering (IEEE-CASE)*, pages 304–309, 2018.

O. Dogru, N. Wieczorek, K. Velswamy, F. Ibrahim, and B. Huang. Online reinforcement learning for a continuous space system with experimental validation. *Journal of Process Control*, 104:86–100, 2021.

M. Dotoli, A. Fay, M. Miśkowicz, and C. Seatzu. A Survey on Advanced Control Approaches in Factory Automation. *IFAC-PapersOnLine*, 48(3):394–399, 2015. ISSN 2405-8963.

L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry. Implementation matters in deep policy gradients: A case study on ppo and trpo. In *International Conference on Learning Representations (ICLR)*, pages 1–12, 2019.

D. Ernst, M. Glavic, P. Geurts, and L. Wehenkel. Approximate Value Iteration in the Reinforcement Learning Context. Application to Electrical Power System Control. *International Journal of Emerging Electric Power Systems*, 3(1):1–35, 2005.

B. Eysenbach, S. Gu, J. Ibarz, and S. Levine. Leave no trace: Learning to reset for safe and autonomous reinforcement learning. In *International Conference on Learning Representations (ICLR)*, pages 1–18, 2018.

R. Fakoor, P. Chaudhari, and A. J. Smola. P3o: Policy-on policy-off policy optimization. In *Conference on Uncertainty in Artificial Intelligence*, pages 1017–1027, 2020.

M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and S. Legg. Noisy networks for exploration. In *Proceedings of the International Conference on Representation Learning (ICLR 2018)*, Vancouver (Canada), 2018.

R. Fox, A. Pakman, and N. Tishby. Taming the noise in reinforcement learning via soft updates. In *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence*, pages 202–211, 2016.

J. Fu, A. Kumar, M. Soh, and S. Levine. Diagnosing bottlenecks in deep q-learning algorithms. volume 97 of *Proceedings of 36th International Conference on Machine Learning*, pages 2021–2030, 2019.

S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 1587–1596, 2018.

J. Garcia and F. Fernandez. Safe exploration of state and action spaces in reinforcement learning. *Journal of Artificial Intelligence Research*, 45:515–564, 2012. ISSN 10769757.

J. García and F. Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16:1437–1480, 2015.

P. Geibel and F. Wysotzki. Risk-sensitive reinforcement learning applied to control under constraints. *Journal of Artificial Intelligence Research*, 24:81–108, 2005.

M. Geist, B. Scherrer, and O. Pietquin. A theory of regularized Markov decision processes. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 2160–2169, 2019.

S. S. Gu, T. Lillicrap, R. E. Turner, Z. Ghahramani, B. Schölkopf, and S. Levine. Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 3846–3855, 2017.

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, pages 1352–1361, 2017.

T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 1861–1870, 2018.

D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint, arXiv:1912.01603*, 2020.

S. A. Harp, S. Brignone, B. F. Wollenberg, and T. Samad. Sepia. a simulator for electric power industry agents. *IEEE Control Systems Magazine*, 20(4):53–69, 2000.

P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. *CoRR*, abs/1709.06560, 2017. URL http://arxiv.org/abs/1709.06560.

J. C. Hoskins and D. M. Himmelblau. Process control via artificial neural networks and reinforcement learning. *Computers and Chemical Engineering*, 16(4):241–251, 1992.

C. D. Hubbs, C. Li, N. V. Sahinidis, I. E. Grossmann, and J. M. Wassick. A deep reinforcement learning approach for chemical production scheduling. *Computers and Chemical Engineering*, 141:106982, 2020.

M. S. Kakade. *On the Sample Complexity of Reinforcement Learning.* PhD thesis, University College London, 2003.

S. Kakade and J. Langford. Approximately optimal approximate reinforcement learning. In *19th International Conference on Machine Learning (ICML)*, pages 267–274, 2002.

M. Kano and Y. Nakagawa. Data-based process monitoring, process control, and quality improvement: Recent developments and applications in steel industry. *Computers and Chemical Engineering*, 32(1):12 – 24, 2008.

M. Kano and M. Ogawa. The State of the Art in Advanced Chemical Process Control in Japan. *IFAC Proceedings Volumes*, 42(11):10–25, 2009. ISSN 1474-6670.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *Int. J. Rob. Res.*, 32(11):1238–1274, sep 2013. ISSN 0278-3649.

T. Kozuno, E. Uchibe, and K. Doya. Theoretical analysis of efficiency and robustness of softmax and gap-increasing operators in reinforcement learning. In *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, volume 89, pages 2995–3003, 2019.

E. Kreyszig. *Introductory Functional Analysis with Applications*. Wiley, 1991. ISBN 9780471504597.

V. Krishnamurthy. *Partially Observed Markov Decision Processes: From Filtering to Controlled Sensing*. Cambridge University Press, 2016.

S. Kubosawa, T. Onishi, and Y. Tsuruoka. Synthesizing chemical plant operation procedures using knowledge, dynamic simulation and deep reinforcement learning. In *SICE Annual Conference*, pages 1376–1379, 2018.

S. Kullback. *Information Theory and Statistics*. Wiley, New York, 1959.

M. G. Lagoudakis and R. Parr. Least-squares policy iteration. *The Journal of Machine Learning Research*, 4(44):1107–1149, 2003.

A. Lazaric, M. Ghavamzadeh, and R. Munos. Finite-sample analysis of least-squares policy iteration. *Journal of Machine Learning Research*, 13(98):3041–3074, 2012.

Q. V. Le, T. Sarlos, and A. Smola. Fastfood-computing hilbert space expansions in log-linear time. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, 2013. ISBN 0086538217.

J. H. Lee and W. Wong. Approximate dynamic programming approach for process control. *Journal of Process Control*, 20(9):1038–1048, 2010.

K. Lee, S. Choi, and S. Oh. Sparse markov decision processes with causal sparse tsallis entropy regularization for reinforcement learning. *IEEE Robotics and Automation Letters*, 3:1466–1473, 2018.

K. Lee, S. Kim, S. Lim, S. Choi, M. Hong, J. I. Kim, Y. Park, and S. Oh. Generalized tsallis entropy reinforcement learning and its application to soft mobile robots. In *Robotics: Science and Systems XVI*, pages 1–10, 2020.

S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *International Journal of Robotics Research*, 2018. ISSN 17413176.

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. In *International Conference on Learning Representations (ICLR)*, 2016.

Z. C. Lipton, J. Gao, L. Li, J. Chen, and L. Deng. Combating reinforcement learning's sisyphean curse with intrinsic fear. *CoRR*, abs/1611.01211, 2016.

W. Liu, Y. Tan, and Q. Qiu. Enhanced Q-learning algorithm for dynamic power management with performance constraint. In *the Conference on Design, Automation and Test in Europe*, pages 602–605, 2010.

M. L. Luyben and B. D. Tyréus. An industrial design/control study for the vinyl acetate monomer process. *Computers and Chemical Engineering*, 22(7-8):867–877, 1998.

Y. Machida, S. Ootakara, H. Seki, Y. Hashimoto, M. Kano, Y. Miyake, N. Anzai, M. Sawai, T. Katsuno, and T. Omata. Vinyl Acetate Monomer (VAM) Plant Model: A New Benchmark Problem for Control and Operation Study. *International Federation of Automatic Control (IFAC)*, 49(7):533–538, 2016.

H. Mania, A. Guy, and B. Recht. Simple random search of static linear policies is competitive for reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 31, pages 1–10. Curran Associates, Inc., 2018.

A. F. T. Martins and R. F. Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 1614–1623, 2016.

T. Matsubara, V. Gómez, and H. J. Kappen. Latent Kullback-Leibler Control for Continuous-State Systems using Probabilistic Graphical Models. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 583–592, 2014.

T. J. McAvoy. Synthesis of plantwide control systems using optimization. *Industrial and Engineering Chemistry Research*, 38(8):2984–2994, 1999.

H. B. McMahan. A survey of algorithms and analysis for adaptive online learning. *Journal of Machine Learning Research*, 18(90):1–50, 2017.

J. Mei, C. Xiao, R. Huang, D. Schuurmans, and M. Müller. On principled entropy exploration in policy optimization. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 3130–3136, 2019.

J. Mei, C. Xiao, C. Szepesvari, and D. Schuurmans. On the global convergence rates of softmax policy gradient methods. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 6820–6829, 2020.

A. M. Metelli, M. Mutti, and M. Restelli. Configurable Markov decision processes. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3491–3500, 2018.

M. Metzger and G. Polakow. A survey on applications of agent technology in industrial process control. *IEEE Transactions on Industrial Informatics*, 7(4):570–581, 2011.

V. Mnih, K. Kavukcuoglu, D. Silver, and Others. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015a.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, and Others. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015b.

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48, pages 1928–1937, 2016.

T. M. Moldovan and P. Abbeel. Safe exploration in markov decision processes. In *International Coference on International Conference on Machine Learning (ICML)*, pages 1451–1458, 2012.

R. Munos, T. Stepleton, A. Harutyunyan, and M. Bellemare. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1054–1062, 2016.

O. Nachum, M. Norouzi, K. Xu, and D. Schuurmans. Bridging the gap between value and policy based reinforcement learning. In *Advances in Neural Information Processing Systems 30*, pages 2775–2785. 2017.

O. Nachum, M. Norouzi, K. Xu, and D. Schuurmans. Trust-pcl: An off-policy trust region method for continuous control. In *International Conference on Learning Representations*, pages 1–11, 2018.

Y. Nesterov. *Lectures on Convex Optimization*. Springer Publishing Company, Incorporated, 2nd edition, 2018. ISBN 3319915770.

G. Neu, A. Jonsson, and V. Gómez. A unified view of entropy-regularized markov decision processes. *arXiv:1705.07798*, 2017. URL http://arxiv.org/abs/1705.07798.

R. Nian, J. Liu, and B. Huang. A review on reinforcement learning: Introduction and applications in industrial process control. *Computers and Chemical Engineering*, 139: 106886, 2020.

D. G. Olsen, W. Y. Svrcek, and B. R. Young. Plantwide control study of a vinyl acetate monomer process design. *Chemical Engineering Communications*, 192(10): 1243–1257, 2005.

OpenAI. Openai five blog. https://openai.com/blog/openai-five/, 2018.

M. Papini, M. Pirotta, and M. Restelli. Adaptive batch size for safe policy gradients. In *Advances in Neural Information Processing Systems*, volume 30, pages 1–10, 2017.

M. Papini, A. Battistello, and M. Restelli. Balancing learning speed and stability in policy gradient via adaptive exploration. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 1188–1199, 2020.

T. J. Perkins and A. G. Barto. Lyapunov Design for Safe Reinforcement Learning. *Journal of Machine Learning Research*, 3:803–832, 2002.

M. Pirotta, M. Restelli, and L. Bascetta. Adaptive step-size for policy gradient methods. In *Advances in Neural Information Processing Systems*, volume 26, pages 1–9, 2013a.

M. Pirotta, M. Restelli, A. Pecorino, and D. Calandriello. Safe policy iteration. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28, pages 307–315, 2013b.

W. B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality (Wiley Series in Probability and Statistics)*. Wiley-Interscience, USA, 2007. ISBN 0470171553.

D. Precup. Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*, page 80, 2000.

D. Precup, R. S. Sutton, and S. Dasgupta. Off-policy temporal difference learning with function approximation. In *International Conference on Machine Learning*, page 417–424, 2001.

M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1177–1184, 2008.

R. T. Rockafellar. *Convex analysis*. Princeton Mathematical Series. Princeton University Press, Princeton, N. J., 1970.

G. Rummery and M. Niranjan. On-line q-learning using connectionist systems. *Technical Report CUED/F-INFENG/TR 166*, 11 1994.

I. Sason and S. Verdú. f-divergence inequalities. *IEEE Transactions on Information Theory*, 62:5973–6006, 2016.

B. Scherrer and M. Geist. Local policy search in a convex space and conservative policy iteration as boosted policy search. In *Machine Learning and Knowledge Discovery in Databases*, pages 35–50, 2014.

B. Scherrer, M. Ghavamzadeh, V. Gabillon, B. Lesner, and M. Geist. Approximate modified policy iteration and its application to the game of tetris. *Journal of Machine Learning Research*, 16(1):1629–1676, 2015.

J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pages 1889–1897, 2015.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017. URL https://arxiv.org/pdf/1707.06347.pdf.

H. Seki, M. Ogawa, T. Itoh, S. Ootakara, H. Murata, Y. Hashimoto, and M. Kano. Plantwide control system design of the benchmark vinyl acetate monomer production plant. *Computers and Chemical Engineering*, 34(8):1282–1295, 2010.

S. Shalev-Shwartz. Online learning and online convex optimization. *Foundations and Trends® in Machine Learning*, 4(2):107–194, 2012.

L. Shani, Y. Efroni, and S. Mannor. Adaptive trust region policy optimization: Global convergence and faster rates for regularized mdps. *CoRR*, abs/1909.02769, 2019. URL http://arxiv.org/abs/1909.02769.

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, and Others. Mastering the game of Go without human knowledge. *Nature*, 550:354–359, 2017.

Z. Song, R. E. Parr, and L. Carin. Revisiting the softmax bellman operator: New benefits and new perspective. In *Proceedings of the 36th International Conference on Machine Learning*, pages 1–10, 2019.

S. Spielberg, A. Tulsyan, N. P. Lawrence, P. D. Loewen, and R. Bhushan Gopaluni. Toward self-driving processes: A deep reinforcement learning approach to control. *AIChE Journal*, 65(10):e16689, 2019.

S. P. K. Spielberg, R. B. Gopaluni, and P. D. Loewen. Deep reinforcement learning approaches for process control. In *International Symposium on Advanced Control of Industrial Processes (AdCONIP)*, pages 201–206, 2017.

R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1038–1044, 1996.

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.

R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1057–1063, 1999.

S. Syafiie, F. Tadeo, and E. Martinez. Model-free learning control of neutralization processes using reinforcement learning. *Engineering Applications of Artificial Intelligence*, 20(6):767 – 782, 2007.

C. Szepesvari. *Algorithms for Reinforcement Learning*. Morgan and Claypool Publishers, 2010. ISBN 1608454924.

Y. Tang and S. Agrawal. Discretizing continuous action space for on-policy optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5981–5988, 2020.

C. Tessler, D. Mankowitz, and S. Mannor. Reward constrained policy optimization. In *Proceedings of the International Conference on Representation Learning (ICLR 2019)*, pages 1–11, 2019.

E. Todorov. Linearly-solvable Markov decision problems. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1369–1376, 2006.

C. Tsallis. Possible generalization of boltzmann-gibbs statistics. *Journal of Statistical Physics*, 52:479–487, 1988.

C. Tsallis. *Introduction to Nonextensive Statistical Mechanics: Approaching a Complex World*. Springer New York, 2009. ISBN 9780387853581.

Y. Tsurumine, Y. Cui, E. Uchibe, and T. Matsubara. Deep reinforcement learning with smooth policy update: Application to robotic cloth manipulation. *Robotics and Autonomous Systems*, 112:72 – 83, 2019.

A. B. Tsybakov. *Introduction to Nonparametric Estimation*. Springer Publishing Company, Incorporated, 1st edition, 2008. ISBN 0387790519.

E. Uchibe. Model-free deep inverse reinforcement learning by logistic regression. *Neural Processing Letters*, 47(3):891–905, 2018.

E. Uchibe and K. Doya. Forward and inverse reinforcement learning sharing network weights and hyperparameters. *Neural Networks*, 144:138–153, 2021.

L. van der Maaten. Accelerating t-sne using tree-based algorithms. *Journal of Machine Learning Research*, 15:3221–3245, 2014.

N. Vieillard, O. Pietquin, and M. Geist. On connections between constrained optimization and reinforcement learning. In *Optimization Foundation for Reinforcement Learning Workshop at NeurIPS*, 2019. URL https://arxiv.org/abs/1910.08476.

N. Vieillard, T. Kozuno, B. Scherrer, O. Pietquin, R. Munos, and M. Geist. Leverage the average: an analysis of regularization in rl. In *Advances in Neural Information Processing Systems 33*, pages 1–12, 2020a.

N. Vieillard, O. Pietquin, and M. Geist. Deep conservative policy iteration. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI'20*, pages 6070–6077, 2020b.

N. Vieillard, O. Pietquin, and M. Geist. Munchausen reinforcement learning. In *Advances in Neural Information Processing Systems 33*, pages 1–11. 2020c.

N. Vieillard, B. Scherrer, O. Pietquin, and M. Geist. Momentum in reinforcement learning. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108, pages 2529–2538, 2020d.

P. Wagner. A reinterpretation of the policy oscillation phenomenon in approximate policy iteration. In *Advances in Neural Information Processing Systems 24*, pages 2573–2581, 2011.

Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas. Sample efficient actor-critic with experience replay. In *International Conference on Learning Representations*, pages 1–13, 2017.

C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

J. Wen, B. Dai, L. Li, and D. Schuurmans. Batch stationary distribution estimation. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 10203–10213, 2020.

P. Winder. *Reinforcement Learning: Industrial Applications of Intelligent Agents*. O'Reilly Media, Incorporated, 2020. ISBN 9781098114831.

P. Wurman, S. Barrett, K. Kawamoto, J. MacGlashan, K. Subramanian, T. Walsh, R. Capobianco, A. Devlic, F. Eckert, F. Fuchs, L. Gilpin, P. Khandelwal, V. Kompella, H. Lin, P. MacAlpine, D. Oller, T. Seno, C. Sherstan, M. Thomure, and H. Kitano. Outracing champion gran turismo drivers with deep reinforcement learning. *Nature*, 602:223–228, 2022.

X. Xu, D. Hu, and X. Lu. Kernel-based least squares policy iteration for reinforcement learning. *IEEE Transactions on Neural Networks*, 18(4):973–992, 2007. ISSN 1045-9227.

Y. Ye. The simplex and policy-iteration methods are strongly polynomial for the markov decision problem with a fixed discount rate. *Mathematics of Operations Research*, 36: 593–603, 2011.

H. Yoo, B. Kim, J. W. Kim, and J. H. Lee. Reinforcement learning based optimal control of batch processes using monte-carlo deep deterministic policy gradient with phase segmentation. *Computers and Chemical Engineering*, 144:107133, 2021.

C. Zhang, Y. Cai, L. Huang, and J. Li. Exploration by maximizing renyi entropy for reward-free rl framework. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10859–10867, 2021.

A. Zheng, R. V. Mahajanam, and J. M. Douglas. Hierarchical procedure for plantwide control system synthesis. *AIChE Journal*, 45(6):1255–1265, 1999.

L. Zhu and T. Matsubara. Ensuring monotonic policy improvement in entropy-regularized value-based reinforcement learning. *arXiv preprint, arXiv:2008.10806*, 2020. URL https://arxiv.org/pdf/2008.10806.pdf.

L. Zhu, Y. Cui, G. Takami, H. Kanokogi, and T. Matsubara. Scalable reinforcement learning for plant-wide control of vinyl acetate monomer process. *Control Engineering Practice*, 97:104331–104340, 2020.

L. Zhu, G. Takami, M. Kawahara, H. Kanokogi, and T. Matsubara. Alleviating parameter-tuning burden in reinforcement learning for large-scale process control. *Computers & Chemical Engineering*, 158:107658, 2022.

B. D. Ziebart. *Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy*. PhD thesis, Carnegie Mellon University, 2010.