Sample Complexity of Incremental Policy Gradient Methods for Solving Multi-Task Reinforcement Learning

Yitao Bai

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Electrical Engineering

Thinh T. Doan, Chair
Daniel J. Stilwell
Ming Jin

March 25, 2024
Blacksburg, Virginia

Keywords: Markov Decision Processes, Multi-Task Reinforcement Learning

Sample Complexity of Incremental Policy Gradient Methods for Solving Multi-Task Reinforcement Learning

Yitao Bai

(ABSTRACT)

We consider a multi-task learning problem, where an agent is presented a number of N reinforcement learning tasks. To solve this problem, we are interested in studying the gradient approach, which iteratively updates an estimate of the optimal policy using the gradients of the value functions. The classic policy gradient method, however, may be expensive to implement in the multi-task settings as it requires access to the gradients of all the tasks at every iteration. To circumvent this issue, in this paper we propose to study an incremental policy gradient method, where the agent only uses the gradient of only one task at each iteration. Our main contribution is to provide theoretical results to characterize the performance of the proposed method. In particular, we show that incremental policy gradient methods converge to the optimal value of the multi-task reinforcement learning objectives at a sublinear rate $\mathcal{O}(1/\sqrt{k})$, where k is the number of iterations. To illustrate its performance, we apply the proposed method to solve a simple multi-task variant of GridWorld problems, where an agent seeks to find an policy to navigate effectively in different environments.

Sample Complexity of Incremental Policy Gradient Methods for Solving Multi-Task Reinforcement Learning

Yitao Bai

(GENERAL AUDIENCE ABSTRACT)

First, we introduce a popular machine learning technique called Reinforcement Learning (RL), where an agent, such as a robot, uses a policy to choose an action, like moving forward, based on observations from sensors like cameras. The agent receives a reward that helps judge if the policy is good or bad. The objective of the agent is to find a policy that maximizes the cumulative reward it receives by repeating the above process. RL has many applications, including Cruise autonomous cars, Google industry automation, training ChatGPT language models, and Walmart inventory management. However, RL suffers from task sensitivity and requires a lot of training data. For example, if the task changes slightly, the agent needs to train the policy from the beginning. This motivates the technique called Multi-Task Reinforcement Learning (MTRL), where different tasks give different rewards and the agent maximizes the sum of cumulative rewards of all the tasks. We focus on the incremental setting where the agent can only access the tasks one by one randomly. In this case, we only need one agent and it is not required to know which task it is performing. We show that the incremental policy gradient methods we proposed converge to the optimal value of the MTRL objectives at a sublinear rate $\mathcal{O}(1/\sqrt{k})$, where k is the number of iterations. To illustrate its performance, we apply the proposed method to solve a simple multi-task variant of GridWorld problems, where an agent seeks to find an policy to navigate effectively in different environments.

Acknowledgments

I would like to acknowledge the significant contributions of my faculty advisor, Thinh Doan, whose guidance and support have been invaluable throughout the research and writing process of this thesis. Dr. Doan's expertise and insights have helped shape the direction and methodology of this work. I also extend my acknowledgment to the members of my thesis committee, Daniel J. Stilwell, Ryan Williams, and Ming Jin, for their valuable feedback and input. Their expertise and constructive criticism have contributed to the refinement of this research. Finally, I acknowledge the support of my family and friends for their understanding and encouragement throughout this academic journey.

Contents

List of Figures

Chapter 1

Introduction

Reinforcement learning (RL) is a machine learning paradigm inspired by behavioral psychology, focusing on how agents learn to make sequential decisions in dynamic environments to achieve specific goals. Unlike traditional supervised learning, where models are trained on labeled datasets, RL algorithms learn through interaction with their environment. For example, the wild fire monitoring in Figure 1.1, drone in the left picture is the agent, and the smoke on the right represents the agent's observation from the environment which is camera view. At each iteration, an RL agent observes the current state from the environment, decides an action based on its current policy, receives feedback in the form of rewards or loss, and updates its policy accordingly to maximize the sum of rewards through the time. Through this process, RL agents can learn complex behaviors and optimal decision-making strategies, making RL particularly well-suited for tasks involving uncertainty, partial observability, and long-term planning. RL has found applications in diverse domains, including robotics [9], gaming playing [19], path plan navigation [1, 11, 32], autonomous driving [14], field coverage [24], and drone-based delivery [21].

While RL provides a powerful learning framework, it suffers a fundamental challenge in its data efficiency. The existing RL methods like Q-learning and policy gradient are known to require a significant amount of data and computational resources in their training. In addition, policies learned in one task might not be applicable to solve other tasks so we often have to restart learning when faced with slight task variations, requiring the agent to start

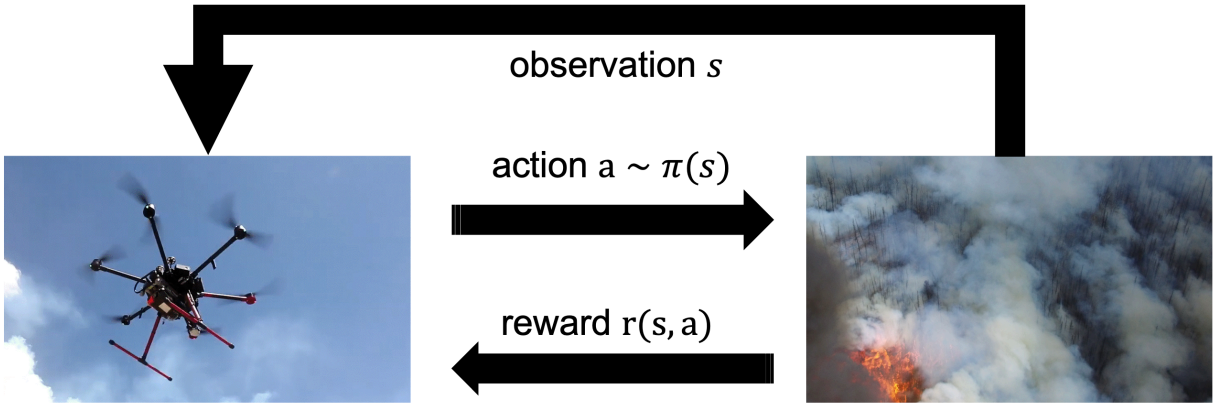


Figure 1.1: Illustration of the reinforcement learning process

afresh without leveraging prior experiences from similar tasks. This lack of adaptability necessitates time-consuming retraining for each minor task alteration.

The limitation of RL has motivated the study of multi-task RL (MTRL) framework, where an agent aims to learn multiple tasks simultaneously. If the tasks are related in some ways, then learning them jointly should be more efficient than learning individually. For example, the agent is learning to avoid a specific kind of obstavle in different environment like city and forest, at least it can use the knowledge of obstacle detection in different environment instead of learning from beginning. MTRL aims to improve generalization and efficiency by exploiting the inherent relationships between multiple tasks [5, 13].

Motivated by recent studies on policy gradient methods in single-task RL settings [17], in this thesis we propose to study an incremental policy gradient method to solve MTRL problems. The incremental method we proposed only uses the gradient of one random task per iteration in its update under some assumption to the period of accessing all tasks, unlike the classic policy gradient approach where the agent is required to access the gradients of all the tasks at every iteration. Thus, the proposed incremental policy gradient method can be implemented efficiently when agent cannot access which task it is perform or agent

have difficulty to accessing all the tasks at every step is challenging. For example, agent is very expansive spacecraft, we cannot build thousands of spacecrafts to let them perform reinforcement learning in different environments and space craft cannot know what kind of environment encountered until get into regions of charged particles.

1.1 Contribution

This thesis proposes to study an incremental policy mirror descent (IPMD) method for solving MTRL problems. Our main contribution is to provide theoretical results to characterize the performance of the proposed method. In particular, we show that incremental policy gradient methods converge to the optimal value of the multi-task reinforcement learning objectives at a sublinear rate $\mathcal{O}(1/\sqrt{k})$, where k is the number of iterations. To illustrate its performance, we apply the proposed method to solve a simple multi-task variant of GridWorld problems, where an agent seeks to find a policy to navigate effectively in different environments.

Chapter 2

Review of Literature

2.1 Markov Decision Process

A Markov Decision Process (MDP) serves as a foundational concept within the idea of reinforcement learning (RL), representing a mathematical framework utilized to model decision-making processes with stochastic policy and transition of state. At its core, an MDP can be view as a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$. States $s \in \mathcal{S}$ represents the possible configurations or situations within the environment, while actions $a \in \mathcal{A}$ represent the possible choices available to the agent. Transition probabilities $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ dictate the probability of transitioning from one state to another state upon selecting a particular action, capturing the stochastic nature of the environment. Rewards $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ denote the immediate feedback received by the agent upon executing an action in a specific state, providing a quantitative measure of the desirability of certain outcomes. The discount factor serves to weight the importance of future rewards relative to the rewards of current iteration, facilitating long-term decision-making. Agent will apply some chosen policy $\pi : \mathcal{S} \to \mathcal{A}$ to decide the action take based on the state observed. Each choice of policy in task i induces a long-term expected discounted reward

$$V_\pi(s) = \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t) \mid s_0 = s, a_t \sim \pi(\cdot \mid s_t), s_{t+1} \sim \mathcal{P}(\cdot \mid s_t, a_t)\right],$$

and the state-action value function Q

$$Q_\pi(s,a) = \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t,a_t) \mid s_0 = s, a_0 = a, s_{t+1} \sim \mathcal{P}(\cdot \mid s_t, a_t), a_{t+1} \sim \pi(\cdot \mid s_t)\right]. \quad (2.1)$$

This Q satisfies a well known function called Bellman Equation [3]

$$Q_\pi(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s' \mid s,a) V_\pi(s'). \quad (2.2)$$

Then we can define the objective function as

$$f(\pi) = \max_\pi \mathbf{E}_{s\sim\rho}[V_\pi(s)],$$

where ρ is initial distribution of the initial state.

2.2 Policy Gradient for Single MDP

Policy gradient methods are a class of algorithms which directly learn a policy without explicitly computing value functions. policy gradient methods focus on learning the parameters of a policy directly.

In single Markov decision process (MDP), a policy gradient algorithm aims to optimize a parameterized policy $\pi_\theta(a \mid s)$, where θ represents the parameters of the policy. The goal is to find the parameters θ that maximize the value function V_{π_θ}, obtained by following the policy. The key idea behind policy gradient methods is to parameterize the policy and then update these parameters based on the gradient which represent the direction that increases the expected return. This is typically done by computing the gradient of the expected return with respect to the policy parameters. Several well known policy gradient

based reinforcement learning algorithms are Monte-Carlo[29], actor-critic method[20], and proximal policy optimization (PPO)[25].

In this thesis we want to focus on one specific policy gradient called policy mirror descent [17]. This method gives an exponential convergence. So, at each iteration, we compute the gradient of V which we can use Q since

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(\cdot \mid s) Q_\pi(s, a).$$

So we have the following algorithm

Algorithm 2.1. Policy Mirror Descent Method (PMD)

Input: π_0, step sizes $\{\alpha_k\}_{k \geq 0}$

For $k = 0, 1, ..., K-1$ **do**

[1] Compute $Q_k \triangleq Q_{\pi_k}$using the policy π_k.

[2] Update π_k for all $s \in \mathcal{S}$ as

$$\pi_{k+1}(\cdot \mid s) = \text{argmax}_{\mu(\cdot|s) \in \triangle_{|\mathcal{A}|}} \{\alpha_k \langle Q_k(s, \cdot), \mu(\cdot, s) \rangle - D^\mu_{\pi_k}(s)\}. \tag{2.3}$$

Then we need to find a way to find Q function which we used as the gradient of the objective function. Notice that compute Q function is an expansive step and the following two subsections will explain how to solve this problem.

2.2.1 Q-Learning

Q-learning is a temporal difference (TD) based algorithm, which is one of the most popular methods to directly compute the Q function[28]. Q-learning estimates the function $Q(s, a)$ introduced in Equation (2.1). The action-value function represents the expected cumulative reward should be received by selecting an action a when the current state is s and the action is chosen by policy π afterwards. Here is the algorithm

---

Algorithm 2.2. Q-Learning

Input: Initial action-value function Q_0, exploration-exploitation constant ϵ, stepsize α

For $k = 0, 1, ..., K-1$ **do**

[1] Draw $\epsilon_k \in (0, 1)$.

[2] Choose action

$$a_k = \begin{cases} \text{random action } a \in \mathcal{A} & \text{if } \epsilon_t < \epsilon, \\ a \sim \pi(\cdot \mid s_k) & \text{otherwise.} \end{cases}$$

[3] Take action a_k and observe reward r_k and next state s_{k+1}.

[2] Update Q_k using $a' \sim \pi(\cdot \mid s_{k+1})$ as

$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha(r_k + \gamma Q_k(s_{k+1}, a') - Q_k(s_k, a_k)).$$

The agent will get the current state and select actions based on an exploration-exploitation strategy, which is called ϵ-greedy strategy. The agent will exploit the the Q function using policy with probability of ϵ, and agent will explore the environment by taking random action

with probability $1-\epsilon$. By using Q-learning we can find the gradient in step [1] of Algorithm 2.1.

2.2.2 Actor-Critic Method

Another way to solve the problem is to find the Q function and perform the policy gradient simultaneously. This approach is also known as the actor-critic method [15]. In this method, the actor estimates the value function using the current policy, and the critic updates the policy based on the estimated value function. Below is the algorithm

---

Algorithm 2.3. Actor-Critic Method

Input: Initial action-value function Q_0, initial policy π_0, exploration-exploitation constant ϵ, stepsize α

For $k = 0, 1, ..., K-1$ **do**

[1] Take action $a_k \sim \pi_k(\cdot \mid s_k)$ and observe reward r_k and next state s_{k+1}.

[2] Update Q_k using $a' \sim \pi(\cdot \mid s_{k+1})$ as

$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha(r_k + \gamma Q_k(s_{k+1}, a') - Q_k(s_k, a_k)).$$

[3] Update π_k for all $s \in \mathcal{S}$ as

$$\pi_{k+1}(\cdot \mid s) = \text{argmax}_{\mu(\cdot|s)\in\triangle_{|\mathcal{A}|}}\{\alpha_k\langle Q_k(s,\cdot), \mu(\cdot, s)\rangle - D^{\mu}_{\pi_k}(s)\}. \tag{2.4}$$

This algorithm effectively employs the actor-critic method by integrating Q-learning and policy mirror descent. It is noteworthy that there are numerous other actor-critic algorithms available. For instance, the natural actor-critic method [23], soft actor-critic method [10], and advantage actor-critic method [2] are all variations of the actor-critic method, each with their unique features and applications.

2.3 Multi-Task MDP

Similar to single MDP, we can model a multi-task reinforcement learning problem as a multi-task MDP, where an agent is presented N tasks, each is modeled by a discounted Markov decision process (MDP). In particular, the MDP $\mathcal{M}^i$ is a collection of 5-tuples, $\mathcal{M}^i = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}^i, \gamma)$ where $\mathcal{S}$ and $\mathcal{A}$ are sets of states and actions. The transition probability kernel $\mathcal{P}$ specifies the probability of which the next state s' will be based on the state and action at current iteration, i.e., $s' \sim \mathcal{P}(\cdot \mid s, a)$. In addition, $\mathcal{R}^i : \mathcal{S} \times \mathcal{A} \to [0, 1]$ is the reward function for task i and the discount factor $\gamma \in (0, 1)$ weights the importance of future reward. Here, without loss of generality, we consider the reward between $(0, 1)$, but it can be extended to any bounded interval. A policy $\pi(\cdot \mid s)$ is a probability that an action is chosen within the action space $\mathcal{A}$ at each different state $s \in \mathcal{S}$. Each choice of policy in task i induces a long-term expected discounted reward

$$V_\pi^i(s) = \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}^i(s_t, a_t) \mid s_0 = s, a_t \sim \pi(\cdot \mid s_t), s_{t+1} \sim \mathcal{P}(\cdot \mid s_t, a_t)\right],$$

and the state-action value function Q

$$Q_\pi^i(s, a) = \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}^i(s_t, a_t) \mid s_0 = s, a_0 = a, s_{t+1} \sim \mathcal{P}(\cdot \mid s_t, a_t), a_{t+1} \sim \pi(\cdot \mid s_t)\right].$$

Thus, $V_\pi^i(s) = \mathbb{E}_{a \sim \pi(\cdot|s)}[Q_\pi^i(s, a)]$. Since $\mathcal{R}^i \in (0, 1)$ we have $|Q_\pi^i(s, a)| \leq 1/(1-\gamma)$ for all s, a. Whats more, it is known that $Q_\pi^i(s, a)$ satisfies the Bellman equation

$$Q_\pi^i(s, a) = \mathcal{R}^i(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s' \mid s, a) V_\pi^i(s').$$

The goal of MTRL is to find a policy that simultaneously optimizes the aggregate of the value functions of the tasks at every state, i.e., the agent aims to find $\pi^\star$ such that $\sum_i V_{\pi^\star}^i(s)$

is maximized for every state $s \in \mathcal{S}$. It is known that for the finite MDP setting there exists such an optimal policy $\pi^\star$ [4]. Finding $\pi^\star$ is essentially equivalent to solve

$$\max_{\pi} f(\pi) := \sum_{i=1}^{N} f^i(\pi). \tag{2.5}$$

where $f^i(\pi) = \mathbf{E}_{s\sim\rho^\star(\cdot)}[V_\pi^i(s)]$ and $\rho^\star$ is the stationary distribution corresponding to the optimal policy $\pi^\star$. Note that the knowledge of $\rho^\star$ is not required to the implementation of the proposed algorithm studied in the next section.

We conclude this section by introducing a few notation that will facilitate our algorithmic development in the next section. Given a policy π, we denote by $d_\pi^s(s')$ the discounted state visitation

$$d_\pi^s(s') = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t \mathcal{P}_\pi(s_t = s' \mid s_0 = s),$$

which represents the amount of time that the agent visits state s' when it starts from s. Given two policies π, μ, let D_π^μ be the Bregman's distance defined as

$$D_\pi^\mu(s) = \omega(\mu(\cdot \mid s)) - \omega(\pi(\cdot \mid s)) - \langle \nabla_\pi \omega(\pi(\cdot \mid s)), \mu(\cdot \mid s) - \pi(\cdot \mid s)\rangle, \quad \forall s \in \mathcal{S},$$

where ω is a strongly convex function. In this section, we will consider ω as an entropy function

$$\omega(\pi(\cdot \mid s)) = \sum_{a\in\mathcal{A}} \pi(a \mid s)\log(\pi(a \mid s)).$$

Under this choice, D_π^μ can be simplified to

$$D_\pi^\mu(s) = \sum_{a\in\mathcal{A}} \mu(a \mid s)\log\frac{\mu(a \mid s)}{\pi(a \mid s)},$$

which by using the Pinsker's inequality we obtain

$$\|\mu(\cdot \mid s) - \pi(\cdot \mid s)\|_2^2 \leq \|\mu(\cdot \mid s) - \pi(\cdot \mid s)\|_1^2 \leq 2D_\pi^\mu(s), \tag{2.6}$$

where the first inequality is due to the fact that $\|\cdot\|_2 \leq \|\cdot\|_1$

There are several existing methods to solve Multi-Task MDP problem.

2.3.1 Transfer Learning for Multi-Task MDP

Transfer learning is a technique that aims to expedite the learning process by transferring knowledge from similar tasks. For instance, an individual with experience driving in a city can learn to drive on a highway much faster than starting from scratch. In this context, the agent learns the first task and uses the learned policy as the initial policy for subsequent tasks. This method is particularly useful when the available data is limited. A survey paper by [22] provides numerous applications of transfer learning, while another survey paper by [26] presents several papers that apply transfer learning to reinforcement learning (RL) problems. In this scenario, the agent learns new tasks but forgets the previous ones. Our aim is to find a policy that maximizes the average objective function across all tasks.

2.3.2 Life-Long Learning for Multi-Task MDP

Life-Long Learning [27] is a concept wherein an agent performs tasks sequentially, leveraging knowledge from earlier tasks to learn new ones. The study by [18] delves into the balance between old and new tasks, while the work by [8] presents an optimal exploration strategy for tackling new but related problems.

2.3.3 Centralized Policy Gradient for Multi-Task MDP

Centralized Policy Gradient for Multi-Task MDP can be reduced to the single MDP by averaging the gradient. Theoretical results of policy gradient methods are well understood, where these methods are shown to converge to the optimal policy at a linear rate [17]. One can apply policy gradient methods to solve MTRL problems, however, this approach requires to access the gradients of all the tasks at each iteration. This can be expensive to implement as the number of tasks can be very large. Our focus in this thesis is to study an incremental policy gradient approach, where each iteration requires the gradient of only one task in to update the underlying policy variable.

2.3.4 Distributed Policy Gradient Method for Multi-Task MDP

Another approach to solve MTRL problems is to use distributed policy gradient methods [31]. This approach often has a network of agents, each performs one task. The agents are then collaborate either directly or indirectly through a centralized coordinator to aggregate their local solutions. Prior study in distributed policy gradient methods have investigated convergence rates under various scenarios. For instance, research by [16] demonstrated $\mathcal{O}(1/k)$ convergence rates for their specific algorithm FedNPG-ADMM. Similarly, [31] explored the convergence behavior of their decentralized entropy regularized policy gradient methods, revealing $\mathcal{O}(1/\sqrt{k})$ convergence rates for certain architectures and loss functions. On the other hand, we study a different setting where there is only one agent that aims to learn a number of tasks by access the tasks randomly one by one. Therefore, the theoretical results in the distributed methods are not applicable to our setting in this thesis.

Chapter 3

Incremental Policy Mirror Descent

3.1 Incremental Policy Mirror Descent

As mentioned in section 2.2, to solve problem (2.5), one can apply the policy mirror descent method studied in [17]. This method iteratively updates π_k, an estimate of $\pi^\star$, starting from an arbitrary policy π_0 as

$$\pi_{k+1}(\cdot \mid s) = \text{argmax}_{\mu(\cdot|s)\in\Delta_{|\mathcal{A}|}} \{\alpha_k \Big\langle \sum_i Q^i_{\pi_k}(s,\cdot), \mu(\cdot,s) \Big\rangle - D^\mu_{\pi_k}(s)\},$$

To implement this update, one needs to estimate the state-value function Q^i for every task i at any iteration k, which is equivalent to solving N policy evaluation problems. This can be expensive in practice as the number of tasks can be large.

In this section, we are interested in studying an incremental variant of the policy mirror descent methods, where the agent can only have access to the state-value function of one task at a time. Our algorithm is formally stated in Algorithm 3.1. At any iteration k, the agent chooses a task i to compute the state-value function $Q^i_{\pi_k}$ using the current policy π_k. The agent then implements a policy mirror descent step to update π_k as in (3.1).

In Algorithm 3.1, we allow the agent to choose the task index i at any iteration arbitrarily. For example, the agent can apply the cyclic rule (i.e., choosing the task in increasing order) or random rule (i.e., randomly picking the task index). However, we do enforce that each

task is chosen infinitely often to guarantee that the agent will perform all the tasks. One way to have this condition is to consider the following assumption.

Algorithm 3.1. Incremental Policy Mirror Descent (IPMD)

Input: π_0, step sizes $\{\alpha_k\}_{k\geq 0}$

For $k = 0, 1, ..., K-1$ **do**

[1] Draw $i \in \{1, \cdots, N\}$

[2] Compute $Q_k^i \triangleq Q_{\pi_k}^i$ using the policy π_k

[3] Update π_k for all $s \in \mathcal{S}$ as

$$\pi_{k+1}(\cdot \mid s) = \text{argmax}_{\mu(\cdot|s)\in\triangle_{|\mathcal{A}|}}\{\alpha_k\langle Q_k^i(s,\cdot), \mu(\cdot,s)\rangle - D_{\pi_k}^{\mu}(s)\}. \tag{3.1}$$

Assumption 1. *Given a positive integer $\tau \geq N$, each $i \in \{1, \ldots, N\}$ in Step 1 of Algorithm 3.1 will be drawn at least one time within every interval $[k-\tau, k)$ for every $k \geq \tau$.*

One can view that τ represents an upper bound on the delay in computing the value of the state-action function Q^i at time k. We denote τ_k^i as the last time that task i is drawn at time k. If i is not selected at time k then $Q_{\tau_k^i}^i$ is the most recent Q value of task i using policy $\pi_{\tau_k^i}$. If $\tau = N$, then Algorithm 3.1 reduces to the classic incremental gradient method with cyclic rules.

Finally, we present below some preliminary results that will be useful in deriving our main result studied in the next section. First, we consider the so-called three-point lemma, which is used to characterize the updates of mirror descent. Its proof can be found in [17].

Lemma 3.2. *For any policy μ, the sequence $\{\pi_k\}$ generated by Algorithm 3.1 satisfies for all $s \in \mathcal{S}$*

$$\alpha_k \left\langle Q_k^i, \mu(\cdot \mid s) - \pi_{k+1}(\cdot \mid s) \right\rangle \leq D_{\pi_k}^{\mu}(s) - D_{\pi_{k+1}}^{\mu}(s) - D_{\pi_k}^{\pi_{k+1}}(s).$$

Proof: By optimality condition of argmax

$$\langle \alpha_k Q_k^i(s, \cdot) - \nabla_\pi D_{\pi_k}^{\pi_{k+1}}(s), \mu(\cdot \mid s) - \pi_{k+1}(\cdot \mid s) \rangle \leq 0.$$

Then by reorder them we have

$$\begin{aligned}
\alpha_k \langle Q_k^i(s, \cdot), \mu(\cdot \mid s) - \pi_{k+1}(\cdot \mid s) \rangle &\leq \langle \nabla_\pi D_{\pi_k}^{\pi_{k+1}}(s), \mu(\cdot \mid s) - \pi_{k+1}(\cdot \mid s) \rangle \\
&= \langle \nabla_\pi \omega(\pi_{k+1}(\cdot \mid s)) - \nabla_\pi \omega(\pi_k(\cdot \mid s)), \mu(\cdot \mid s) - \pi_{k+1}(\cdot \mid s) \rangle \\
&= \omega(\mu(\cdot \mid s)) - \omega(\pi_k(\cdot \mid s)) - \langle \nabla_\pi \omega(\pi_k(\cdot \mid s)), \mu(\cdot \mid s) - \pi_k(\cdot \mid s) \rangle \\
&\quad - \omega(\pi_{k+1}(\cdot \mid s)) + \omega(\pi_k(\cdot \mid s)) + \langle \nabla_\pi \omega(\pi_k(\cdot \mid s)), \pi_{k+1}(\cdot \mid s) - \pi_k(\cdot \mid s) \rangle \\
&\quad - \omega(\mu(\cdot \mid s)) + \omega(\pi_{k+1}(\cdot \mid s)) + \langle \nabla_\pi \omega(\pi_{k+1}(\cdot \mid s)), \mu(\cdot \mid s) - \pi_{k+1}(\cdot \mid s) \rangle \\
&= D_{\pi_k}^{\mu}(s) - D_{\pi_k}^{\pi_{k+1}}(s) - D_{\pi_{k+1}}^{\mu}(s).
\end{aligned}$$

■

Next, we present the popular performance difference lemma in reinforcement learning [17].

Lemma 3.3. *For any two policies π and π', we have*

$$V_{\pi'}^i(s) - V_\pi^i(s) = \frac{1}{1-\gamma} \sum_{s' \in \mathcal{S}} d_{\pi'}^s(s') \langle Q_\pi^i(s', \cdot), \pi'(\cdot \mid s') - \pi(\cdot \mid s') \rangle.$$

Proof: For simplicity denote $\zeta^{\pi'}(s_0)$ the random process (s_k, a_k, s_{k+1})

$$
\begin{aligned}
V_{\pi'}(s) - V_{\pi}(s) &= \mathbb{E}_{\xi_{\pi'}(s)}\left[\sum_{k=0}^{\infty}\gamma_k\left[\mathcal{R}^i\left(s_k, a_k\right)\right]\right] - V_{\pi}(s) \\
&= \mathbb{E}_{\xi_{\pi'}(s)}\left[\sum_{k=0}^{\infty}\gamma_k\left[\mathcal{R}^i\left(s_k, a_k\right) + V_{\pi}\left(s_k\right) - V_{\pi}\left(s_k\right)\right]\right] - V_{\pi}(s) \\
&= \mathbb{E}_{\xi_{\pi'}(s)}\left[\sum_{k=0}^{\infty}\gamma_k\left[\mathcal{R}^i\left(s_k, a_k\right) + \gamma V_{\pi}\left(s_{k+1}\right) - V_{\pi}\left(s_k\right)\right]\right] \\
&\quad + \mathbb{E}_{\xi_{\pi'}(s)}\left[V_{\pi}\left(s_0\right)\right] - V_{\pi}(s) \\
&= \mathbb{E}_{\xi_{\pi'}(s)}\left[\sum_{k=0}^{\infty}\gamma_k\left[\mathcal{R}^i\left(s_k, a_k\right) + \gamma V_{\pi}\left(s_{k+1}\right) - V_{\pi}\left(s_k\right)\right]\right] \\
&= \mathbb{E}_{\xi_{\pi'}(s)}\left[\sum_{k=0}^{\infty}\gamma_k\left[Q_{\pi}\left(s_k, a_k\right) - V_{\pi}\left(s_k\right)\right]\right] \\
&= \frac{1}{1-\gamma}\sum_{s'\in\mathcal{S}}\sum_{a'\in\mathcal{A}} d^s_{\pi'}(s')\pi'(a' \mid s')\left[Q_{\pi}\left(s', a'\right) - V_{\pi}\left(s'\right)\right] \\
&= \frac{1}{1-\gamma}\sum_{s'\in\mathcal{S}} d^s_{\pi'}(s')\left[\left\langle Q_{\pi}\left(s', \cdot\right), \pi'(\cdot \mid s')\right\rangle - V_{\pi}\left(s'\right)\right] \\
&= \frac{1}{1-\gamma}\sum_{s'\in\mathcal{S}} d^s_{\pi'}(s')\left[\left\langle Q_{\pi}\left(s', \cdot\right), \pi'(\cdot \mid s')\right\rangle - \left\langle Q_{\pi}\left(s', \cdot\right), \pi(\cdot \mid s')\right\rangle\right] \\
&= \frac{1}{1-\gamma}\sum_{s'\in\mathcal{S}} d^s_{\pi'}(s')\left\langle Q_{\pi}\left(s', \cdot\right), \pi'(\cdot \mid s') - \pi(\cdot \mid s')\right\rangle.
\end{aligned}
$$

■

Finally, the following two lemmas are to characterize the properties of the sequence $\{\pi_k\}$ generated by Algorithm 3.1.

Lemma 3.4. *For any $s \in \mathcal{S}$, the sequence $\left\{\pi_{\tau_k^i}\right\}$ generated by Algorithm 3.1 satisfies for all $s \in \mathcal{S}$*

$$
V^i_{\tau_k^i+1}(s) - V^i_{\tau_k^i}(s) \leq \left\langle Q^i_{\tau_k^i}(s, \cdot), \pi_{\tau_k^i+1}(\cdot \mid s) - \pi_{\tau_k^i}(\cdot \mid s)\right\rangle \leq \frac{-1}{\alpha_k}\left[D^{\pi_{\tau_k^i}}_{\pi_{\tau_k^i+1}}(s) + D^{\pi_{\tau_k^i+1}}_{\pi_{\tau_k^i}}(s)\right],
$$

Proof: By Lemma choose $\pi' = \pi_{\tau_k^i+1}$ and $\pi = \pi_{\tau_k^i}$ we have

$$V^i_{\tau_k^i+1}(s) - V^i_{\tau_k^i}(s) = \frac{1}{1-\gamma}\sum_{s'\in\mathcal{S}} d^s_{\tau_k^i+1}(s')\langle Q^i_{\tau_k^i}(s',\cdot), \pi_{\tau_k^i+1}(\cdot \mid s') - \pi_{\tau_k^i}(\cdot \mid s')\rangle.$$

Then we have

$$\begin{aligned}
V^i_{\tau_k^i}(s) - V^i_{\tau_k^i+1}(s) &= \frac{1}{1-\gamma}\sum_{s'\in\mathcal{S}} d^s_{\tau_k^i+1}(s')\langle Q^i_{\tau_k^i}(s',\cdot), \pi_{\tau_k^i}(\cdot \mid s') - \pi_{\tau_k^i+1}(\cdot \mid s')\rangle \\
&\leq \frac{1}{1-\gamma} d^s_{\tau_k^i+1}(s)\langle Q^i_{\tau_k^i}(s,\cdot), \pi_{\tau_k^i}(\cdot \mid s) - \pi_{\tau_k^i+1}(\cdot \mid s)\rangle \\
&\leq \langle Q^i_{\tau_k^i}(s,\cdot), \pi_{\tau_k^i}(\cdot \mid s) - \pi_{\tau_k^i+1}(\cdot \mid s)\rangle \\
&\leq -\frac{1}{\alpha_k}\left[D^{\pi_{\tau_k^i}}_{\pi_{\tau_k^i+1}}(s) + D^{\pi_{\tau_k^i+1}}_{\pi_{\tau_k^i}}(s)\right],
\end{aligned}$$

where the third inequality is by lemma 1 at iteration τ_k^i and choose $\mu = \tau_k^i$, we have

$$\langle Q^i_{\tau_k^i}(s',\cdot), \pi_{\tau_k^i}(\cdot \mid s') - \pi_{\tau_k^i+1}(\cdot \mid s')\rangle \leq -\frac{1}{\alpha_k}\left[D^{\pi_{\tau_k^i}}_{\pi_{\tau_k^i+1}}(s) + D^{\pi_{\tau_k^i+1}}_{\pi_{\tau_k^i}}(s)\right] \leq 0.$$

Then the last inequality above holds since D is always positive due to definition of Bregman's distance, we just drop some negative term in the first inequality. The second inequality is by definition of discounted state visitation d, we have

$$\begin{aligned}
d^s_{\tau_k^i+1}(s) &= (1-\gamma)\sum_{k=0}^{\infty}\gamma_k \mathcal{P}_{\tau_k^i+1}(s_k = s \mid s_0 = s) \\
&= (1-\gamma) + (1-\gamma)\sum_{k=1}^{\infty}\gamma_k \mathcal{P}_{\tau_k^i+1}(s_k = s \mid s_0 = s) \\
&\geq 1-\gamma,
\end{aligned}$$

then $-d^s_{\tau_k^i+1}(s) \leq 1-\gamma$. ■

Lemma 3.5. *The sequence* $\{\pi_k\}$ *generated by Algorithm 3.1 satisfies for all* $s \in \mathcal{S}$

$$\|\pi_{k+1}(\cdot \mid s) - \pi_k(\cdot \mid s)\|_2 \leq \|\pi_{k+1}(\cdot \mid s) - \pi_k(\cdot \mid s)\|_1 \leq \frac{\alpha_k}{1-\gamma}.$$

Proof: Without loss of generality, let task i be chosen at time $t \in [k-\tau, k]$. Thus, using Lemma 1 with $\mu = \pi_t$ we have

$$\alpha_t \left\langle Q^i_{\pi_t}(s,\cdot), \pi_t(\cdot \mid s) - \pi_{t+1}(\cdot \mid s) \right\rangle \leq -D^{\pi_t}_{\pi_{t+1}}(s) - D^{\pi_{t+1}}_{\pi_t}(s) \leq -\|\pi_{t+1}(\cdot \mid s) - \pi_t(\cdot \mid s)\|_1^2,$$

where the last inequality is due to 2.6. Since $\left|Q^i_{\pi_t}(s,a)\right| \leq \frac{1}{1-\gamma}$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$, multiply both side by -1 and use the Holder's inequality the preceding relation gives

$$\begin{aligned}\|\pi_{t+1}(\cdot \mid s) - \pi_t(\cdot \mid s)\|_1^2 &\leq \alpha_t \|\pi_{t+1}(\cdot \mid s) - \pi_t(\cdot \mid s)\|_1 \left\|Q^i_{\pi_t}(s,\cdot)\right\|_\infty \\ &\leq \frac{\alpha_t}{1-\gamma} \|\pi_{t+1}(\cdot \mid s) - \pi_t(\cdot \mid s)\|_1 .\end{aligned}$$

■

3.2 Main Results

In this section, we will present the main result of this thesis, where we will study the convergence rate of Algorithm 3.1. In particular, we show that under Assumption 1, Algorithm 3.1 can return an optimal policy of problem (2.5) at a sublinear rate. The following theorem is to present this result.

Theorem 3.6. *Suppose that Assumption 1 holds. Let $\{\pi_k\}_{k\geq 0}$ be generated by Algorithm 3.1 and step sizes be chosen as $\alpha_k = \frac{2}{\sqrt{k+\tau+1}}$. Then we have*

$$\max_{t=1,\dots,k}\left[f(\pi^\star)-f(\pi_t)\right] \leq \frac{1}{2(1-\gamma)\sqrt{k}}\left(\sqrt{8}[f(\pi^\star)-f(\pi_0)] + N\mathbb{E}_{s\sim\rho^\star(\cdot)}\left[D_{\pi_0}^{\pi^\star}(s)\right]\right) + \frac{4\sqrt{2}N + N\tau(8(\tau+1)+\ln(k))}{2(1-\gamma)^3\sqrt{k}}. \tag{3.2}$$

Proof: Recall that τ_k^i is the last time that task i is selected in the time interval $[k-\tau, k]$. Using Lemma 3.2 with $\mu = \pi^\star$ we obtain for all $s \in \mathcal{S}$

$$\begin{aligned}
&D_{\pi_{\tau_k^i}}^{\pi^\star}(s) - D_{\pi_{\tau_k^i+1}}^{\pi^\star}(s) - D_{\pi_{\tau_k^i+1}}^{\pi_{\tau_k^i}}(s)\\
&\geq \alpha_{\tau_k^i}\left\langle Q_{\tau_k^i}^i(s,\cdot), \pi^\star(\cdot\mid s) - \pi_{\tau_k^i+1}(\cdot\mid s)\right\rangle\\
&\geq \alpha_{\tau_k^i}\left\langle Q_{\tau_k^i}^i(s,\cdot), \pi^\star(\cdot\mid s) - \pi_{\tau_k^i}(\cdot\mid s)\right\rangle + \alpha_{\tau_k^i}\left\langle Q_{\tau_k^i}^i(s,\cdot), \pi_{\tau_k^i}(\cdot\mid s) - \pi_{\tau_k^i+1}(\cdot\mid s)\right\rangle\\
&\geq \alpha_{\tau_k^i k}\left\langle Q_{\tau_k^i}^i(s,\cdot), \pi^\star(\cdot\mid s) - \pi_{\tau_k^i}(\cdot\mid s)\right\rangle + \alpha_{\tau_k^i}\left(V_{\tau_k^i}^i(s) - V_{\tau_k^i+1}^i(s)\right),
\end{aligned} \tag{3.3}$$

where the last inequality used Lemma 3.4. By Lemma 3.3 we have

$$(1-\gamma)\left(V_{\pi^\star}^i(s) - V_{\tau_k^i}^i(s)\right) = \mathbb{E}_{s'\sim d_{\pi^\star}^s(\cdot)}\left[\left\langle Q_{\tau_k^i}^i(s',\cdot), \pi^\star(\cdot\mid s') - \pi_{\tau_k^i}(\cdot\mid s')\right\rangle\right],$$

which by using the fact that $\mathbb{E}_{s\sim\rho^\star(\cdot)}\mathbb{E}_{s'\sim d^s_{\pi^\star}(\cdot)} = \mathbb{E}_{s\sim\rho^\star(\cdot)}$ gives

$$(1-\gamma)\mathbb{E}_{s\sim\rho^\star(\cdot)}\left[V^i_{\pi^\star}(s) - V^i_{\tau^i_k}(s)\right] = \mathbb{E}_{s\sim\rho^\star(\cdot)}\left[\left\langle Q^i_{\tau^i_k}(s,\cdot), \pi^\star(\cdot \mid s) - \pi_{\tau^i_k}(\cdot \mid s)\right\rangle\right].$$

Taking the expectation w.r.t. $s \sim \rho^\star(\cdot)$ on both sides of equation (3.3) and using the preceding inequality we obtain

$$\begin{aligned}
&\mathbb{E}_{s\sim\rho^\star(\cdot)}\left[D^{\pi^\star}_{\pi_{\tau^i_k}}(s) - D^{\pi^\star}_{\pi_{\tau^i_k+1}}(s) - D^{\pi_{\tau^i_k}}_{\pi_{\tau^i_k+1}}(s)\right] \\
&\quad\geq (1-\gamma)\alpha_{\tau^i_k}\mathbb{E}_{s\sim\rho^\star(\cdot)}\left[V^i_{\pi^\star}(s) - V^i_{\tau^i_k}(s)\right] + \alpha_{\tau^i_k}\mathbb{E}_{s\sim\rho^\star(\cdot)}\left[V^i_{\tau^i_k}(s) - V^i_{\tau^i_k+1}(s)\right] \\
&\quad= \alpha_{\tau^i_k}\mathbb{E}_{s\sim\rho^\star(\cdot)}\left[V^i_{\pi^\star}(s) - V^i_{\tau^i_k+1}(s)\right] - \gamma\alpha_{\tau^i_k}\mathbb{E}_{s\sim\rho^\star(\cdot)}\left[V^i_{\pi^\star}(s) - V^i_{\tau^i_k}(s)\right] \\
&\quad= \alpha_{\tau^i_k}\mathbb{E}_{s\sim\rho^\star(\cdot)}\left[V^i_{\pi^\star}(s) - V^i_{\pi_{k+1}}(s)\right] - \gamma\alpha_{\tau^i_k}\mathbb{E}_{s\sim\rho^\star(\cdot)}\left[V^i_{\pi^\star}(s) - V^i_{\pi_k}(s)\right] \\
&\qquad+ \alpha_{\tau^i_k}\mathbb{E}_{s\sim\rho^\star(\cdot)}\left[V^i_{\pi_{k+1}}(s) - V^i_{\tau^i_k+1}(s)\right] - \gamma\alpha_{\tau^i_k}\mathbb{E}_{s\sim\rho^\star(\cdot)}\left[V^i_{\pi_k}(s) - V^i_{\tau^i_k}(s)\right].
\end{aligned}$$

Reorganizing both sides we obtain

$$\begin{aligned}
&\alpha_{\tau^i_k}\mathbb{E}_{s\sim\rho^\star(\cdot)}\left[V^i_{\pi^\star}(s) - V^i_{\pi_{k+1}}(s)\right] \\
&\quad\leq \gamma\alpha_{\tau^i_k}\mathbb{E}_{s\sim\rho^\star(\cdot)}\left[V^i_{\pi^\star}(s) - V^i_{\pi_k}(s)\right] + \mathbb{E}_{s\sim\rho^\star(\cdot)}\left[D^{\pi^\star}_{\pi_{\tau^i_k}}(s) - D^{\pi^\star}_{\pi_{\tau^i_k+1}}(s) - D^{\pi_{\tau^i_k}}_{\pi_{\tau^i_k+1}}(s)\right] \\
&\qquad+ \gamma\alpha_{\tau^i_k}\mathbb{E}_{s\sim\rho^\star(\cdot)}\left[V^i_{\pi_k}(s) - V^i_{\tau^i_k}(s)\right] + \alpha_{\tau^i_k}\mathbb{E}_{s\sim\rho^\star(\cdot)}\left[V^i_{\tau^i_k+1}(s) - V^i_{\pi_{k+1}}(s)\right].
\end{aligned} \tag{3.4}$$

We next analyze each term on the right-hand side of the preceding equation. First, by Lemma 3.3 we obtain

$$\begin{aligned}
\left|V^i_{\pi_{k+1}}(s) - V^i_{\tau^i_k+1}(s)\right| &= \left|\frac{1}{1-\gamma}\mathbb{E}_{s'\sim d^s_{\pi_{k+1}}(\cdot)}\left[\left\langle Q^i_{\tau^i_k+1}(s',\cdot), \pi_{k+1}(\cdot \mid s') - \pi_{\tau^i_k+1}(\cdot \mid s')\right\rangle\right]\right| \\
&\leq \frac{1}{1-\gamma}\mathbb{E}_{s'\sim d^s_{\pi_{k+1}}(\cdot)}\left\|Q^i_{\tau^i_k+1}(s',\cdot)\right\|_\infty \left\|\pi_{k+1}(\cdot \mid s') - \pi_{\tau^i_k+1}(\cdot \mid s')\right\|_1
\end{aligned}$$

$$
\begin{aligned}
&\leq \frac{1}{(1-\gamma)^2}\mathbb{E}_{s'\sim d^s_{\pi_{k+1}}(\cdot)}\left\|\pi_{k+1}\left(\cdot \mid s'\right)-\pi_{\tau_k^i+1}\left(\cdot \mid s'\right)\right\|_1 \\
&= \frac{1}{(1-\gamma)^2}\mathbb{E}_{s'\sim d^s_{\pi_{k+1}}(\cdot)}\left\|\sum_{t=\tau_k^i+1}^{k}\pi_{\ell+1}\left(\cdot \mid s'\right)-\pi_{\ell}\left(\cdot \mid s'\right)\right\|_1 \\
&\leq \frac{1}{(1-\gamma)^2}\mathbb{E}_{s'\sim d^s_{\pi_{k+1}}(\cdot)}\sum_{t=\tau_k^i+1}^{k}\left\|\pi_{\ell+1}\left(\cdot \mid s'\right)-\pi_{\ell}\left(\cdot \mid s'\right)\right\|_1 \\
&\leq \frac{1}{(1-\gamma)^2}\mathbb{E}_{s'\sim d^s_{\pi_{k+1}}(\cdot)}\sum_{t=k-\tau+1}^{k}\left\|\pi_{\ell+1}\left(\cdot \mid s'\right)-\pi_{\ell}\left(\cdot \mid s'\right)\right\|_1 \\
&\leq \frac{1}{(1-\gamma)^3}\mathbb{E}_{s'\sim d^s_{\pi_{k+1}}(\cdot)}\sum_{t=k-\tau+1}^{k}\alpha_t \\
&\leq \frac{\tau\alpha_{k-\tau}}{(1-\gamma)^2}, \qquad (3.5)
\end{aligned}
$$

where the first inequality we used Holder's inequality, the second we use $\left|Q^i_{\pi_t}(s,a)\right| \leq \frac{1}{1-\gamma}$ for all $(s,a)\in\mathcal{S}\times\mathcal{A}$, the third inequality used triangle inequality, the fifth inequality used Lemma 3.5, the last inequality is due to $\alpha_k \leq \alpha_{\tau_k^i} \leq \alpha_{k-\tau}$ for all $\tau_k^i \in (k-\tau, k]$. Similarly, we have

$$
\left|V^i_{\tau_k}(s) - V^i_{\pi_k}(s)\right| \leq \frac{\tau\alpha_{k-\tau}}{(1-\gamma)^2} \qquad (3.6)
$$

Substituting Eqs. (3.5) and (3.6) into Eq. (3.4) and use the fact that $\alpha_{\tau_k^i} \leq \alpha_{k-\tau}$ we obtain

$$
\begin{aligned}
\alpha_{\tau_k^i}\mathbb{E}_{s\sim\rho^\star(\cdot)}\left[V^i_{\pi^\star}(s) - V^i_{\pi_{k+1}}(s)\right] \leq \gamma\alpha_{\tau_k^i}\mathbb{E}_{s\sim\rho^\star(\cdot)}\left[V^i_{\pi^\star}(s) - V^i_{\pi_k}(s)\right] + \frac{(1+\gamma)\tau(\alpha_{k-\tau})^2}{(1-\gamma)^2} \\
+ \mathbb{E}_{s\sim\rho^\star(\cdot)}\left[D^{\pi^\star}_{\pi_{\tau_k^i}}(s) - D^{\pi^\star}_{\pi_{\tau_k^i+1}}(s) - D^{\pi_{\tau_k^i}}_{\pi_{\tau_k^i+1}}(s)\right].
\end{aligned}
$$

Combining with that fact that $\alpha_{\tau_k^i} \geq \alpha_k \geq \alpha_{k+1}$ gives

$$
\begin{aligned}
&\alpha_{k+1}\mathbb{E}_{s\sim\rho^\star(\cdot)}\left[V^i_{\pi^\star}(s) - V^i_{\pi_{k+1}}(s)\right] \\
&\quad\leq \gamma\alpha_k\mathbb{E}_{s\sim\rho^\star(\cdot)}\left[V^i_{\pi^\star}(s) - V^i_{\pi_k}(s)\right] + \frac{(1+\gamma)\tau(\alpha_{k-\tau})^2}{(1-\gamma)^2} + \mathbb{E}_{s\sim\rho^\star(\cdot)}\left[D^{\pi^\star}_{\pi_{\tau_k^i}}(s) - D^{\pi^\star}_{\pi_{\tau_k^i+1}}(s) - D^{\pi_{\tau_k^i}}_{\pi_{\tau_k^i+1}}(s)\right] \\
&\qquad+ \gamma(\alpha_{\tau_k^i} - \alpha_k)\mathbb{E}_{s\sim\rho^\star(\cdot)}\left[V^i_{\pi^\star}(s) - V^i_{\pi_k}(s)\right] \\
&\quad\leq \gamma\alpha_k\mathbb{E}_{s\sim\rho^\star(\cdot)}\left[V^i_{\pi^\star}(s) - V^i_{\pi_k}(s)\right] + \frac{(1+\gamma)\tau(\alpha_{k-\tau})^2}{(1-\gamma)^2} + \mathbb{E}_{s\sim\rho^\star(\cdot)}\left[D^{\pi^\star}_{\pi_{\tau_k^i}}(s) - D^{\pi^\star}_{\pi_{\tau_k^i+1}}(s) - D^{\pi_{\tau_k^i}}_{\pi_{\tau_k^i+1}}(s)\right] \\
&\qquad+ \frac{2\gamma}{1-\gamma}(\alpha_{\tau_k^i} - \alpha_k),
\end{aligned}
$$

where the last inequality we use $|V^i_\pi(s)| \leq 1/(1-\gamma)$ for any policy π and $s \in \mathcal{S}$. Taking the average of the preceding equations over $i \in [1, N]$ we obtain

$$
\begin{aligned}
\alpha_{k+1}\left[f(\pi^\star) - f(\pi_{k+1})\right] \leq \gamma\alpha_k\left[f(\pi^\star) - f(\pi_k)\right] &+ \sum_{i=1}^N \frac{(1+\gamma)\tau(\alpha_{k-\tau})^2}{(1-\gamma)^2} \\
&+ \sum_{i=1}^N \mathbb{E}_{s\sim\rho^\star(\cdot)}\left[D^{\pi^\star}_{\pi_{\tau_k^i}}(s) - D^{\pi^\star}_{\pi_{\tau_k^i+1}}(s) - D^{\pi_{\tau_k^i}}_{\pi_{\tau_k^i+1}}(s)\right] + \sum_{i=1}^N \frac{2\gamma}{1-\gamma}(\alpha_{\tau_k^i} - \alpha_k).
\end{aligned}
$$

Then minus both side by $\gamma\alpha_{k+1}[f(\pi^\star) - f(\pi_{k+1})]$, use the fact that $\alpha_{\tau_k^i} \leq \alpha_{k-\tau}$ and drop $-\sum_{i=1}^N \mathbb{E}_{s\sim\rho^\star(\cdot)}\left[D^{\pi_{\tau_k^i}}_{\pi_{\tau_k^i+1}}(s)\right]$ we have

$$
\begin{aligned}
&(1-\gamma)\alpha_{k+1}\left[f(\pi^\star) - f(\pi_{k+1})\right] \\
&\quad\leq \gamma\left(\alpha_k[f(\pi^\star) - f(\pi_k)] - \alpha_{k+1}[f(\pi^\star) - f(\pi_{k+1})]\right) + \frac{N(1+\gamma)\tau(\alpha_{k-\tau})^2}{(1-\gamma)^2} \\
&\qquad+ \sum_{i=1}^N \mathbb{E}_{s\sim\rho^\star(\cdot)}\left[D^{\pi^\star}_{\pi_{\tau_k^i}}(s) - D^{\pi^\star}_{\pi_{\tau_k^i+1}}(s)\right] + \frac{2N\gamma}{1-\gamma}(\alpha_{k-\tau} - \alpha_k).
\end{aligned}
$$

Summing up both sides over $\tau_k^i = k - \tau + 1, \ldots, k$ gives

$$
\begin{aligned}
&(1-\gamma)\tau\alpha_{k+1}\left[f(\pi^\star)-f(\pi_{k+1})\right] \\
&\quad\le \gamma\tau\left(\alpha_k[f(\pi^\star)-f(\pi_k)]-\alpha_{k+1}[f(\pi^\star)-f(\pi_{k+1})]\right)+\frac{2N\gamma}{1-\gamma}(\alpha_{k-\tau}-\alpha_k) \\
&\qquad+\frac{N(1+\gamma)\tau^2(\alpha_{k-\tau})^2}{(1-\gamma)^2}+\sum_{i=1}^{N}\mathbb{E}_{s\sim\rho^\star(\cdot)}\left[D^{\pi^\star}_{\pi_{k-\tau+1}}(s)-D^{\pi^\star}_{\pi_{k+1}}(s)\right].
\end{aligned}
$$

Further summing up the time from 0 to k we have

$$
\begin{aligned}
&(1-\gamma)\tau\sum_{t=0}^{k-1}\alpha_{t+1}\left[f(\pi^\star)-f(\pi_{t+1})\right] \\
&\quad\le \gamma\tau\left(\alpha_0[f(\pi^\star)-f(\pi_0)]-\alpha_{k+1}[f(\pi^\star)-f(\pi_{k+1})]\right)+\sum_{t=0}^{k-1}\frac{2N\gamma}{1-\gamma}(\alpha_{t-\tau}-\alpha_t) \\
&\qquad+\sum_{t=0}^{k-1}\frac{N(1+\gamma)\tau^2(\alpha_{t-\tau})^2}{(1-\gamma)^2}+\sum_{i=1}^{N}\sum_{t=0}^{k-1}\mathbb{E}_{s\sim\rho^\star(\cdot)}\left[D^{\pi^\star}_{\pi_{t-\tau+1}}(s)-D^{\pi^\star}_{\pi_{t+1}}(s)\right] \\
&\quad=\gamma\tau\left(\alpha_0[f(\pi^\star)-f(\pi_0)]-\alpha_{k+1}[f(\pi^\star)-f(\pi_{k+1})]\right)+\sum_{t=0}^{k-1}\frac{2N\gamma}{1-\gamma}(\alpha_{t-\tau}-\alpha_t) \\
&\qquad+\sum_{t=0}^{k-1}\frac{N(1+\gamma)\tau^2(\alpha_{t-\tau})^2}{(1-\gamma)^2}+\sum_{i=1}^{N}\mathbb{E}_{s\sim\rho^\star(\cdot)}\left[\sum_{t=0}^{\tau-1}D^{\pi^\star}_{\pi_{t-\tau+1}}(s)-\sum_{t=k-\tau+1}^{k-1}D^{\pi^\star}_{\pi_{t+1}}(s)\right].
\end{aligned}
$$

We can set $\pi_t=\pi_0$ for all $t\le 0$ and drop negative term, we have

$$
\begin{aligned}
(1-\gamma)\tau\sum_{t=0}^{k-1}\alpha_{t+1}\left[f(\pi^\star)-f(\pi_{t+1})\right]&\le\gamma\tau\alpha_0[f(\pi^\star)-f(\pi_0)]+N\tau\mathbb{E}_{s\sim\rho^\star(\cdot)}\left[D^{\pi^\star}_{\pi_0}(s)\right] \\
&\quad+\sum_{t=0}^{k-1}\frac{2N\gamma}{1-\gamma}(\alpha_{t-\tau}-\alpha_t)+\sum_{t=0}^{k-1}\frac{N(1+\gamma)\tau^2(\alpha_{t-\tau})^2}{(1-\gamma)^2}. \qquad (3.7)
\end{aligned}
$$

Then we choose $\alpha_k=\frac{2}{\sqrt{k+\tau+1}}$ for all $k\ge 0$ and $\alpha_k=\alpha_0$ for $k\le 0$, since α_k is decreasing

function w.r.t. k so we can use the bound of integral test, we have

$$\begin{aligned}
\sum_{t=0}^{k-1}\alpha_{t+1} &\geq \frac{2}{\sqrt{\tau+2}} + \int_1^k \frac{2}{\sqrt{t+\tau+2}}dt = \frac{2}{\sqrt{\tau+2}} + 4\sqrt{t+\tau+2}\Big|_1^k \\
&= \frac{2}{\sqrt{\tau+2}} + 4\sqrt{k+\tau+2} - 4\sqrt{\tau+3} \\
&\geq 4\sqrt{k+\tau+2} - 4\sqrt{\tau+3} \\
\sum_{t=0}^{k-1}(\alpha_{t-\tau})^2 &= \sum_{t=0}^{\tau-1}(\alpha_0)^2 + \sum_{t=\tau}^{k}(\alpha_{t-\tau})^2 \leq \sum_{t=0}^{\tau-1}(\alpha_0)^2 + (\alpha_0)^2 + \int_\tau^{k-1}\frac{4}{t+\tau+1}dt \\
&\leq \frac{4(\tau+1)}{\tau+1} + 4\ln(t+\tau+1)\Big|_\tau^{k-1} = 4 + 4\ln(k+\tau) - 4\ln(2\tau+1) \\
&\leq 4 + 4\ln(k+\tau) \\
\sum_{t=0}^{k-1}(\alpha_{t-\tau} - \alpha_t) &= \sum_{t=0}^{\tau-1}\alpha_{t-\tau} + \sum_{t=\tau}^{k-1}\alpha_{t-\tau} - \sum_{t=0}^{k-1}\alpha_t = \sum_{t=0}^{\tau-1}\alpha_0 + \sum_{t=0}^{k-\tau-1}\alpha_t - \sum_{t=0}^{k-1}\alpha_t \\
&= \frac{2\tau}{\sqrt{\tau+1}} - \sum_{t=k-\tau}^{k-1}\alpha_t \leq \frac{2\tau}{\sqrt{\tau+1}} \\
&\leq 2\sqrt{\tau}.
\end{aligned}$$

Plug back to equation (3.7) we have

$$\begin{aligned}
&4(1-\gamma)\tau(\sqrt{k+\tau+2} - \sqrt{\tau+3})\max_{t=1,\cdot,k}[f(\pi^\star) - f(\pi_t)] \\
&\leq \gamma\tau\alpha_0[f(\pi^\star) - f(\pi_0)] + N\tau\mathbb{E}_{s\sim\rho^\star(\cdot)}\left[D_{\pi_0}^{\pi^\star}(s)\right] + \frac{4N\gamma\sqrt{\tau}}{1-\gamma} + \frac{4N(1+\gamma)\tau^2(1+\ln(k+\tau))}{(1-\gamma)^2}
\end{aligned}$$

Divide both side by $4(1-\gamma)\tau(\sqrt{k+\tau+2} - \sqrt{\tau+3})$ we have

$$\begin{aligned}
&\max_{t=1,\cdot,k}[f(\pi^\star) - f(\pi_t)] \\
&\leq \frac{\gamma\tau\alpha_0}{4(1-\gamma)\tau(\sqrt{k+\tau+2} - \sqrt{\tau+3})}[f(\pi^\star) - f(\pi_0)] \\
&\quad + \frac{N\tau}{4(1-\gamma)\tau(\sqrt{k+\tau+2} - \sqrt{\tau+3})}\mathbb{E}_{s\sim\rho^\star(\cdot)}\left[D_{\pi_0}^{\pi^\star}(s)\right]
\end{aligned}$$

$$+\frac{N\gamma\sqrt{\tau}}{(1-\gamma)^2\tau(\sqrt{k+\tau+2}-\sqrt{\tau+3})}+\frac{N(1+\gamma)\tau^2(1+\ln(k+\tau))}{(1-\gamma)^3\tau(\sqrt{k+\tau+2}-\sqrt{\tau+3})}.$$

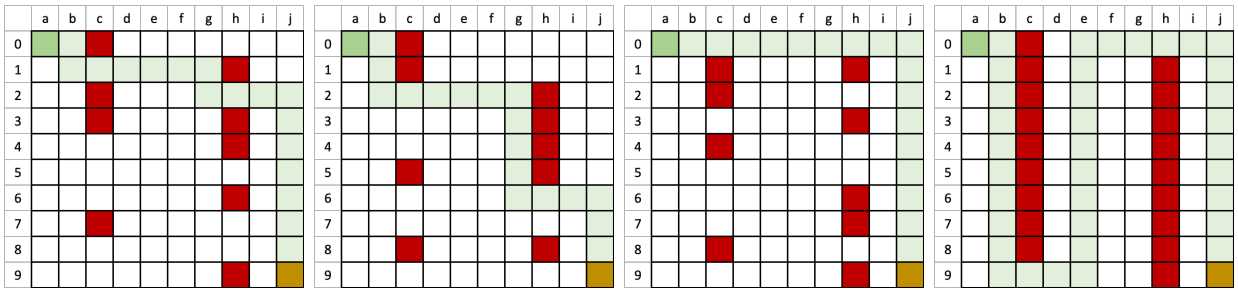
■

Chapter 4

Simulation

In this section, we will illustrate the convergence of IPMD in Algorithm 3.1 in solving a multi-task GridWorld problem. The goal of this simulation is to investigate the performance of IPMD.

GridWorld Environments. We create six GridWorld environments, each has size 10×10. Each environment has the same initial starting point (the blue square in the top left corner) and the goal (the yellow square in the bottom right corner). The goal of the agent is to find a policy that can help it navigate from the starting point to the target while avoiding all the obstacles (the red square). The obstacles in each environment are located in different positions in the grid. The first three figures in Figure 4.1 presents some example of GridWorld environments, while the last figure in Figure 4.1 shows the obstacles at all the environments put together. There is an optimal path that can solve the task simultaneously (e.g., light green path in the last figure), while there are multiple different solutions for each task (e.g., light green paths in the first three figures). To solve this multi-task problem, the agent seeks to find the light green path in the last figure. For our MTRL setting, we assign the reward being $+1$ at the target while the agent gets a -1 if it runs into the obstacles. For our simulation, we implement Algorithm 3.1 with a cyclic rule, where the environments are chosen in increasing order. In addition, since the transition probability matrices of these environments are known, we can easily compute the state-action value function Q at each



GridWorld 0 GridWorld 1 GridWorld 2 Total GridWorld

Figure 4.1: GridWorlds

iteration, e.g., by solving the Bellman equation. The results of our simulation are shown in Figure 4.2, where we show the total aggregated rewards on the left and the error (the difference between these rewards and the optimal value) on the right. We can see that the proposed algorithm returns an optimal policy that can solve the multi-task GridWorld problem, which agrees with our theoretical results in Theorem 3.6. In this simulation, the rate of convergence seems to be faster than $1/\sqrt{k}$, which implies that the current analysis might not be tight. This simulation suggests that one might need a new analysis and method to study the best theoretical results on the performance of IPMD.

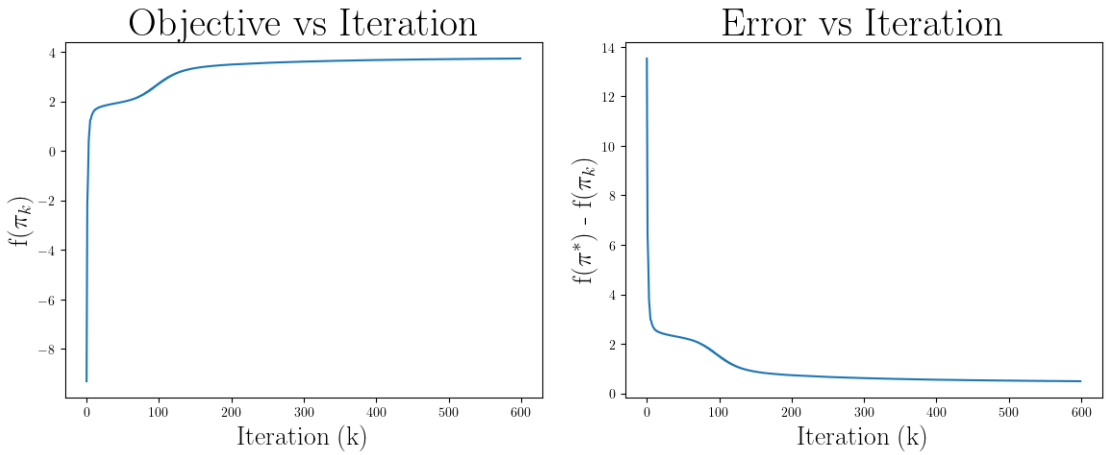


Figure 4.2: Performance of IPMD Methods

Chapter 5

Discussion and Future Direction

For our main result, it can be seen from Eq. (3.2) that the IPMD finds the optimal value of problem (2.5) at a rate $\mathcal{O}(1/\sqrt{k})$. We note that this rate is much weaker than the result of centralized policy mirror descent studied in [17, 30], where the rate is exponential. However, we note that the existing analysis requires access to the full gradient of the objective in (2.5), i.e., $\sum_i Q^i_{\pi_k}$, at every iteration. On the other hand, the implementation of IPMD requires the Q function of only one task per iteration. Indeed, IPMD can be viewed as a stochastic variant of the existing policy mirror descent. To achieve an exponential convergence rate of the stochastic counterpart, the work in [17] assumes that the variance of the gradient samples decay exponentially fast. This assumption obviously does not hold in the context of IPMD. One can potentially apply the variance reduction techniques in [6, 12] to achieve this condition. This approach, however, is nontrivial since the analysis in [17] is not applicable due to the heterogeneity of the value functions of the tasks. Finally, our theoretical result also shows that the rate scales linearly with the number of tasks, quadratically on the delay interval τ, and cubically on the problem horizon $1/(1-\gamma)$. The convergence of policy mirror descent, however, only scales linearly with $1/(1-\gamma)$ [17, 30]. Addressing these gaps of convergence will be an interesting topic, which we leave for our future studies.

Note that there is a limitation by the assumption that within the time period τ all tasks need been visited. Which means that every task need to be visited infinitely often. For future work, we aim to relax the assumption so that we no longer need to access each tasks

infinitely often. One approach is to reuse the data previously collected when the task is not appear within the time period τ. This method will cause distribution error similar to off policy training since the policy not matched. Another approach is when agent can only visit a task for limited times, then we can still let the policy converge to some sub-optimal policy. It is interesting to study the relation between the times limited to visit and the difference between the optimal policy and the sub-optimal policy. By addressing these challenges, we can further advance the applicability and effectiveness of multi-task learning algorithms in complex and dynamic settings.

In our simulations, we demonstrated the efficacy of our algorithm in addressing multi-task learning problems where task objectives did not conflict. However, real-world scenarios often present challenges where objectives across tasks may conflict. In such instances, our algorithm prioritizes maximizing the average objective function across all tasks, potentially leading to compromises where one task's objectives are favored over another's. To address this issue, one potential avenue for exploration involves enabling the agent to employ distinct policies for different tasks so the problem reduce to single MDP for each tasks. However, this approach requires the agent to possess additional knowledge to discern which task it is currently undertaking.

Looking ahead, I aim to extend our research by considering scenarios where the agent receives noisy Q-functions for each task, introducing uncertainty into the learning process. Leveraging the work in [7] on two time scales, I plan to explore methods to mimic the impact of noise and enhance the algorithm's robustness. This extension holds promise for improving the algorithm's performance in real-world environments characterized by uncertainty and conflicting objectives.

As another future directions, I am concurrently exploring meta-learning alongside the ongoing development of our current algorithm. Specifically, my interest lies in leveraging

meta-learning techniques to enhance learning efficiency and adaptability. Currently, I am investigating the utilization of a common task representation matrix and individual task specification vectors to characterize the value function for a fixed policy across tasks. This endeavor involves integrating this representation framework into the policy improvement step, thereby imbuing the algorithm with Actor-Critic capabilities. By working in parallel on both meta-learning and algorithm development, we aim to advance in both areas, ultimately leading to more robust and versatile learning systems capable of addressing the complexities of reinforcement learning effectively.

Bibliography

[1] Aqeel Anwar and Arijit Raychowdhury. Autonomous navigation via deep reinforcement learning for resource constraint edge nodes using transfer learning. *IEEE Access*, 8: 26549–26560, 2020.

[2] Mohammad Babaeizadeh, Iuri Frosio, Stephen Tyree, Jason Clemons, and Jan Kautz. Reinforcement learning through asynchronous advantage actor-critic on a gpu. *arXiv preprint arXiv:1611.06256*, 2016.

[3] Richard Bellman. Dynamic programming. *science*, 153(3731):34–37, 1966.

[4] Richard Bellman and Stuart Dreyfus. Functional approximations and dynamic programming. *Mathematical Tables and Other Aids to Computation*, 13(68):247–251, 1959.

[5] Rich Caruana. Multitask learning. *Machine learning*, 28:41–75, 1997.

[6] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. *Advances in neural information processing systems*, 27, 2014.

[7] Thinh T Doan. Fast nonlinear two-time-scale stochastic approximation: Achieving

\

mathcal {O}(1/k) $finite - samplecomplexity.$ *arXiv preprint arXiv:2401.12764*, 2024.

[8] Francisco Garcia and Philip S Thomas. A meta-mdp approach to exploration for lifelong reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[9] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE, 2017.

[10] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[11] Dooyoung Hong, Seonhoon Lee, Young Hoo Cho, Donkyu Baek, Jaemin Kim, and Naehyuck Chang. Energy-efficient online path planning of multiple drones using reinforcement learning. *IEEE Transactions on Vehicular Technology*, 70(10):9725–9740, 2021.

[12] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. *Advances in neural information processing systems*, 26, 2013.

[13] Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards continual reinforcement learning: A review and perspectives. *Journal of Artificial Intelligent Research*, 75:1401–1476, 2020.

[14] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6):4909–4926, 2021.

[15] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

[16] Guangchen Lan, Han Wang, James Anderson, Christopher Brinton, and Vaneet Aggarwal. Improved communication efficiency in federated natural policy gradient via admm-based gradient updates. *arXiv preprint arXiv:2310.19807*, 2023.

[17] Guanghui Lan. Policy mirror descent for reinforcement learning: Linear convergence, new sampling complexity, and generalized problem classes. *Mathematical programming*, 198(1): 1059–1106, 2023.

[18] Fan Lyu, Shuai Wang, Wei Feng, Zihan Ye, Fuyuan Hu, and Song Wang. Multi-domain multi-task rehearsal for lifelong learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 8819–8827, 2021.

[19] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[20] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[21] Guillem Muñoz, Cristina Barrado, Ender Çetin, and Esther Salami. Deep reinforcement learning for drone delivery. *Drones*, 3(3):72, 2019.

[22] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[23] Jan Peters and Stefan Schaal. Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190, 2008.

[24] Huy Xuan Pham, Hung Manh La, David Feil-Seifer, and Aria Nefian. Cooperative and distributed reinforcement learning of drones for field coverage. *arXiv preprint arXiv:1803.07250*, 2018.

[25] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[26] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7), 2009.

[27] Sebastian Thrun. Lifelong learning algorithms. In *Learning to learn*, pages 181–209. Springer, 1998.

[28] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[29] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

[30] Lin Xiao. On the convergence rates of policy gradient methods. *The Journal of Machine Learning Research*, 23(1):12887–12922, 2022.

[31] Sihan Zeng, Malik Aqeel Anwar, Thinh T. Doan, Arijit Raychowdhury, and Justin Romberg. A decentralized policy gradient approach to multi-task reinforcement learning. In Cassio de Campos and Marloes H. Maathuis, editors, *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*, volume 161 of *Proceedings of Machine Learning Research*, pages 1002–1012. PMLR, 27–30 Jul 2021. URL https://proceedings.mlr.press/v161/zeng21a.html.

[32] Guang Zhan, Xinmiao Zhang, Zhongchao Li, Lin Xu, Deyun Zhou, and Zhen Yang. Multiple-uav reinforcement learning algorithm based on improved ppo in ray framework. *Drones*, 6 (7):166, 2022.

Appendices

Appendix A

Python code

A.1 IPMD.py

```
"""
A Decentralized Policy Gradient Approach to Multi-task Reinforcement Learning
Source code for the GridWorld Problem

To run the code use
python main.py

The Maze visualization part of the problem is not coded by the authors and is
taken from the following reference
https://morvanzhou.github.io/tutorials/
"""
import random

import matplotlib.pyplot as plt
from tqdm import tqdm
from maze_env import Maze
from RL_brain import ThetaLearningTable
```

```
import numpy as np
import imageio
from matplotlib import rc
from PIL import Image


rc("text", usetex=True)
rc("font", family="cmr10")

# Define the coordinates for wall and goal positions
HELL_COORD = [
    [[2, 0],[2, 2],[2, 3],[2, 7],       [7, 9],[7, 6],[7, 4],[7, 3],[7, 1]],
    [[2, 1],[2, 0],[2, 5],[2, 8],       [7, 8],[7, 5],[7, 4],[7, 2],[7, 3]],
    [[2, 2],[2, 1],[2, 4],[2, 8],       [7, 7],[7, 9],[7, 6],[7, 1],[7, 3]],
    [[2, 3],[2, 0],[2, 5],[2, 6],[2, 7],[7, 6],[7, 8],[7, 5],[7, 2]],
    [[2, 4],[2, 1],[2, 3],[2, 7],[2, 6],[7, 5],[7, 9],[7, 7],[7, 2]],
    [[2, 5],[2, 2],[2, 4],[2, 8],[2, 6],[7, 4],[7, 8],[7, 7],[7, 1]]
]
HELL_COORD_TOTAL = [[2, 0],[2, 1],[2, 2],[2, 3],[2, 4],[2, 5],[2, 6],[2, 7],[2, 8],
                    [7, 9],[7, 8],[7, 7],[7, 6],[7, 5],[7, 4],[7, 3],[7, 2],[7, 1]]
GOAL_COORD = [[9, 9], [9, 9], [9, 9], [9, 9], [9, 9], [9, 9]]
num_agents = len(HELL_COORD)
img = [[] for _ in range(num_agents)]
img_opt = [[] for _ in range(num_agents)]

# print all goal and hell in one Maze
```

```
env_total = Maze(
            name='Maze Total',
            MAZE_H=10,
            MAZE_W=10,
            UNIT=40,
            hell_coord=HELL_COORD_TOTAL,
            goal_coord=GOAL_COORD,
        )
# env_total.reset()
# env_total.render()
# env_total.canvas.postscript(file="a_.eps")
# img_total = np.asarray(Image.open("a_.eps"))
# imageio.imsave('Results/Maze_Total.jpg',img_total)

# Set learning parameters
num_episode = 100
max_iteration = 100
num_try_policy = 100
env_list = []
GAMMA = 0.99
plt.interactive(True)
line_color = "#94B6D2"
Display = False
cum_f = np.array([])

if __name__ == "__main__":
```

```
for n in range(num_agents):
    name = "Maze " + str(n)
    hell_coord = HELL_COORD[n]
    goal_coord = [GOAL_COORD[n]]

    # Generate the maze of size MAZE_H times MAZE_W with goal and wall positions as
    env = Maze(
        name=name,
        MAZE_H=10,
        MAZE_W=10,
        UNIT=40,
        hell_coord=hell_coord,
        goal_coord=goal_coord,
    )

    # Set up the tabular parameter for RL
    env_list.append(env)
Theta_RL = ThetaLearningTable(env_list[0].n_actions, env_list[0].n_states, name="Maz

# # print GridWorlds
# for i,env in enumerate(env_list):
#     env.reset()
#     env.render()
#     env.canvas.postscript(file="a_.eps")
#     img1 = np.asarray(Image.open("a_.eps"))
#     imageio.imsave('Results/Maze_'+str(i)+'.jpg', img1)
```

```
# Since determinant transition probability, compute Qi Vi for gradient and evaluatic
for episode in tqdm(range(num_episode)):
    # epsilon = 0.05 +(1-0.05)*np.exp(-0.05*episode)
    for i,env in enumerate(env_list):
        # estimate sum of discounted reward Q according to state and action
        Qi = env.compute_Qi(GAMMA, Theta_RL.theta_table, env)
        # update policy based on the estimated Q
        Theta_RL.learn(Qi, episode, num_agents)
        # evaluate the updated policy using all tasks (objective)
        f = 0
        if episode == num_episode - 1 and i == 5:
            print('Trained policy Vi')
        for j, env1 in enumerate(env_list):
            Vi = env1.compute_Vi(GAMMA, Theta_RL.theta_table, env1)
            f += Vi[0]
            if episode == num_episode-1 and i == 5:
                print(Vi[0])

        cum_f = np.append(cum_f, f)

# optimal policy and it's value
optimal_theta = np.zeros((env_list[0].n_states,env_list[0].n_actions))
optimal_state_actions = [
    [96,86,76,66,56,46,36,26,16],
    [0,10,20,30,40,50,60,70,80,9,19,29,39,49,59,69,79,89],
```

```
    [90,91,92,93,94,95,6,7,8],
    []
]
for i in range(env_list[0].n_actions):
    for j in optimal_state_actions[i]:
        optimal_theta[j]= np.ones(env_list[0].n_actions)*10**-10
        optimal_theta[j][i] = 1-3*10**-10
for i in range(env_list[0].n_states):
    if np.sum(optimal_theta[i])==0:
        optimal_theta[i] = np.ones(env_list[0].n_actions)*0.25

# # plot optimal path for Maze total
# path = []
# path_state = [96,86,76,66,56,46,36,26,16,0,10,20,30,40,50,60,70,80,9,19,29,39,49,5
# for i in range(len(path_state)):
#     path.append([np.remainder(path_state[i],env_total.MAZE_H),np.floor(path_state[
# env_total.plot_path(path)
# env_total.render()
# env_total.canvas.postscript(file="a_.eps")
# img_total = np.asarray(Image.open("a_.eps"))
# imageio.imsave('Results/Maze_Total_Path.jpg', img_total)
# # plot optimal path for Maze 0
# path = []
# path_state = [0,10,11,12,13,14,15,16,26,27,28,29,39,49,59,69,79,89,99]
# for i in range(len(path_state)):
#     path.append([np.remainder(path_state[i], env_total.MAZE_H), np.floor(path_stat
```

```
# env_total.plot_path(path)
# env_total.render()
# env_total.canvas.postscript(file="a_.eps")
# img_total = np.asarray(Image.open("a_.eps"))
# imageio.imsave('Results/Maze_0_Path.jpg', img_total)

# optimal policy have f
f_opt = 0
for i, env in enumerate(env_list):
    Vi = env.compute_Vi(GAMMA, optimal_theta, env)
    f_opt += Vi[0]

# # optimal policy display
# for i,env in enumerate(env_list):
#     state, observation = env.reset()
#     for j in range(max_iteration):
#         # refresh env displayed
#         env.render()
#         env.canvas.postscript(file="a_.eps")
#         # use PIL to convert to PNG
#         img[i].append(np.asarray(Image.open("a_.eps")))
#         # RL choose action based on observation
#         action = np.argmax(optimal_theta[state])
#         # RL take action and get next observation and reward
#         state_, observation_, reward, done = env.step(action)
#         # swap observation
```

```
#         state = state_
#         if done or (j == max_iteration - 1):
#             env.render()
#             env.canvas.postscript(file="a_.eps")
#             # use PIL to convert to PNG
#             img[i].append(np.asarray(Image.open("a_.eps")))
#             # break loop when end of this episode
#             break

# visualize the learned policy
if Display:
    policy_actions = []
    for i,env in enumerate(env_list):
        for k in range(num_try_policy):
            actions = []
            state, observation = env.reset()
            for j in range(max_iteration):
                # RL choose action based on observation
                action = Theta_RL.choose_action(state)
                actions.append(action)
                # RL take action and get next observation and reward
                state_, observation_, reward, done = env.step(action)
                # swap observation
                state = state_
                if done or (j == max_iteration - 1):
                    policy_actions.append(actions)
```

```
                break


        if len(policy_actions) == i+1:
            state, observation = env.reset()
            for j in range(max_iteration):
                # refresh env displayed
                env.render()
                env.canvas.postscript(file="a_.eps")
                # use PIL to convert to PNG
                img[i].append(np.asarray(Image.open("a_.eps")))
                # RL choose action based on observation
                action = policy_actions[i][j]
                # RL take action and get next observation and reward
                state_, observation_, reward, done = env.step(action)
                # swap observation
                state = state_
                if done or (j == max_iteration - 1):
                    env.render()
                    env.canvas.postscript(file="a_.eps")
                    # use PIL to convert to PNG
                    img[i].append(np.asarray(Image.open("a_.eps")))
                    # break loop when end of this episode
                    break
            break

# Create animated GIF of the last few frames
```

```
    print("Creating gif")
    for i in range(len(env_list)):
        name_init = "Results/Maze_" + str(i)
        name = name_init + ".gif"
        imageio.mimsave(name, img[i], duration=0.05)

# plot the f
fig = plt.figure()
plt.plot(cum_f)
plt.title('Objective vs Iteration', fontsize=30)
plt.xlabel('Iteration (k)', fontsize=20)
plt.ylabel('f('r'$\pi_k$)', fontsize=20)
plt.show()
fig.savefig('Results/Objective_vs_Iteration.png', bbox_inches='tight')

# plot the error f_opt - f
fig = plt.figure()
plt.plot(np.abs(f_opt-cum_f))
plt.title('Error vs Iteration', fontsize=30)
plt.xlabel('Iteration (k)', fontsize=20)
plt.ylabel('f('r'$\pi^*$) - f('r'$\pi_k$)', fontsize=20)
plt.show()
fig.savefig('Results/Error_vs_Iteration.png', bbox_inches='tight')
```

A.2 maze_env.py

```
"""
A Decentralized Policy Gradient Approach to Multi-task Reinforcement Learning
Source code for the GridWorld Problem

To run the code use
python main.py

The Maze visualization part of the problem is not coded by the authors and is taken from
https://morvanzhou.github.io/tutorials/
"""
import numpy as np
import time
import sys
from scipy.optimize import minimize
from scipy.linalg import lu_factor, lu_solve

def rosen(x, a, b):
    return np.linalg.norm(np.dot(x,a)-b)

if sys.version_info.major == 2:
    import Tkinter as tk
else:
    import tkinter as tk
```

```
# Define colors to be used in maze
GOAL_COLOR = "#D8B25C"
AGENT_COLOR = "#A5AB81"
WALL_COLOR = "#F24F5C"
GRID_COLOR = "#595959"
PATH_COLOR = "#4169E1"



class Maze(tk.Tk, object):
    def __init__(self, name, MAZE_H, MAZE_W, UNIT, hell_coord, goal_coord):
        super(Maze, self).__init__()
        self.action_space = ["u", "d", "l", "r"]
        self.n_actions = len(self.action_space)
        self.n_states = MAZE_W * MAZE_H
        self.title(name)
        self.geometry("{0}x{1}".format(MAZE_H * UNIT, MAZE_H * UNIT))
        self.MAZE_H = MAZE_H
        self.MAZE_W = MAZE_W
        self.UNIT = UNIT
        self.hell_coord = hell_coord
        self.goal_coord = goal_coord
        self.hell_array = []
        self._build_maze([0,0])

    def _build_maze(self,start_position=[0,0]):
        self.canvas = tk.Canvas(
```

```
        self,
        bg="white",
        height=self.MAZE_H * self.UNIT,
        width=self.MAZE_W * self.UNIT,
    )

    # create grids
    for c in range(0, (self.MAZE_W+1) * self.UNIT, self.UNIT):
        x0, y0, x1, y1 = c, 0, c, self.MAZE_H * self.UNIT
        self.canvas.create_line(x0, y0, x1, y1, fill=GRID_COLOR)
    for r in range(0, (self.MAZE_H+1) * self.UNIT, self.UNIT):
        x0, y0, x1, y1 = 0, r, self.MAZE_W * self.UNIT, r
        self.canvas.create_line(x0, y0, x1, y1, fill=GRID_COLOR)

    # create origin
    origin = np.array([20, 20])
    start = np.array([20 + self.UNIT * start_position[0], 20 + self.UNIT * start_pos

    # create walls
    for h_coord in self.hell_coord:
        hell_center = origin + np.array(
            [self.UNIT * h_coord[0], self.UNIT * h_coord[1]]
        )
        self.hell = self.canvas.create_rectangle(
            hell_center[0] - 15,
            hell_center[1] - 15,
```

```
            hell_center[0] + 15,
            hell_center[1] + 15,
            fill=WALL_COLOR,
            outline="",
        )
        self.hell_array.append(self.canvas.coords(self.hell))

    # create goal
    for g_coord in self.goal_coord:
        oval_center = origin + np.array(
            [self.UNIT * g_coord[0], self.UNIT * g_coord[1]]
        )
        self.oval = self.canvas.create_oval(
            oval_center[0] - 15,
            oval_center[1] - 15,
            oval_center[0] + 15,
            oval_center[1] + 15,
            fill=GOAL_COLOR,
            outline="",
        )

    # create agent rect
    self.rect = self.canvas.create_rectangle(
        start[0] - 15,
        start[1] - 15,
        start[0] + 15,
```

```
            start[1] + 15,
            fill=AGENT_COLOR,
            outline="",
        )

        # pack all
        self.canvas.pack()

    def plot_path(self, path, start_position=[0,0]):
        self.canvas.destroy()
        self.canvas = tk.Canvas(
            self,
            bg="white",
            height=self.MAZE_H * self.UNIT,
            width=self.MAZE_W * self.UNIT,
        )

        # create origin
        origin = np.array([20, 20])
        start = np.array([20+self.UNIT*start_position[0], 20+self.UNIT*start_position[1]

        # create path
        for p in path:
            p_center = origin + np.array(
                [self.UNIT * p[0], self.UNIT * p[1]]
            )
```

```
    self.canvas.create_rectangle(
        p_center[0] - 19,
        p_center[1] - 19,
        p_center[0] + 19,
        p_center[1] + 19,
        fill=PATH_COLOR,
        outline="",
    )

# create grids
for c in range(0, (self.MAZE_W+1) * self.UNIT, self.UNIT):
    x0, y0, x1, y1 = c, 0, c, self.MAZE_H * self.UNIT
    self.canvas.create_line(x0, y0, x1, y1, fill=GRID_COLOR)
for r in range(0, (self.MAZE_H+1) * self.UNIT, self.UNIT):
    x0, y0, x1, y1 = 0, r, self.MAZE_W * self.UNIT, r
    self.canvas.create_line(x0, y0, x1, y1, fill=GRID_COLOR)



# create walls
for h_coord in self.hell_coord:
    hell_center = origin + np.array(
        [self.UNIT * h_coord[0], self.UNIT * h_coord[1]]
    )
    self.hell = self.canvas.create_rectangle(
        hell_center[0] - 15,
        hell_center[1] - 15,
```

```
            hell_center[0] + 15,
            hell_center[1] + 15,
            fill=WALL_COLOR,
            outline="",
        )
        self.hell_array.append(self.canvas.coords(self.hell))

    # create goal
    for g_coord in self.goal_coord:
        oval_center = origin + np.array(
            [self.UNIT * g_coord[0], self.UNIT * g_coord[1]]
        )
        self.oval = self.canvas.create_oval(
            oval_center[0] - 15,
            oval_center[1] - 15,
            oval_center[0] + 15,
            oval_center[1] + 15,
            fill=GOAL_COLOR,
            outline="",
        )

    # create agent rect
    self.rect = self.canvas.create_rectangle(
        start[0] - 15,
        start[1] - 15,
        start[0] + 15,
```

```
            start[1] + 15,
            fill=AGENT_COLOR,
            outline="",
        )

        # pack all
        self.canvas.pack()

    def reset(self, start_position=[0,0]):
        #self.update()
        # time.sleep(0.01)
        self.canvas.delete(self.rect)
        origin = np.array([20+self.UNIT*start_position[0], 20+self.UNIT*start_position[1
        self.rect = self.canvas.create_rectangle(
            origin[0] - 15,
            origin[1] - 15,
            origin[0] + 15,
            origin[1] + 15,
            fill=AGENT_COLOR,
        )
        # return observation
        state_coords = self.canvas.coords(self.rect)
        # state = state_coords[1]/origin[0]
        return int(start_position[1]*self.MAZE_H+start_position[0]), self.canvas.coords(

    def step(self, action):
```

```
        s = self.canvas.coords(self.rect)
        base_action = np.array([0, 0])
        if action == 0:  # up
            if s[1] > self.UNIT:
                base_action[1] -= self.UNIT
        elif action == 1:  # down
            if s[1] < (self.MAZE_H - 1) * self.UNIT:
                base_action[1] += self.UNIT
        elif action == 2:  # right
            if s[0] < (self.MAZE_W - 1) * self.UNIT:
                base_action[0] += self.UNIT
        elif action == 3:  # left
            if s[0] > self.UNIT:
                base_action[0] -= self.UNIT

        self.canvas.move(self.rect, base_action[0], base_action[1])  # move agent

        s_ = self.canvas.coords(self.rect)  # next state
        state_ = int(self.MAZE_H * (s_[1] // self.UNIT) + s_[0] // self.UNIT)

        # Define rewards
        reward,done = self.reward(s_)

        return state_, s_, reward, done

    def reward(self,state):
```

```
    if state == self.canvas.coords(self.oval):
        reward = 1
        done = True
    elif state in self.hell_array:
        # reward = 0
        reward = -1
        done = True
    else:
        # reward = 0
        reward = 0
        done = False
    return reward,done

def render(self):
    # time.sleep(0.01)
    self.update()

def compute_Vi(self, gamma, theta, env):
    # first construct the transition probability matrix and the reward vector under
    P = np.zeros((self.n_states, self.n_states))
    R_list = np.zeros((self.n_states, 1))
    for state in range(self.n_states):
        for action in range(self.n_actions):  # up, down, left, right
            if [np.remainder(state,env.MAZE_H),np.floor(state/env.MAZE_H)] in env.go
                P[state][state]=1
            else:
```

```
                    env.reset([np.remainder(state,env.MAZE_H),np.floor(state/env.MAZE_H)
                    state_, observation_, reward, done = env.step(action)
                    P[state, state_] += theta[state][action]
                    # P[state_, state] += theta[state][action]
                    R_list[state] += theta[state][action] * reward

        return np.matmul(np.linalg.inv(np.identity(self.n_states) - gamma * P), R_list)

    def compute_Qi(self, gamma, theta, env):
        # first construct the transition probability matrix and the reward vector under
        Vi = env.compute_Vi(gamma,theta,env)
        Qi = np.zeros((self.n_states,self.n_actions))
        for state in range(self.n_states):
            for action in range(self.n_actions):
                env.reset([np.remainder(state,env.MAZE_H),np.floor(state/env.MAZE_H)])
                state_, observation_, reward, done = env.step(action)
                Qi[state][action] = reward + gamma*Vi[state_]

        return Qi
```

A.3 RL_brain.py

```
"""
A Decentralized Policy Gradient Approach to Multi-task Reinforcement Learning
Source code for the GridWorld Problem
```

```
To run the code use
python main.py

The Maze visualization part of the problem is not coded by the authors and is taken from
https://morvanzhou.github.io/tutorials/
"""

import numpy as np
from scipy.optimize import minimize
from scipy.optimize import Bounds

def softmax(x):
    e_x = np.exp(x - np.max(x))
    return e_x / e_x.sum()

def rosen_with_args(x, a, b, c):
    return -a*np.inner(b,x)+np.inner(x,np.log(np.divide(x,c)))

cons = ({'type': 'eq', 'fun': lambda x:  x[0] + x[1] + x[2]+x[3]-1})
bounds = Bounds([0, 0,0,0], [1, 1,1,1])

class ThetaLearningTable:
    def __init__(self, num_actions, num_states, learning_rate=0.2, name=''):
        self.name = name
        self.num_actions = num_actions
```

```
        self.num_states = num_states
        self.lr = learning_rate
        self.theta_table = 0.25 * np.ones((num_states, num_actions), dtype=np.float64) #

    def assign(self, theta):
        self.theta_table = theta

    def load(self, path='model/theta.npy'):
        self.theta_table = np.load(path)

    def save_theta(self, path):
        path = path + '_theta.npy'
        np.save(path, self.theta_table)

    def choose_action(self, state):
        num_actions = self.num_actions
        probs = self.theta_table[state]
        # s = sum(self.theta_table[state])
        # probs = [i / s for i in self.theta_table[state]]
        action = np.random.choice(num_actions, 1, p=probs)[0]
        return action

    def learn(self, Qi, episode, num_agents):
        # Linear decay step size
        alpha_k = 1/np.sqrt(episode+1)
        # alpha_k = 2 / (episode + num_agents + 1)
```

```
# alpha_k = 0.02
theta = np.zeros((self.num_states,self.num_actions))
x0 = self.theta_table
for i in range(self.num_states):
    res = minimize(rosen_with_args, x0[i,:], method='SLSQP',
                   args=(alpha_k, Qi[i,:], x0[i,:]), constraints=cons, bounds=bo
    theta[i,:] = np.array(res.x, dtype=np.float64)

self.theta_table = theta
```