Off-Policy Temporal Difference Learning For Robotics And Autonomous Systems

Heejin Jeong
University of Pennsylvania

Off-Policy Temporal Difference Learning For Robotics And Autonomous Systems

Abstract

Reinforcement learning (RL) is a rapidly advancing field with implications in autonomous vehicles, medicine, finance, along with several other applications. Particularly, off-policy temporal difference (TD) learning, a specific type of RL technique, has been widely used in a variety of autonomous tasks. However, there remain significant challenges that must be overcome before it can be successfully applied to various real-world applications. In this thesis, we specifically address several major challenges in off-policy TD learning.

In the first part of the thesis, we introduce an efficient method of learning complex stand-up motion of humanoid robots by Q-learning. Standing up after falling is an essential ability for humanoid robots yet it is difficult to learn flexible stand-up motions for various fallen positions due to the complexity of the task. We reduce sample complexity of learning by applying a clustering method and utilizing the bilateral symmetric feature of humanoid robots. The learned policy is demonstrated in both simulation and on a physical robot.

The greedy update of Q-learning, however, often causes overoptimism and instability. In the second part of the thesis, we propose a novel Bayesian approach to Q-learning, called ADFQ, which improves the greedy update issues by providing a principled way of updating Q-values based on uncertainty of Q-belief distributions. The algorithm converges to Q-learning as the uncertainty approaches zero, and its efficient computational complexity enables the algorithm to be extended with a neural network. Both ADFQ and its neural network extension outperform their comparing algorithms by improving the estimation bias and converging faster to optimal Q-values.

In the last part of the thesis, we apply off-policy TD methods to solve the active information acquisition problem where an autonomous agent is tasked with acquiring information about targets of interests. Off-policy TD learning provides solutions for classical challenges in this problem -- system model dependence and the difficulty of computing information-theoretic cost functions for a long planning horizon. In particular, we introduce a method of learning a unified policy for in-sight tracking, navigation, and exploration. The policy shows robust behavior for tracking agile and anomalous targets with a partially known target model.

Degree Type

Dissertation

Degree Name

Doctor of Philosophy (PhD)

Graduate Group

Electrical & Systems Engineering

First Advisor

Daniel D. Lee

Second Advisor

George J. Pappas

Keywords

Bayesian Inference, Deep Reinforcement Learning, Information Acquisition, Machine Learning,

OFF-POLICY TEMPORAL DIFFERENCE LEARNING FOR ROBOTICS AND AUTONOMOUS SYSTEMS

Heejin Jeong

A DISSERTATION

in

Electrical and Systems Engineering

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2020

[illegible]

Daniel D. Lee, Co-Supervisor of Dissertation
Professor

[illegible]

George J. Pappas, Co-Supervisor of Dissertation
Professor

[illegible]

Victor Preciado, Graduate Group Chairperson
Professor

Dissertation Committee

Manfred Morari (Chair), Professor in Electrical and Systems Engineering
Daniel D. Lee, Professor in Electrical and Computer Engineering, Cornell University
George J. Pappas, Professor in Electrical and Systems Engineering
Pieter Abbeel, Professor in Electrical Engineering and Computer Science, University of California, Berkeley

Acknowledgment

This work would not have been possible without the support and help from many people. First and foremost, I would like to thank my thesis advisors, Prof. Daniel D. Lee and Prof. George J. Pappas, for their guidance and advice during my Ph.D. program. With their continued direction and involvement, I learned how to critically think and approach challenging problems, and was able to work on exciting topics and applications.

I also thank Prof. Manfred Morari for serving on my thesis committee as the chair, for always providing his valuable insights on my research, and for being excited and willing to learn about my work. In addition to our technical discussions, I also always enjoyed our conversations about different topics. I would like to thank Prof. Pieter Abbeel for being in my committee, for wanting to learn about my ideas, and for reviewing my work. Even though we are at different universities, I always enjoyed our conversations at various conferences.

Many thanks to my collaborators, Prof. Hamed Hassani, Clark Zhang, Brent Schlotfeldt, and participants of DARPA Robotics Challenge Finals and RoboCup 2015. I learned a lot from their instrumental feedback and comments, and our efforts towards the same goals kept me motivated.

Being a member of GRASP Lab has been a wonderful experience. I was able to interact with bright students from different fields and different labs, and learned more things from them than textbooks. I would like to thank many past and current GRASP members for sharing their knowledge and for making the environment intellectually stimulating.

I would like to thank the current and past Ph.D. students in the department of Electrical and Systems Engineering, and the current and past members in both Daniel D. Lee and George J. Pappas groups for the times we spent working on homework together, for discussing research ideas, and for pondering about the future.

I thank my close friends for their tremendous support and for listening to my complaints especially during the tough times of my Ph.D.

I would like to thank Blake Bleier for always being there for me and for his endless support and encouragement. This journey would not have been possible without him. I thank him for his selfless help and for being so caring, loving, and kind through the ups and downs in this journey. I also thank John and Loretta Bleier for their continued support.

Lastly, I thank my dad, mom, and sister. Even though I selfishly left Korea to pursue my own interests, they have been nothing but supportive. I especially want to recognize my dad and honor him for his endless devotion to the family and for his support. Unfortunately and tragically, he passed away in 2018 in Korea and I was not able to spend much time with him in his last years. Even though he was sick and struggling so much with his illness, he always ensured me that he was okay, to keep me from worrying and to focus on my work. I could not have been here without his lessons, advice, support and love. As painful as the loss, it enlightened me in great values in my life. I am privileged to be who I am because of my dad, mom, and sister, and I will do good for people and help people remembering him.

ABSTRACT

OFF-POLICY TEMPORAL DIFFERENCE LEARNING FOR ROBOTICS AND AUTONOMOUS SYSTEMS

Heejin Jeong

Daniel D. Lee, George J. Pappas

Reinforcement learning (RL) is a rapidly advancing field with implications in autonomous vehicles, medicine, finance, along with several other applications. Particularly, off-policy temporal difference (TD) learning, a specific type of RL technique, has been widely used in a variety of autonomous tasks. However, there remain significant challenges that must be overcome before it can be successfully applied to various real-world applications. In this thesis, we specifically address several major challenges in off-policy TD learning.

In the first part of the thesis, we introduce an efficient method of learning complex stand-up motion of humanoid robots by Q-learning. Standing up after falling is an essential ability for humanoid robots yet it is difficult to learn flexible stand-up motions for various fallen positions due to the complexity of the task. We reduce sample complexity of learning by applying a clustering method and utilizing the bilateral symmetric feature of humanoid robots. The learned policy is demonstrated in both simulation and on a physical robot.

The greedy update of Q-learning, however, often causes overoptimism and instability. In the second part of the thesis, we propose a novel Bayesian approach to Q-learning, called ADFQ, which improves the greedy update issues by providing a principled way of updating Q-values based on uncertainty of Q-belief distributions. The algorithm converges to Q-learning as the uncertainty approaches zero, and its efficient computational complexity enables the algorithm to be extended with a neural network. Both ADFQ and its neural network extension outperform their comparing algorithms by improving the estimation bias and converging faster to optimal Q-values.

In the last part of the thesis, we apply off-policy TD methods to solve the active information acquisition problem where an autonomous agent is tasked with acquiring information about targets of interests. Off-policy TD learning provides solutions for classical challenges in this problem – system model dependence and the difficulty of computing information-theoretic cost functions for a long planning horizon. In particular, we introduce a method of learning a unified policy for in-sight tracking, navigation, and exploration. The policy shows robust behavior for tracking agile and anomalous targets with a partially known target model.

Contents

List of Figures

Chapter 1

Introduction

Reinforcement learning stems from psychological and neuroscientific perspectives on the decision-making process of humans and animals [115]. In this framework, a learning subject seeks an optimal or near-optimal behavior through trial-and-error interaction with a dynamic environment (Fig.1.1). Therefore, it is a general mathematical framework for learning sequential decision-making tasks. There have been numerous studies in neuroscience and psychology linking reinforcement learning to decision making because of its resemblance to human and animal learning behavior [23, 24, 69, 86, 115]. It has been shown that there is a connection between *temporal-difference (TD) learning*, a specific type of reinforcement learning technique, and neural signals that occur during decision-making [108, 115]. The TD algorithm [122] is a model-free method that estimates the expected value of cumulative

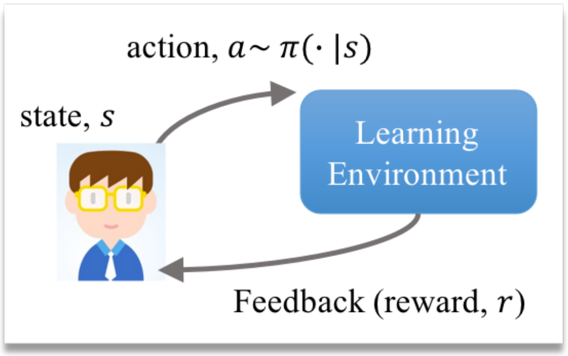

Figure 1.1: Reinforcement Learning Framework

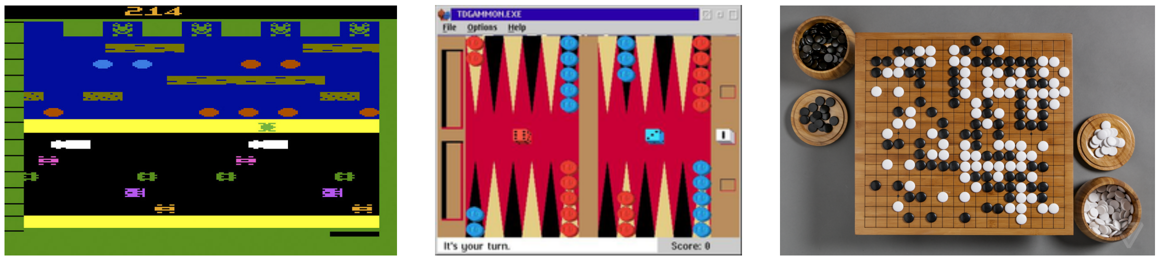

Figure 1.2: Successful reinforcement learning appliation examples. Left: Parker Prothers Frogger (Atari2600), Middle: Backgammon computer game, Right: Game of Go

future rewards of the current state by bootstrapping. Model-free methods in reinforcement learning directly learn near-optimal policies or value functions without explicitly estimating the dynamics of an environment. This contrasts with model-based methods in reinforcement learning that learn a model of a dynamic environment and use the model to estimate near-optimal policies. Therefore, model-free methods are often simpler to implement and can handle more complex environments where learning their models are infeasible. Such model-free methods have been applied in a wealth of complex tasks such as computer games [92], board games (TD Gammon [123], AlphaGo of Google Deepmind [116]), resource management [87], robotics applications [28, 43, 61, 71] and so on.

In reinforcement learning for control problems, we consider two types of policies – behavior policy and target policy. The behavior policy is a policy to choose an action during the interaction with the environment. The target policy is a policy used for evaluating the current value estimates. On-policy methods set these two policies to be identical, and therefore, a learning agent must choose an action based on the current policy it has been updating. Sarsa, a TD learning algorithm, and policy-based methods are examples of on-policy reinforcement learning. On the other hand, the behavior and target policies can be different in off-policy methods. Therefore, off-policy can be desirable when the safety of the behavior policy must be guaranteed (robotics examples). We can also re-use previously generated samples for the current value update. However, off-policy does not always provide a convergence guarantee when it is used with TD learning and function approximation [7]. Further details of TD learning and off-policy are explained in Sec.2.3.

Despite the lack of convergence guarantees, the recent success of deep neural networks enabled reinforcement learning to be applied to diverse and complex tasks showing human-level performance in some domains such as the game of Go, Atari games, and Dota [92, 93, 100]. However, there remain significant challenges that must be overcome before it can be successfully applied to various real-world tasks beyond video games. The challenges include sample complexity, fast reinforcement learning, continual learning, partial observability, uncertainty, and design of reward functions. In this thesis, we specifically address three of these major challenges – sample complexity, uncertainty, and partial observability.

Unlike supervised learning, reinforcement learning requires to interact with a dynamic environment. In general, obtaining direct interaction experience from an environment is expensive, and thus, it is desired to have a simulation environment that can generate an infinite number of sample experiences. However, simulation programs often differ from real environments causing issues in transferring a learned policy. For some tasks such as the game of Go, Dota, dialogue generation, rewards are supposed to be given by (somewhat) experts in tasks such as humans which is extremely expensive. Some methods tried to alleviate such issues by using self-play or adversarial techniques, but it can easily diverge if a reward function is not well-structured [81, 93, 100]. Robotics is another important field where sample complexity is crucial. Most robotic control tasks require to handle continuous variables, and thus, they face the curse of dimensionality issue. Previously, discretization or simple function approximation methods have been applied to solve several robotics tasks. However, they were not able to be scaled or were limited to very specific models or tasks [28, 95]. Deep reinforcement learning methods alleviate such limitations by providing generalized frameworks but demand a significant number of samples. As robotic tasks require physical interactions with real environments, it is difficult to collect millions of data points or learn real-time using a real robot unless enough resources are available [80]. Even in a simulation program, accurately simulating complex dynamics of a robot and environment requires high computation time, and thus repeating a training-evaluation process with millions of examples at a time is extremely time-consuming. There have been numerous studies in order to reduce the sample complexity of reinforcement learning. Meta learning is a method of learning to learn which experiences diverse environments during learning and is able to quickly adapt to a new environment in testing [34, 35]. Some studies leverage human demonstration to reduce the number of samples [17, 48].

Another major stumbling block for reinforcement learning to move into real-world tasks is addressing the uncertainty of learning parameters. In reinforcement learning, uncertainty comes from three sources – stochastic state transition, stochastic reward, and bootstrapping values (for TD learning). Many of recent benchmark environments for reinforcement learning algorithms are deterministic, but this is hardly valid in practice. Most tasks in the real world, especially in autonomous systems, contain a greater range of randomness. Standard reinforcement learning considers the expected values of the stochastic features, and therefore, learning can be inefficient in highly stochastic environments. Moreover, relying on a single best guess rather than considering different uncertainty measures for different beliefs is naturally inefficient. It has been pointed out by numerous studies that one of the most popular reinforcement learning algorithms, Q-learning, often suffers from overoptimism especially in a stochastic environment due to its greedy value evaluation [50, 51, 127, 129]. Bayesian reinforcement learning is one of the approaches in reinforcement learning that deploys Bayesian inference to incorporate new information into prior information for learning parameters by explicitly quantifying the uncertainty of parameters. Therefore, it provides a principled way of handling one of the major challenges in reinforcement learning, exploration-exploitation trade-off. Additionally, it enables us to use a previously learned distribution as an informed prior during learning or for a new learning problem in shared structures or appropriately transformed structures [134]. This can potentially lead to more sample efficient methods. As we will study in the later chapter, a proposed algorithm, ADFQ, in this thesis, utilizes explicit uncertainty measures not only in exploration but also in value updates by considering all possible next action with their uncertainty measures instead of greedily choosing the best next action. This is, in fact, one of the essential components of probabilistic approaches [125] – A learning agent must consider possible outcomes and weigh them differently based on their uncertainties.

The last important challenge in reinforcement learning that will be discussed in this thesis is partial observability. In general, an autonomous agent solves its tasks based on information collected through its sensors. Such sensors include an RGB-D camera, LiDAR, GPS, IMU, and so on. Since the information is not directly obtained by the agent and it passes through a sensor, some part of the information can be missing or noisy. Although learning robotics tasks directly from RGB camera images have shown promising results with a full observability assumption in recent years [67, 80, 106], no hardware is considered as noise-free and perfect in practice. It is crucial to always consider sensor noise, physical limitations, and failure. Information gathering is one of the fields in which accounting partial observability plays a key role in the agent's controls since its main objective is to minimize the uncertainty about surroundings or a target of interests through information acquired by its noisy sensors. As we do not have access to the ground truth, this problem is a partially observable task. When targets are mobile, an agent must keep tracking the targets to maintain low uncertainty. Therefore, having a good belief distribution on targets is helpful. Existing approaches for such active target tracking tasks rely on knowledge on system models which limit their practicality in real-world tasks. An additional complication arises from their computational complexity tied with a planning horizon. It is desirable to plan with a long planning horizon rather than use a myopic approach, especially for dynamic targets [2, 21, 59, 74]. However, most existing search- and sampling-based planning methods require sacrificing non-myopic behavior in order to achieve computational efficiency or vice versa. Therefore, reinforcement learning is a desirable solution for such tasks as it maximizes an objective over an infinite horizon and it is a method for finding an optimal policy when a complete knowledge on a model is not available.

1.1 Outline and Contributions

In Chapter 2, we briefly overview reinforcement learning focusing on TD learning.

- In Chapter 3, we study an efficient method of learning stand-up motion for humanoid robots that are flexible to various fallen positions using Q-learning. Instead of using a complex function approximation that requires a large number of training examples, we define finite representative states and actions, and learn an optimal policy using the standard Q-learning directly in an open-source 3D robot simulator. With the policy, a humanoid robot is able to successfully stand up from different initial fallen positions both in simulation and on a physical humanoid robot.

- In Chapter 4, we introduce a novel Bayesian counterpart algorithm of Q-learning, *Assumed Density Filtering Q-learning* (ADFQ), which provides a principled way for a non-greedy value update using the uncertainty of learning parameters. ADFQ maintains beliefs on state-action values, Q, and updates them through an online Bayesian inference method known as assumed density filtering. The uncertainty measures not only are used in exploration during learning but also give a natural regularization for the Q-value update reducing the estimation bias of Q-learning. We prove that ADFQ converges to Q-learning as the uncertainty of the beliefs approach zero and suggest that ADFQ is a general form of Q-learning. Experimental results in finite state-action domains demonstrate that ADFQ not only improves the estimation bias of Q-learning but also achieving the optimal Q-values and optimal policies faster than all comparing algorithms.

- One of the major drawbacks of existing Bayesian reinforcement learning methods is their high computational complexities, and thus, they struggle to be used with a complex function approximator such as a neural network. ADFQ is computationally efficient and is extended with a neural network – Deep ADFQ – in Chapter 5. Similar to the experimental results in the finite domains in Chapter 4, Deep ADFQ outperforms Deep Q-network [92] and Double Deep Q-network [129] in most domains of continuous control problems and Atari 2600 games. Especially, its improvements are more significant in domains with a large action space and/or stochastic domains. It also empirically shows that Deep ADFQ improves the Q-value estimation overall.

- In Chapter 6, we introduce a framework of applying reinforcement learning to active information acquisition where an agent is tasked with acquiring information about targets of interests. Major challenges of existing methods in active information acquisition for mobile agents and targets are the trade-off between computational complexity and non-myopic behavior as well as a strong dependency of their methods on system models. In contrast, reinforcement learning provides solutions for these challenges as the length of its effective planning horizon does not affect the computational complexity, and it drops the strong dependency of an algorithm on system models. We discuss the potential benefits of the proposed framework and compare the performance of the novel algorithm to an existing information acquisition method for multi-target tracking scenarios.

- Utilizing the advantages of using reinforcement learning, we tackle more challenging problem settings in Chapter 7 and relax assumptions considered in previous studies. In particular, complete knowledge on a target model is no longer available and target motion is highly agile and anomalous. We introduce Active Tracking Target Network (ATTN), a unified reinforcement learning policy that is capable of solving major subtasks of active target tracking – in-sight tracking, navigation, and exploration. The policy shows robust behavior for tracking dynamic targets with a partially known target model. Additionally, the same policy is able to navigate in obstacle environments to reach distant targets as well as explore the environment when targets are positioned in unexpected locations.

This thesis closes with conclusions and potential future directions in Chapter 8.

Chapter 2

Reinforcement Learning : A Brief Overview

In this chapter, we will briefly overview important parts of reinforcement learning focusing on model-free value-based methods. Yet there exist many different fields and recent studies in reinforcement learning, and the field has been growing quickly. A reader who wants to learn more about fundamentals of reinforcement learning is recommended to read the latest version of [122].

2.1 Markov Decision Process

Reinforcement learning problems can be formulated in terms of an Markov Decision Process (MDP). MDP is a stochastic process with the Markov assumption which the future state is independent from the past states given the present state. More formally, if $X = (X_t : t \leq 0)$ is a stochastic process under the Markov assumption then,

$$P(X_{t+1}|X_1, \cdots, X_t) = P(X_{t+1}|X_t)$$

This conditional independence dramatically simplifies computations of the stochastic process. In this thesis, we will only consider finite-time MDPs.

An MDP is described by the tuple, $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, respectively. $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition probability kernel and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to I\!R$ is a reward function. The reward function can be a function of state only or of state, action, and the next state. $\gamma \in [0, 1)$ is a discount factor of future rewards. γ closer to 0 desires a myopic policy. A policy can be defined as a stochastic policy, $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ or as a deterministic policy, $\pi : \mathcal{S} \to \mathcal{A}$. At each time step t, an agent takes an action a_t using its policy π from its state s_t and then receives an reward r_t. We define a return, G_t, which is a sum of discounted future rewards from time t:

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^T r_{t+T}$$

$T < \infty$ for episodic tasks and $T = \infty$ for continuing tasks. The value function is defined as an expected return, which is

$$\begin{aligned} V^{\pi}(s) =& \mathbb{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^t r_t(s_t, a_t) | s_0 = s] \\ =& \mathbb{E}_{\pi}[G_0 | s_0 = s] \qquad\qquad \forall s \in \mathcal{S} \end{aligned}$$

In other words, the value function for state s is an expected value of cumulative future rewards starting at s and following π thereafter. The state-action value (Q) function is defined as the value for a state-action pair,

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^t r_t(s_t, a_t) | s_0 = s, a_0 = a] \qquad \forall s \in \mathcal{S}, a \in \mathcal{A}$$

The value function can be re-written as a self-consistent equation, which means,

$$\begin{aligned} V^{\pi}(s) &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a) \left[r + \gamma \mathbb{E}_{\pi} \left[G_0 \left| s_0 = s' \right.\right]\right] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a) \left[r + \gamma V^{\pi}(s') \right] \\ &= \mathbb{E} \left[r + \gamma V^{\pi}(s') \right] \end{aligned} \tag{2.1}$$

This is the Bellman equation for v^π. The equation represents that the value of the current state s is a function of the value of the subsequent state s'.

The objective of a learning agent in reinforcement learning is to find an optimal policy $\pi^* = \operatorname{argmax}_\pi V^\pi$. Finding the optimal values, $V^*(\cdot)$ and $Q^*(\cdot,\cdot)$, requires solving the Bellman optimality equation:

$$Q^*(s,a) = \mathbb{E}_{s'\sim P(\cdot|s,a)}[R(s,a) + \gamma \max_{a'\in\mathcal{A}} Q^*(s',a')] \qquad \forall s \in \mathcal{S} \quad \forall a \in \mathcal{A} \tag{2.2}$$

$$V^*(s) = \max_{a\in\mathcal{A}(s)} Q^*(s,a) \qquad \forall s \in \mathcal{S} \tag{2.3}$$

If Q^* is known for all $s \in \mathcal{S}, a \in \mathcal{A}$, we can find a deterministic optimal policy. $\pi^*(s) = \operatorname{argmax}_{a\in\mathcal{A}} Q^*(s,a)$.

2.2 Dynamic Programming

When an MDP is simple and fully known, we can solve the MDP using a dynamic programming (DP) algorithm. If the MDP is unknown or too complicated to be solved by a DP method, we can use a reinforcement learning approach. In this section, we will study two DP methods for solving an MDP – Policy iteration and Value iteration.

DP is a method of solving complex problems by breaking down in to sub-problems, solving the sub-problems, and combining the solutions for the main problem. It has been applied to many different problems including scheduling and bioinformatics. Dynamic programming can be also applied to an MDP since the Bellman equation Eq.2.1 is a recursive decomposition.

2.2.1 Policy Iteration

One way to find an optimal policy using a DP algorithm is *policy iteration*. In policy iteration, *policy evaluation* and *policy improvement* steps are iterated until the values and the policy converge. A policy π can be evaluated using the Bellman equation Eq.2.1. At

Algorithm 1 Policy Iteration

```
Initialize randomly $V(s)$ and $\pi(s)$ $\forall s \in \mathcal{S}$
repeat
    Policy Evaluation
    repeat
        $\Delta = 0$
        for each $s \in \mathcal{S}$ do
            $v \leftarrow V(s)$
            $V(s) = \sum_{s',r} p(s', r|s, \pi(s)) [r + \gamma V(s')]$
            $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
        end for
    until $\Delta <$ a small positive number
    Policy Improvement
    for each $s \in \mathcal{S}$ do
        $a \leftarrow \pi(s)$
        $\pi(s) \leftarrow \text{argmax}_a \sum_{s',r} p(s', r|s, a) [r + \gamma V(s')]$
    end for
until $a = \pi(s)$
return $V$ and $\pi$
```

the kth iteration,

$$V_{k+1}^{\pi}(s) = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma V_k^{\pi}(s')\right]$$

Using the estimated values, the policy is improved by acting greedily, $\pi'(s) = \text{argmax}_a\, Q^{\pi}(s, a)$. The principle of optimality theorem guarantees that the convergence of policy iteration to the optimal policy. In practice, policy iteration often converges within a few iterations. Its algorithm is presented in Algorithm 1 [122].

2.2.2 Value Iteration

In value iteration, the policy improvement step is proceeded after only one sweep of the policy evaluation step. In other words,

$$V_{k+1}^{\pi}(s) = \max_a \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma V_k^{\pi}(s')\right]$$

Value iteration often converges faster than policy iteration as it does not wait until the values converge under a suboptimal policy before the policy improvement step. It can be

Algorithm 2 Value Iteration

```
Initialize randomly $V(s)\ \forall s \in \mathcal{S}$
repeat
    $\Delta = 0$
    for each $s \in \mathcal{S}$ do
        $v \leftarrow V(s)$
        $V(s) = \max_a \sum_{s',r} p(s', r|s, \pi(s)) \left[r + \gamma V(s')\right]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end for
until $\Delta <$ a small positive number
return $\pi^* = \operatorname{argmax}_a \sum_{s',r} p(s', r|s, a) \left[r + \gamma V(s')\right]$
```

also interpreted with the Bellman optimality equation instead of the Bellman expectation equation. The complete algorithm is presented in Algorithm 2 [122].

2.3 Temporal Difference Methods

In the previous section, we studied two dynamic programming methods which solve the Bellman equation with a known MDP, $\mathcal{M}$. However, when we do not have complete knowledge of the model, those methods cannot be applied. Model-free reinforcement learning methods aim to solve a unknown MDP by directly learning either V^*, Q^*, or π^*. The difficulty of solving the Bellman equations without model knowledge is from the computation of the expectation. Monte-Carlo methods solve the difficulty by sampling sequences of states, actions, and rewards following a policy from a simulated or real environment and averaging them out. Similar to policy iteration framework, the policy is evaluated by collecting samples from the interactions and improved by the averaged value. As they require a complete

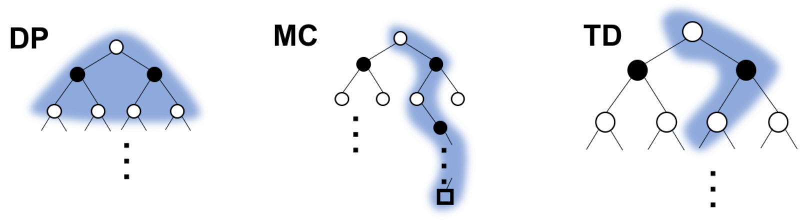

Figure 2.1: Backup diagrams of dynamic programming (DP), Monte-Carlo (MC), and temporal difference (TD(0)) methods. The white and black circles indicate states and actions independently, and the square in MC represents a terminal state. The nodes in the blue shaded areas are considered at each update step for the corresponding methods.

Algorithm 3 Sarsa, on-policy TD(0) control

```
Initialize $Q(s, a)$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}$.
for each episode do
    Initialize $s \leftarrow s_0$
    Choose an action $a$ for $s$ using the current policy derived from $Q$
    repeat each step of episode
        Execute $a$, and observe $r$ and $s_{t+1}$
        Choose $a'$ for $s'$ using the current policy derived from $Q$
        $Q(s, a) \leftarrow Q(s, a) + \alpha (r + \gamma Q(s', a') - Q(s, a))$
        $s \leftarrow s'$ and $a \leftarrow a'$
    until $s$ is terminal
end for
```

return, they are limited to episodic tasks.

On the other hand, temporal difference (TD) methods bootstrap like DP methods instead of sampling until a final outcome. Therefore, one can view TD methods as a combination of Monte-Carlo methods (sampling) and DP methods (bootstrapping). Fig.2.1 shows the differences of the backup diagrams of the three methods. The simplest TD method uses a sample after one step, $< s_t, a_t, r_t, s_{t+1} >$ and is called as TD(0). In other words, a learning agent at state s_t performs an action a_t and receives r_t and s_{t+1}. We can then use $r_t + \gamma V(s_{t+1})$ as a stochastic sample of the Bellman equation 2.1 and update the values through stochastic ascent.

$$V(s_t) \leftarrow V(s_t) + \alpha (r_t + \gamma V(s_{t+1}) - V(s_t))$$

where $\alpha \in [0, 1)$ is a step-size parameter or learning rate. Since it updates toward $(r_t + \gamma V(s_{t+1})$, we call this as the *TD target*, and the error between the TD target and the current estimate, $r_t + \gamma V(s_{t+1}) - V(s_t)$ as the TD error and is denoted as δ_t.

In order to use the TD prediction in control problems, we implement the idea of generalized policy iteration. During learning, all of them should be repeatedly visited by the agent to evaluate the values for all state and action pairs. However, in order to increase the sum of future outcomes, the agent should exploit at some point and chooses optimal actions only based on her current policy. This is called as *exploration-exploitation trade-off* and it is one of the major challenges in reinforcement learning. There are two methods to ensure the agent to explore all the state and action pairs in a control problem – *on-policy* control and *off-policy* control. In general, we call a policy used in choosing an action as *behavior* policy and a policy learned as *target* policy. On-policy control methods use the same policy for both target and behavior policy while off-policy control methods have different policies for them.

The TD target of on-policy TD(0) control algorithm is, therefore, $r_t + \gamma Q(s_{t+1}, a_{t+1})$ where a_{t+1} is chosen by the current behavior policy. This algorithm is called *Sarsa* since it uses the quintuple of events, $< s_t, a_t, r_t, s_{t+1}, a_{t+1} >$, in its update. The complete algorithm is presented in Algorithm 3.

The off-policy TD(0) control is known as *Q-learning*, one of the most popular reinforcement learning algorithms. In Q-learning, the TD target directly approximates the Bellman optimality equation (Eq.2.2) and its update rule is:

$$\begin{aligned} Q(s_t, a_t) \leftarrow & Q(s_t, a_t) + \alpha \cdot \text{TD error} \\ \leftarrow & Q(s_t, a_t) + \alpha \left(r_{t+1} + \gamma V(s_{t+1}) - Q(s_t, a_t) \right) \end{aligned} \tag{2.4}$$

As mentioned above, any policy can be chosen for the behavior policy since it is an off-policy method. The algorithm is presented in Algorithm 4. The convergence of the algorithm to the optimal Q-values have been proved under a specific conditions [131].

Theorem 1. *Given bounded rewards $|r_t| < \infty$, learning rates $0 \leq \alpha_t < 1$, and*

$$\sum_{t=1}^{\infty} \alpha_t = \infty, \quad \sum_{t=1}^{\infty} (\alpha_t)^2 < \infty \tag{2.5}$$

then $Q_t(s, a) \rightarrow Q^*(s, a)$ as $t \rightarrow \infty$, $\forall s, a$ with probability 1.

The proof is presented in [131]. A common choice of the learning rate is a function that output value decreases with the number of visits to a corresponding state-action pair.

Algorithm 4 Q-learning, off-policy TD(0) control

```
Initialize $Q(s,a)$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}$.
**for** each episode **do**
    Initialize $s \leftarrow s_0$
    **repeat**each step of episode
        Choose an action $a$ for $s$ using a behavior policy derived from $Q$
        Execute $a$, and observe $r$ and $s_{t+1}$
        $Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$
        $s \leftarrow s'$
    **until** $s$ is terminal
**end for**
```

2.4 Value Function Approximation

So far we have only considered finite state and action spaces, and we learn the values or Q-values for all states and actions. Since we can represents the values in a table, we refer this as a tabular case. When the number of finite states and/or actions is very large or the state and/or action space is continuous, it is infeasible to represent and learn values and a policy for all possible states and actions. This issue is stated as a *curse of dimensionality* problem and is one of the major challenges in reinforcement learning. One solution for large and continuous MDPs is value function approximation where we represent the value or Q-value using a function with parameters θ:

$$\hat{V}(s;\theta) \approx V^{\pi}(s)$$
$$\hat{Q}(s,a;\theta) \approx Q^{\pi}(s,a)$$

For example, suppose that a task of a learning agent is to escape a maze navigating among obstacles in two-dimensional environment. We define a state as the xy-coordinate, $s_t = [x_t, y_t]$ and $x_t \in I\!R$, $y_t \in I\!R$. Intuitively, value at $s = [3.0, 3.0]$ should be close to $s = [3.05, 3.0]$. Therefore, in value function approximation, we learn the parameters of a function approximator instead of the exact values for all states and actions.

The input for the function can be either s or (s,a), and the output of the function can be $V(s)$, $[Q(s,a_1), \cdots Q(s,a_n)]$, or $Q(s,a)$. There are many possible function approximators including linear combinations of features, neural network, decision tree, nearest neighbor, and fourier basis. For differentiable function approximators such as neural network and linear combinations, we can use stochastic gradient descent to find the optimal values for approximation parameters, θ^*. The objective here is minimizing the mean square value error:

$$J(\theta) = \mathbb{E}_{s\sim\mu(\cdot)}\left[\left(V^\pi(s) - \hat{V}(s;\theta)\right)^2\right] \tag{2.6}$$

where $\mu(\cdot)$ is a state distribution and often chosen to be the fraction of time spent in a certain state. The gradient of $J(\theta)$ with respect to θ is

$$\frac{\partial J(\theta)}{\partial\theta} = 2\mathbb{E}_{s\sim\mu(\cdot)}\left[V^\pi(s) - \hat{V}(s;\theta)\right]\left(-\nabla_\theta \hat{V}(s;\theta)\right) \tag{2.7}$$

when $\hat{V}(s;\theta)$ is differentiable. Then, we update θ with the direction of minimizing the error,

$$\Delta\theta = -\frac{1}{2}\alpha\nabla_\theta J(\theta) \tag{2.8}$$

where α is a positive step-size. 1/2 is multiplied here in order to cancel the factor 2 in Eq.2.7. However, computing the exact expected value in Eq.2.7 is often challenging and inefficient. Instead, we use stochastic gradient descent (SGD) to minimize the error by adjusting the parameters θ with a small amount after each example is observed. In other words,

$$\theta \leftarrow \theta - \alpha\left(V^\pi(s) - \hat{V}(s;\theta)\right)\left(-\nabla_\theta \hat{V}(s;\theta)\right) \tag{2.9}$$

Mathematically, its expected update is identical to the full gradient update.

The simplest function approximator is a linear function, a linear combination of features $\phi(s) = [\phi_1(s), \cdots, \phi_n(s)]^T$. The value function is represented by

$$\hat{V}(s;\theta) = \phi(s)^T\theta$$

Then, the gradient of $J(\theta)$ is now linear in θ,

$$\Delta\theta = \alpha \left(V^{\pi}(s) - \phi(s)^T\theta\right)\phi(s)$$

and SGD converges on global optimum. However, what is the value for $V^{\pi}(s)$ in the above equation? This is again a target value for the update, and as we have seen in the classical reinforcement learning algorithms, we can use a sampled value for the target such as Monte-Carlo target or TD target. For TD(0) learning, the update becomes:

$$\Delta\theta = \alpha \left(r_t + \gamma\hat{V}(s_{t+1};\theta) - \phi(s)^T\theta\right)\phi(s)$$

For control problems, the same approximation techniques are applied to Q functions instead of V, for example, $\hat{Q}(s, a; \theta) \approx Q^{\pi}(s, a)$. For on-policy TD(0) control, the update is:

$$\Delta\theta = \alpha \left(r_t + \gamma\hat{Q}(s_{t+1}, a_{t+1};\theta) - \phi(s)^T\theta\right)\phi(s)$$

and for off-policy TD(0) control, the update is:

$$\Delta\theta = \alpha \left(r_t + \gamma \max_{a'} \hat{Q}(s_{t+1}, a';\theta) - \phi(s)^T\theta\right)\phi(s)$$

MC control, Sarsa, and Q-learning guarantee to converge to optimal values in the tabular cases. However, with a linear function approximator, MC control and Sarsa chatters around near-optimal values, and Q-learning is not guaranteed to converge. Moreover, none of them is guaranteed to converge to optimal values with non-linear function approximator such as neural network.

2.5 Deep Reinforcement Learning

Despite the convergence issue, neural networks have been successfully applied to reinforcement learning contributing to the field, *deep reinforcement learning.* One of the early successes in this application is TD-Gammon [123] which combined the TD(λ) algorithm [122], a model-free TD learning, and a multi-layer neural network for the game of backgammon and achieved near the level of the world's best grandmasters. However, the TD-Gammon method failed to be successful in other applications. The major challenges of applying deep learning methods to reinforcement learning are 1) noisy, sparse, and delayed reward, 2) correlated data samples, and 3) changes in the data distribution. Deep Q-network (DQN), which combines Q-learning with a deep convolutional neural network or multi-layer neural network, solves these challenges by using experience replay, a fixed target Q-network, and normalized rewards [92, 93]. DQN is the first deep reinforcement learning algorithm that achieved near-human performance in 49 different Atari 2600 games with raw sensory inputs. As it uses the raw sensory inputs, it requires extended training stages, but it drops the dependency of the algorithm on the design of feature vectors. Following this success, numerous studies have introduced deep reinforcement learning algorithms (mostly scaling up prior work in reinforcement learning) or applied them to interesting applications such as complex video games and robotic controls. AlphaGo, an artificial intelligence machine trained by using both supervised and reinforcement learning, defeated a grandmaster of the game of Go [116]. Double DQN extends Double Q-learning [51] applying similar techniques used in DQN and improves the stability and estimation bias of DQN through a use of a double estimator [129]. Dueling DQN extends the DQN architecture by learning networks for the value function, V^π, and the advantage function, A^π where $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$, instead of Q^π [130]. Besides these algorithms, many studies improve DQN with various approaches, and the Rainbow algorithm combines improvements of different DQN extensions [54].

Deep learning has also been applied to other types of reinforcement learning such as policy search and actor-critic methods. Trust region policy optimization (TRPO) optimizes a local approximation to the expected return of the policy proving monotonic improvement [113]. Proximal Policy Optimization (PPO) optimizes a surrogate objective function using stochastic gradient ascent [114]. It improves TRPO in a way that it is simpler to implement and empirically shows better sample complexity. Deep Deterministic Policy Gradient

(DDPG) is an actor-critic method that learns a Q-network (critic network) as well as a policy network. It extends the DQN algorithm with a policy gradient method and also proposes a soft target network update [82]. As it is learning a policy network separately, it is applicable to continuous action space problems. Normalized Advantage Functions (NAF) is another continuous variant of DQN which applies a simple and effective adaptive exploration method with an actor-critic architecture [44]. The Asynchronous Advantage Actor-Critic (A3C) algorithm uses multiple learning agents where each agent has its own network and asynchronously interacts with a learning environment [91]. It can be faster and more robust than similar algorithms as it employs the diversification of knowledge of multiple agents.

Deep reinforcement learning is a fast-growing field as of the time of this writing. However, most empirical results or demonstrations of algorithms in this field are limited in simulation, and efforts towards real-world applications such as meta reinforcement learning and transfer learning have received increasing attention lately. It has also been pointed out that the reproducibilities of popular algorithms remain in question. As discussed in this paper [53], it is important for authors to check reproducibility, to use proper experimental techniques, and to report details of procedures when proposing a new algorithm and claiming improvements over existing methods.

Chapter 3

Efficient Learning of Stand-up Motion for Humanoid Robots

3.1 Introduction

There has been much interest in using humanoid robots for tasks in various applications such as disaster response, personal assistance, and education. In these scenarios, robots are required to walk around and complete complex tasks while alternating between different body poses. This requires complicated motions exposing the robots to situations where they are likely to fall down. For example, twenty-four teams recently competed in Defense Advanced Research Project Agency (DARPA) Robotics Challenge Final. The mission of the challenge was to complete a course of eight tasks related to disaster response. Most of the robots that competed were humanoid since the environment was designed to be similar to human environments. However, many robots fell during the competition (Fig.3.1), and only one robot from Team Tartan was able to stand up after falling. Although the robot spent considerable time to stand up, it was able to resume the remaining tasks without restarting the challenge. Other robots may have been able to complete each individual task, but did not have the opportunity since they could not stand up and had to restart the challenge after falling. This example shows that having more efficient and versatile stand-up motions is clearly important for humanoid robots.

Stand-up motions for humanoid robots have previously been studied with a number of different types of robots. Stückler et. al. suggested four episodic phases in each stand-up motion from the prone and supine positions, respectively [121]. Hirukawa et. al. also developed a stand-up motion for the HRP-2P human-size humanoid robot [56]. However, such stand-up motions cannot be easily implemented to other robot models because of their high dependency on specific robots. Moreover, they are only applicable under strict conditions and they consist of only one or two courses of actions with constant joint angles. This significantly limits the number of fallen body configurations available. Humanoid robots can be damaged by executing their fixed stand-up motion sequences regardless of what their current body configurations are. It had not been a major problem because most humanoid robots were designed to perform only a few specific tasks using either their upper bodies only (e.g. manipulator robots) or their lower bodies only (e.g. robot soccer). However, as the DARPA Robotics Challenge example shows, humanoid robots are now expected to perform various tasks using their both upper and lower bodies simultaneously increasing the possibility of falling with numerous postures. Thus, it is necessary to find stand-up motions which can be safely applied to different body configuration of humanoid robots, can be easily implemented to different types of humanoid robots, and can be extended to more complex environments.

One possible approach to finding such stand-up motions is learning the motions using reinforcement learning. Reinforcement learning enables robots to learn their optimal course

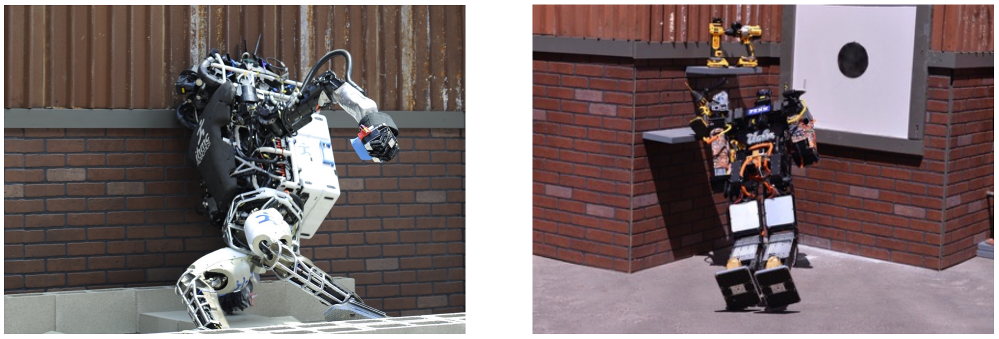
Figure 3.1: Robot falls during the DRC Finals: Team IHMC (left) and Team THOR (right).

of actions by interacting with their environment [122]. One major benefit of using reinforcement learning rather than supervised learning is that there is no need for having a knowledgeable external supervisor. In other words, humans are not always correct about which courses of actions are optimal for many different initial fallen poses. Reinforcement learning has been applied to many robotic tasks, and due to complexities such as continuous, high-dimensional state and action spaces, various learning approaches have been introduced and evaluated on robotic systems [71]. Peters et. al. introduced the Natural Actor-Critic algorithm and applied the algorithm to learning nonlinear motor primitives for arm movements of a humanoid robot [105]. The algorithm has also been applied to other humanoid robotics applications [45, 128]. Morimoto and Doya proposed a stand-up motion of a three-link-two-joint robot obtained using Hierarchical reinforcement learning [95]. They used Q-learning with discrete states and actions for the upper level and TD(λ)-learning with actor-critic method with continuous states and actions for the lower level. However, stand-up motions of a full body humanoid robot (having both legs and arms) requires one to consider movements of all its joints and thus these previous approaches may not be applicable to learning stand-up motions. Mordatch et. al. presented a method for motion control of humanoid robots [94]. They used a model-based policy search with trajectory optimization and neural network for high-level commands, and a model-free technique for low-level joint control. They demonstrated their method on a physical robot, but complex tasks such as standing up were only tested in simulation.

In this chapter, we introduce a method for learning stand-up motion for humanoid robots. In Sec 3.2, we define representative states and actions using a clustering technique instead of applying a function approximation method. A reward function is designed by considering various key features which affect stand-up motion. Our reward function for stand-up motions enables our low cost biped platform, DarwIn-OP humanoid robot, to receive feedback from the environment without requiring additional sensors such as a force sensor detecting ground contact [94]. Additionally, we present the *LS* (*L*earning in *S*ymmetric spaces) method which utilizes the bilateral symmetric feature of humanoid robots and learns more efficiently. We learned the policy in simulation using Q-learning and were able to efficiently update Q values using the symmetric state-action pairs.

This chapter is based on the papers presented in Thirtieth AAAI Conference on Artificial Intelligence (AAAI) and 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) [61, 62].

3.2 Representative States and Actions by Clustering

In order for a state to correspond a certain pose or a group of poses of a humanoid robot, it must contain the information of all joint positions and body attitude angles of the robot. Degrees of freedom of most full body humanoid robots range from 18 to more than 50. Thus, if we define a state space with all these variables, the state space will be continuous with at least twenty components. Function approximation techniques have been employed in many reinforcement learning problems in continuous domains. The simplest function approximator for a value function is a linear function approximator with a basis functions such as polynomial, Gaussian, radial, and Fourier basis. Although it has been widely used in reinforcement learning and has shown reasonable results, dimension of feature vectors from these basis functions is still large because of the dimension of the original state space is large. For example, for nth order Fourier expansion in d dimension, the function approximation requires $(n+1)^d$ basis functions [73].

Instead of using a function approximation method, we define *representative states* by discretizing continuous raw input data of a robot (all joint positions and body attitude angles) using a clustering method, *Expectation-Maximization* (EM) algorithm for Gaussian Mixtures [62]. EM is powerful learning algorithm to find the maximum likelihood for models with hidden variables [11]. This iterative algorithm optimizes the marginal likelihood by alternating two steps, called the E step and M step. Each centroid of clusters, then, represents a class of poses that can be visited by the robot during standing up after falling. The algorithm is presented in Algorithm 5.

Algorithm 5 Expectation-Maximization for Gaussian Mixtures [11]

```
1: Initialize the means, $\mu_k$, covariances, $\Sigma_k$, and mixing coefficients, $\pi_k$ for $k = 1, \cdots, K$
   where $K$ is the total number of clusters.
2: Given a data set of observations $X = \{x_1, \cdots, x_N\}$
3: Evaluate the initial log likelihood : $ll = \log p(X|\mu, \Sigma, \pi)$
4: repeat
5:     E step: Evaluate the responsibilities for $n = 1, \cdots, N$ and $k = 1, \cdots, K$:
           $$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_n|\mu_j, \Sigma_j)}$$
6:     M step: Re-estimate the parameters using $\gamma(z_{nk})$ (where $N_k = \sum_{n=1}^{N} \gamma(z_{nk})$):
           $$\mu_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) x_n$$
           $$\Sigma_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})(x_n - \mu_k^{new})(x_n - \mu_k^{new})^T$$
           $$\pi_k^{new} = \frac{N_k}{N}$$
7:     Evaluate the log likelihood $ll^{new} = \log p(X|\mu^{new}, \Sigma^{new}, \pi)$
8: until $|ll^{new} - ll| < \Delta$
```

First, we define a continuous input vector, $\mathbf{u}$:

$$\mathbf{u} = [q_1, q_2, \cdots, q_{N_J}, \varphi, \theta]^T, \quad \mathbf{u} \in U =^{(N_J+2)}$$

where N_J is the number of joints of a robot, $q_i \in [-\pi, \pi]$ is ith joint angle (rad), and $\varphi, \theta \in [-\pi, \pi]$ represent roll and pitch of the robot with respect to the global frame, respectively. We do not include yaw angle of the robot since how much the robot is rotated around the gravity axis does not affect to describe its pose under the assumption. However, yaw angle should be included in more complex environment such as inclined surface and obstacle environments. Since it is obvious that there is no reason to explore forward actions when the robot falls down backward and vice versa, we consider the stand-up as two separate cases: standing up from chest-down fallen body positions ($\theta_{pitch} \geq 0$ subscript F) and standing up from back-down fallen body positions ($\theta_{pitch} < 0$ subscript B).

We collect continuous pose data points, $mathbfu$, by randomly generating random motions from numerous fallen position (details are discussed in the experiment section). Applying EM with initial spherical covariance matrices and priors in uniform frequency to the collected continuous data, we construct N_c number of clusters where $N_{c,F}$ clusters are for the chest-down fallen position case and $N_{c,B}$ clusters are for the other case ($N_c = N_{c,F} + N_{c,B}$). Let $\mathbf{z}_k$ be the centroid vector of the kth cluster. Then, its first N_J components correspond to joint positions and the last two components correspond to body roll and pitch angles. In practice, some centroids cannot describe static poses or cause self collisions, so it is necessary to modify values of such centroids to values of their closest static poses. The representative states are defined by the centroids as:

$$s_t = s^{(i)} \text{ if } i = \underset{1 \leq k \leq N_c}{\operatorname{argmin}} \|\mathbf{z}_k - \mathbf{u}_t\|_2$$

and its state space, S, is defined as:

$$s^{(i)} \in S = S_F \cup S_B, \qquad \text{for } i = 1, \cdots, N_c$$

where S_F and S_B are subspaces corresponding to the forward and backward regions, respectively. States in their intersection, $S_F \cap S_B$, are finalizing states, close to neutral postures with $\varphi = 0$, which can be reached from both cases.

Representative action is defined as a transition to a state, and thus, each action correspond to one of the states in S. Each centroid vector of the clusters defines goal positions of all the joints at each action as:

$$a^{(k)}\text{: Move all joints to goal positions, } \mathbf{q}_{goal}^{(k)}$$

$$(\mathbf{q}_{goal}^{(k)})_j = (\mathbf{z}_k)_j \quad \text{for } j = 1, \cdots, N_J$$

Similar to the state space, the action space, A, consists of two subspaces with finalizing actions in $A_F \cap A_B$:

$$a^{(i)} \in A = A_F \cup A_B, \quad \text{for } i = 1, \cdots, N_c$$

3.3 Reward Function for Stand-up Motion

3.3.1 Reward Variables

A reward function in reinforcement learning informs a learning agent whether executing an action at a current state is beneficial for achieving a goal [122]. When the goal of the robot is to learn appropriate stand-up motions, the following five factors are affected by stand-up motions [62]. At step t when $s_t = s^{(i)}, a_t = a^{(j)}$, the reward variables $(\xi_{1,t}, \cdots, \xi_{5,t})$ are:

- Accumulated body angular acceleration differences

$$\xi_{1,t} = \sum_{k=1}^{m-1} \frac{\|(\omega(\tau_{k+1}) - \omega(\tau_k))\|_2}{\tau_{k+1} - \tau_k}$$

  where $\tau_1 = \tau(t), \tau_m = \tau(t+1)$ are real time (sec) at step t and step $t+1$, respectively, when each joint moves with m linear interpolation.

- Height Difference,

$$\xi_{2,t} = H_{t+1} - H_t$$

  where H_t is the height of the center of mass of the robot before it executes the action a_t.

- Body Pitch difference

$$\xi_{3,t} = |\theta_t| - |\theta_{t+1}| + c_\theta(\text{sgn}(\theta_{t+1}\theta_t) - 1)$$

- Body Roll difference

$$\xi_{4,t} = |\varphi_t| - |\varphi_{t+1}| + c_\varphi(\text{sgn}(\varphi_{t+1}\varphi_t) - 1)$$

- Difference from goal attitude of $a^{(j)}$

$$\xi_{5,t} = \left\| [z_{j,n_J+1}, z_{j,n_J+2}]^T_{goal} - [\varphi, \theta]^T_{t+1} \right\|$$

If the angular acceleration of the body is too high, the robot can potentially hit the ground with too large impact force, fall down again, or execute a very unstable movement. This enables us to evaluate motions in low-cost humanoid robot platforms such as Darwin-OP without requiring force sensors. Moreover, intuitively, the height of the robot should be increased and its body pitch and roll angles have to be closer to 0 over time in order to stand up. Thus, $\xi_{2,t}$ is positive if its height has been increased, and $\xi_{3,t}$ and $\xi_{4,t}$ are positive if the body pitch and roll become closer to zero. The last terms of the third and fourth equations represent penalties of moving to the opposite side (*e.g.* moving from $\theta_t = 60$ to $\theta_{t+1} = 10$ is better than moving from $\theta_t = 60$ to $\theta_{t+1} = -10$). $\xi_{5,t}$ represents how close the robot is to its goal by executing the action at the current state.

However, none of them can determine a proper reward value by itself. For example, $\xi_{1,t}$ will be high when the robot moves from its current pose to a good pose omitting intermediate poses. Similarly, it is not always beneficial to increase its height. It is possible that it moves its head to a lower position and its feet at a higher position. Therefore, we consider the trade-off between those variables.

3.3.2 Reward Function

The total reward function is defined as :

$$r_t = \sum_{i=1}^{5} r_i = \sum_{i=1}^{5} w_i(\tanh\big(c_i(\xi_{i,t} - \nu_i)\big) + b_i)$$

where w_i, c_i, ν_i, and b_i are constants which determine the trade-off between the effects of the five variables. Hyper tangent function is used to saturate too small or too large values. For example, we ignored small angular velocity differences and assigned the maximum penalty, -1, for values above a threshold. ξ_1 and ξ_5 are used only for penalty and others are used for

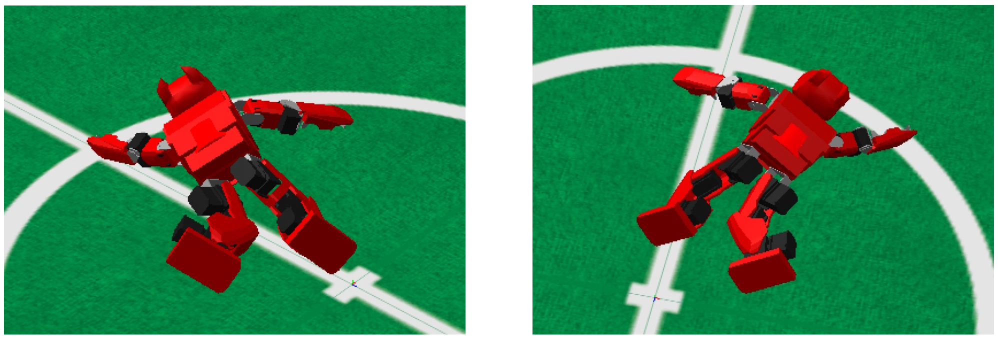

Figure 3.2: Symmetric poses for the DarwIn-OP robot

reward as well as penalty ($r_i \in [-1, 0]$ for $i = 1, 5$ and $r_i \in [-1, 1]$ for $i = 2, 3, 4$).

3.4 Bilateral Symmetric Feature of Humanoid Robots

In general, humanoid robots have bilateral symmetry. Thus if there is a course of actions, $\{a_1, \cdots, a_n\}$, which enables a robot to stand up from a certain fallen position m, the robot is also able to stand up from a fallen position symmetric to m by performing a course of actions symmetric to $\{a_1, \cdots, a_n\}$ (Fig.3.2). This means that one can obtain two optimal motions for two symmetric fallen positions with just one side. In addition, a robot can attempt more actions during learning which may allow the robot to find a better action, especially in cases with discrete actions. Therefore, we defined another state space, X, symmetric to S excluding some states in S bilaterally symmetric to themselves. Let $\bar{s}$ denote the symmetric state of the state s, then $X = X_F \cup X_B$ and $X_F \cap X_B = \varnothing$ where

$$X_F = \{x^{(i)} | x^{(i)} = \bar{s}^{(i)} \text{ non-bilaterally symmetric } \forall s^{(i)} \in S_F\}$$
$$X_B = \{x^{(i)} | x^{(i)} = \bar{s}^{(i)} \text{ non-bilaterally symmetric } \forall s^{(i)} \in S_B\}$$

X_F and X_B do not share any state unlike S_F and S_B do because the finalizing states are bilaterally symmetric to themselves.

A state at step t, therefore, is determined by finding the closest cluster to its continuous input data at t as follow:

$$\begin{aligned} s_t = & \min(s^{(i)}, \bar{s}^{(j)}) \\ \text{where} \quad & i = \text{argmin}_{1 \le k \le N_c} \|\mathbf{z}_k - \mathbf{u}_t\|_2 \\ & j = \text{argmin}_{1 \le k \le |X|} \|\bar{\mathbf{z}}_k - \mathbf{u}_t\|_2 \end{aligned}$$

There also exists the symmetric action space, $B = B_F \cup B_B$ and $B_F \cap B_B = \varnothing$ where

$$B_F = \{b^{(i)} | b^{(i)} = \bar{a}^{(i)} \text{ non-bilaterally symmetric } \forall a^{(i)} \in A_F\}$$
$$B_B = \{b^{(i)} | b^{(i)} = \bar{a}^{(i)} \text{ non-bilaterally symmetric } \forall a^{(i)} \in A_B\}$$

3.5 Learning in Symmetric Spaces

In order to find the optimal policy for standing up motions, Q-learning, an off-policy TD control algorithm, is used ($\gamma = 0.9$) with the Boltzmann action policy with $\tau = 0.3$ [122]. The learning algorithm is summarized in Algorithm 6.

3.5.1 Updating Symmetric State-Action Pairs in Learning

In this learning algorithm, we applied the bilateral symmetric feature of humanoid robots to learn quicker than an algorithm with the same number of states and actions but without the symmetric feature. Moreover, it can learn more flexible motions than ones learned by an algorithm with only S and A. In reinforcement learning, policy π defines the behavior of a learning agent (the robot in this case). In this framework, a state-action pair has the same policy with its symmetric pair.

$$\pi(s^{(i)}, a^{(j)}) = \pi(\bar{s}^{(i)}, \bar{a}^{(j)}) \quad \text{and} \quad \pi(s^{(i)}, \bar{a}^{(j)}) = \pi(\bar{s}^{(i)}, a^{(j)})$$

Algorithm 6 Learning in Symmetric Spaces with Q-learning

```
1: Initialize $Q(s,a)\ \forall s \in \mathcal{S},\ \forall a \in \mathcal{A}$
2: for each step $t$ do
3:     Input: $\mathbf{u}_t = [q_1(t), \cdots, q_J(t), \phi(t)\theta(t)]^T$
4:     Assign $s_t = s^{(i)}$ where $i = \text{argmin}_{l \in Z \cup \bar{Z}} \|\mathbf{z}_l - \mathbf{u}_t\|_2$
5:     Choose $a_t = a^{(j)}$ from $\pi^{action}$
6:     Execute $a_t$ and observe $\mathbf{u}_{t+1},\ s_{t+1} = s^{(k)},\ r_t = f_r(s_t, a_t, s_{t+1})$
7:     if $s_t \in X$ then
8:         $s = \bar{s}^{(i)}, a = \bar{a}^{(j)}$
9:     else
10:        $s = s^{(i)}, a = a^{(j)}$
11:    end if
12:    if $s_{t+1} \in X$ then
13:        $s' = \bar{s}^{(k)}$
14:    else
15:        $s' = s^{(k)}$
16:    end if
17:    Q-Update: $Q(s,a) \leftarrow Q(s,a) + \alpha_t\big[r_t + \gamma \cdot \max_{a' \in \mathcal{A}} Q(s',a') - Q(s,a)\big]$
18: end for
```

Therefore, instead of updating Q values once for each of the four different pairs, we can update the values of the two different pairs twice. This allows us to reduce the size of the Q matrix. Thus, instead of having $(|S| + |X|) \times (|A| + |B|)$ parameters to learn, we have $|S| \times (|A| + |B|)$ parameters. From line 7 to 16 in Algorithm 6 show a procedure of transferring a state-action pair to its symmetric pair to update a corresponding Q value at the step t.

3.5.2 Learning Rate For Representative States and Actions

We used a modified learning rate related to the clustering method for the Q-update:

$$\alpha_t = \alpha(s_t, a_t, \mathbf{u_t}) = c \cdot p(\mathbf{u_t}|s_t) \cdot \frac{1}{(1 + visits(s_t, a_t))}$$

where c is a scaling constant and $p(\mathbf{u}|s)$ is the probability density function of having the discrete state s given the continuous input $\mathbf{u}$. This probability can be considered as a weight of the learning rate. If the current continuous input is far from the assigned cluster centroid, its learning result affects to the corresponding Q value less. Since EM for Gaussian Mixtures is used for clustering, we define this probability follows the Gaussian distribution with mean $\mathbf{z}$ and a spherical covariance σ^2.

$$p(\mathbf{u_t}|s_t = s^{(k)}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\big(-\frac{\|\mathbf{u}_t - \mathbf{z}_k\|_2^2}{2\sigma^2}\big)$$

This learning rate satisfies Watkins' theorem on convergence of Q-values [131] with selecting proper values for σ and c. Using $\sigma = 0.2$ and $c = 0.5$, the upper bound can be proved as following:

$$\begin{aligned}\alpha &= \frac{c}{\sqrt{2\pi\sigma^2}} \exp\big(-\frac{\|\mathbf{u} - \mathbf{z}\|_2^2}{2\sigma^2}\big) \frac{1}{(1 + visits(s,a))} \\ &\leq \frac{c}{\sqrt{2\pi\sigma^2}} \frac{1}{(1 + visits(s,a))} \leq \frac{c}{\sqrt{2\pi\sigma^2}} < 1\end{aligned}$$

The lower bound is simply proved since the constant terms and $1/(1+visits(s,a))$ are larger than 0 and the exponential function is always larger than 0. This learning rate significantly improves the learning performance.

3.6 Experiments

3.6.1 Experimental System Details

We applied our method to the DarwIn-OP humanoid robot. The robot consists of twenty MX-28 servo motors - three for each arm, six for each leg, and two for its neck. We did not use the neck joints for stand-up motions. Yi developed two stand-up motions for prone and supine fallen positions of this robot using direct joint level control [136]. The motions consist of five and seven motion sequences, respectively, and each sequence is a set of constant joint positions as similar to other approaches mentioned in the introduction. We denote these two motion sequences as M_F and M_B, respectively. They are used for prior knowledge as well as for a comparison when evaluating the result of the proposed learning method.

3.6.2 Generating Clusters

In order to find reasonable representative states by clustering, we generate random motions of a robot from many different fallen positions by randomly choosing one of left arm, right arm, left leg and right leg to move as well as its direction. Self-collisions and joint limits are used as constraints. By doing this, we can collect continuous data with the form of the input vector, **u** which informs many possible poses that the robot can make while standing up.

We constructed 14 clusters ($N_{c,F} = 10$, $N_{c,B} = 8$) from the collected dataset (Fig.3.3). Then, we added the motion sequences in M_F and M_B as centroids of additional clusters. They are not included in X and B since their motions are bilateral symmetric. We regarded two motion sequences which were in both M_F and M_B as finalizing states/actions. The subspaces and state (action) numbers are summarized in Fig.3.4.

In order to improve the learning efficiency further, we can use our prior knowledge for the initial Q values. Since executing an action in A_B is obviously not an optimal solution when the robot has a state in S_F, and vice versa, a negative value is assigned to the initial Q values of this case. The initial Q values we used are shown in Fig.3.5.

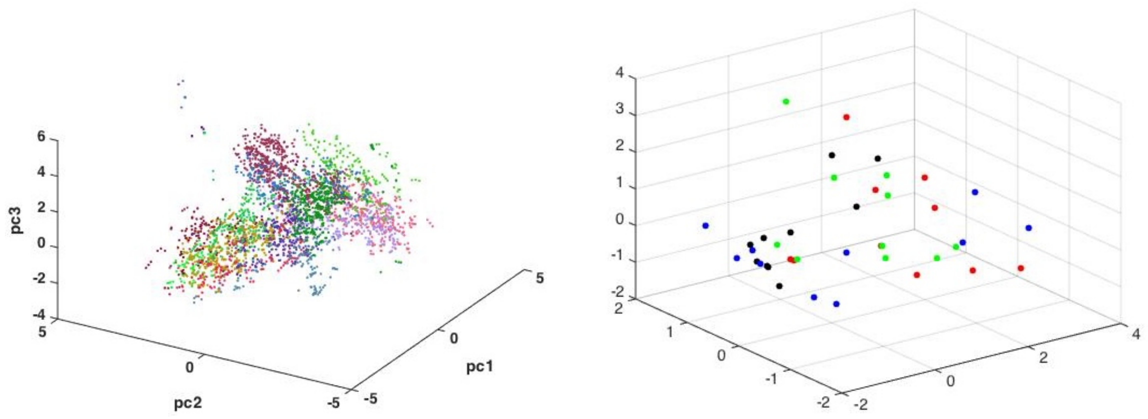

Figure 3.3: (Shown as converted by PCA(3)) The continuous data we collected by generating random motions from the robot (Left). The centroids of the final clusters before adding M_F and M_B (red(F) and blue(B)) with their symmetric centroids (green(F) and black(B)) (Right)

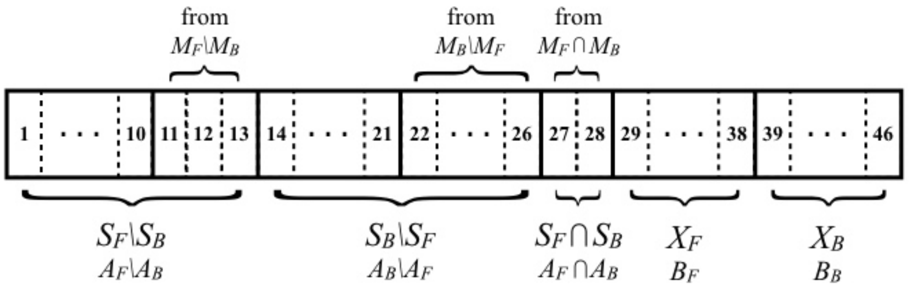

Figure 3.4: Assigned state(or action) indicies in state(or action) spaces

3.6.3 Results and Evaluation

In order to evaluate the significance of using the symmetric feature in LS method, two different non-LS methods were also tested. These methods do not consider the symmetric feature in the same model including the reward function and the initial Q values. The first method learned an optimal policy with states and actions only in S and A. Therefore, the number of states and the number of actions were 28 for each (non-$LS(28)$). The second method used the same state and action space definitions with LS method but did not include the symmetric features during learning. Therefore, the robot learned for 46 by 46 state-action pairs (non-$LS(46)$). Fig.3.5 shows the Q values (z axis) of all state-action pairs at $t = 850$ updated by LS method. The n umber of learning steps is determined by the number of visits to state-action pairs. A lot of states are not reachable from other states, so

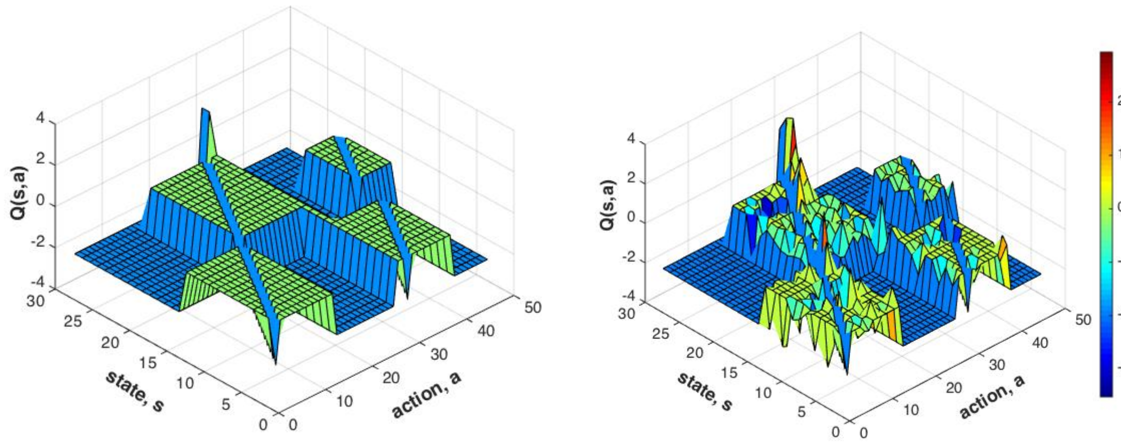

Figure 3.5: Left : Initial Q values, Right : Final Q values

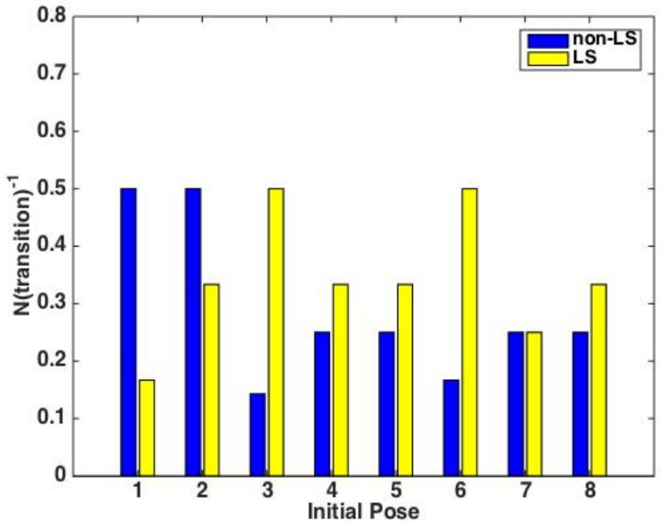

Figure 3.6: Evaluation on speed of performed motions with the real robot (red: LS, green: Manually designed Stand-up motion, blue: non-$LS(28)$, black: non-$LS(46)$)

the actual steps (samples) required is far less than $|S| \times (|A| + |B|)$.

As a result, the policy of the non-$LS(28)$ method resulted 7 failures among 15 trials while the policy of the LS method resulted 2 failures among 15 trials. One major reason of the failure in the non-$LS(28)$ method is its limited number of action choices. Moreover, the robot almost exploits to such actions rather than explores other actions. In addition to the success/failure test, we also evaluated how fast the robot was able to stand up in trials where both of the policies succeeded (8 trials among 15 trials, the plot in Fig.3.6). The y-axis of the plot is the reciprocal of the number of transitions in each trial (higher is faster). As the figure shows, LS method is faster than non-$LS(28)$ method in 5 trials and equal to the method in 1 trial. We applied these policies learned in the simulation as well as the manually designed stand-up motion, denoted as MS, to the real physical robot, and tested them with five different initial fallen poses. The bottom plot in Fig.3.7 shows the speed efficiencies of each method in each trial. Cases where the robot failed to stand up are not plotted. Thus, only LS method succeeded in all the trials and faster or equal to other methods which succeeded in each trial. Fig.3.8 is a sequence of screenshot images of an example result of the LS method.

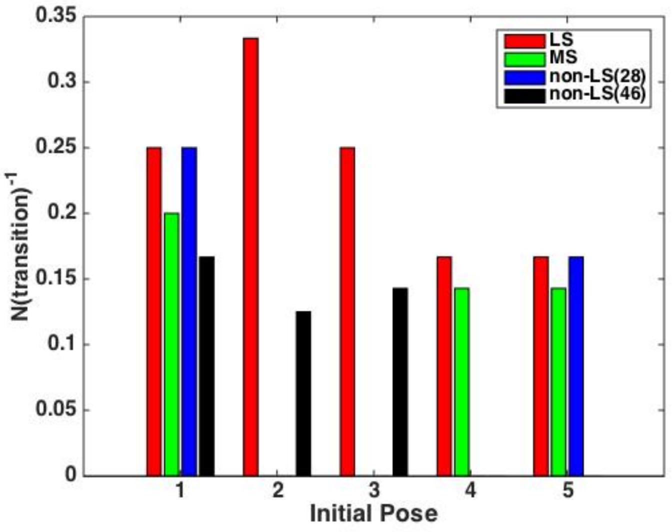

Figure 3.7: Evaluation on speed of performed motions with the real robot (red: LS, green: Manually designed Stand-up motion, blue: non-LS(28), black: non-LS(46))

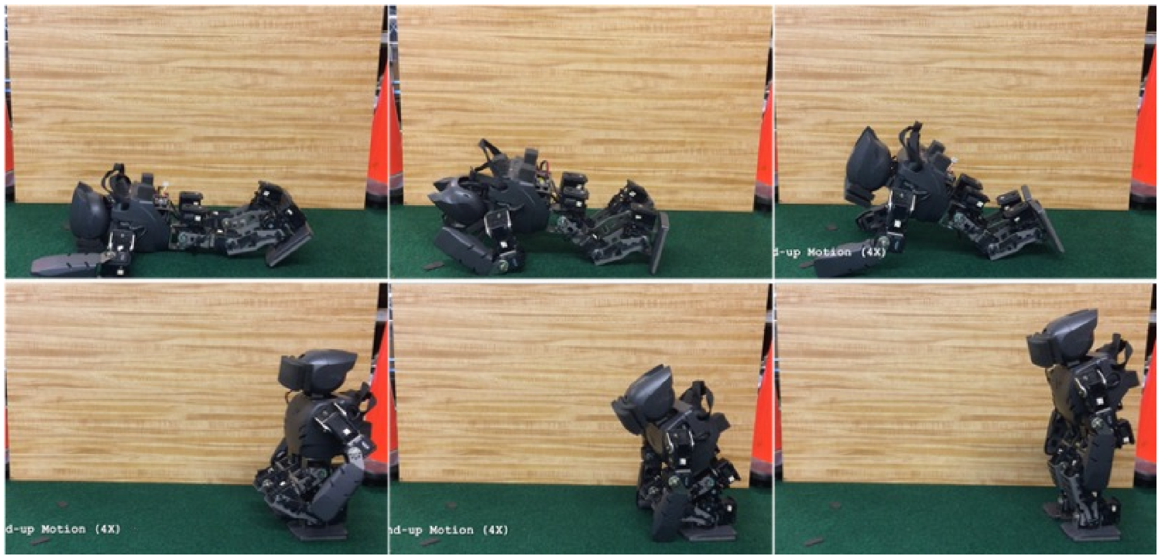

Figure 3.8: A sequence of a successful stand-up motion of a real-robot experiment example (from top left to bottom right).

3.7 Summary

In this chapter, we studied a new method of learning stand-up motions for humanoid robots using Q-learning and utilizing their bilateral symmetry. We first defined discrete representative states and actions applying EM for Gaussian Mixtures to high dimensional and continuous robot pose information data. Then, we defined their symmetric states and actions and incorporated this symmetry when updating the value function. We demonstrated our method by implementing it on DarwIn-OP humanoid robot. The result shows that the stand-up motion learned by our method has less number of motion transitions as well as much higher successful rate on standing up from various initial fallen poses than other stand-up motions - manual stand-up motions and non-LS methods.

Although the kinematic model of the robot was required in order to measure its height and to check self-collisions since we used low-cost biped without using any additional sensor, our method can be easily implemented to any type of humanoid robots. Moreover, this reinforcement learning approach enables the stand-up motions to be extended to more complex environment applying other learning methods such as transfer learning and Hierarchical reinforcement learning.

Chapter 4

Assumed Density Filtering Q-learning

4.1 Introduction

Despite the recent success of reinforcement learning, there remain several key questions. First, one of major challenges in reinforcement learning is the exploration-exploitation trade-off. When is the appropriate time for a learning agent to stop exploring and choose the best action for each state according to its current knowledge? Or should the learning agent continue exploring new actions in order to find a better policy? If it should continue exploring, which action should the learning agent select to explore? Second, some prior knowledge might only be partially known for certain states and/or actions, or might be known with different uncertainties. How can we incorporate this knowledge into a learning process so as to provide a good baseline?

One approach that can provide answers to these questions is Bayesian reinforcement learning. Bayesian reinforcement learning is one of the approaches in reinforcement learning that deploys Bayesian inference in order to incorporate new information into prior information. It explicitly quantifies the uncertainty of learning parameters unlike standard (frequentist) reinforcement learning which uses point estimates of the parameters. Therefore, an *explicit quantification of the uncertainty* can optimize the exploration-exploitation trade-off by exploring actions with larger uncertainty more often than actions with smaller uncertainty. Moreover, prior knowledge on learning parameters can be utilized in the update based on its uncertainty when there is a new example [134]. In the off-policy temporal difference learning, this problem is related to the overoptimism issue of Q-learning which have been discussed in a number of papers [50, 51, 127, 129]. The greedy max operator in Q-learning 2.4 leads to overestimated Q-values in the early stage of learning. It has been shown that it becomes especially problematic when combined with a function approximation method and/or when used for stochastic MDPs. Double Q-learning [51] improves the overestimation issue by having two estimators and using one for selecting the subsequent action and the other one for estimating the value of the subsequent state. However, as it is pointed out in the original paper, Double Q-learning is not a full solution of the overestimation issue of Q-learning and it sometimes underestimates the values. The double estimator still does not contain the degree of the prior knowledge of the learning agent on Q-values.

A number of algorithms have been proposed in both model-based Bayesian reinforcement learning [25, 30, 46, 107, 120] and model-free Bayesian reinforcement learning [20, 26, 32, 33, 36, 37]. However, Bayesian approaches to *off-policy temporal difference (TD) learning* have been less studied compared to alternative methods due to difficulty in handling the max non-linearity in the Bellman optimality equation. Dearden's Bayesian Q-Learning [26], Kalman Temporal Difference Q-learning (KTD-Q) [36], and Gaussian Process Q-learning (GPQ) [20] are perhaps the closest Bayesian counterparts to Q-learning. Dearden's approach assumes that $Q_{s,a}$ follows a Gaussian distribution and updates the distribution by numerically compute the moments. Although it outperforms Q-learning in several finite domains utilizing the uncertainty for exploration during learning, it does not guarantee the convergence to an optimal policy. Moreover, the computation of the algorithm is notoriously taxing. GPQ models $Q_{s,a}$ as a Gaussian Process and its convergence to optimal Q-values are provided. However, as a common limitation of the Gaussian Process, it does not scale to the high-dimensional state space. KTD-Q uses the unscented Kalman filter scheme. By analytically updating the mean and covariance of learning parameters, it requires less computation time than the Dearden's method. However, the Cholesky decomposition in the unscented transform demands high computational complexity. Additionally, the convergence of KTD-

Q to optimal Q-values is only shown in a deterministic environment (Further details of these algorithms are explained in Sec.4.2). These limitations prevent the algorithms from being used in practical settings and being extended to more complex environments. Yet off-policy TD methods in standard reinforcement learning such as Q-learning [131] have been widely used in standard reinforcement learning, including extensions integrating neural network function approximations such as Deep Q-Networks (DQN) [92].

In this chapter, we introduce a novel approximate Bayesian Q-learning algorithm, denoted as ADFQ, which updates belief distributions of Q (action-value function) and approximates their posteriors using an online Bayesian inference algorithm known as assumed density filtering (ADF). In order to reduce the computational burden of estimating parameters of the approximated posterior, we propose a method to analytically estimate the parameters. Unlike Q-learning, ADFQ considers all possible actions for the next state, and returns a soft-max behavior and regularization determined by the uncertainty measures of the Q-beliefs. This can alleviate the overoptimism and instability issues from the greedy update of Q-learning. We prove the convergence of ADFQ to the optimal Q-values by showing that ADFQ becomes identical to Q-learning as all state and action pairs are visited infinitely often. We demonstrate the performance of the algorithm in a set of deterministic and stochastic finite MDPs. In Chapter 5, we extend ADFQ with a neural network and show its promising performance in continuous state domains and Atari 2600 games.

This chapter is based on the paper [64] published in the proceedings of the 28th International Joint Conference on Artificial Intelligence. The ADFQ source code can be found in https://github.com/coco66/ADFQ.

4.2 Related Work

In this section, we will review some of Bayesian reinforcement learning algorithms that correspond to Q-learning in the classical reinforcement learning. This will help the readers to understand how Bayesian methods are integrated into the update of Q-values and how the uncertainty measures can be used to accelerate the learning.

Dearden et al. has proposed Bayesian Q-Learning (Dearden's BQL) assuming that the total discounted future reward, denoted as $R_{s,a}$ when executing action a at state s, is a Gaussian random variable with mean $\mu_{s,a}$ and precision $\tau_{s,a}$ [26]. The conjugate prior of $R_{s,a}$, $p(\mu_{s,a}, \tau_{s,a})$, is a normal-gamma distribution with parameters $< \theta_{s,a}, \lambda_{s,a}, \alpha_{s,a}, \beta_{s,a} >$. By the definition of the action-value function in reinforcement learning, $Q_{s,a} = \mathbb{E}[R_{s,a}] = \mu_{s,a}$ and is normally distributed with mean $\theta_{s,a}$ and precision $\lambda_{s,a}\tau$ [27]. Among the four learning algorithms suggested in the paper, Myopic-VPI action selection with Mixture updating (VPI+Mix) led to the best overall performance. The Myopic-VPI selection considers quantity of policy improvement when selecting an action. The value of perfect information (VPI) is defined as the expected gain:

$$VPI_{s,a} = \int_{-\infty}^{\infty} Gain_{s,a}(\mu^*_{s,a} = x) Pr(\mu_{s,a} = x) dx$$

where the gain of choosing action a at state s when the true mean, , is given is defined as:

$$Gain_{s,a}(\mu^*_{s,a}) = \begin{cases} \mathbb{E}[\mu_{s,a_2}] - \mu^*_{s,a} & \text{if } a = a_1 \text{ and } \mu^*_{s,a} < \mathbb{E}[\mu_{s,a_2}] \\ \mu^*_{s,a} - \mathbb{E}[\mu_{s,a_1}] & \text{if } a \neq a_1 \text{ and } \mu^*_{s,a} > \mathbb{E}[\mu_{s,a_1}] \\ 0 & \text{otherwise} \end{cases}$$

a_1 and a_2 are the best and the second best actions, respectively. Then, an action is chosen to maximize $VPI_{s,a}$ – expected cost, $a^* = \text{argmax}_a\{VPI_{s,a} - (\max_{a'} \mathbb{E}[Q_{s,a'}] - \mathbb{E}[Q_{s,a}])\} = \text{argmax}_a\{VPI_{s,a} + \mathbb{E}[Q_{s,a}]\}$.In the Mixture updating, the posterior distribution over $\mu_{s,a}, \tau_{s,a}$ is updated after an immediate reward, r, and is observed as:

$$p^{mix}_{r,s'}(\mu_{s,a}, \tau_{s,a}) = \int_{-\infty}^{\infty} p(\mu_{s,a}, \tau_{s,a} | r + \gamma x) p(R_{s'} = x) dx$$

where $R_{s'}$ is equal to $R_{s',\pi^*(s')}$ assuming that it follows the apparently optimal policy π^*. The distribution does not have a simple closed form, so they approximated it to the closest normal-gamma distribution using KL-divergence.

Although Dearden's BQL with VPI+Mix has been successfully implemented by other researchers [42, 88], it has limitations on theories and implementation. First, the Myopic-VPI selection does not guarantee that $\mu_{s,a}$ will converge to the true Q-value since it may not try each action infinitely often. It always selects an optimal action according to its current knowledge rather than occasionally selecting a suboptimal action. Second, the assumption on the distribution over $R_{s,a}$ and the Mixture updating does not satisfy the Bellman optimality equation. The maximum of Gaussian random variables with different means and variances does not have a Gaussian distribution. Additionally, $r+\gamma \max_{a'} \mathbb{E}[R_{s',a'}]$ has to be considered in the Mixture updating instead of $r + \gamma R_{s'}$ according to the Bellman optimality equation. Third, the assumption on $R_{s'} = R_{s',\pi^*(s')}$ ignores the stochasticity of choosing the best action at state s' according to his belief. Finally, the implementation of this approach is nontrivial, especially due to the numerical integration of $p^{mix}_{r,s'}(\mu_{s,a}, \tau_{s,a})$.

Gaussian Process model of the value function learning, denoted as GPQ, is an off-policy Bayesian non-parametric approximate reinforcement learning algorithm which models a true action-value function, $Q^*_{s,a}$, as a Gaussian Process with mean $m^*_{s,a}$ and positive semi-definite covariance kernel $k([s, a], [s', a'])$ [20]. The paper presents two GPQ algorithms in batch and online settings–Batch GPQ and Online GPQ.

Let the current estimate of the mean of the Q-function be $\hat{Q}(s, a) = \hat{m}(s, a)$. In Batch GPQ, at each step, the model of $\hat{Q}(s, a)$ is updated with observed data as $\hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma \max_{a'}(\hat{Q}(s_{t+1}, a'))$. In Online GPQ, the method used in Batch GPQ cannot be applied because of its expensive computational requirements for the convergence. Therefore, they suggested a modified algorithm online GP which chooses a basis vector to be added in the active bases set using the sparse online GP algorithm [22]. This online GPQ algorithm uses ϵ-greedy method for its exploration strategy which is simple but efficient. Therefore, they suggested another exploration strategy, called optimistic online GPQ, that chooses an action that are greedy with respect to a upper confidence tail, $m(s) + 2\Sigma(s_{\tau+1})$, while updating GP as $\hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma \max_{a'}(\hat{Q}(s_{t+1}, a') + 2\Sigma(s_{t+1}, a'))$.

The online GPQs with the ϵ-greedy and with the optimistic exploration strategies were tested in three domains - Gridworld(5×5), Inverted pendulum, and Puddle World. The optimistic GPQ showed the best performance in the first two domains, less complex problems than the Puddle World domain. In the last domain, pervious Q-learning variants showed better performance because the wide range of values around the puddle made it difficult to place basis. The authors concluded that GPQ performs almost as well as their best case with less information and far outperforms their worst-case results. However, the challenges in basis placement may provide a significant limitation when we apply the algorithm to complex domains.

Kalman Temporal Difference algorithm, denoted as KTD, approximates the value function using the Kalman filtering scheme. It considers parameters of the value function as hidden states and tracks them through indirect observations, or rewards from the environment.

Let θ_t be the hidden variables (or function approximation parameters) at time t. Then, from the one-step look ahead sample of the Bellman optimality equation, r_t can be expressed as

$$r_t = \hat{V_{\theta_t}}(s_t) - \gamma \hat{V_{\theta_t}}(s_{t+1})$$

or,

$$r_t = \hat{Q_{\theta_t}}(s_t, a_t) - \gamma \max_{a' \in A} \hat{Q}_{\theta_t}(s_{t+1}, a')$$

In KTD, r_t is considered as an observed variable and replaced by a function of the parameters, $g_t(\theta)$. The idea behind KTD is **to express value function approximation as a filtering problem**: the parameters are the hidden state to be tracked (modeled as random variables following a random walk), the observation being the reward linked to the parameters through a Bellman equation. And the variables are defined as:

$$\theta_t = \theta_{t-1} + v_t \quad r_t = g_t(\theta_t) + n_t$$

where the evolution noise, v_t, is white, independent and of variance matrix Σ_{v_t}, and the observation noise, n_t, is also white, independent and of scalar variance, σ_{n_t}. Given deterministic transitions, this model noise arises because the solution of the Bellman equation does not necessarily exist in the hypothesis space induced by the parameterization:

- random vector, θ_t and its estimation:

$$\hat{\theta}_{t|t-1} = \mathbf{E}[\theta_{t-1} + v_t | r_{1:t-1}] = \hat{\theta}_{t-1|t-1}$$

$$\hat{r}_{t|t-1} = \mathbf{E}[g_t(\theta_t) + n_t | r_{1:t-1}] = \mathbf{E}[g_t(\theta_t) | r_{1:t-1}]$$

- variance:

$$\begin{aligned} P_{t|t} &= cov(\theta_t - \hat{\theta}_{t|t} | r_{1:t}) \\ &= P_{t|t-1} - K_t P_{r_t} K_t^T \end{aligned}$$

  Note that this variance matrix encodes the uncertainty over parameter estimates, and not the intrinsic uncertainty of the considered MDP. It is not the variance of the random process from which the value function is the mean.

KTD-Q is the extension of KTD to Q-learning. They handle the nonlinearity of the max-operator in the update by the unscented transform. However, KTD, KTD-Sarsa, and KTD-Q are defined under the deterministic assumption of an MDP (which transition probabilities between states are either 0 or 1). XKTD and XKTD-Sarsa are proposed separately as extensions of the original algorithms for the stochastic case. However, KTD-Q was failed to be extended to the stochastic case (the section 4.3.2 in [36]). Another limitation of KTD-Q is that it requires many parameter values to be chosen and it is computationally expensive.

4.3 Belief Updates on Q-values

We define $Q_{s,a}$ as a Gaussian random variable with mean $\mu_{s,a}$ and variance $\sigma^2_{s,a}$ corresponding to the action value function $Q(s,a)$ for $s \in \mathcal{S}$ and $a \in \mathcal{A}$. We assume that the random variables for different states and actions are independent and have different means and variances, $Q_{s,a} \sim \mathcal{N}(\mu_{s,a}, \sigma^2_{s,a})$ where $\mu_{s,a} \neq \mu_{s',a'}$ if $s \neq s'$ or $a \neq a'$ $\forall s \in \mathcal{S}, \forall a \in \mathcal{A}$.

According to the Bellman optimality equation in Eq.2.2, we can define a random variable for $V(s)$ as $V_s = \max_a Q_{s,a}$. In general, the probability density function for the maximum of Gaussian random variables, $M = \max_{1 \leq k \leq N} X_k$ where $X_k \sim \mathcal{N}(\mu_k, \sigma_k^2)$, is no longer Gaussian:

$$\begin{aligned} Pr\left(M \leq x\right) &= \prod_{i=1}^{N} Pr(X_i \leq x) = \prod_{i=1}^{N} \Phi\left(\frac{x-\mu_i}{\sigma_i}\right) \\ p\left(M = x\right) &= \frac{d}{dx}\left(Pr\left(M \leq x\right)\right) \\ &= \sum_{i=1}^{N} \frac{1}{\sigma_i}\phi\left(\frac{x-\mu_i}{\sigma_i}\right)\prod_{i \neq j}^{N} \Phi\left(\frac{x-\mu_j}{\sigma_j}\right) \neq \text{Gaussian} \end{aligned} \tag{4.1}$$

where $\phi(\cdot)$ is the standard Gaussian probability density function (PDF) and $\Phi(\cdot)$ is the standard Gaussian cumulative distribution function (CDF).

For one-step Bayesian TD learning, the beliefs on $\mathbf{Q} = \{Q_{s,a}\}_{\forall s \in \mathcal{S}, \forall a \in \mathcal{A}}$ can be updated at time t after observing a reward r_t and the next state s_{t+1} using Bayes' rule. In order to reduce notation, we drop the dependency on t denoting $s_t = s$, $a_t = a$, $s_{t+1} = s'$, $r_t = r$, yielding the causally related 4-tuple $\tau =< s, a, r, s' >$. We use the one-step TD target with a small Gaussian white noise, $r + \gamma V_{s'} + W$ where $W \sim \mathcal{N}(0, \sigma_w^2)$, as the likelihood for $Q_{s,a}$. The noise parameter, σ_w, reflects stochasticity of an MDP. Thus, a larger value is required for σ_w for higher stochasticity in the reward function or the state transition. We will first derive the belief updates on Q-values with $\sigma_w = 0$ and then extend the result to the general case. The likelihood distribution can be represented as a distribution over $V_{s'}$, $p(r + \gamma V_{s'}|q, \theta) = p_{V_{s'}}((q-r)/\gamma|s', \theta)$ where q is a value corresponding to $Q_{s,a}$ and θ is a set of mean and variance of $\mathbf{Q}$. From the independence assumptions on $\mathbf{Q}$, the posterior update is reduced to an update for the belief on $Q_{s,a}$:

$$\hat{p}_{Q_{s,a}}(q|\theta, r, s') \propto p_{V_{s'}}\left(\left.\frac{q-r}{\gamma}\right| q, s', \theta\right) p_{Q_{s,a}}(q|\theta)$$

According to the Bellman optimality in Eq.2.2, $V_{s'}$ follows the distribution presented in Eq.4.1. The resulting posterior distribution is given as follows (derivation details in Appendix A.1):

$$\hat{p}_{Q_{s,a}}(q|\theta, r, s') = \frac{1}{Z} \sum_{b \in \mathcal{A}} \frac{c_{\tau,b}}{\bar{\sigma}_{\tau,b}} \phi\left(\frac{q - \bar{\mu}_{\tau,b}}{\bar{\sigma}_{\tau,b}}\right) \prod_{\substack{b' \in \mathcal{A} \\ b' \neq b}} \Phi\left(\frac{q - (r + \gamma\mu_{s',b'})}{\gamma\sigma_{s',b'}}\right) \tag{4.2}$$

where Z is a normalization constant and

$$c_{\tau,b} = \frac{1}{\sqrt{\sigma_{s,a}^2 + \gamma^2\sigma_{s',b}^2}} \phi\left(\frac{(r + \gamma\mu_{s',b}) - \mu_{s,a}}{\sqrt{\sigma_{s,a}^2 + \gamma^2\sigma_{s',b}^2}}\right) \tag{4.3}$$

$$\bar{\mu}_{\tau,b} = \bar{\sigma}_{\tau,b}^2 \left(\frac{\mu_{s,a}}{\sigma_{s,a}^2} + \frac{r + \gamma\mu_{s',b}}{\gamma^2\sigma_{s',b}^2}\right) \qquad \frac{1}{\bar{\sigma}_{\tau,b}^2} = \frac{1}{\sigma_{s,a}^2} + \frac{1}{\gamma^2\sigma_{s',b}^2} \tag{4.4}$$

Note that all next actions are considered in Eq.4.2 unlike the conventional Q-learning update which only considers the subsequent action resulting in the maximum Q-value at the next step ($\max_b Q(s', b)$). This can lead to a more stable update rule as updating with only the maximum Q-value has inherent instability [50, 127]. The Bayesian update considers the scenario where the true maximum Q-value may not be the one with the highest estimated mean, and weights each subsequence Q-value accordingly. Each term for action b inside the summation in Eq.4.2 has three important features. First of all, $\bar{\mu}_{\tau,b}$ is an inverse-variance weighted (IVW) average of the prior mean and the TD target mean. Therefore, the Gaussian PDF part becomes closer to the TD target distribution if it has a lower uncertainty than the prior, and vice versa as compared in the first row (a) and (b) of Fig.4.1. Next, the TD error, $\delta_{\tau,b} = (r + \gamma\mu_{s',b}) - \mu_{s,a}$, is naturally incorporated in the posterior distribution with the form of a Gaussian PDF in the weight $c_{\tau,b}$. Thus, a subsequent action which results in a smaller TD error contributes more to the update. The sensitivity of a weight value is determined by the prior and target uncertainties. An example case is described in the second row of Fig.4.1 where $\delta_{\tau,1} = \delta_{\tau,2} > \delta_{\tau,3}$ and $\sigma_{s',1} > \sigma_{s',2} = \sigma_{s',3}$. Finally, the product of Gaussian CDFs provides a soft-max operation. The red curve with dots in the third row of Fig.4.1 represents $\prod_{b' \neq b} \Phi(q|r + \gamma\mu_{\tau,b'}, \gamma\sigma_{\tau,b'})$ for each b. For a certain q value (x-axis), the term returns a larger value for a larger $\mu_{s',b}$ as seen in the black circles. This result has

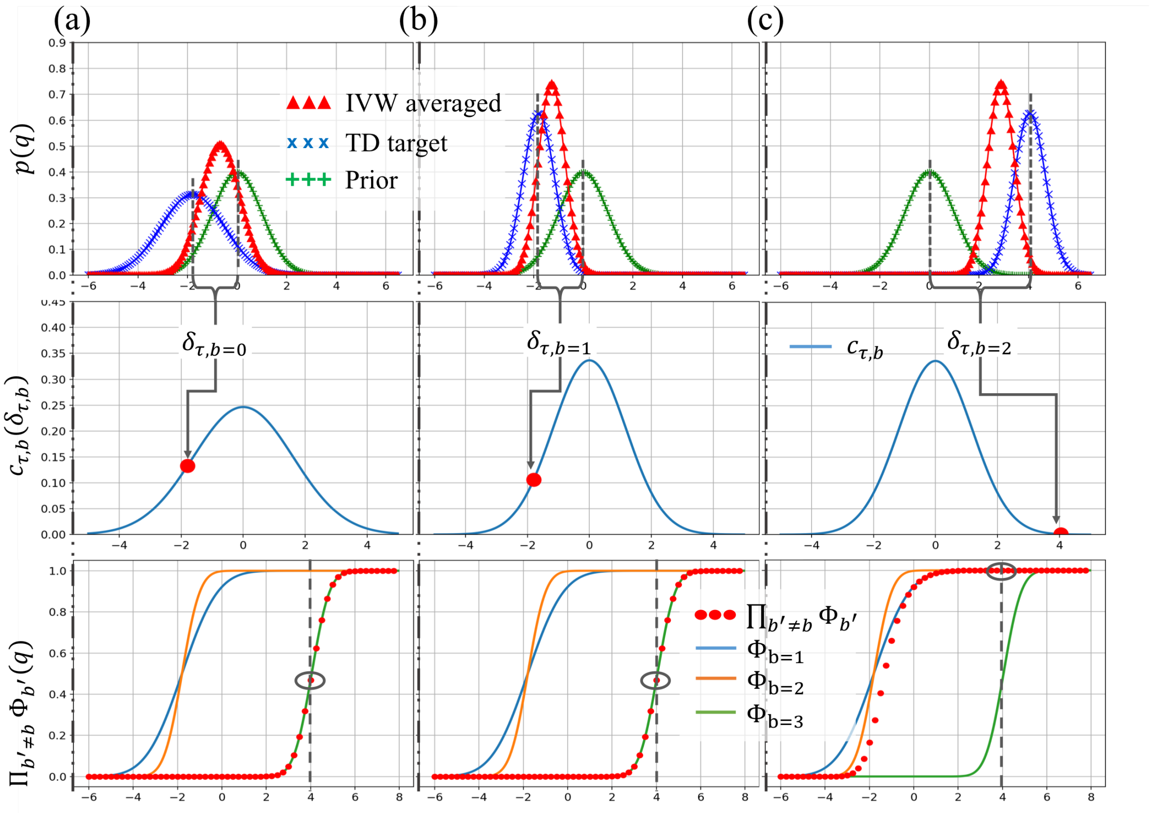

Figure 4.1: An example of the belief update in Eq.4.2 when $|\mathcal{A}| = 3, r = 0.0, \gamma = 0.9$ and prior (+ green) has $\mu_{s,a} = 0.0, \sigma^2_{s,a} = 1.0$. Each column corresponds to a subsequent state and action pair, **(a)** $b = 1$: $\mu_{s',b} = -2.0, \sigma^2_b = 2.0$, **(b)** $b = 2$: $\mu_{s',b} = -2.0, \sigma^2_b = 0.5$, **(c)** $b = 3$: $\mu_{s',b} = 4.5, \sigma^2_b = 0.5$. The first row presents how $\bar{\mu}_{\tau,b}$ and $\bar{\sigma}_{\tau,b}$ are determined from prior and a part of the likelihood. The second row shows how $c_{\tau,b}$ (y-axis of the dot) is assigned by TD error ($\delta_{\tau,b}$, x-axis of the dot) and a joint uncertainty measure $\sigma^2_{s,a} + \gamma^2\sigma^2_{s',b}$. The third row shows a softmax-like behavior of the product of CDFs.

a similarity with the soft Bellman equation [139], but the degree of softness in this case is determined by the uncertainty measures rather than a hyperparameter.

When $\sigma_w > 0$ the likelihood distribution is obtained by solving the following integral:

$$p(r + \gamma V_{s'}|q, \theta) = \int_{-\infty}^{\infty} p(r + \gamma V_{s'} + w) p_W(w) dw \tag{4.5}$$

$$= \int_{-\infty}^{\infty} \sum_{b\in\mathcal{A}} \frac{l_{s',b}}{\bar{v}_{s',b}} \phi\left(\frac{w - \bar{w}_{s',b}}{\bar{v}_{s',b}}\right) \prod_{b'\neq b} \left(1 - \Phi\left(\frac{w - (q - r - \gamma\mu_{s',b'})}{\gamma\sigma_{s',b'}}\right)\right) dw \tag{4.6}$$

where

$$l_{s',b} = \frac{1}{\sqrt{\sigma_w^2 + \gamma^2 \sigma_{s',b}^2}} \phi \left(\frac{q - (r + \gamma \mu_{s',b})}{\sqrt{\sigma_w^2 + \gamma^2 \sigma_{s',b}^2}} \right) \tag{4.7}$$

$$\bar{w}_{s',b} = \bar{v}_{s',b}^2 \left(\frac{q - (r + \gamma \mu_{s',b})}{\gamma^2 \sigma_{s',b}^2} \right) \qquad \frac{1}{\bar{v}_b^2} = \frac{1}{\gamma^2 \sigma_{s',b}^2} + \frac{1}{\sigma_w^2} \tag{4.8}$$

Then, the posterior distribution of $Q_{s,a}$ is:

$$\begin{aligned} \hat{p}_{Q_{s,a}}(q|\theta, r, s') = & \frac{1}{Z} \sum_{b \in \mathcal{A}} \frac{c_{\tau,b}}{\bar{\sigma}_{\tau,b}} \phi \left(\frac{q - \bar{\mu}_{\tau,b}}{\bar{\sigma}_{\tau,b}} \right) \\ & \times \int_{-\infty}^{\infty} \frac{1}{\bar{v}_{s',b}} \phi \left(\frac{w - \bar{w}_{s',b}}{\bar{v}_{s',b}} \right) \prod_{b' \neq b} \Phi \left(- \frac{w - (q - (r + \gamma \mu_{s',b'}))}{\gamma \sigma_{s',b'}} \right) dw \end{aligned} \tag{4.9}$$

where

$$c_{\tau,b} = \frac{1}{\sqrt{\sigma_{s,a}^2 + \gamma^2 \sigma_{s',b}^2 + \sigma_w^2}} \phi \left(\frac{(r + \gamma \mu_{s',b}) - \mu_{s,a}}{\sqrt{\sigma_{s,a}^2 + \gamma^2 \sigma_{s',b}^2 + \sigma_w^2}} \right) \tag{4.10}$$

$$\bar{\mu}_{\tau,b} = \bar{\sigma}_{\tau,b}^2 \left(\frac{\mu_{s,a}}{\sigma_{s,a}^2} + \frac{r + \gamma \mu_{s',b}}{\gamma^2 \sigma_{s',b}^2 + \sigma_w^2} \right) \qquad \frac{1}{\bar{\sigma}_{\tau,b}^2} = \frac{1}{\sigma_{s,a}^2} + \frac{1}{\gamma^2 \sigma_{s',b}^2 + \sigma_w^2} \tag{4.11}$$

Note that $c_{\tau,b}$, $\bar{\mu}_{\tau,b}$, and $\bar{\sigma}_{\tau,b}$ in Eq.4.10-4.11 become identical with the ones in Eq.4.3-4.4 when $\sigma_w = 0$.

4.4 Assumed Density Filtering on Q-Belief Updates

The posterior distribution in Eq.4.2, however, is no longer Gaussian. In order to continue the Bayesian update, we approximate the posterior with a Gaussian distribution using Assumed Density Filtering.

4.4.1 Assumed Density Filtering

Assumed Density Filtering (ADF) is a general technique for approximating the true posterior with a tractable parametric distribution in Bayesian networks. ADF has been independently rediscovered for a number of applications and is also known as *moment matching*, *online Bayesian learning*, and *weak marginalization* [13, 89, 101]. Suppose that a hidden variable $\mathbf{x}$ follows a tractable parametric distribution $p(\mathbf{x}|\theta_t)$ where θ_t is a set of parameters at time

t. In the Bayesian framework, the distribution can be updated after observing some new data (D_t) using Bayes' rule, $\hat{p}(\mathbf{x}|\theta_t, D_t) \propto p(D_t|\mathbf{x}, \theta_t)p(\mathbf{x}|\theta_t)$. In online settings, a Bayesian update is typically performed after a new data point is observed, and the updated posterior is then used as a prior for the following iteration.

When the posterior computed by Bayes' rule does not belong to the original parametric family, it can be approximated by a distribution belonging to the parametric family. In ADF, the posterior is projected onto the closest distribution in the family chosen by minimizing the reverse *Kullback-Leibler* divergence denoted as $KL(\hat{p}||p)$ where $\hat{p}$ is the original posterior distribution and p is a distribution in a parametric family of interest. Thus, for online Bayesian filtering, the parameters for the ADF estimate is given by

$$\theta_{t+1} = \operatorname*{argmin}_{\theta} KL(\hat{p}(\cdot|\theta_t, D_t)||p(\cdot|\theta)) \tag{4.12}$$

4.4.2 Online Belief Update

When the parametric family of interest is spherical Gaussian, it is shown that the ADF parameters are obtained by matching moments. Thus, the mean and variance of the approximate posterior are given by those of the true posterior, $\mathbb{E}_{\hat{p}_{Q_{s,a}}}[q]$ and $\mathrm{Var}_{\hat{p}_{Q_{s,a}}}[q]$, respectively. It is fairly easy to derive the mean and variance when $|\mathcal{A}| = 2$. The derivation is presented in Appendix A.2. However, to our knowledge, there is no analytically tractable solution for $|\mathcal{A}| > 2$.

In the next sections, we prove the convergence of the means to the optimal Q-values for the case $|\mathcal{A}| = 2$ with the exact solutions for the ADF parameters. Then, we show how to derive an analytic approximation for the ADF parameters which becomes exact in the small variance limit.

4.4.3 Convergence to Optimal Q-values

The convergence theorem of the Q-learning algorithm has previously been proven [131]. We, therefore, show that the online Bayesian update using ADF with the posterior in Eq.4.2 converges to Q-learning when $|\mathcal{A}| = 2$. We apply an approximation from Lemma 1 in order

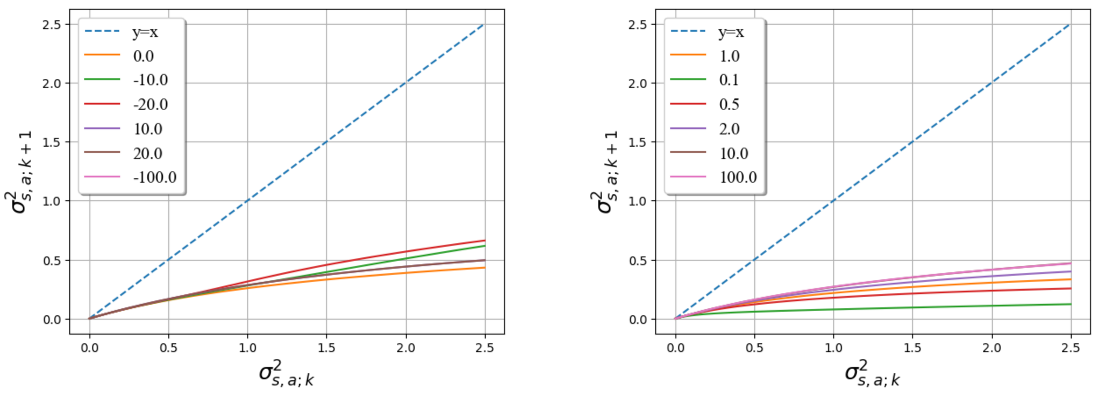

Figure 4.2: Relationship between $\sigma^2_{s,a;k}$ and its updated value, $\sigma^2_{s,a;k+1}$ for $|\mathcal{A}| = 2$. Each solid curve represents a different set of parameters. Left: Differing values of $\mu_{s',2} - \mu_{s',1}$. Right: Differing values of $\sigma^2_{s',1}/\sigma^2_{s',2}$

to prove Theorem 2. Proofs for Lemma 1 and Theorem 2 are presented in Appendix C.

Lemma 1. *Let X be a random variable following a normal distribution, $\mathcal{N}(\mu, \sigma^2)$. Then we have:*

$$\lim_{\sigma \to 0} \left[\Phi\left(\frac{x-\mu}{\sigma}\right) - \exp\left\{ -\frac{1}{2} \left[-\frac{x-\mu}{\sigma} \right]_+^2 \right\} \right] = 0 \tag{4.13}$$

where $[x]_+ = \max(0, x)$ is the ReLU *nonlinearity.*

Theorem 2. *Suppose that the mean and variance of $Q_{s,a}$ $\forall s \in \mathcal{S}, \forall a \in \mathcal{A}$ are iteratively updated by the mean and variance of $\hat{p}_{Q_{s,a}}$ after observing r and s' at every step. When $|\mathcal{A}| = 2$, the update rule of the means is equivalent to the Q-learning update if all state-action pairs are visited infinitely often and the variances approach 0. In other words, at the kth update on $\mu_{s,a}$:*

$$\lim_{k\to\infty, \{\sigma\}\to 0} \mu_{s,a;k+1} = (1 - \alpha_{\tau;k})\, \mu_{s,a;k} + \alpha_{\tau;k}\big(r + \gamma \max_{b\in\mathcal{A}} \mu_{s',b;k}\big)$$

where $\alpha_{\tau;k} = \sigma^2_{s,a;k} / \left(\sigma^2_{s,a;k} + \gamma^2 \sigma^2_{s',b^+;k} + \sigma^2_w\right)$ and $b^+ = \mathrm{argmax}_{b\in\mathcal{A}}\, \mu_{s',b}$.

Interestingly, α_τ approaches 1 when $\sigma_{s,a}/\sigma_{s',b^+} \to \infty$ and 0 when $\sigma_{s,a}/\sigma_{s',b^+} \to 0$ for $\sigma_w = 0$. Such behavior remains when $\sigma_w > 0$ but α_τ eventually approaches 0 as the number of visits to (s, a) goes to infinity. This not only satisfies the convergence condition of Q-learning but also provides a natural learning rate - the smaller the variance of the next state (the higher the confidence), the more $Q_{s,a}$ is updated from the target information rather than the current belief.

Fig. 4.2 shows empirical evidence that the contraction condition on variance for Theorem 2 holds. The updated variance is less than the current variance for a large range of different values for the related parameters. In addition, it is easily shown that 0 is the fixed point of the variance from Eq.A.5-A.6 in Appendix A.2.

4.5 Analytic ADF Parameter Estimates

When $|\mathcal{A}| > 2$, the update can be solved by numerical approximation of the true posterior mean and variance using a number of samples. However, its computation becomes unwieldy due to the large number of samples needed for accurate estimates. This becomes especially problematic with small variances as the number of visits to corresponding state-action pairs grows. Therefore, in this section, we show how to accurately estimate the ADF parameters using an analytic approximation. This estimate becomes exact in the small variance limit.

4.5.1 Analytic Approximation of Posterior

Using Lemma 1, the true posterior in Eq.4.2 is approximated as the following distribution:

$$\tilde{p}_{Q_{s,a}}(q) = \frac{1}{Z}\sum_{b\in\mathcal{A}} \frac{c_{\tau,b}}{\sqrt{2\pi}\bar{\sigma}_{\tau,b}} \exp\left\{-\frac{(q-\bar{\mu}_{\tau,b})^2}{2\bar{\sigma}^2_{\tau,b}} - \sum_{b'\neq b}\frac{\left[r+\gamma\mu_{s',b'}-q\right]_+^2}{2\gamma^2\sigma^2_{s',b'}}\right\} \tag{4.14}$$

Each term for $b \in \mathcal{A}$ inside the summation can then be approximated by a Gaussian PDF. Similar to Laplace's method, we approximate each term as a Gaussian distribution by matching the maximum values as well as the curvature at the peak of the distribution. In other words, the maximum of the distribution is modeled locally near its peak by the quadratic concave function:

$$-\frac{(q-\bar{\mu}_{\tau,b})^2}{2\bar{\sigma}^2_{\tau,b}} - \sum_{b'\neq b}\frac{\left[r+\gamma\mu_{s',b}-q\right]_+^2}{2\gamma^2\sigma^2_{s',b}} \approx -\frac{(q-\mu_b^+)^2}{2{\sigma^*_{\tau,b}}^2} \tag{4.15}$$

We find $\mu^*_{\tau,b}$ and $\sigma^*_{\tau,b}$ by matching the first and the second derivatives, respectively (the coefficient of the quadratic term gives the local curvature):

$$\frac{\mu^*_{\tau,b} - \bar{\mu}_{\tau,b}}{\bar{\sigma}^2_{\tau,b}} = \sum_{b' \neq b} \frac{\left[r + \gamma\mu_{s',b'} - \mu^*_b\right]_+}{\gamma^2\sigma^2_{s',b'}} \tag{4.16}$$

$$\frac{1}{{\sigma^*_b}^2} = \frac{1}{\bar{\sigma}^2_{\tau,b}} + \sum_{b' \neq b} \frac{H\left(r + \gamma\mu_{s',b'} - \mu^*_{\tau,b}\right)}{\gamma^2\sigma^2_{s',b'}} \tag{4.17}$$

where $H(\cdot)$ is a Heaviside step function. The self-consistent piece-wise linear equation for $\mu^*_{\tau,b}$ can be rewritten as follows:

$$\mu^*_b = \left(\frac{1}{\bar{\sigma}^2_{\tau,b}} + \sum_{b' \neq b} \frac{H(r + \gamma\mu_{s',b'} - \mu^*_{\tau,b})}{\gamma^2\sigma^2_{s',b'}}\right)^{-1} \left(\frac{\bar{\mu}_{\tau,b}}{\bar{\sigma}^2_{\tau,b}} + \sum_{b' \neq b} \frac{(r + \gamma\mu_{s',b'})}{\gamma^2\sigma^2_{s',b'}} H(r + \gamma\mu_{s',b'} - \mu^*_{\tau,b})\right) \tag{4.18}$$

This is an IVW average mean of the prior, the TD target distribution of b, and other TD target distributions whose means are larger than $\mu^*_{\tau,b}$. The height of the peak is computed for $q = \mu^*_b$,

$$k^*_{\tau,b} = \frac{c_{\tau,b}\sigma^*_{\tau,b}}{\bar{\sigma}_{\tau,b}} \exp\left\{-\frac{(\mu^*_b - \bar{\mu}_{\tau,b})^2}{2\bar{\sigma}^2_{\tau,b}} - \sum_{b' \neq b} \frac{\left[r + \gamma\mu_{s',b'} - \mu^*_b\right]^2_+}{2\gamma^2\sigma^2_{s',b'}}\right\} \tag{4.19}$$

The final approximated distribution is a Gaussian mixture model with $\mu^*_{\tau,b}, \sigma^*_{\tau,b}, w^*_{\tau,b}$ for all $b \in \mathcal{A}$ where $w^*_{\tau,b} = k^*_{\tau,b} / \sum_{b'} k^*_{\tau,b'}$:

$$\tilde{p}_{Q_{s,a}} = \sum_{b \in \mathcal{A}} \frac{w^*_{\tau,b}}{\sigma^*_{\tau,b}} \phi\left(\frac{q - \mu^*_{\tau,b}}{\sigma^*_{\tau,b}}\right) \tag{4.20}$$

Finally, we can update the belief distribution over $Q_{s,a}$ with the mean and variance of Eq.4.20:

$$\mathbb{E}_{\tilde{p}}[q] = \sum_{b \in \mathcal{A}} w^*_{\tau,b}\mu^*_{\tau,b} \tag{4.21}$$

$$\mathrm{Var}_{\tilde{p}}[q] = \sum_{b \in \mathcal{A}} w^*_{\tau,b}{\sigma^*_{\tau,b}}^2 + \sum_{b \in \mathcal{A}} w^*_{\tau,b}{\mu^*_{\tau,b}}^2 - \left(\mathbb{E}_{\tilde{p}}[q]\right)^2 \tag{4.22}$$

Algorithm 7 ADFQ algorithm

```
1: Initialize randomly $\mu_{s,a}$, $\sigma_{s,a}$ $\forall s \in \mathcal{S}$ and $\forall a \in \mathcal{A}$
2: for each episode do
3:     Initialize $s_0$
4:     for each time step $t$ do
5:         Choose an action, $a_t \sim \pi^{action}(s_t; \theta_t)$
6:         Perform the action and observe $r_t$ and $s_{t+1}$
7:         for each $b \in \mathcal{A}$ do
8:             Compute $\mu^*_{\tau,b}$, $\sigma^*_{\tau,b}$, $k^*_{\tau,b}$ using Eq.4.16-4.19
9:         end for
10:        Update $\mu_{s_t,a_t}$ and $\sigma_{s_t,a_t}$ using Eq.4.21 and Eq.4.22
11:    end for
12: end for
```

The final mean is the weighted sum of each individual mean with a weight from $k^*_{\tau,b}$ and the final variance is the weighted sum of each individual variance added to a non-negative term accounting for the dispersion of the means. The first term of Eq.4.22 is always smaller than $\sigma_{s,a}$ since $\sigma_{\tau,b}* < \sigma_{s,a}$. The difference amount depends on the following mean dispersion term and is determined by various factor such as TD errors, relative distances between means, variances. However, since $\mu^*_{\tau,b}$ is an IVW average mean and $k^*_{\tau,b}$ is small for a large TD error and large distance to other TD targets, the mean dispersion part is relatively small. As shown in Eq.4.19, the weights are determined by TD errors, variances, relative distances to larger TD targets. It has the TD error penalizing term, $c_{\tau,b}$, and also decreases as the number of TD targets larger than $\mu^*_{\tau,b}$ increases. Therefore, the weight provides a softened maximum property over b.

The final algorithm is summarized in Table.7. Its space complexity is $O(|\mathcal{S}||\mathcal{A}|)$. The computational complexity of each update is $O(|\mathcal{A}|^2)$ which is higher than Q-learning but only by a factor of $|\mathcal{A}|$ and constant in the number of states.

4.5.2 Approximate Likelihood

Similar to the posterior in Eq.4.2, a closed form solution of the expected likelihood in Eq.4.6 is not available when $|\mathcal{A}| > 2$. Thus, we will derive an analytic solution for $|\mathcal{A}| = 2$, and then generalize it to an arbitrary number of actions by applying the small variance approximation in Lemma 1. Derivation details are presented in Appendix B.

The expected likelihood for $|\mathcal{A}| = 2$ is derived as :

$$p(r + \gamma V_{s'}|q, \theta) = l_1 \Phi\left(\frac{q - \mu_2^w}{\sigma_2^w}\right) + l_2 \Phi\left(\frac{q - \mu_1^w}{\sigma_1^w}\right) \tag{4.23}$$

where

$$\mu_2^w \equiv \left(1 - \frac{\bar{v}_1^2}{\gamma^2 \sigma_{s',1}^2}\right)^{-1} \left(r + \gamma\mu_{s',2} - \frac{\bar{v}_1^2}{\gamma^2 \sigma_{s',1}^2}(r + \gamma\mu_{s',1})\right)$$

$$\sigma_2^w \equiv \left(1 - \frac{\bar{v}_1^2}{\gamma^2 \sigma_{s',1}^2}\right)^{-1} \sqrt{\bar{v}_1^2 + \gamma^2 \sigma_{s',2}^2}$$

and $l_{s',b}$, $\bar{w}_{s',b}$, and $\bar{v}_{s',b}$ are defined in Eq.4.7-4.8.

In an asymptotic limit of $\sigma_w/\sigma_{s',b} \to 0$, $\forall b \in \mathcal{A}$ and $|\mathcal{A}| = 2$, the expected likelihood distribution for $\sigma_w > 0$ (Eq.4.23) is similar to $p(r + \gamma V_{s'}|q, \theta)$ with $\sigma_w = 0$ but the variance of its Gaussian PDF term is $\gamma^2\sigma_{s',b}^2 + \sigma_w^2$ instead of $\gamma^2\sigma_{s',b}^2$ (see Appendix B for details). In other words, $\lim_{\sigma_w/\sigma_{s',b} \to 0} p(r + \gamma V_{s'}|q, \theta)$ is :

$$\sum_{b \in \mathcal{A}} \frac{\gamma}{\sqrt{\gamma^2\sigma_{s',b}^2 + \sigma_w^2}} \phi\left(\frac{q - (r + \gamma\mu_{s',b})}{\sqrt{\gamma^2\sigma_{s',b}^2 + \sigma_w^2}}\right) \prod_{b' \neq b, b' \in \mathcal{A}} \Phi\left(\frac{q - (r + \gamma\mu_{s',b'})}{\gamma\sigma_{s',b'}}\right) \tag{4.24}$$

The posterior distribution, $\hat{p}_{Q_{s,a}}(q)$, derived from this likelihood has the same form with Eq.4.2 but it uses $\gamma^2\sigma_{s',b}^2 + \sigma_w^2$ instead of $\gamma^2\sigma_{s',b}^2$ in $c_{\tau,b}$, $\bar{\mu}_{\tau,b}$, and $\bar{\sigma}_{\tau,b}$:

$$c_{\tau,b} = \frac{1}{\sqrt{\sigma_{s,a}^2 + \gamma^2\sigma_{s',b}^2 + \sigma_w^2}} \phi\left(\frac{(r + \gamma\mu_{s',b}) - \mu_{s,a}}{\sqrt{\sigma_{s,a}^2 + \gamma^2\sigma_{s',b}^2 + \sigma_w^2}}\right) \tag{4.25}$$

$$\bar{\mu}_{\tau,b} = \bar{\sigma}_{\tau,b}^2 \Big(\frac{\mu_{s,a}}{\sigma_{s,a}^2} + \frac{r + \gamma\mu_{s',b}}{\gamma^2\sigma_{s',b}^2 + \sigma_w^2}\Big) \qquad \bar{\sigma}_{\tau,b}^2 = \Big(\frac{1}{\sigma_{s,a}^2} + \frac{1}{\gamma^2\sigma_{s',b}^2 + \sigma_w^2}\Big)^{-1} \tag{4.26}$$

Again, it becomes identical to the posterior in Eq.4.2 when $\sigma_w = 0$.

Extending this result to the general case ($|\mathcal{A}| = n$ for $n \in \mathbb{N}$), the posterior distribution for $\sigma_w > 0$ is same with Eq.4.2 but $\gamma^2\sigma_{s',b}^2$ is replaced by $\gamma^2\sigma_{s',b}^2 + \sigma_w^2$ in $c_{\tau,b}$, $\bar{\mu}_{\tau,b}$, and $\bar{\sigma}_{\tau,b}$ (Eq.4.3-4.4). Therefore, $\mu_{\tau,b}^*$, $\sigma_{\tau,b}^*$, and $k_{\tau,b}^*$ in the ADFQ algorithm (Table.7) are also changed accordingly.

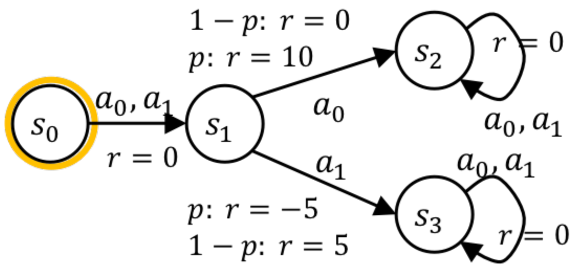

Figure 4.3: A simple MDP with stochastic rewards

4.5.3 Convergence of ADFQ

Theorem 2 extends to the ADFQ algorithm. The contraction behavior of the variances in the case of Theorem 2 is also empirically observed in ADFQ (Proof in Appendix C).

Theorem 3. *The ADFQ update on the mean $\mu_{s,a}$ $\forall s \in \mathcal{S}$, $\forall a \in \mathcal{A}$ for $|\mathcal{A}| = 2$ is equivalent to the Q-learning update if the variances approach 0 and if all state-action pairs are visited infinitely often. In other words, we have :*

$$\lim_{k\to\infty, \{\sigma\}\to 0} \mu_{s,a;k+1} = (1-\alpha_{\tau;k})\,\mu_{s,a;k} + \alpha_{\tau;k}\left(r + \gamma \max_{b\in\mathcal{A}} \mu_{s',b;k}\right)$$

where $\alpha_{\tau;k} = \sigma^2_{s,a;k} / \left(\sigma^2_{s,a;k} + \gamma^2 \sigma^2_{s',b^+;k} + \sigma^2_w\right)$ *and* $b^+ = \text{argmax}_{b\in\mathcal{A}}\, \mu_{s',b}$.

As we have observed the behavior of α_τ in Theorem 2, the learning rate α_τ again provides a natural learning rate with the ADFQ update. We can therefore think of Q-learning as a special case of ADFQ.

4.6 A Concrete Demonstration in a Discrete MDP

To demonstrate the behavior of the ADFQ update, we provides a simple MDP ($\gamma = 0.9$) with stochastic rewards in Fig.4.3. An episode starts at s_0 and terminates at either s_2 or s_3. At s_1, each action returns a stochastic reward with $p = 0.2$. The optimal deterministic policy at s_1 is a_1 and $Q^*(s_0,\cdot) = 2.7, Q^*(s_1,a_0) = 2.0, Q^*(s_1,a_1) = 3.0, Q^*(s_2,\cdot) = Q^*(s_3,\cdot) = 0.0$. Suppose that a reinforcement learning learner is at $(s_t,a_t) = (s_1,a_1)$ and has visited (s_1,a_1) three times with all $r = 5$ until $t-1$ updating its Q-beliefs. The plots in Fig.4.4 show the

ADFQ update for Q_{s_0,a_0} at $t+1$ when $r_t = +5$ (left) and $r_t = -5$ (right). When it receives a less expected reward, -5, $\sigma_{s_1,a_1;t}$ is updated to a larger value than the one in the $r_t = +5$ case. Then, at $t+1$, ADFQ considers both Q_{s_1,a_0} and Q_{s_1,a_1} for updating Q_{s_0,a_0}. Due to the relatively large TD error and variance of Q_{s_1,a_1}, a lower value is assigned to $w^*_{\tau,b=1}$. In this same scenario, Q-learning would update $Q(s_0, a_0)$ only from $Q(s_1, a_0)$ and regulate the update amount with the learning rate which is usually fixed or determined on the number of visits.

In order to show the benefits of the update rule, we examined ADFQ, Q-learning, Double Q-learning and KTD-Q for the convergence to the optimal Q-values in the presented MDP and a similar MDP but with 10 terminating states and 10 actions. Random exploration is used in order to evaluate only the update part of each algorithm. The averaged results over 10 trials are plotted in Fig.4.5. In both cases, Q-learning and ADFQ converge near the optimal Q-values. At the early stage of the learning, ADFQ quickly reduces the overestimated amount compared to Q-learning and converges to the optimal values slightly faster than Q-learning. Double Q-learning approaches the optimal values as it learns more, but the plots clearly show its underestimation behavior which causes significantly slower learning than other algorithms. As the convergence of KTD-Q in a stochastic MDP is not guaranteed, it failed to converge in both domains.

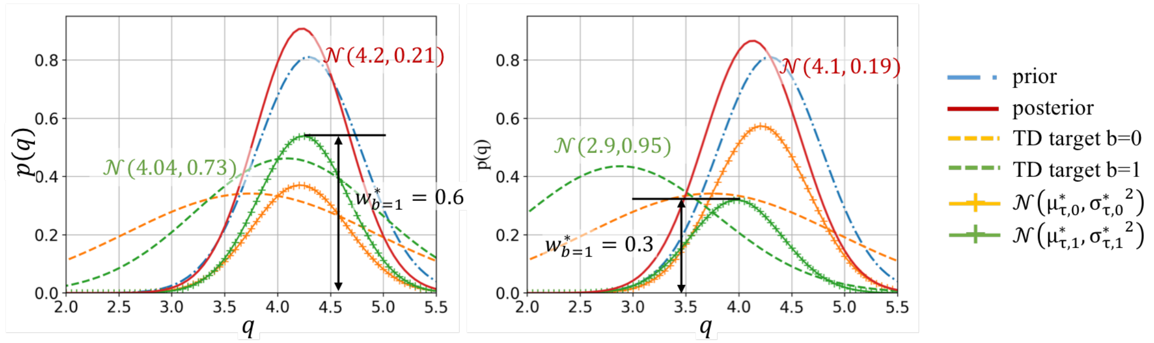

Figure 4.4: ADFQ update example for $s_{t+1} = s_0, a_{t+1} = a_0$, (left) $r_t = 5$, (right) $r_t = -5$ in the simple MDP

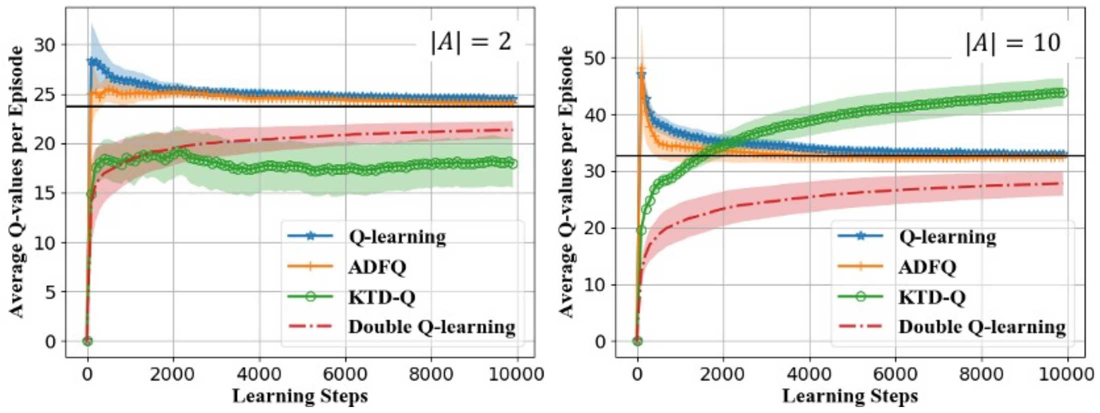

Figure 4.5: $\sum_{s\in\mathcal{S},a\in\mathcal{A}} Q_t(s,a)$ during learning in the simple MDP of $|\mathcal{A}| = 2$ (left) and an MDP $|\mathcal{A}| = 10$ (right), averaged over 10 trials. The solid lines and markers represent mean performance and the shaded areas represent the standard deviation across trials. The black horizontal line is the average of the optimal Q-values in each case, $\sum_{s\in\mathcal{S},a\in\mathcal{A}} Q^*(s,a)$.

4.7 Experiments in Discrete MDPs

4.7.1 Algorithms

ADFQ is evaluated with two action policies: *Thompson Sampling (TS)* [124] selects $a_t = \text{argmax}_a\, q_{s_t,a}$ where $q_{s_t,a} \sim p_{Q_{s_t,a}}(\cdot|\theta_t)$, and ϵ-*greedy* selects a random action with ϵ probability and selects the action with the highest mean otherwise. In implementation, we fixed the initial variance to 100.0 and the variances are bounded by 10^{-10} since their values dramatically drop and eventually exceed the precision range of computers.

For comparison, we test Q-learning with ϵ-greedy and Boltzmann action policies. The learning rate decreases as the number of visits to a state-action pair increases $\alpha_t = \alpha_0(n_0 + 1)/(n_0 + t)$, $\alpha_0 = 0.5$ [76]. Additionally, KTD-Q with ϵ-greedy and its active learning scheme are also examined. KTD-Q is an extension of Kalman Temporal Difference (KTD) [36] and one of the recent influential algorithms for Bayesian off-policy TD learning. KTD approximates the value function using the Kalman filtering scheme, and handles the non-linearity in the Bellman optimality equation by applying the Unscented Transform. The same hyperparameter values as the ones in the original paper are used if presented. All other hyperparameters are selected through cross-validation (presented in Appendix E).

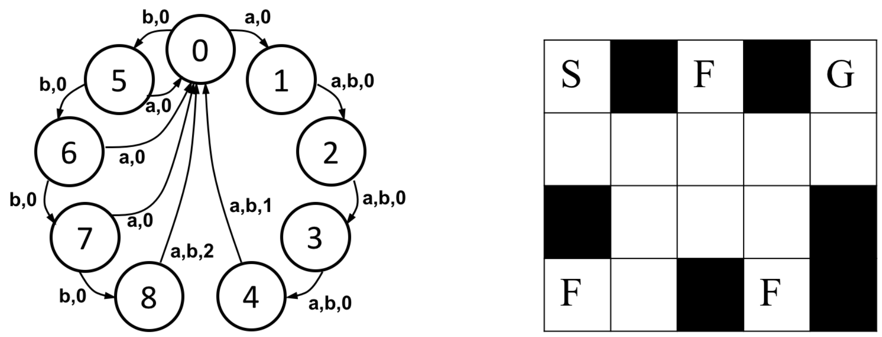

Figure 4.6: Loop and Maze domain diagrams

Further details of KTD are explained in Sec.4.2.

4.7.2 Domains

We test the algorithms in Loop and Maze ($\gamma = 0.95$, Fig.4.6) from [26] with and without stochasticity in the domains for finite learning steps ($T_{H,loop} = 10000$, $T_{H,maze} = 30000$). The Loop domain consists of 9 states and 2 actions (a,b). There are +1 reward at state 4 and +2 reward at state 8. For a stochastic case, a learning agent performs the other action with a probability 0.1. In Maze, the agent's goal is to collect the flags "F" and escape the maze through the goal position "G" starting from "S". It receives a reward equivalent to the number of flags it has collected at "G". The agent remains at the current state if it performs an action toward a wall (black block). For a stochastic case, the agent slips with a probability 0.1 and moves to the right perpendicular direction.

4.7.3 Results

We first examined the convergence to the optimal Q-values using randomly generated fixed trajectories $< s_0, a_0, r_0, s_1, \cdots >$ for all algorithms in order to evaluate only the update part of each algorithm. During learning, we computed the root mean square error (RMSE) between the estimated Q-values (or means) and the true optimal Q-values, and the results were averaged over 10 trials. The true optimal Q-values were obtained using the policy iteration method described in Sec.2.2.1. In addition, we evaluated the performance of each algorithm with different action policies during learning. At every $T_H/100$ steps, the current policy was greedily evaluated where the maximum number of steps was bounded by 1.5

times of the optimal path length or it was terminated when the goal was reached. The entire experiment was repeated 10 times for each domain and the averaged results were plotted in Fig.4.8.

As shown in Fig.4.7, ADFQ converged to the optimal Q-values quicker than all other algorithms including Q-learning. Moreover, ADFQ with ϵ-greedy and ADFQ with TS showed similar results and converged to the optimal performance faster than the comparing algorithms in all cases. Q-learning with ϵ-greedy learned an optimal policy almost as fast as ADFQ in the deterministic cases, but the performance of ADFQ was improved dramatically in the stochastic cases. KTD-Q approached to the optimal values in the deterministic Loop domain, but diverged in others since its derivative-free approximation nature does not scale well with the number of parameters. It is also proposed under a deterministic environment assumption. The author proposed XKTD-V and XKTD-SARSA which are extended versions of KTD-V and KTD-SARSA, respectively, for a stochastic environment. Yet, KTD-Q was not able to be extended to XKTD-Q (see the section 4.3.2 in [36] for details). Despite the convergence issue, the KTD-Q with active learning scheme worked better than Q-learning and converged to an optimal policy in Loop. These results imply that KTD-Q does not scale with the number of parameters even though it works well in smaller domains, and its convergence to the optimal Q-values is not guaranteed in stochastic domains.

4.8 Fast ADFQ

In this section, we introduced another approximation approach to the true posterior, Eq.4.2 using a stronger approximation. The major advantage of this method is that it is computationally as efficient as Q-learning. ADFQ becomes identical with the following algorithm as variance decreases.

When the variance of a Gaussian random variable, $X \sim \mathcal{N}(\mu, \sigma^2)$, approaches 0, we can approximate its CDF and PDF to a Heaviside step function, $H(\cdot)$ and a dirac delta function, $\delta(\cdot)$, respectively. Suppose that $\sigma_{s,a} \ll 1$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$. The product of the Gaussian CDFs in the Eq.4.2 is approximated to 1 if $q \geq r + \gamma\mu_{s',b'}$ for all $b' \in \mathcal{A}$, $b' \neq b$,

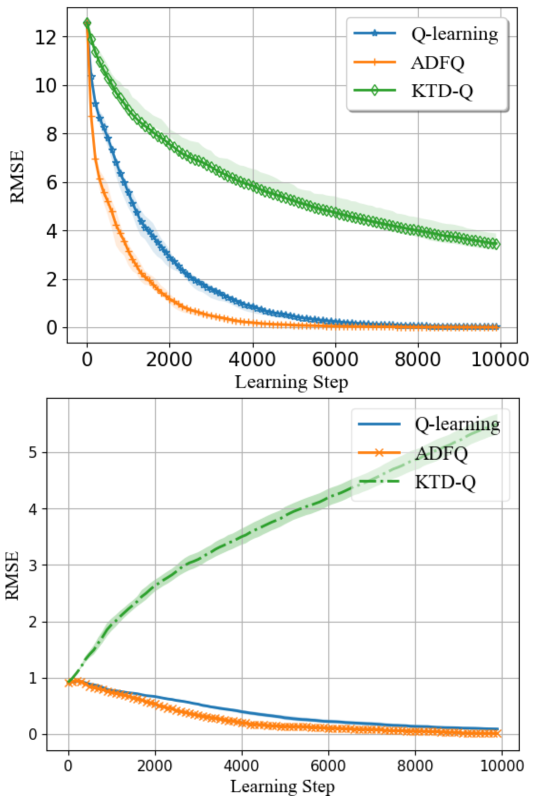

Figure 4.7: Root Mean Square Error (RMSE), $||Q - Q*||_2$ or $||\mu - Q^*||_2$, of ADFQ, Q-learning, and KTD-Q. Left: deterministic, Right: stochastic, Top: Loop, Bottom: Maze.

and 0 otherwise. However, when $q = \bar{\mu}_{\tau,b}$, we cannot simply apply the approximation since the PDF approaches infinity:

$$\lim_{\substack{\bar{\sigma}_{\tau,b},\sigma_{s',b'}\\ \to 0}} \frac{1}{\bar{\sigma}_{\tau,b}}\phi\left(\frac{q-\bar{\mu}_{\tau,b}}{\bar{\sigma}_{\tau,b}}\right)\cdot\prod_{b'\neq b}\Phi\left(\frac{q-(r+\gamma\mu_{s',b'})}{\gamma\sigma_{s',b'}}\right) = \infty\cdot 0 \neq 0$$

We define a function $f(\cdot)$ which is the approximation of the above term inside of the limit when the term of the product of the Gaussian CDFs approaches to 0 (e.g. $q < r + \gamma\mu_{s',b'}$ for all $b' \neq b$).

$$f(q;\mu,\sigma) = \begin{cases} \frac{\epsilon}{\sigma}\phi\left(\frac{q-\mu}{\sigma}\right) & \text{for } q \in [\mu-\epsilon, \mu+\epsilon], \epsilon \ll 1 \\ 0 & \text{otherwise} \end{cases} \tag{4.27}$$

Let $b^+ \doteq \text{argmax}_{b\in\mathcal{A}}\,\mu_{s',b}$ and $b^{2+} \doteq \text{argmax}_{b\in\mathcal{A},b\neq b^+}\,\mu_{s',b}$. Then, the true posterior distri-

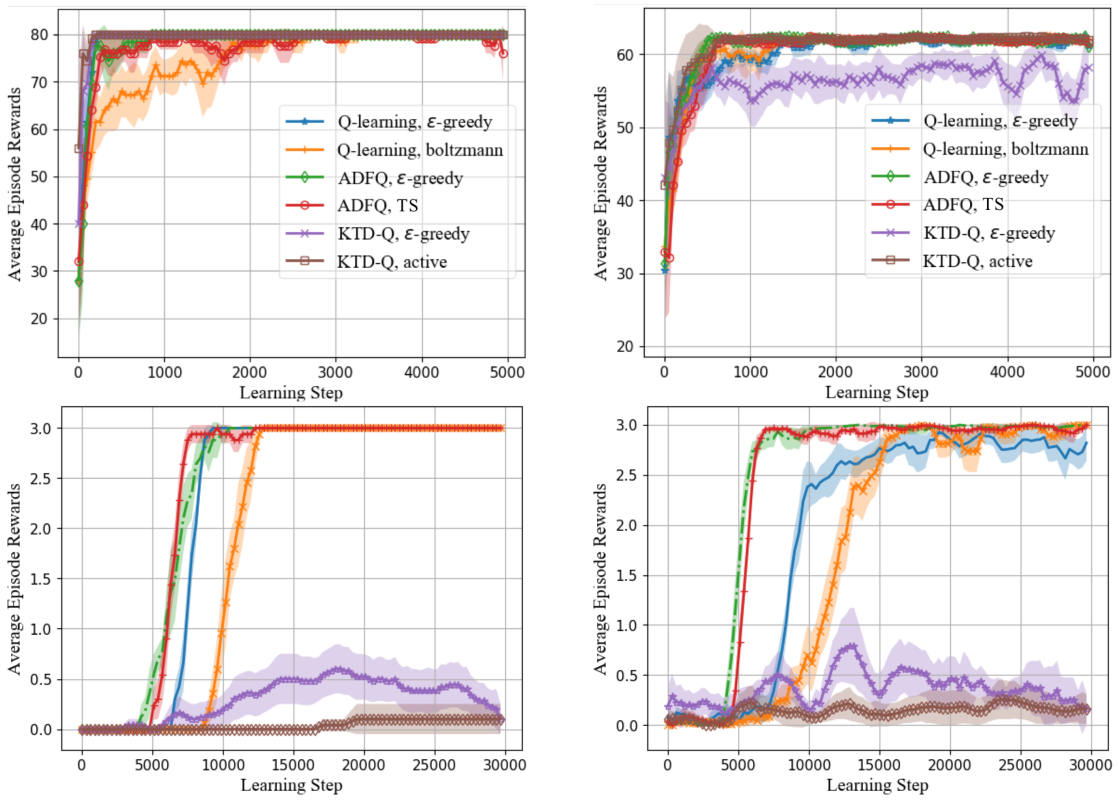

Figure 4.8: Semi-greedy evaluation of ADFQ, Q-learning, and KTD-Q during learning smoothed by a moving average with window 4. Left: deterministic, Right: stochastic, Top: Loop, Bottom: Maze.

bution in Eq.4.2 is approximated to $\tilde{p}_{Q_{s,a}}$ in three different ranges of q.

1. $q \in (-\infty, r + \gamma\mu_{s',b^{2+}})$

$$\tilde{p}_{Q_{s,a}}(q) = \frac{1}{Z}\sum_{b} c_{\tau,b} f(q; \bar{\mu}_{\tau,b}, \bar{\sigma}_{\tau,b}) \tag{4.28}$$

2. $q \in [r + \gamma\mu_{s',b^{2+}}, r + \gamma\mu_{s',b^+})$

$$\tilde{p}_{Q_{s,a}}(q) = \frac{1}{Z}\sum_{b \neq b^+} c_{\tau,b} f(q; \bar{\mu}_{\tau,b}, \bar{\sigma}_{\tau,b}) + \frac{1}{Z}\frac{c_{\tau,b^+}}{\bar{\sigma}_{\tau,b^+}}\phi\left(\frac{q - \bar{\mu}_{\tau,b^+}}{\bar{\sigma}_{\tau,b^+}}\right) \tag{4.29}$$

3. $q \in [r + \gamma\mu_{s',b^+}, +\infty)$

$$\tilde{p}_{Q_{s,a}} = \frac{1}{Z}\sum_{b} \frac{c_{\tau,b}}{\bar{\sigma}_{\tau,b}}\phi\left(\frac{q - \bar{\mu}_{\tau,b}}{\bar{\sigma}_{\tau,b}}\right) \tag{4.30}$$

Applying ADF to the approximated distribution, $\tilde{p}$, we need to find only mean and variance of the distribution. The first and the second moments of $\tilde{p}_{Q_{s,a}}$ are:

$$\mathbb{E}_{q\sim\tilde{p}_{Q_{s,a}}(\cdot)}[q] \approx \frac{\sum_b c_{\tau,b}\nu_{\tau,b}\bar{\mu}_b}{\sum_b c_{\tau,b}\nu_{\tau,b}} \qquad \mathbb{E}_{q\sim\tilde{p}_{Q_{s,a}}(\cdot)}[q^2] \approx \frac{\sum_b c_{\tau,b}\nu_{\tau,b}(\bar{\mu}^2_{\tau,b}+\bar{\sigma}^2_{\tau,b})}{\sum_b c_{\tau,b}\nu_{\tau,b}} \tag{4.31}$$

where

$$\nu_{\tau,b} = \begin{cases} 1-(1-\epsilon)H(r+\gamma\mu_{s',b^+}-\bar{\mu}_{\tau,b}) & \text{for } b \neq b^+ \\ \\ 1-(1-\epsilon)H(r+\gamma\mu_{s',b^{2+}}-\bar{\mu}_{\tau,b}) & \text{for } b = b^+ \end{cases}$$

The variance is computed by $\mathbb{E}_{q\sim\tilde{p}_{Q_{s,a}}(\cdot)}[q^2] - (\mathbb{E}_{q\sim\tilde{p}_{Q_{s,a}}(\cdot)}[q])^2$. Note that they are linear combinations of the IVW average values, $\bar{\mu}_{\tau,b}$ and $\bar{\sigma}^2_{\tau,b}$, with weights $c_{\tau,b}, \nu_{\tau,b}$, and its computational complexity for an each update is $O(|\mathcal{A}|)$ – which is same with the Q-learning algorithm. Detailed derivations are presented in Appendix D.

We evaluated the convergence of the algorithm, denoted as ADFQ-V2, to the optimal Q-values as well as its performance during learning in the Maze domain as we did in the main paper (Fig.4.9). ADFQ-V2 converged to the optimal Q-values in both cases. In the bottom row of the figure, we compared the performance of ADFQ-V2 to that of ADFQ. ADFQ-V2 with ϵ-greedy showed a similar performance, but ADFQ-V2 with *BS* showed a slower convergence to the optimal performance (3.0). This may be due to the stronger assumption on the variance in the algorithm update. However, ADFQ-V2 is computationally cheaper than ADFQ, and it sometimes takes less absolute time to reach an optimal performance than ADFQ in some domains.

4.9 Summary

In this chapter, we studied a Bayesian off-policy TD method called *Assumed Density Filtering Q-learning* (ADFQ). ADFQ maintains a belief distribution over each $Q_{s,a}$ and updates the Q-values considering all possible subsequent actions with their corresponding uncertainties rather than taking the best action account. ADFQ demonstrated that it could improve the overoptimism issue from the greedy update of Q-learning and the underestimation issue of Double Q-learning by showing the quicker convergence to Q^* in MDPs with stochastic

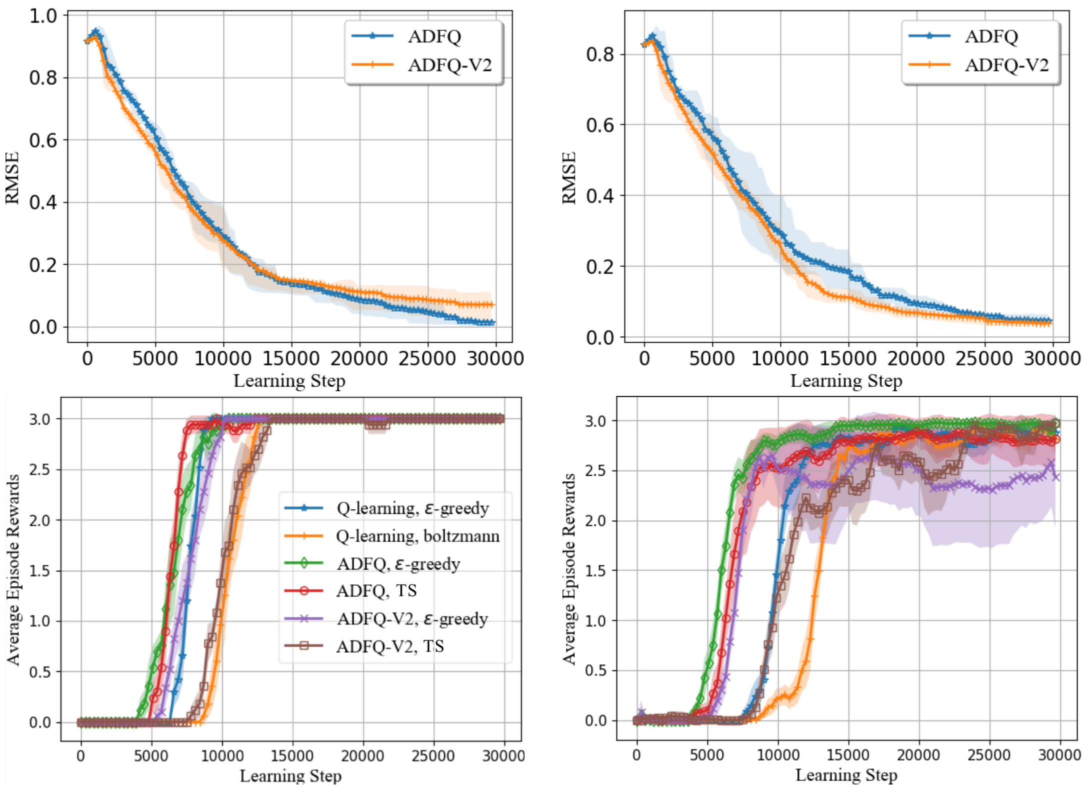

Figure 4.9: Top: Root Mean Square Error (RMSE), $||\mu - Q^*||_2$, of ADFQ and Fast ADFQ in deterministic (left) and in stochastic (right) Maze. Bottom: Greedy evaluation plots of ADFQ and Fast ADFQ during learning. The curves were smoothed by a moving average with window 4 in deterministic (left), stochastic (right) Maze.

rewards. It also outperformed comparing algorithms in finite MDPs for both deterministic and stochastic cases. The presented ADFQ algorithm shows several intriguing results.

- **Non-greedy Update.** Unlike the conventional Q-learning algorithm, ADFQ incorporates the information of all possible actions for the subsequent state in the update with weights depending on TD errors and uncertainty measures (Eq.4.2 and Eq.4.21-4.22). Therefore, it gives a principled way for a non-greedy update and helps to reduce the overestimation bias induced by the max operator.

- **Regularization with uncertainty.** ADFQ provides an intuitive update - a state-action pair with higher uncertainty in its Q belief has a smaller weight contributing less to the update. Therefore, we make use of our uncertainty measures not only in exploration but also in the value update with natural regularization based on the current beliefs.

- **Convergence to Q-learning.** We prove that ADFQ converges to Q-learning as the variances decrease and can be seen as a more general form of Q-learning.
- **Scalability** One of the major drawbacks of Bayesian reinforcement learning approaches is their high computational and space complexities [26, 36, 38]. The computational and space complexities of ADFQ are significantly than other Bayesian methods, and therefore it is extended with a complex function approximator in the next chapter.

Chapter 5

ADFQ with Neural Networks

5.1 Introduction

Reinforcement learning had attracted many scientists and engineers due to its connection to biological decision making. However, it had suffered from the curse of dimensionality when it is applied to high dimensional or continuous state and/or action domains. Numerous function approximation methods have been proposed to tackle such limitations, but the range of application domains was still significantly limited [76, 83, 109, 123]. Deep Q-network [92] must be the first reinforcement learning algorithm that is applied to very high dimensional state space without feature engineering and achieved near human performance in the Arcade Learning Environment (ALE) [10]. Since 2014, most studies in reinforcement learning have focused on or included reinforcement learning methods with a neural network, referred to as deep reinforcement learning, and demonstrated in complex sequential decision-making tasks (see Chapter 2 for further information about deep reinforcement learning).

Despite the significant progress made in deep reinforcement learning in recent years, the classical Bayesian reinforcement learning algorithms have not been extended with a neural network. This is mainly due to their high computational and space complexities.

On the other hand, we have shown that ADFQ is computationally efficient in Chapter 4, and thus it is extended with a neural network and applied to complex environments. There are previous works that implement Bayesian approaches to deep reinforcement learning by using uncertainty in the neural network weights and show promising performance in several Atari games [5, 97, 102]. However, their methods can be considered as an application of Bayesian neural network methods to reinforcement learning rather than approaching a reinforcement learning problem as a Bayesian perspective. Therefore, they only focus on exploration, and uncertainty information does not directly apply to update reinforcement learning parameters.

Our method differs from these approaches as it explicitly computes the variances of the Q-beliefs and uses them both for exploration and in the value update. Another recent work [9] proposed a gradient-based categorical DQN algorithm using a distributional perspective. The value distribution in their work represents the distribution of the random return that a learning agent receives, while the Q-belief defined in our approach is a belief distribution of a learning agent on a certain state-action pair. Therefore, ADFQ is not a replacement algorithm for distributional reinforcement learning methods, rather they can be used synergistically.

In this chapter, we show how to use ADFQ with a neural network and refer the method as Deep ADFQ. Since ADFQ is a Bayesian counterpart of Q-learning, we use a similar approach with Deep Q-Network [92]. Deep ADFQ is evaluated in inverted pendulum tasks as well as several Atari 2600 games and compare the performance with Deep Q-network and Double Deep Q-network. ADFQ outperforms the comparing algorithms by significant amounts in most domains and shows more stable learning curves.

This chapter is based on the paper [64] published in the proceedings of the 28th International Joint Conference on Artificial Intelligence. The Deep ADFQ source code can be found in https://github.com/coco66/ADFQ.

5.2 Related Work : Neural Networks to Q-learning

Many attempts to apply neural networks to reinforcement learning as function approximation had been made to solve tasks with high-dimensional or continuous state inputs [76, 83, 109, 123]. However, the temporal correlations of online reinforcement learning algorithms violates the main assumption on independent, identically distributed samples. As a learning agent performs an action, moves to the next state, and receives feedback from an environment, we obtain a sequence of correlated data. Additionally, the sample distribution constantly changes as samples are generated according to the agent's policy which is often updated over time. Deep Q-Network (DQN) solves these limitations by using an experience replay mechanism [92]. Instead of updating the network parameters online with incoming samples, it stores samples in a replay buffer and randomly selects samples from the buffer in training. We refer this sample as an *experience*, a 4-tuple of $e =< s, a, r, s' >$, and the replay buffer is denoted as $\mathcal{D} = \{e_1, \cdots, e_M\}$. Another important approach of DQN is the use of two networks. The temporal difference (TD) target in the loss function is computed from a different network with parameters ξ' while it updates a network with parameters ξ.

$$L(\xi) = \mathbb{E}_{<s,a,r,s'>\sim\mathcal{D}} \left[\left(\mathbb{E} \left[r + \gamma \max_{a'} Q(s', a'; \xi') \right] - Q(s, a; \xi) \right)^2 \right] \tag{5.1}$$

The network with ξ' is often called as a *target network*. The target network is updated in every N steps, and N remains as a tuning parameter. Later, an approach that gradually updates the target network at every step with a certain fraction is proposed [82]: $\xi' \leftarrow (1-\tau)\xi' + \tau\xi$ for $\tau \ll 1$.

In practice, a memory capacity of hardware is limited and so as the replay buffer size. DQN overwrites old samples with new samples to maintain a limited capacity regardless of how important the samples are. The uniform sampling approach from the replay buffer during training also ignores such differences among samples in terms of their importance. Prioritized experience replay [111] is proposed to tackle this limitation while still breaking the correlations among samples. In particular, the method prioritizes samples by the magnitude of their temporal-difference error. To prevent it from losing the diversity of samples, it employs stochastic prioritization and bias with importance sampling. Although its performance has been demonstrated in the Atari 2600 benchmark suite, its effect can vary depending on a task and sometimes harms learning.

Although DQN shows promising results in the Atari 2600 games, it suffers from instability

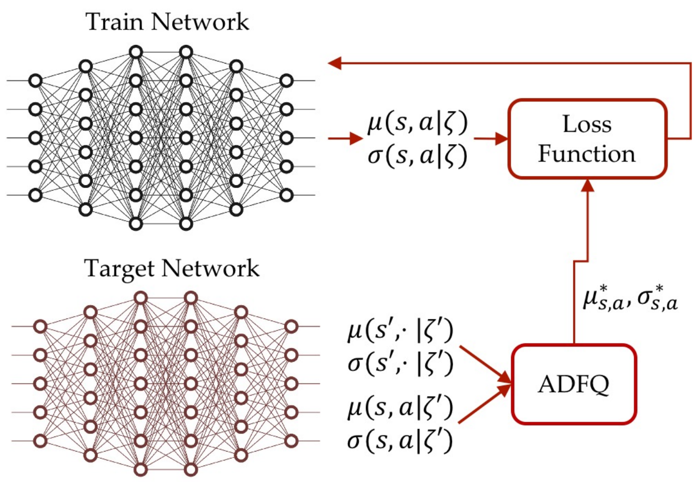

Figure 5.1: A neural network model for ADFQ

in some domains. Hasselt et. al. point out that the overestimation issue of Q-learning extends to DQN, and proposed Double DQN which is a neural network extension of Double Q-learning [129]. In Double Q-learning, it updates two Q value estimators and updates one of the estimators using the greedy estimate of the other estimator. Double DQN utilizes the feature in DQN that uses two networks, and selects the subsequent greedy action from a different network. In other word, the TD target in Double DQN is defined as

$$r + \gamma Q(s', \underset{a'}{\operatorname{argmax}}\, Q(s', a'; \xi); \xi') \tag{5.2}$$

Note that this differs from the TD target in Eq.5.1. It is empirically shown that this decomposition of the max operator into action selection and value evaluation reduces the overestimation bias occurred in DQN resulting in a better performance in multiple Atari 2600 games.

5.3 Deep ADFQ

As ADFQ is a Bayesian counterpart of Q-learning, we apply the Deep Q-network (DQN) structure and key ingredients for the neural network extension of ADFQ. ADFQ with a neural network, called as Deep ADFQ, explicitly estimate both the mean and variance of Q values for a given state. In other words, the output of the network is mean $\mu(s, a; \xi)$

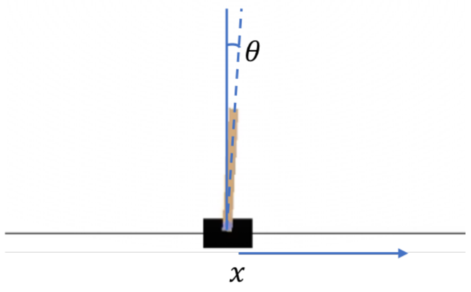

Figure 5.2: Cart-pole simulation environment in OpenAI Gym.

and variance $\sigma^2(s, a; \xi)$ of each action for a given state s where ξ is the set of network parameters. In practice, we use $-\log(\sigma_{s,a})$ instead of $\sigma^2_{s,a}$ for the network output in order to ensure positive values for the variance. As in DQN, we have a train network(ξ) and a target network(ξ'). Mean and variance for s and s' from the target network are used as inputs into the ADFQ algorithm to compute the desired mean, μ^{ADFQ}, and standard deviation, σ^{ADFQ} for the train network. We used the experience replay mechanism and a combined Huber loss functions, $L_\delta(\cdot)$. For each batch sample i, a loss function is:

$$L_i(\xi) = L_\delta\left(\mu^{ADFQ}(s_i, a_i, s'_i, r_i; \theta') - \mu(s, a; \theta)\right) + L_\delta\left(\sigma^{ADFQ}(s_i, a_i, s'_i, r_i; \theta') - \sigma(s, a; \theta)\right)$$

5.4 Experiments

In order to demonstrate the effectiveness of our algorithm, we tested on a cart-pole balancing task and six Atari 2600 games from the OpenAI gym simulator [14]. For baselines, we used DQN and Double DQN with experience replay (prioritized for the Atari games) implemented in OpenAI baselines [29] with their default hyperparameters for all tasks. We used ϵ-greedy action policy with ϵ annealed from 1.0 to 0.01 for the baselines as well as ADFQ. Additionally, we examined the Thompson sampling action policy (TS) for ADFQ [124]. Further details on the network architecture and initialization are provided in the appendix E.

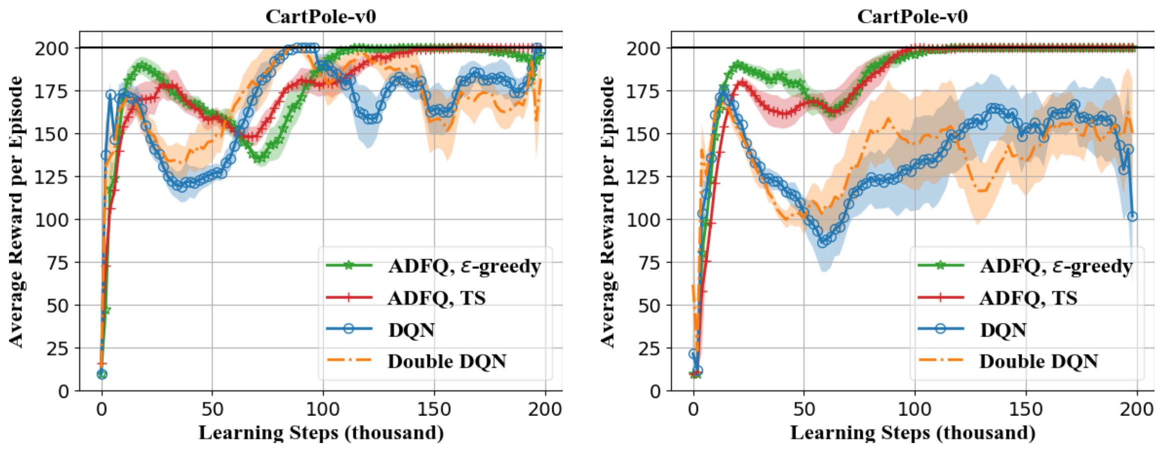

Figure 5.3: Performance of ADFQ, DQN, and Double DQN during learning in the OpenAI gym CartPole-v0 environment (Left: Deterministic, Right: Stochastic). The curves are smoothed by a moving average with window 6. The solid lines and markers represent mean performance and the shaded areas represent the standard deviation across 5 trials.

5.4.1 Cartpole : Deterministic and Stochastic Environments

The reinforcement learning state of this task consists of the 1-D position, x, and the velocity, $\dot{x}$, of the cart, and the rotational angle and its velocity of the pole, θ and $\dot{\theta}$, respectively (see Fig.5.2). The action space is defined with two discrete actions $\mathcal{A} = \{-10, 10\}[N]$ where an action is a horizontal directional force to the cart. An episode ends if the pole has fallen, or it receives $+1$ reward. Therefore, the goal of a reinforcement learning policy is to choose a sequence of actions that keeps balancing the pole on the cart. The episode ends after 200 steps, and thus the maximum cumulative reward at each episode is 200. This environment is deterministic.

Additionally, we formulate a stochastic cart-pole environment by adding randomness to the actions. The force applied to the cart with an action a is defined as:

$$F = a(n + 0.5) \quad n_t \sim \mathcal{U}(0, 1) \tag{5.3}$$

All algorithms are trained for 200,000 steps. Every 2000 steps, the learning is paused and the current policy is evaluated for 5 episodes with ϵ-greedy policy with $\epsilon = 0.05$. Note that an action policy differs from a final deterministic policy ($a^* = \pi^*(s) = \text{argmax}_a Q(s, a)$), and we use the final deterministic policy for the evaluation during learning. Each learning is repeated 5 times with different random seeds, and the average performance results during learning are presented in Fig.5.3.

In the deterministic environment (the left figure in Fig.5.3), DQN and Double DQN reach the highest reward, 200, faster than ADFQ. However, the performances drop shortly after and continue with unstable results. On the other hand, ADFQ with ϵ-greedy and ADFQ with TS stay in the optimal performance after reaching the point even though they reach the point slower than DQN and Double DQN. We see a similar result in the stochastic environment. The right figure of Fig.5.3 displays the performance of the algorithms in the stochastic environment. Both DQN and Double DQN show highly unstable performance, and they fail to reach the optimal performance. On the other hand, the evaluation results of the ADFQ policies are stable, and they reach the optimal performance even slightly faster than the ones in the deterministic environment.

Since all the network parameters are initialized near 0.0 (except for the final layer of ADFQ corresponding to the variance outputs, see details in the appendix E), the errors of initial Q estimates are relatively large. All algorithms in both environments experience a major decline period in the early stage of learning. First, the action policies are updated over time and samples are generated from the policies. Second, the range of cumulative episode reward is 0 to 200. Therefore, the network can be quickly overfit by large losses from initially collected sample in the replay buffer, and it eventually needs a correction period as it encounters more diverse examples sampled from the replay buffer.

5.4.2 Atari 2600 Environments

We test the algorithms on six Atari games, Boxing ($|\mathcal{A}| = 18$), Kung-Fu Master ($|\mathcal{A}| = 14$), Enduro ($|\mathcal{A}| = 9$), Asterix ($|\mathcal{A}| = 9$), Pong ($|\mathcal{A}| = 6$), and Breakout ($|\mathcal{A}| = 4$), from the OpenAI gym simulator [14]. As emphasized and demonstrated in Chapter 4, ADFQ updates the Q-values considering all available actions for the subsequent state with their uncertainties and TD errors, instead of the greedy update in Q-learning. Therefore, the effect of the algorithm can be expected to be more significant when the size of the action

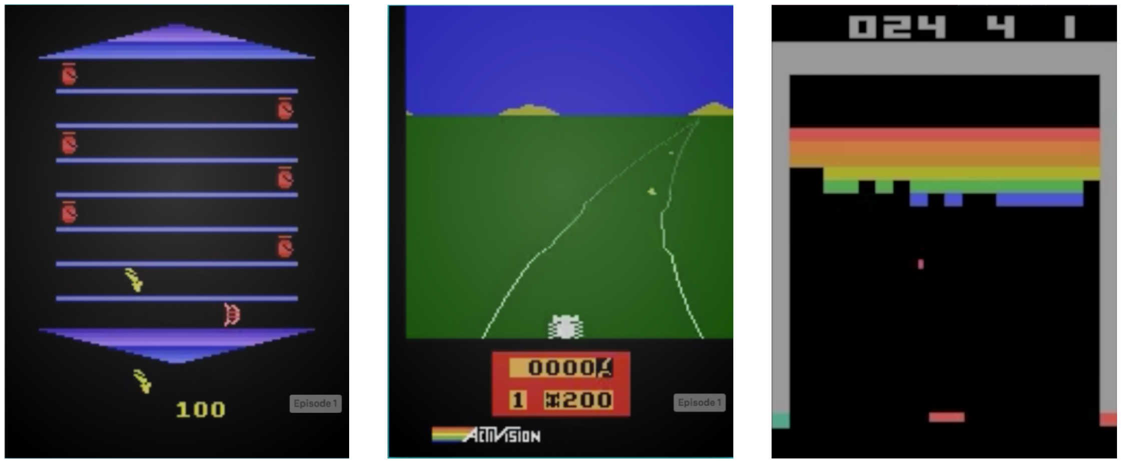

Figure 5.4: Examples of Atari games. From left to right, Asterix, Enduro, Breakout.

space is larger. Therefore, we select the games to have various numbers for available actions, $|\mathcal{A}|$.

The total number of simulation steps is $T_H = 10M$ ($5M$ for Pong), and the networks are trained every 4 steps ($1.25M$ training steps for Pong and $2.5M$ training steps for the others). Each learning is greedily evaluated at every epoch ($T_H/100$ steps) for three different episodes, and their averaged results are presented in Fig.5.5. The entire experiment is repeated for 3 random seeds for each algorithm. Rewards are normalized to $\{-1, 0, 1\}$ and different from raw scores of the games.

Both ADFQ with TS and with ϵ-greedy notably surpass DQN and Double DQN in Enduro, Boxing, Asterix, and Kung-Fu Master and show similar results in Pong. Additionally, ADFQ showed more stable performance in all domains overcoming DQN's instability. ADFQ with TS achieved slightly higher performance than the ϵ-greedy method utilizing the uncertainty in exploration. In the Breakout game which has the least number of actions, Double DQN shows the best performance among the tested algorithm. Although DQN achieves the highest performance during learning, the performance quickly drops and reveals unstable progress.

Fig.5.6 presents average estimated Q-values (μ for ADFQ) for sampled experiences during learning in four Atari games in which ADFQ outperformed both DQN and Double DQN by a significant amount. In the original paper of Double DQN [129], Double DQN constantly

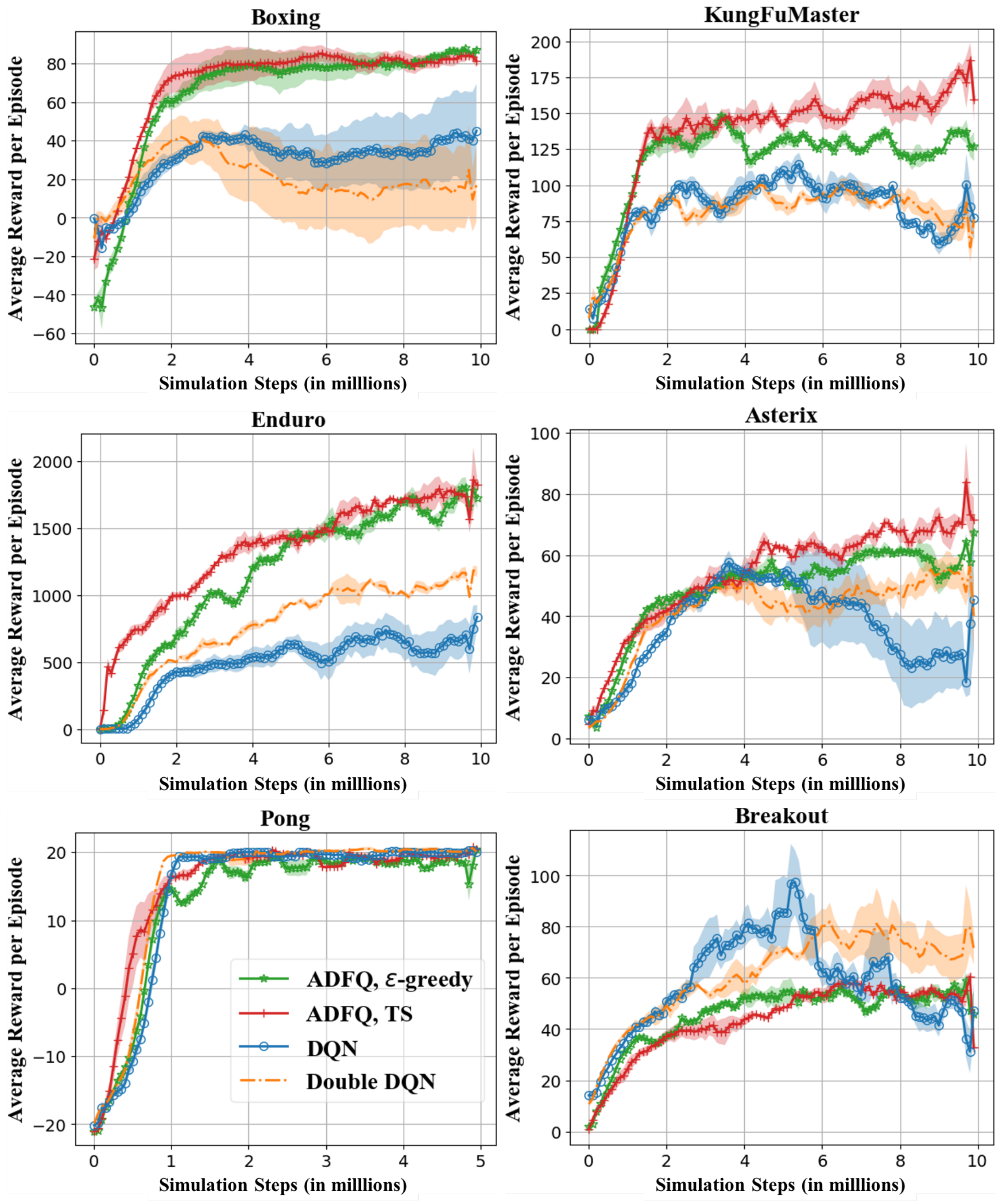

Figure 5.5: Performance of ADFQ, DQN, and Double DQN during learning in Atari games. The curves are smoothed by a moving average with window 6. The solid lines and markers represent mean performance and the shaded areas represent the standard deviation across 3 trials with different random seeds.

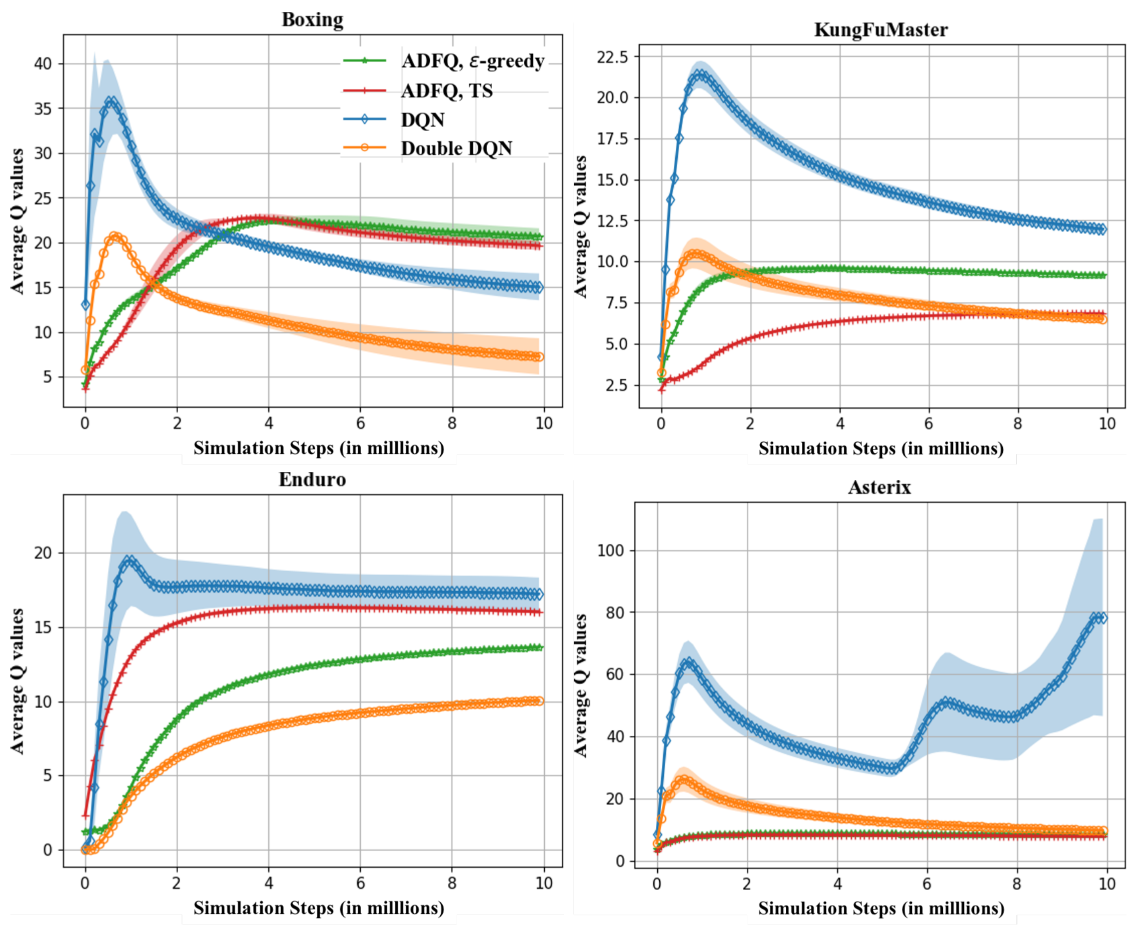

Figure 5.6: Average predicted Q-values at each evaluation of ADFQ, DQN, Double DQN in Atari games. Performance of ADFQ, DQN, and Double DQN during learning in Atari games. The curves are smoothed by a moving average with window 6. The solid lines and markers represent mean performance and the shaded areas represent the standard deviation across trials.

estimates lower Q-values than DQN and showed better performance in most Atari 2600 games. Similarly, the averaged estimated Q-values by DQN are significantly higher than the other algorithms in most cases. In all plots, the estimated values by DQN rapidly increase in the early stage of learning and the model corrects the bias after reaching a maximum value resulting in a decrease in overestimation value with time. Interestingly, even though Double DQN persistently estimates lower Q-values than DQN, it shows the same trend in three out of four cases. As discussed in Chapter 4, even though the Double Q-learning method guarantees to estimate lower values than Q-learning by using two estimators, it may underestimate the values, or it may not correct the overestimation bias enough since it still greedily select the value with the other estimator. In contrast, ADFQ do not show a rapid increase in the value estimation in any of the cases. It suggests that updating the values with a non-greedy approach based on uncertainty, TD error, and TD target provides a principled way to solve the overestimation bias.

5.5 Summary

In this chapter, we extend the Assumed Density Filtering Q-learning (ADFQ) algorithm introduced in Chapter 4 to ADFQ with a neural network function approximator. We apply the similar architectures and training mechanisms with the ones of DQN. Both the mean and variance of Q values are directly estimated as outputs of a neural network. The empirical results in the deterministic and stochastic continuous domains as well as the Atari 2600 games support that the contributions of ADFQ hold in the extended algorithm with a neural network.

We would like to highlight the fact that ADFQ is a Bayesian counterpart of Q-learning and is orthogonal to most other advancements made in deep reinforcement learning. ADFQ merely changes the loss function and we compare with basic architectures here to provide insight as to how it may improve the performance. ADFQ can be used in conjunction with other extensions and techniques applied to Q-learning and DQN.

Chapter 6

Reinforcement Learning Approach to Active Information Acquisition

6.1 Introduction

Active information acquisition is a challenging problem with diverse applications from medical diagnosis to robotics [4, 68, 75, 104, 117]. It is a sequential decision making problem where agents are tasked with acquiring information about a certain process of interest, called as *target*, through their on-board sensors. The objective function for such problems typically takes on the information-theoretic form such as mutual information and Shannon entropy. The theory of optimal experiment design also studies cost functions based on the trace, determinant, or eigenvalues of the information matrix that describes the current information state.

A major challenge in active information acquisition problems is the computation of cost functions that are difficult to compute for arbitrary probability distributions and a long planning horizon. Therefore, existing planning algorithms require prior knowledge on target models and often need to approximate a cost function online, which requires additional assumptions to make the computation tractable [21, 59, 74]. Myopic policies have been proposed to lessen the computational burden. However, they are susceptible to local minima resulting in inefficient planning trajectories or failing to achieve the optimal performance. For instance, in an active target tracking scenario, an agent should maintain enough distance from a mobile target instead of tightly following it, especially when the target drastically changes its direction often. This increases the chance of capturing the target when it moves unexpectedly. To improve the limitation of the greedy actions, approximate non-myopic algorithms are presented and they reduce the complexity of obtaining an optimal policy while providing strong performance guarantees [2, 112]. An iterative sampling-based algorithm was also proposed, increasing efficiency under a given budget constraints [58]. The complexity of their methods, however, are still not free from the planning horizon.

In this chapter, we tackle the active information acquisition problem with reinforcement learning (RL). We formulate the active target tracking problem as a Markov decision process (MDP) with an information-theoretic objective function where the value function is a discounted sum of future mutual information between target states and measurement history. One of the major advantages of using RL is its non-myopic behavior by its nature. A learned policy is executable online, and its computational complexity is independent of the planning horizon. Therefore, even though RL-based approaches demand an extended training stage, they dramatically extend the available problem domains, for example, running a robotics system in real-time. Moreover, we can avoid over-dependence on system models, providing flexibility in the choice of target process and state estimation methods such as particle filters [57] and learning-based estimations for highly non-linear systems [49, 90, 99].

In Sec.6.3, we first formally state the active information acquisition problem with mutual information as its objective. Then, we formulate the problem as a Markov decision process (MDP). Sec.6.4 presents details of applying off-policy temporal difference methods for learning a policy for the MDP in an active target tracking scenario. The performance of the learned policies are evaluated and compared with an existing search-based planning algorithm in Sec.6.5.

This chapter is based on the papers 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) [63].

6.2 Related Work

The field of active target tracking has been explored by various domains such as state estimation, sensor management, active perception, planning, and machine learning. Previous works that are closely related to this study are search-based target tracking algorithms for dynamic targets, Reduced Value Iteration (RVI) and Anytime Reduced Value Iteration (ARVI) [2, 112]. Both algorithms formulate the active information acquisition problem as a deterministic optimal control problem and compute an open-loop policy that maximizes a mutual information objective. With known system models of an agent, targets, and sensors (observation), they build a search tree of the possible trajectories of the agent and apply a pruning method to reduce the size of the search tree and to ensure finite execution time while guaranteeing suboptimality. ARVI improves RVI by eliminating the need to tune pruning parameters in RVI while reducing the computation time by re-using computations from prior iterations. Although the algorithm shows promising results in simulation and is demonstrated in real robot experiments, it requires adequate knowledge of a true target model. Since the search tree is built with a predicted target trajectory by a Kalman filter, inaccurate knowledge of the target model can lead to a path where the target does not exist. In many cases, however, target models may not be known resulting in practical limitations.

While learning-based methods have not been extensively studied in the problems of dynamic targets, numerous studies have integrated machine learning methods into tasks with static objects. They mainly focus on complications of target objects such as arbitrarily deforming objects, self-occluded objects, and semantic understanding [39, 79, 119]. The recent substantial achievement of deep learning in computer vision and perception enable solutions to track more sophisticated objects or to study complex scenarios. Following the trend of using data-driven approaches, some of the recent literature in active perception apply deep RL methods to directly learn a control policy from raw sensory inputs [60, 84, 132, 137]. Zhang et. al. proposed a RL method for visual tracking in videos that uses REINFORCE [133] and a recurrent convolutional neural network. Jayaraman et. al. also proposed a deep RL solution which learns an exploratory behavior for active completion of panoramic natural scenes and 3D object shapes. Imitation learning is also applied for information gathering tasks [19, 52]. These are supervised learning methods which train a policy to mimic the provided expert trajectories, and thus, require a large and labeled dataset. This method could be more sample efficient than reinforcement learning, but it is limited to problems having a access to labeled datasets.

6.3 Active Information Acquisition

6.3.1 Problem Formulation

We are interested in a problem where both an agent and a target are mobile following discrete-time dynamic models. We consider a single agent and N targets, and denote their states at time t as x_t and $y_t = [y_{1,t}^T, \cdots, y_{N,t}^T]^T$, respectively, where $y_{i,t}$ is an individual target state for $i = 1, \cdots, N$. x_t can be defined in a different form depending on its degree of freedom or its type. For instance, $x_t \in \mathbb{R}^2 \times SO(2)$ for a ground robot and $x_t \in \mathbb{R}^3 \times SO(3)$ for a quadrotor where $SO(n)$ is the special orthogonal group in dimension n. The goal of the agent is to maximize the cumulative mutual information for a time horizon T between the belief states at t and a measurement history $z_{1:t}$. The subscript notation $t1 : t2$ indicates a collection of data from time t_1 to time t_2. More formally, the objective of the problem is to find a sequence of control inputs to the robot, $u_{1:T}$, which satisfies,

$$\text{maximize}_{u_{0:T-1}} \sum_{t=1,\cdots,T} \mathrm{I}(y_t; z_{1:t} | x_{1:t}) \tag{6.1}$$

$$\begin{aligned} s.t. \quad & x_{t+1} = f(x_t, u_t) && t = 0, \cdots, T-1 \\ & y_{i,t+1} = g(y_{i,t}) && t = 0, \cdots, T-1 \\ & z_{i,t} = h(x_t, y_{i,t}) && t = 1, \cdots, T \end{aligned}$$

where $f(\cdot)$ and $g(\cdot)$ are dynamic models of the agent and the targets, respectively, and $h(\cdot)$ is an observation model. Since the agent does not have access to the ground truth of the target states, the agent infers the target states from its internal belief distributions. We denote the belief distribution for the i-th target as $B(y_{i,t}) = p(y_{i,t}|z_{1:t}, x_{1:t})$ and its predicted distribution for the subsequent step as $\bar{B}(y_{i,t+1}) = p(y_{i,t+1}|z_{1:t}, x_{1:t})$. The belief distribution is updated by a Bayes filter from a given prior distribution, $B(y_0)$:

$$\text{Prediction:}\ \ \bar{B}(y_t) = \int p(y_t|y_{t-1})B(y_{t-1})dy_{t-1} \quad t = 1, \cdots, 1 \tag{6.2}$$

$$\text{Update:}\ \ B(y_t) = Z^{-1}p(z_t|y_t, x_t)\bar{B}(y_t) \quad t = 1, \cdots, T \tag{6.3}$$

6.3.2 Active Information Acquisition as a Markov Decision Process

In order to solve the active information acquisition problem using reinforcement learning, we formulate the problem as an MDP. Information about the target states is essential for the agent to make an optimal decision. The exact target states, however, are unknown to the agent, resulting in the problem being formulated as a partially observable Markov decision process (POMDP) [65]. Unfortunately, it is known that finding an optimal solution for a general POMDP is intractable [103, 122]. Instead, we explicitly include the parameters of belief distributions on the targets in the RL state, s_t, and solve it as an MDP. The target states evolve with their own dynamics independently from the agent's actions. Thus, the agent's decision at time t should be based on predicted target states for $t + 1$ rather than the current target states. Therefore, we define a function, f_s, that maps the agent state and the belief prediction given map information, M, to the RL state, $s_t = f_s(x_t, \bar{B}(y_{t+1}); M)$.

The goal of solving the MDP is to find an optimal policy π^* that maximizes the cumulative mutual information in Eq.6.1. By defining $r_{t+1} = R(s_t, a_t) = I(y_t; z_t|x_t)$, we approximate the objective as a discounted sum. In other words, the value function is :

$$V^{\pi}(s) = \mathbb{E}_{\pi}[\sum_{t=0}^{T-1} \gamma^t I(y_t; z_t|x_t)|s_0 = s] \tag{6.4}$$

The action space, $\mathcal{A}$, is defined as a set of motion primitives for available control inputs.

The procedure of the problem with RL in discrete-time is illustrated in Fig.6.1. At each

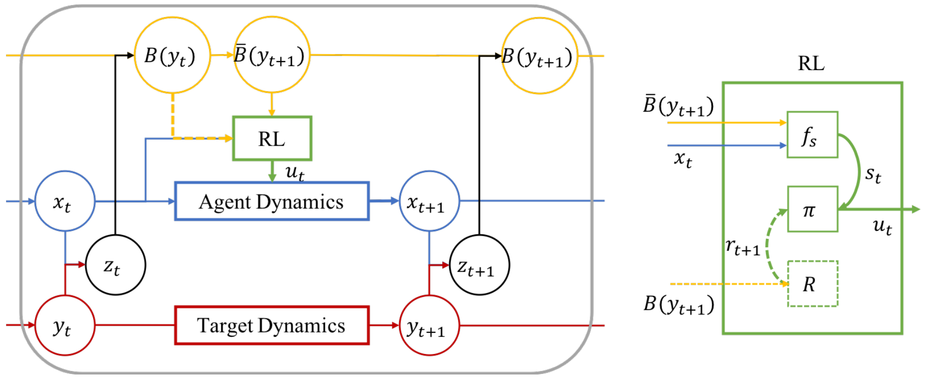

Figure 6.1: Illustration of the dynamics in active information acquisition with RL.

step t, the agent state, x_t, and its prediction on the targets, $\bar{B}(y_{i,t+1})$, are mapped to the RL state s_t. Then, the agent chooses a control input, u_t, according to the RL policy, π to maximize the objective (6.1). At the same time, the target states will evolve to the next step. Then, the robot receives measurements, z_{t+1}, from the sensor. If some targets are observed, the corresponding belief distributions are updated with the new measurements. During learning, the updated posterior belief distributions are used to compute a reward and the policy is updated according to the reward. This process is repeated at every step. Note that we do not study mapping and localization in this chapter and assume that the map and the exact odometry of the robot are known to the robot.

6.4 Application: Active Target Tracking

In this section, we present a specific reinforcement learning approach to the active information acquisition problem by focusing on the target tracking application in two-dimensional environment with Gaussian belief distributions. Assuming y_{t+1} is independent of $x_{1:t}$, the optimization problem in Eq.6.1 can be reduced to minimizing the cumulative differential entropy, $H(y_{t+1}|z_{1:t}, x_{1:t})$ [2]. Furthermore, when the belief is Gaussian $B(y_{i,t}) = \mathcal{N}(\hat{y}_{i,t}, \Sigma_{i,t})$ and the targets are independent to each other, the joint Gaussian belief on y_t is expressed as $B(y_t) = \mathcal{N}(\hat{y}_t, \Sigma_t)$ where $\hat{y}_t = [\hat{y}_{1,t}^T, \cdots, \hat{y}_{N,t}^T]^T$ and $\Sigma_t = \text{diag}_{i=1,\cdots,N}(\Sigma_i)$, and the entropy is:

$$H(y_t|z_{1:t}, x_{1:t}) = \frac{1}{2}\log\left((2\pi e)^N \det(\Sigma_t)\right) \tag{6.5}$$

and the optimization problem is reduced to:

$$\text{minimize} \sum_{t=1,\cdots,T} \log\det(\Sigma_t) \tag{6.6}$$

Therefore, the reward after executing an action a_t at s_t is defined as:

$$r_{t+1} = R(s_t, a_t) = -\kappa_m \sum_i \log\det \Sigma_{i,t+1} - \kappa_d SD_i[\log\det \Sigma_{i,t+1}] - \kappa_o o_{r,t+1}^{-2} \tag{6.7}$$

The extra terms, $SD_i[\log\det \Sigma_{i,t+1}]$ and $o_{r,t+1}^{-2}$, are added to the mutual information objective in order to assist the learning process. In practice, we found that it was able to learn a near-optimal policy without the extra terms, but the terms helped to accelerate the learning. The first two terms penalizes the overall uncertainty of the target beliefs and their dispersion (as standard deviation). The dispersion term prevents the robot from tracking only a few targets when not all the targets are within its sensing range at time. The second term discourages the robot to approach toward obstacles or a wall. κ_m, κ_d, and κ_o are constant factors, and κ_o is set to 0 if no obstacle is detected.

We define a non-geographical feature vector ϕ_t:

$$\phi_t = [\phi_{1,t}^T, \cdots, \phi_{N,t}^T, (o_t^{(x_t)})^T]^T \tag{6.8}$$

$\phi_{i,t}$ is composed of the predicted belief state for the i-th target in the agent's current frame, its covariance, and the observability of the true i-th target :

$$\phi_{i,t} \equiv [(\hat{y}_{t+1|t}^{(x_t)})^T, \log\det \Sigma_{i,t+1|t}, \mathbb{I}(y_{i,t} \in \mathcal{O}(x_t))]^T \tag{6.9}$$

$\mathcal{O}(x)$ is an observable space from the robot state x and $\mathbb{I}(\cdot)$ is a boolean function which returns 1 if its given statement is true and 0 otherwise. $o_t^{(x_t)}$ is a coordinate of the closest obstacle point to the agent in the agent frame. This feature vector is used as s_t.

We suggest off-policy temporal difference methods such as DQN, Double DQN, and

Algorithm 8 Learning a Q-network for Active Target Tracking

```
1: Randomly initialize a train Q-network, $Q(s, a|\xi)$
2: Initialize a target Q-network, $Q(s, a|\xi')$ with weights $\xi' \leftarrow \xi$
3: Initialize a replay buffer $D$
4: for episode =1:H do
5:     Randomly initialize $x_0$, $y_{i,0}$, $\hat{y}_{i,0}$, $\Sigma_{i,0}$ for $i = 1, \cdots, N$
6:     Predict $\bar{B}(y_1)$
7:     Initialize the RL state: $s_0 = f_s(x_0, \bar{B}(y_1); M))$
8:     for step $t = 0 : T - 1$ do
9:         Choose an action $a_t \sim \pi(s_t)$
10:        Update the agent state $x_{t+1} = f(x_t, a_t)$
11:        for $i = 1 : N$ do
12:            Update the $i$-th target state $y_{i,t+1} = g(y_{i,t})$
13:            Observe the $i$-th target $z_{i,t+1} = h(x_{t+1}, y_{i,t+1})$
14:            if $z_{i,t+1}$ exists then
15:                $B(y_{t+1}) = Z^{-1} p(z_{t+1}|x_{t+1}, y_{t+1}) \bar{B}(y_{t+1})$
16:            end if
17:        end for
18:        Compute a reward $r_{t+1} = R(B(y_{t+1}))$
19:        Predict target states $\bar{B}(y_{t+2})$
20:        Update the RL state: $s_{t+1} = f_s(x_{t+1}, \bar{B}(y_{t+2}); M)$
21:        $D \leftarrow D \cup \{< s_t, a_t, r_{t+1}, s_{t+1} >\}$
22:        Train the Q-network
23:    end for
24: end for
```

ADFQ in order to learn an optimal policy for the problem. Although any reinforcement learning algorithms can be used in this framework, such off-policy temporal difference algorithms are known to be more sample efficient than policy-based reinforcement learning methods [96]. Moreover, an action policy can be different from the update policy in off-policy methods which allow a safe exploration during learning. The algorithm is summarized in Table.8. Note that the reinforcement learning agent does not require any knowledge on the system models as long as it can observe its state and a reward. Additionally, the reinforcement learning update is independent from the Bayes filter and it can leverage various state estimation methods.

6.5 Experiments

To demonstrate the proposed framework, we evaluate it with ADFQ, DQN, and Double DQN in target tracking problems with different numbers of targets ($N = 1, 2, 3$). ϵ-greedy action policy is used with ϵ annealed from 1.0 to 0.01 for all algorithms. For ADFQ, we additionally used its Thompson sampling (TS) action policy using its uncertainty estimate for Q-values.

Furthermore, we compare with the Anytime Reduced Value Iteration (ARVI), an open-source target tracking algorithm, which we use as a baseline. The ARVI uses a linear system model and the Kalman Filter to predict a target trajectory, and then evaluates the mutual information over a search tree with some pruning to ensure finite execution time. The performance of ARVI has been verified in target tracking simulations and real robot experiments in [112]. The aim is to show the reinforcement learning outperforms this approach, but rather that it achieves a comparable performance while featuring a much more general problem formulation.

A robot is governed by the following differential drive dynamics:

$$\begin{bmatrix} x_{1,t+1} \\ x_{2,t+1} \\ x_{\theta,t+1} \end{bmatrix} = \begin{bmatrix} x_{1,t} \\ x_{2,t} \\ x_{\theta,t} \end{bmatrix} + \begin{bmatrix} \nu\tau sinc(\frac{\omega\tau}{2}) \cos(x_{\theta,t} + \frac{\omega\tau}{2}) \\ \nu\tau sinc(\frac{\omega\tau}{2}) \sin(x_{\theta,t} + \frac{\omega\tau}{2}) \\ \tau\omega \end{bmatrix} \tag{6.10}$$

where τ is a sampling period, and $x_{1,t}, x_{2,t}, x_{\theta,t}$ correspond to the elements of x_t in x-axis, y-axis and polar coordinate at time t, respectively. We discretized the action space with pre-defined motion primitives, $\mathcal{A} = \{(\nu, \omega) |\ \nu \in \{0, 1, 2, 3\}$ m/s, $\omega \in \{0, -\pi/2, \pi/2\}$ rad/s$\}$. The objective of the robot is to track the positions and velocities of targets which follows double integrator dynamics with Gaussian noise:

$$y_{i,t+1} = Ay_{i,t} + w_{i,t}, \qquad w_{i,t} \sim \mathcal{N}(0, W) \tag{6.11}$$

where

$$A = \begin{bmatrix} I_2 & \tau I_2 \\ 0 & I_2 \end{bmatrix}, \qquad W = q \begin{bmatrix} \tau^3/3 I_2 & \tau^2/2 I_2 \\ \tau^2/2 I_2 & \tau I_2 \end{bmatrix}$$

q is a noise constant factor. When the target is close to a wall or an obstacle, it reflects its direction with a small Gaussian noise. We assumed that the target model is known to the robot and updated the target belief distributions using the Kalman Filter. Note that the Kalman Filter can be simply replaced by other Bayes filters or learning-based state estimation methods within the proposed reinforcement learning framework.

The observation model of the sensor for each target is:

$$z_{i,t} = h(x_t, y_{i,t}) + v_t, \quad v_t \sim \mathcal{N}\Big(0, V(x_t, y_{i,t})\Big) \tag{6.12}$$

where

$$h(x,y) = \begin{bmatrix} r(x,y) \\ \alpha(x,y) \end{bmatrix} := \begin{bmatrix} \sqrt{(y_1 - x_1)^2 + (y_2 - x_2)^2} \\ \tan^{-1}((y_2 - x_2)(y_1 - x_1)) - x_\theta \end{bmatrix}$$

To be used in the Kalman Filter, this model is linearized by computing the Jacobian matrix of $h(y,x)$ with respect to y:

$$\nabla_y h(x,y) = \frac{1}{r(x,y)} \begin{bmatrix} (y_1 - x_1) & (y_2 - x_2) & \mathbf{0}_{1x2} \\ -\sin(x_\theta + \alpha(x,y)) & \cos(x_\theta + \alpha(x,y)) & \mathbf{0}_{1x2} \end{bmatrix}$$

In the experiments, the sensor has a maximum range of 10 meters and its field of view is 120 degree. We assume that the sensor is able to distinguish targets or obstacles (or walls). x_0 is randomly initialized within the given map and the position components of y_0 is also randomly initialized within the maximum offset of 8 meter from the initial robot state. The velocity is initialized to 0.0. The belief target state follows Gaussian. In order to design the experiment more realistic, the mean position is randomly initialized to have the maximum offset of 5 meter from the target and the covariance, Σ, is initialized to $30.0I_4$. We use $\tau = 0.5$ and a constant observation noise, $V = \text{diag}(\sigma_r, \sigma_b)$ with $\sigma_r = 0.2, \sigma_b = 0.01$.

For neural network architecture, we used 3 layers with 128 units and a learning rate 0.001 for a single target, and 3 layers with 256 units and a learning rate 0.0005 for multiple targets. The target Q network is updated every 50 training steps. The batch size and the size of the replay buffer are 64 and 1000, respectively.

All experiments are obtained with 5 different random seeds for the learning algorithms and 10 random seeds for ARVI. The results are plotted in Fig.6.2. The darker lines show the

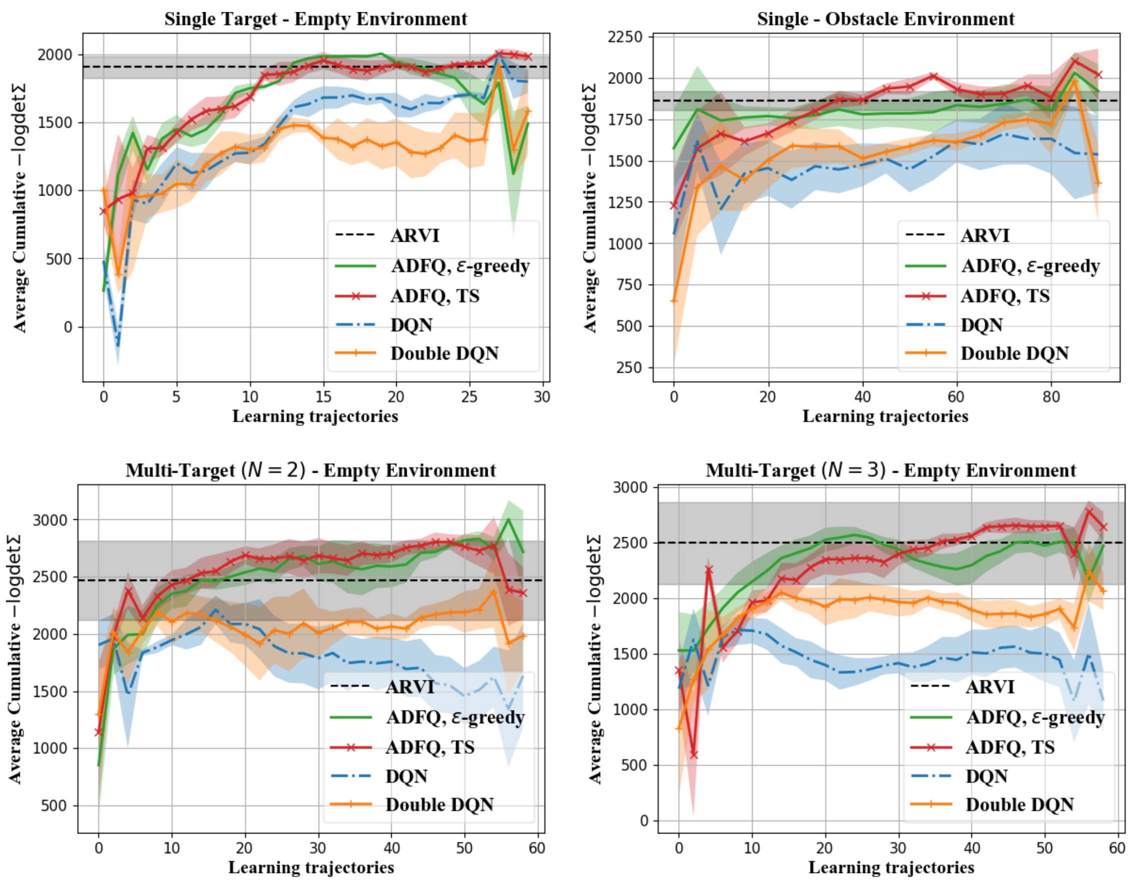

Figure 6.2: Cumulative $-\log\det\Sigma_t$ per trajectory of ADFQ, DQN, and Double DQN during learning compared with ARVI.

mean over seeds and the shaded areas represent standard deviation. The current learned policies from ξ_t were semi-greedily evaluated with $\epsilon = 0.05$ for 5 times after trained with a single trajectory (every two trajectories for multi-target experiments). The curves are smoothed by a moving average with window 4.

6.5.1 Single Target Tracking

We tested the single target problem in an empty domain ($100 \times 100[m^2]$) where there is no obstacle, and therefore, the behavior of the target is more predictable (as there is far less reflection behavior of the target with noise). We also tested a domain with four obstacles as in the first row of Fig.6.3. The noise parameter for the target model is set to $q = 0.01$ for both cases and the length of a trajectory is $T = 100$ steps.

The first plot in Fig.6.2 shows that both ADFQ with TS and ADFQ with ϵ-greedy

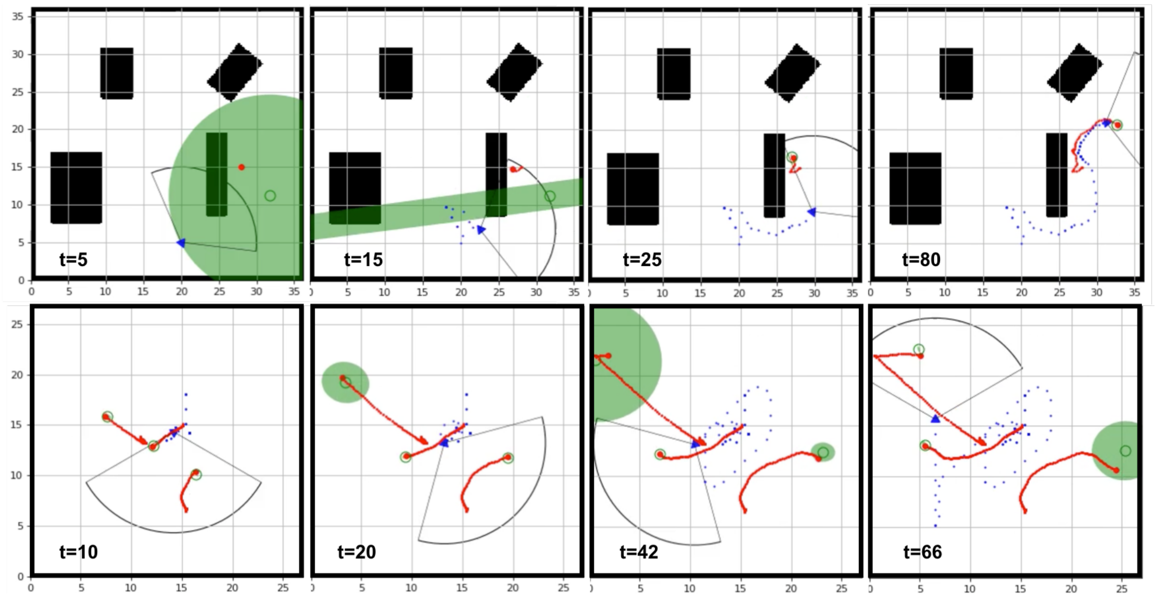

Figure 6.3: Demonstrations of learned ADFQ-TS policies in the obstacle environment with a single target (first row) as well as the empty environment with three target (second row). The time step increases from left to right. Blue triangle: x_t, Blue dot: $x_{1:t-1}$, Red dot (big): y_t, Red dot (small): $y_{1:t}$, Green circle: $\hat{y}_t$, Green shaded area: Σ_t.

achieved the baseline performance after learning with 13 trajectories. ADFQ-TS showed a more stable performance outperforming the baseline toward the end. Since the belief state mean can quickly diverge from the true state while its covariance is quite small, exploration methods based on state-action uncertainty such as Thompson sampling leads a better performance than ϵ-greedy. ADFQ achieved the near baseline performance in the obstacle environment as well. An example case of a learned policy by ADFQ-TS is presented in the first row of Fig.6.3. As shown, even though it missed the target at $t = 15$ and the belief became inaccurate, it quickly adjusted its direction and followed the target keeping it in its range.

DQN and Double DQN failed to reach the baseline performance in both environments. Although their performances increased with the number of learning trajectories in the empty environment, their performances dramatically dropped in the obstacle environments. This is due to the high stochasticity of the environment as the target changes its path abruptly with noise when it faces an obstacle.

6.5.2 Multi-Target Tracking

We tested the cases of two and three targets in an empty domain ($27\times27[m^2]$) with $q = 0.001$. A longer trajectory, $T = 150$, is used in order to evaluate cases where targets diverge and a robot has to keep traveling to minimize the covariances. As shown in Fig.6.2, the mean of both ADFQ algorithms outperformed the mean of baseline after 13 trajectories of learning. For $N = 3$, ADFQ with ϵ-greedy reached the similar performance with the baseline and ADFQ with TS outperformed the baseline. Additionally, the baseline showed large variances in its performance in both cases while ADFQ algorithms showed fairly lower variances across the trials.

The most challenging part of these experiments is when not all targets are observable at time. The results indicate that the reinforcement learning methods can learn a policy which makes a near-optimal decision on when to keep traveling to track all the targets or when to exploit to close targets. The learned policy of ADFQ-TS is demonstrated in the second row of Fig.6.3. When the targets are not simultaneously observable but not too far from each other, the robot must choose to visit each target sequentially to maintain its belief distribution for every target.

6.6 Summary

In this chapter, we introduced a novel reinforcement learning approach for the active information acquisition problem and developed a detailed approach for solving the active target tracking problem with a Q-network-based reinforcement learning algorithm. The experimental results demonstrated that the reinforcement learning-based methods can achieve or sometimes outperform the existing search-based planning algorithm.

As an initial approach, we assume that the target model is mostly known to the Bayes filter (except when it collides into an obstacle) and use a relatively simple environment setup that is used in previous works. In the next chapter, we will extend this approach to a non-linear and partially known target model in more complex environments.

Chapter 7

Learning to Track Agile Targets in a Partially Known Environment

7.1 Introduction

Active target tracking is an information gathering task where a mobile agent makes a sequence of control decisions to track targets of interest using its on-board sensors. For instance, a camera angle can be controlled to find an exit in a room, or an autonomous vehicle can move to view and identify an occluded object. Its various applications include surveillance [55, 110], search-and-rescue task [75], active perception [60, 84, 119, 137], and environmental monitoring [18, 31].

Problems in active tracking for dynamic targets tackle challenges associated with target motions such as estimation of target states, planning horizon, cost function, and stochasticity of the targets, rather than placing the weight on processing and inferencing sophisticated raw sensory data. Previous studies applied various techniques from Bayesian estimation, information theory, and optimal control to solve a spectrum of problems.

One major limitation of previous works is a strong dependency of their methods on a specific problem formulation and system models. Due to such dependencies, they have focused on settings where the prior knowledge of targets are sufficiently accurate and/or the targets are located near the sensing range [2, 21, 112]. However, such prior conditions considerably simplify the problem as the main task becomes tightly following a target. In real-world examples, tracking dynamic targets with partial information in a large environment requires more than chasing targets within the range of a sensor. When targets are initially located far from the agent or when they are occluded by obstacles, the agent must navigate through any obstacles in the domain to reach the targets – *Navigation*. Additionally, an initial belief about a target can be inaccurate, leading to issues when the agent searches for a target near its incorrect belief location. Thus, the agent should be able to explore the environment until it discovers the targets – *Discovery*. These capabilities are also required when targets move quickly or when a target model in the beliefs is highly inaccurate. In such cases, the agent often loses the targets and needs to explore the domain while navigating until it rediscovers them. We hereafter refer to tracking within the range of a sensor as *in-sight tracking*, and consider *in-sight tracking*, *navigation*, and *discovery* as sub-tasks of active target tracking.

Exploratory behavior for information gathering has long been studied in robotics. Various approaches have been presented including heuristic methods [40, 135], formal decision theory [1], and planning [12, 72, 78]. Many of the studies aim to increase the accuracy in building maps in unknown environments. Myopic approach is studied which maximizes the expected Shannon information gain of an occupancy map as well as minimizes the uncertainty of the vehicle pose and map feature uncertainty [12]. It is also solved as a sequential decision making problem, formulated as a POMDP. Kollar and Roy [72] applied policy search dynamic programming [6] to find an optimal trajectory for improving accuracy of a map by minimizing the number of sensor measurements for map features. An approach proposed in [78] solves a POMDP with sequential monte carlo optimization. They use a sampling-based approximate mutual information for its reward function. More recently, learning-based approaches have been presented to solve exploration tasks. Chen et. al. used imitation learning to pre-train an exploration policy, and then applied a policy gradient algorithm to further optimize the policy. Similar to our approach, they use egocentric maps for the network inputs. Their reward function includes a coverage measure as well as a collision penalty with fine-tuned parameters. Our goal for exploration is to find dynamic objects, and differs from these previous works which aim to construct accurate information about static surroundings.

Navigation tasks of mobile robots have also been paid increasing attention from the learning community [39, 47, 66]. Gupta et. al. proposed an architecture that builds a belief map of a world and trains a mapper and a planner to reach a goal [47]. However, one of the challenges for applying such navigation approaches to target tracking scenarios is that a goal position continuously changes as targets move. Re-planning its path for a new goal every step is computationally expensive. Another challenge is that a goal position is not always identical to a belief position. For instance, in a situation with two targets, the optimal goal position for the agent is located somewhere in between the targets and is also dependent on their uncertainties.

In this chapter, we introduce a method of learning a unified reinforcement learning (RL) policy that is capable of all three tasks - *in-sight tracking*, *navigation*, and *discovery.* As we discussed in the previous chapter, one of the benefits of using RL is that we can drop the dependency of an algorithm on prior conditions and system models – agent dynamics, target dynamics, and observation model. Therefore, it is able to learn a *robust* behavior for a scenario where the agent's knowledge of the target model considerably differs from a true target model. We additionally incorporate map information and visit-frequency, which reflects how recently a certain area has been scanned by the sensor, into an RL state. This enables an RL policy to *learn to navigate* and *learn to explore.* Our model uses egocentric information with respect to the agent frame, dropping the strong dependency of a learned policy on its training environment. We demonstrate the proposed method in a target tracking environment with various scenarios including agile targets, imperfect prior knowledge on a target model, and inaccurate initial belief.

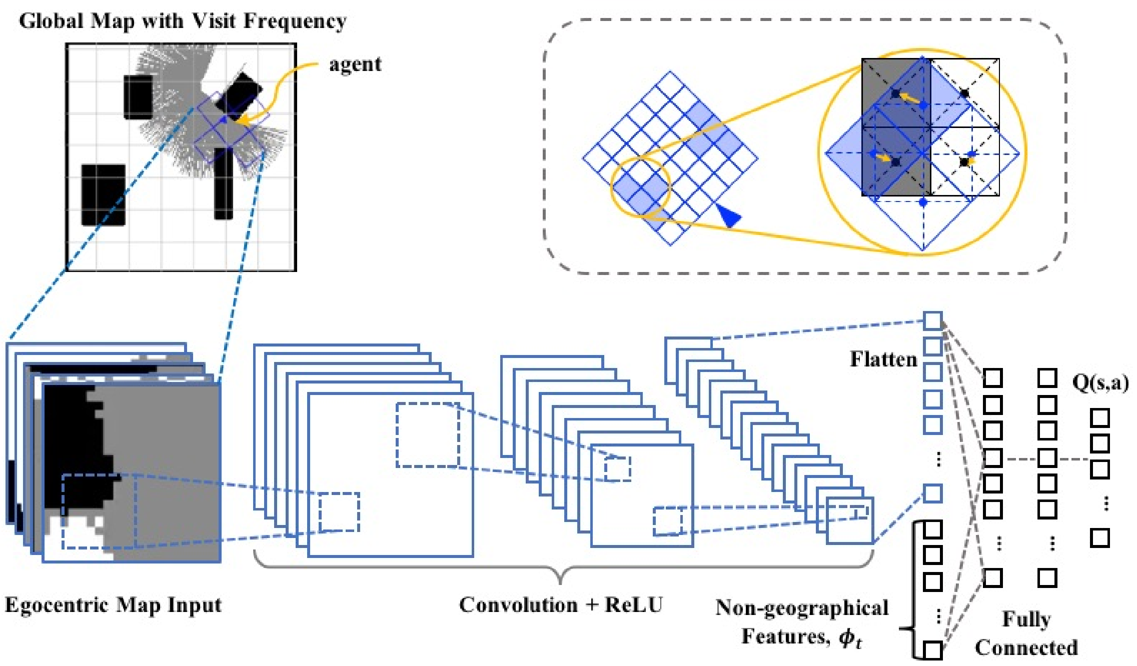

Figure 7.1: Illustration of the active target tracking network architecture.

7.2 Active Target Tracking Network

Numerous papers have pointed out the limitation of RL methods when goals shift. In general, this requires a complete re-training even though the underlying environment remains the same. However, in an active target tracking setting, not only is a goal position changed in every episode, but a goal position can be continuously changed over time because it is dependent on the uncertainty of beliefs as well as target motions. Since the goal should be implicitly determined by the policy, simple solutions such as including a goal as a part of the RL state [138] will not work. Therefore, we use information in the agent's perspective for the RL state and reduce the dependency on the training environment.

In Chapter 6, we have defined a non-geographical feature vector $\phi_t = [\phi_{1,t}^T, \cdots, \phi_{N,t}^T, (o_t^{(x_t)})^T]^T$. $\phi_{i,t}$ is composed of the predicted belief state for the i-th target in the agent's current frame, its covariance, and the observability of the true i-th target :

$$\phi_{i,t} \equiv [(\hat{y}_{t+1|t}^{(x_t)})^T, \log\det \Sigma_{i,t+1|t}, \mathbb{I}(y_{i,t} \in \mathcal{O}(x_t))]^T \tag{7.1}$$

$\mathcal{O}(x)$ is an observable space from the robot state x and $\mathbb{I}(\cdot)$ is a boolean function which returns 1 if its given statement is true and 0 otherwise. $o_t^{(x_t)}$ is a coordinate of the closest obstacle point to the agent in the agent frame.

However, these non-geographical features are not enough for the agent to make an informed decision under all circumstances. As pointed out in the previous sections, the ability to navigate around obstacles is crucial. To achieve this, we need more information than just the closest obstacle point. For example, suppose the agent is located in front of an obstacle and its left corner is closer to the agent than the right corner. It is impossible for the agent to figure out the shorter path to pass the obstacle only with the closest obstacle point. Another important capability of the agent is to explore the current domain when a target is not near the corresponding belief. Note that this exploration means exploring a given environment with a deterministic policy and differs from the exploration problem in RL (action exploration) during learning. To learn to explore in the MDP setting, we build a visit frequency map similar to the occupancy grid mapping. Suppose that λ_c is a visit-frequency value for a cell c. At t,

$$\lambda_{c,t} = \begin{cases} 1 & \text{if c is scanned} \\ \lambda_{c,t-1} \cdot c \exp\left(\frac{\bar{v}_t \tau}{r_{sensor}}\right) & \text{otherwise} \end{cases} \tag{7.2}$$

where $\bar{v}_t$ is the average speed of the targets from the beliefs, τ is a sampling period, r_{sensor} is a sensing range of the sensor, and c is a constant factor. Therefore, the most recently visited cells have the value 1.0 and it decays over time as a function of the current target speed estimate.

To reduce the dependency of a learned policy on training environments, we use egocentric maps of the surrounding areas of the current agent position as inputs to the convolutional neural network (CNN) in our architecture. The flatten output of the CNN is concatenated by the non-geographic features ϕ_t, and then fed to the fully connected network. Fig. 7.1 shows the illustration of the network architecture and the inputs.

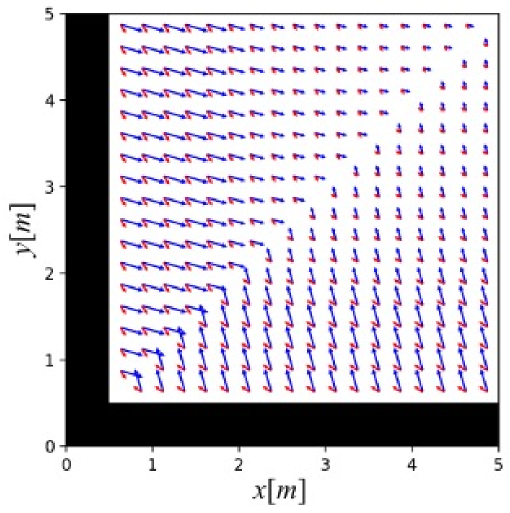

Figure 7.2: An example of the effect of $\zeta(\cdot)$ in Eq.7.4 at different target positions in the grid. Each arrow corresponds to the velocity components of ζ, $[a_t\tau\cos(\theta_{rot,t}), a_t\tau\sin(\theta_{rot,t})]^T$ when $\tau = 0.5, v_{max} = 3.0, r_{margin} = 0.1, r_{min} = 1.0$. The blue arrows are for $v_t = 3.0$ and the red arrows are for $v_t = 1.0$. The black blocks are obstacles and the target orientation are $-3\pi/4$ [rad] for all.

7.3 Target Tracking Environment

In this section, we describe details of the target tracking environment. It is designed for reinforcement learning practice and follows the OpenAI Gym structure, a popular benchmark simulation environment for the RL community [14]. The source codes can be found at `https://github.com/coco66/ttenv`. *Notations:* The xy-position and the orientation in $SE(2)$ are represented with a subscript 1, 2, θ, for example, $(x_1, x_2) \in \mathbb{R}^2$ and $x_\theta \in SO(2)$, respectively.

7.3.1 Target Model

We design a target model based on the double integrator with Gaussian noise. In order to maneuver smoothly around obstacles, we add a non-linear term, $\zeta(\cdot)$, that pushes the target away from a nearby obstacle.

$$y_{t+1} = Ay_t + w_t + \zeta(y_t; M) \qquad w_t \sim \mathcal{N}(0, W(q)) \tag{7.3}$$

where $y_t = [y_{1,t}, y_{2,t}, \dot{y}_{1,t}, \dot{y}_{2,t}]^T$ and

$$A = \begin{bmatrix} I_2 & \tau I_2 \\ 0 & I_2 \end{bmatrix}, \qquad W(q) = q \begin{bmatrix} \tau^3/3I_2 & \tau^2/2I_2 \\ \tau^2/2I_2 & \tau I_2 \end{bmatrix}$$

q is a noise constant and I_n is an $n \times n$ identity matrix. $\zeta(\cdot)$ is a function of the current target state and the map, M, and it directs the target away from its closest obstacle which polar coordinate with respect to the current target frame is denoted as $r_o^{(y_t)}, \theta_o^{(y_t)}$.

$$\zeta(y_t; M) = [0, 0, a_t \tau \cos(\theta_{rot,t}), a_t \tau \sin(\theta_{rot,t})]^T \tag{7.4}$$

where

$$a_t = \frac{\nu_t \cos_+(\theta_o^{(y_t)})}{\max(r_{\min}, r_o^{(y_t)} - r_{\text{margin}})^2} \tag{7.5}$$

$$\theta_{rot,t} = \begin{cases} y_{\theta,t} + \theta_o^{(y_t)} - \theta_{rep,t} & \text{for } \theta_o^{(y_t)} \geq 0 \\ y_{\theta,t} + \theta_o^{(y_t)} + \theta_{rep,t} & \text{otherwise} \end{cases} \tag{7.6}$$

$$\theta_{rep,t} = \frac{\pi}{2}\left(1 + \frac{1}{1 + e^{-(\nu_t - \nu_{\max}/2)}}\right) \tag{7.7}$$

$$\nu_t = \sqrt{\dot{y}_{1,t}^2 + \dot{y}_{2,t}^2} \tag{7.8}$$

r_{min} and r_{margin} are design parameters that determine the maximum value of a_t and a margin from an obstacle point with the maximum effect, respectively. ν_{max} is the maximum target speed, and $\cos_+(x) = \cos(x)$ if $-\pi/2 \leq x \leq \pi/2$ and 0 otherwise. Fig.7.2 illustrates the effect of $\zeta(\cdot)$ around the obstacle in different positions.

Additionally, if y_{t+1} in Eq.7.3 causes a collision with an obstacle, its velocity is changed while its position remains same :

$$y_{t+1} = y_t + (\nu_t + n_t)[0, 0, \cos(\theta_{rot,t}), \sin(\theta_{rot,t})]^T \tag{7.9}$$

where $n_t \sim \mathcal{N}(0,1)$.

7.3.2 Agent and Observation Models

The agent follows the differential drive dynamics,

$$\begin{bmatrix} x_{1,t+1} \\ x_{2,t+1} \\ x_{\theta,t+1} \end{bmatrix} = \begin{bmatrix} x_{1,t} \\ x_{2,t} \\ x_{\theta,t} \end{bmatrix} + \begin{bmatrix} \nu\tau sinc(\frac{\omega\tau}{2})\cos(x_{\theta,t} + \frac{\omega\tau}{2}) \\ \nu\tau sinc(\frac{\omega\tau}{2})\sin(x_{\theta,t} + \frac{\omega\tau}{2}) \\ \tau\omega \end{bmatrix} \tag{7.10}$$

controlled by linear and angular velocity commands, v and ω, respectively.

We use a range-bearing sensor which is commonly used in practice in robotics and assume that the agent can uniquely identify different targets. The observation model of the sensor for each target is:

$$z_{i,t} = h(x_t, y_{i,t}) + v_t, \quad v_t \sim \mathcal{N}\left(0, V\right) \tag{7.11}$$

where V is a observation noise covariance matrix and

$$h(x,y) = \begin{bmatrix} r_{x,y} \\ \alpha_{x,y} \end{bmatrix} := \begin{bmatrix} \sqrt{(y_1 - x_1)^2 + (y_2 - x_2)^2} \\ \tan^{-1}((y_2 - x_2)(y_1 - x_1)) - x_\theta \end{bmatrix}$$

7.3.3 Belief Update

We use a Kalman Filter to update the beliefs on the targets. While previous works assume that the target model is known to the agent [2, 112], we release the assumption by allowing only the partial knowledge, a double integrator with A and $W(q_b)$ in Eq.7.3, and excluding the $\zeta(\cdot)$ term. In a domain with obstacles, the effect of $\zeta(\cdot)$ is significant, and the belief update becomes considerably inaccurate leading to a more challenging task.

The observation model in Eq.7.11 is approximated to a linear model in the Kalman filter and the Jacobian matrix of $h(y,x)$ with respect to y is :

$$\nabla_y h(x,y) = \frac{1}{r_{x,y}} \begin{bmatrix} (y_1 - x_1) & (y_2 - x_2) & \mathbf{0}_{1x2} \\ -\sin(x_\theta + \alpha_{x,y}) & \cos(x_\theta + \alpha_{x,y}) & \mathbf{0}_{1x2} \end{bmatrix}$$

7.4 Experiments

Unlike many benchmark simulation environments in RL, the target tracking environment is highly stochastic and contains considerable randomness. First, the agent, beliefs, and targets can be randomly initialized within a large range in a domain. As discussed further in the following section, different abilities of an agent can be emphasized depending on initialization. Moreover, the noise components in the target motion, the belief model, and the observation model provide additional randomness in performance. Lastly, the agent can discover a target by chance as both the targets and the agent are dynamic and as the target motion is independent of an agent path. Therefore, it is important to have a learning environment that provides diverse samples while to carefully design evaluation environments that reduces a large performance variance across different trials.

We perform experiments in single-target scenarios as well as two-target scenarios. When there is more than one target, the maximum velocity of the agent should be much higher than that of targets in order to continuously gather information about the targets while traveling among them. If targets are too far apart, committing to track nearby targets can return higher mutual information in a given time than tracking all targets. Therefore, we use different training and evaluation settings for single-target and two-target environments.

7.4.1 Algorithms

We learn a ATTN policy using Assumed Density Filtering Q-learning, a Bayesian Q-learning method which has shown promising performance in stochastic environments and tasks with a large action space [63, 64]. It is an off-policy temporal difference learning, and thus, we can have an action policy different from the target policy. For the action policy, we use Thompson sampling [124] which samples Q-values for all actions given a state from the belief distributions and select an action with the maximum sample value. We use $\gamma = 0.99$, the discount factor, which results in the length of the effective horizon as $T_{eff} = 687$ ($T_{eff} = \text{argmax}_t \, \gamma^t$ s.t. $\gamma^t > 0.001$). The dimension of the egocentric map input is $25 \times 25 \times 5$, and the two convolutional layers consist of 20, 4×4 filter, with a stride of 3 and 40, 3×3

filter, with a stride of 2. The following fully connected layers consist of 3 layers with 512 hidden units. The network is trained for 30K steps (or 300 episodes) with three different random seeds (thus, three models).

We compare the ATTN policy with the Anytime Reduced Value Iteration (ARVI) algorithm, an open-source search-based target tracking algorithm that has been verified in both simulations and real robot experiments [112]. Details of the algorithm is explained in Section 6.2. The algorithm optimizes the same mutual information objective of ATTN. It also uses the same observation and agent models and the Kalman filter described in Section 7.3. ARVI finds a near-optimal path at each step given a specified amount of time denoted as T_{ARVI}. Therefore, this allocated planning time is usually equal to or less than the sampling period or a time interval, τ. In this experiment, we set this value to be $T_{ARVI} = \tau = 0.5$ [s]. The planning horizon for ARVI is set to 12 steps following the original paper. Since the algorithm finds a path in a given time, a longer planning horizon sometimes helps to find a better path but it can also lead to a worse plan since the search space becomes too large. We tested with 1, 5, and 10 for the number of controls to be applied at each time step, and 5 showed the best result.

7.4.2 Training Setup

The training environment is designed so that the learning agent can be exposed to various situations. The system models follow the models described in the previous section with the following parameters : $\tau = 0.5$, $V = \text{diag}(0.2, 0.01)$, $r_{\text{margin}} = 1.0$, $\nu_0 = 0.0$ [m/s]. The maximum sensing range is $r_{\text{sensor}} = 10$ [m] and its field of view is 120 degrees. A set of motion primitives, or the action space, is $\mathcal{A} = \{(v, \omega) | v \in \{0, 1, 2, 3\}[m/s], \omega \in \{0, -\pi/2, \pi/2\}[rad/s]\}$ and the time horizon of an episode, T is set to 100. In single-target scenarios, the noise constants of the target model and the belief model are set as $q = q_b = 0.5$, and the maximum target velocity is set as $\nu_{\text{max}} = 3.5$ [m/s]. For two-target scenarios, lower values are used for training: $q = q_b = 0.2$ and $\nu_{\text{max}} = 1.0$ [m/s]. As mentioned above, it is infeasible for an agent to keep tracking multiple targets that are diverging from each other with limited dynamic constraints. Additionally, high noise constant of the target

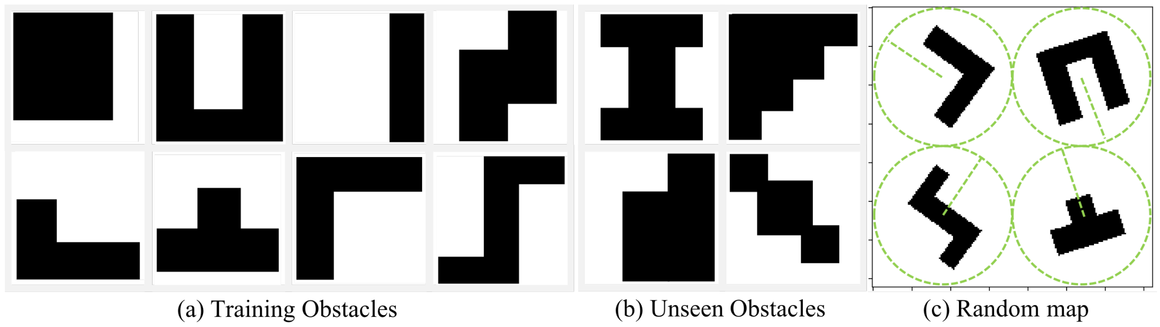

Figure 7.3: Obstacle polygons for generating a random map. (a) Obstacles used during training, (b) Unseen obstacles, (c) An example map of a randomly generated map.

model makes a target to be quickly diverged from its corresponding belief while the agent is tracking another target (when it is not available to cover both targets at once). While this requires an ability to explore near the belief when losing the target, it may also result in a case where committing to one target gives a higher return.

Map. To learn a policy that is robust to various environmental configurations, we randomly generate a map from a set of obstacle candidates as shown in Fig. 7.3 (a). These obstacles include both convex concave polygons resulting in more challenging navigation tasks. At each episode, four randomly selected obstacles are placed at the center of each quadrant of an empty domain with random orientations (see Fig. 7.3(c)). The map resolution (cell size) is 0.4 [m] and the map dimension is 72.4×72.4 [m^2].

Initialization. For each episode, the robot is randomly initialized in a given domain. In single-target domains, the initial belief position is randomly chosen within a distance between 5.0 to 20.0 [m] from the agent, and the initial target position is randomly placed within (0.0, 20.0)[m] range from the belief. Since the agent may be required to travel between targets in two-target domains, smaller ranges are used – (5.0, 10.0)[m] range for a distance between beliefs and the agent, and (0.0, 10.0)[m] range for a distance between a belief and its corresponding target. These ranges are chosen considering the maximum sensing range of the agent. Note that having a target placed too far from its corresponding belief can lead to learning a policy that does not trust the belief.

We added a penalty to the reward function when the agent chooses an action that imme-

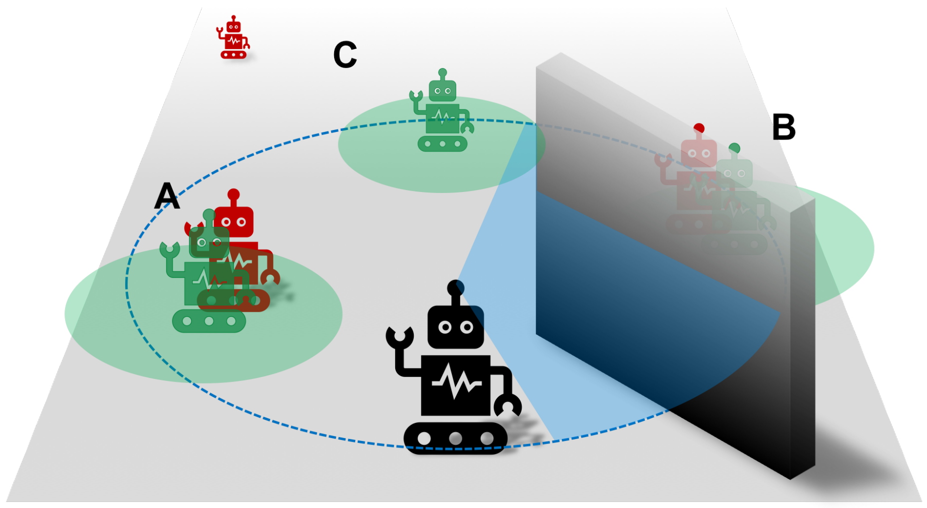

Figure 7.4: Illustration of the three initialization configurations. The black figure corresponds to the robot with a range-bearing sensor on top. The blue dotted circle is the sensing radius and the faded blue sector indicates a covered area by the sensor. Targets are represented with the red figures. The green figures are beliefs with uncertainty represented as the faded green circles. The black cuboid is an obstacle occluding the target and the belief in the configuration B.

diately leads to a position within r_{margin} distance to an obstacle. The penalty accelerated the learning by a marginal amount, but did not make any noticeable difference in performance.

7.4.3 Evaluation Setup for Single-Target Domains

In single-target domains, different sets of initial positions of the robot, targets, and beliefs emphasize different abilities of an agent. For example,

- Occlusion of a target and/or its corresponding belief : This requires an ability to navigate around obstacles.
- The initial distance between $y_{i,t}$ and x_t : A larger value requires the ability to navigate for a long path.
- The initial distance between $y_{i,t}$ and $\hat{y}_{i,t}$: A larger value requires the ability to explore the current domain until reaching the target.

Therefore, we consider three different initialization setups to evaluate capabilities of the testing algorithms in the sub-tasks of active target tracking – *in-sight tracking*, *navigation*, *discovery* – separately.

1. In-sight tracking task: A target and the corresponding belief are randomly initialized within the sensing range ($||y_{i,t} - x_t||_2 \in [3, 10]$), and they are located close to each other ($||y_{i,t} - \hat{y}_{i,t}||_2 \in [0, 3]$). The case A in Fig. 7.4 illustrates this initial condition. We experiment with target model noise level and the maximum target speed limit.

2. Navigation task: Both a target and the corresponding belief are initialized relatively far from the agent ($||y_{i,t} - x_t||_2 \in [15, 20]$), and occluded as described in Fig. 7.4 B. They are located close to each other ($||y_{i,t} - \hat{y}_{i,t}||_2 \in [0, 3]$).

3. Discovery task : The initial belief is located within the sensing range ($||\hat{y}_{i,t} - x_t||_2 \in [3, 10]$), but the target is initialized far from the belief ($||y_{i,t} - \hat{y}_{i,t}||_2 \in [15, 20]$) as illustrated in Fig. 7.4 C. Thus, the agent won't be able to discover the target by simply reaching the belief location and scanning around. As the agent has beliefs on targets, we exclude cases where beliefs are significantly far from real targets.

We first generated a set of 10 episodes for each evaluation task. In each episode, the target trajectory, initial robot and belief states, and map configuration are randomly generated and they differ across episodes. Three trained models of ATTN with different random seeds are evaluated on the evaluation sets. ARVI is also evaluated on the evaluation sets with three random seeds. To prevent a case where the target approaches the robot and is found by luck, the target starts to move once it is observed by the robot for the first time in each episode.

7.4.4 Results in Single-Target Domains

The robustness of the ATTN policy and ARVI is evaluated in the in-sight tracking task. In particular, we are interested in tracking fast and anomalous targets. Thus, we tested ATTN and ARVI with different values for both the target noise constant q in the target model Eq.7.3 and the maximum target speed, $\nu_{\max}$. The larger the q value, the more deviated the belief state is from the target state. Furthermore, q affects how much the target speed evolves over time. If the q value is small, the target may never reach the maximum target speed in a given time horizon. q_b is set to be 0.5 as same as the value used in training.

To quantitatively measure the performance of the algorithms, we consider mutual information, or in particular, a normalized sum of negative log of determinant of covariance in predictions within an episode:

$$\bar{J} = \frac{-\sum_{t=1,\cdots,T} \log\det(\Sigma_{t+1|t}) - J_{\min}}{J_{\max} - J_{\min}} \in [0,1] \tag{7.12}$$

The lower bound, $J_{\min}$, is found when there is no observation in an episode, and the belief is updated only by the prediction step in the Kalman filter. The upper bound, $J_{\max}$, is met when the target is observed at every step. According to Theorem 4 in [118], $J_{\max} = -T \log\det(W)$ and $J_{\min} = -\sum_{t=1,\cdots,T} \log\det(\Sigma_{t+1})$ for $\Sigma_{t+1} = A\Sigma_t A^T + W$. We additionally evaluate resilience, $\eta \in [0,1]$, defined as the number of times the target is re-discovered by the sensor divided by the number of times the target is lost.

The left plot in Fig. 7.5 shows the average performances of both algorithms for $q \in \{0.02, 0.1, 0.2, 1.0, 2.0\}$ and $\nu_{\max} = 3.0$, and the right plot shows their average performances for $q = 0.2$ and $\nu_{\max} \in \{2.5, 2.75, 3.0, 3.25, 3.5\}$. Note that the maximum robot linear speed

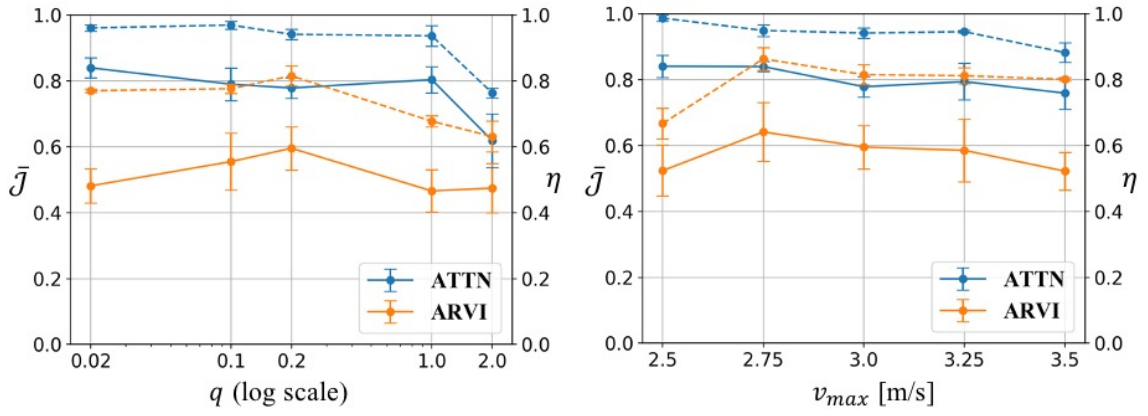

Figure 7.5: $\bar{J}$ (solid line) and η (dotted line) of ATTN (blue) and ARVI (yellow) in environments with different q and $\nu_{\max}$ values. The error bars represent the standard deviation across 10 episodes. The mean values are averaged values over different seeds and 10 episodes. Left: $\nu_{\max} = 3.0$[m/s] and $q \in \{0.02, 0.1, 0.2, 1.0, 2.0\}$. Right: $q = 0.2$, $\nu_{max} \in \{2.5, 2.75, 3.0, 3.25, 3.5\}$

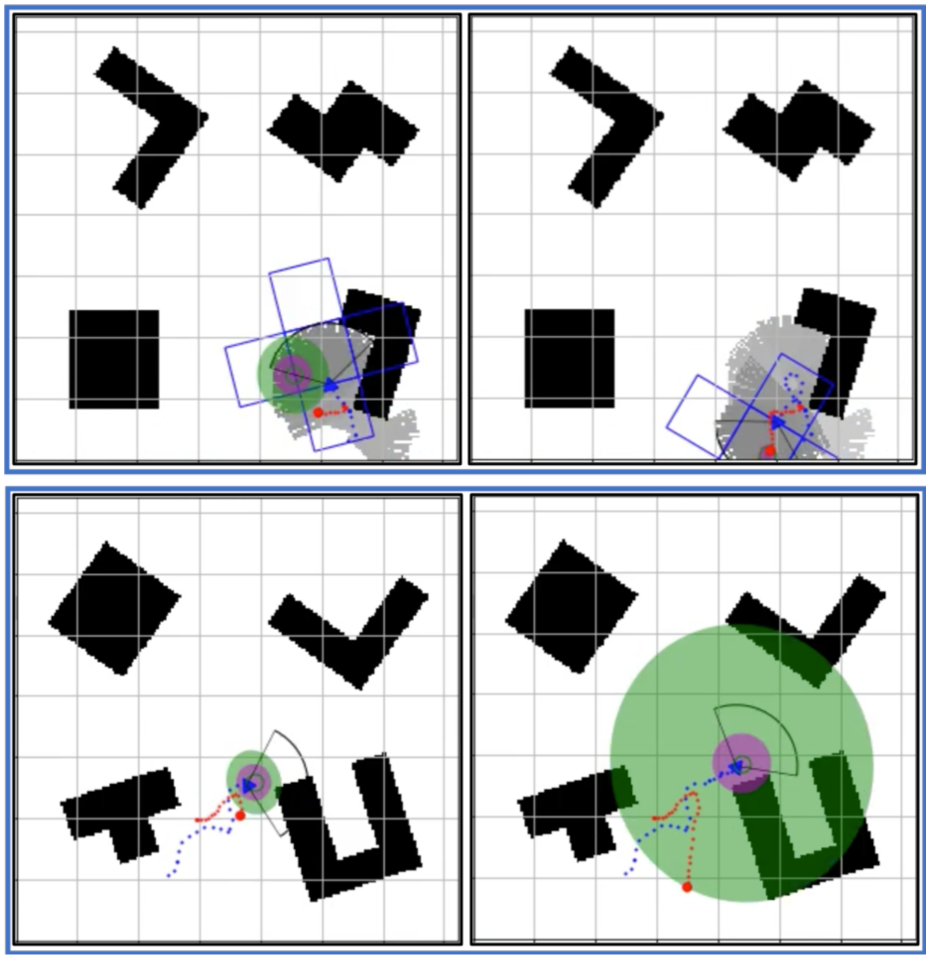

Figure 7.6: Examples of ATTN (top) and ARVI (bottom) when the robot loses the target. The left figures are a few steps after the robot loses the target, and the right figures are after 10 steps passed. The blue triangle is the robot and the red circle is the current target position. The blue and red dots are paths of the robot and target so far. The green circle indicates the belief position. The green and purple shaded circles represent the position and velocity uncertainty of the belief, respectively. The circular sector is the sensing area. In the top figures, visited cells are filled with gray color based on $\lambda_{c,t}$ in Eq.7.2, and the five blue squares indicate areas in the local map input.

is 3.0 [m/s], and thus, it is likely that the robot will often lose the target when $\nu_{\max} = 3.25$ and $\nu_{\max} = 3.5$. For both $\bar{J}$ and η, ATTN outperformed ARVI. As expected, $\bar{J}$ and η decrease as the target motion becomes noisier. Surprisingly, $\nu_{\max}$ did not have a significant impact on either $\bar{J}$ or η. This shows that the robot is capable of tracking a target that is faster than itself. The resilience tracks similarly with $\bar{J}$ indicating that the resilience is a significant factor of the performance. ATTN learns to explore near the belief when the target is not observed at the belief location and the uncertainty is high. In Fig. 7.6, the belief is

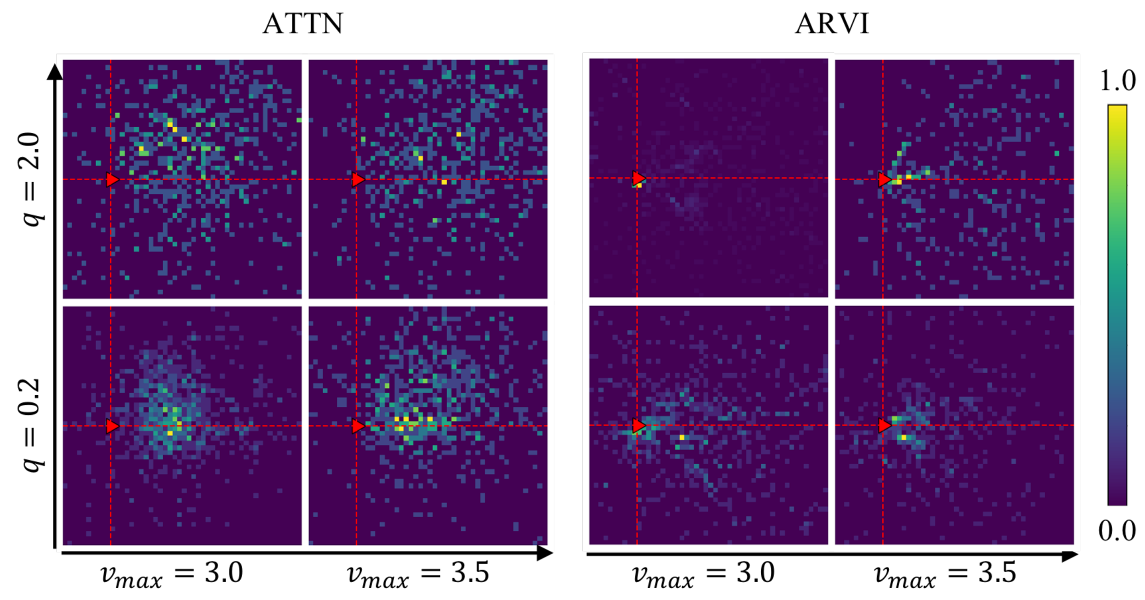

Figure 7.7: Density plots of belief positions in the agent frame during 10 episodes with different values for q and ν_{max}. The red triangle is the robot position (0.0, 0.0) and the horizontal and vertical red dotted lines are x and y axis of the agent frame, respectively. $x \in (-2.0, 8.0)$ and $y \in (-5.0, 5.0)$[m].

located to the left of the robot, but the robot chooses to turn right instead to explore the surrounding areas. After a few steps, the robot re-discovers the target. On the other hand, ARVI greedily follows the belief, failing to discover the target even though the uncertainty is relatively high and no observation is received. Additionally, ATTN does not track the belief as tightly as ARVI, and instead leaves some buffer space to provide maneuverability when the target quickly changes its direction. Fig. 7.7 is density plots of belief positions with respect to the robot during 10 episodes. The red dotted lines are the x-axis and y-axis of the robot frame, and the x-axis is the robot heading direction. Overall, ATTN results in scattered density plots while the belief positions in the ARVI plots are mostly concentrated in a few small areas.

For the navigation and discovery tasks, we measure a discovery rate to evaluate the performance of the algorithms. This discovery rate is defined as the number of episodes in which the target is found divided by the number of total episodes (=10). The faster the robot finds the target, the higher the resulting $\bar{J}$. However, $\bar{J}$ can vary significantly

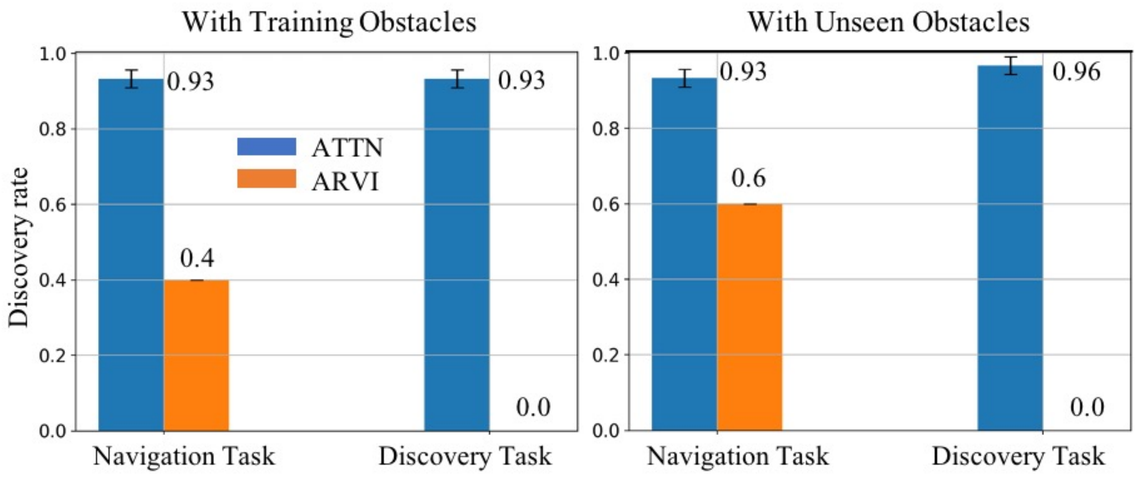

Figure 7.8: Performance evaluation for *Discovery* and *Navigation* tasks. Left: Evaluated in 10 different environments randomly generated by obstacles used in training. Right: Evaluated in 10 different environments randomly generated by unseen obstacles during training.

depending on the initial positions of the setup, and therefore is a poor measure for these tasks.

The results of the navigation task are depicted in the left figure of Fig.7.8. The ATTN policy finds the target 93% of the time on average (models trained with three different random seeds). On the other hand ARVI successfully finds the target only 40% of the time. We observe that ARVI fails to find a path to the target given T_{ARVI} when target is located far from the agent and is occluded by concave polygon obstacles.

Since ARVI use a forward simulation, it guarantees to avoid any collision with a perfect map information. While ATTN does not provide the guarantee for the collision avoidance, ATTN results in 0.4 times of collision attempts on average in the navigation task and no collision attempt in the discovery task.

In the discovery task, as defined earlier, the initial target is placed far from the initial belief location requiring the robot to explore to find the target. As shown in Fig. 7.8, ARVI fails to find the target in any of the episodes in this task while ATTN finds it 93% of the time on average. The four figures in Fig. 7.9 are density maps of areas that the robot's sensor has scanned in the global map. The top figures are examples of a single episode (left: ATTN, right: ARVI) and the bottom figures are examples of 20 episodes with random

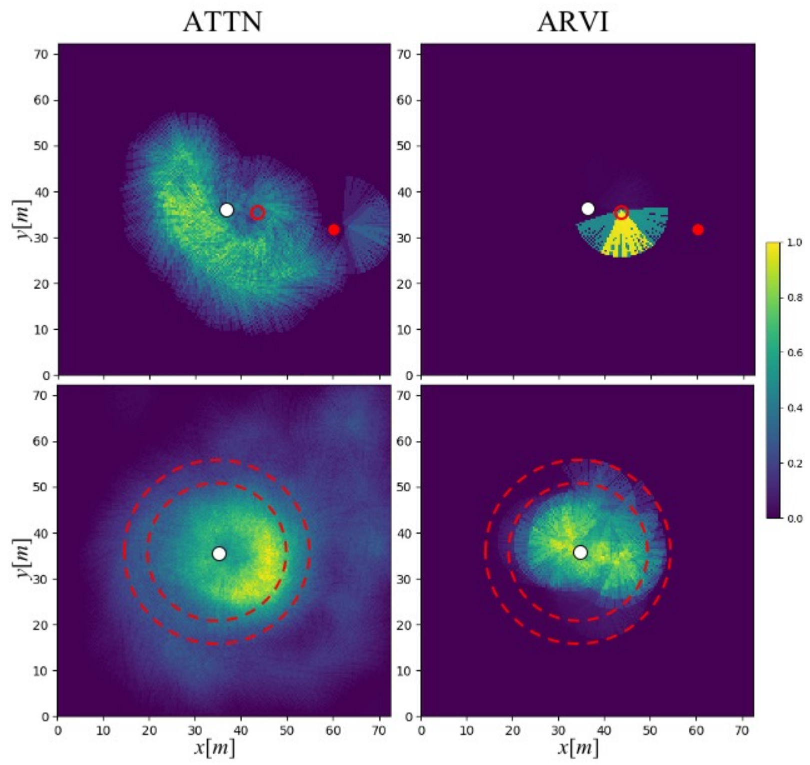

Figure 7.9: Density maps of scanned areas by the robot's sensor over single episode (top) and 20 episodes (bottom). The white circle are the initial position of the robot. The red filled circles in the top figures are the initial target positions, and the red hollow circles in the top figures are the initial belief positions. In the bottom figures, the target is randomly initialized in the area between the red dotted circles.

initialization. To solely evaluate the exploration capability, no obstacles are present in the map. The figures show that ATTN explores the global domain broadly when the target is not found near the belief while ARVI commits to the incorrect belief and leaves the global domain unexplored. This difference explains why ATTN achieves an 93% discovery rate while ARVI never finds the target. The computation time for planning of ATTN is 0.12 [sec] on average, which is much faster than the allocated computation used for ARVI (=0.5 [sec]) while ATNN has a significantly longer planning horizon.

7.4.5 Unseen Environments

Additional to the experiments described in the previous sections, we test the learned ATTN policy in environments with unseen obstacles during training (see Fig.7.3). The results in

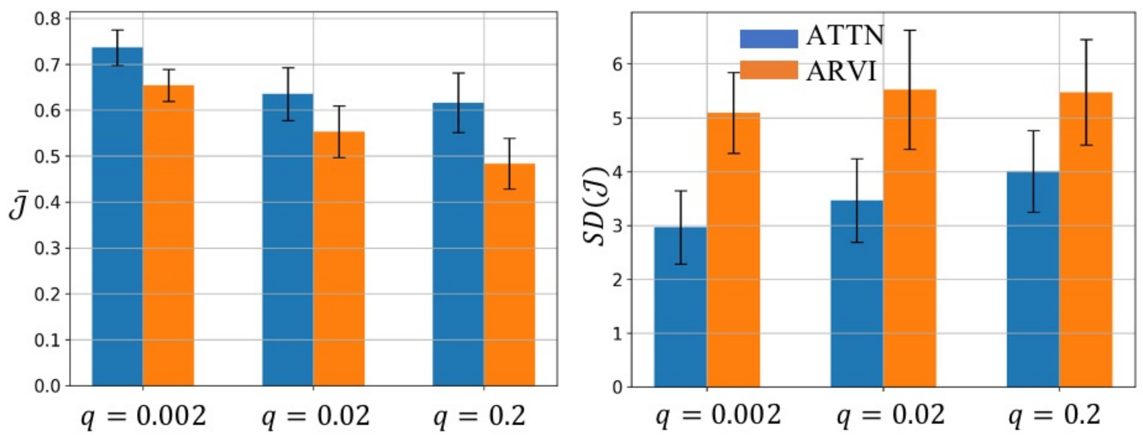

Figure 7.10: Performance evaluation on $N = 2$ targets. Left: The normalized mean of log determinant of belief covariances averaged over 10 episodes and the error bars indicate standard deviations. Right: Standard deviation of log determinant of belief covariances averaged over 10 episodes and the error bars indicate their standard deviations.

the right figure of Fig.7.8 shows that the discovery rates of ATTN in both tasks are 93% and 96%, similar to the results in the training environments. Likewise, ARVI results in 60% and 0% discovery rates. Additionally, ATTN shows 0.4 and 0.0 collision attempts on average for the navigation and discovery tasks, respectively. These results demonstrate that the learned policy performs well in unseen environments promoting the benefit of using local information. Note that ARVI is an online search-based planning algorithm without having a training stage, and therefore it is not affected by a new set of obstacles.

7.4.6 Results in Two-Target Domains

Unlike the single-target domains (where once the agent is nearby the target, the navigation and discovery abilities are not much needed), the agent may need to use all three abilities interchangeably throughout an episode or to use them at the same time. For instance, the agent must navigate from one target to another one when they are far apart. Moreover, while the agent is tracking one target, the other belief diverges from its corresponding target which requires the agent to explore to find the target. Lastly, the agent may explore to discover one target while tracking the other target when they both are around the agent's sensing range. Therefore, instead of evaluating the algorithms in the three subtasks, we evaluate the algorithms with an initialization similar to the in-sight task in the single-target evaluation

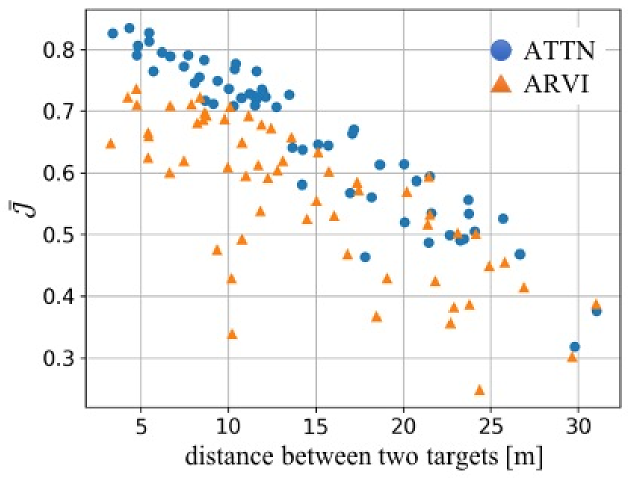

Figure 7.11: The relation between average distance between two targets during an episode and $\bar{J}$.

at different noise constants $q = \{0.002, 0.02, 0.2\}$ and $v_{\max} = 1.0$ [m/s].

The average normalized objective, $\bar{J}$, are presented in the left figure of Fig. 7.10. In all q values, ATTN constantly outperforms ARVI. The right figure shows the average standard deviation between $\bar{J}$ of two targets. A lower value for $SD(\bar{J})$ indicates that the agent tracks the targets while balancing the objective brought by both targets. The reason of the ARVI performance is similar to the one in the single target case. ARVI aggressively tracks the beliefs and it especially harms the performance in a multi-target tracking case since the agent is often required to travel between two targets and a belief diverges from its corresponding target while the agent is tracking another one.

Fig.7.11 shows how the performance of ATTN and ARVI in terms of $\bar{J}$ decreases as a distance between two targets increases. Although ATTN shows a higher performance than ARVI, the value drops in general which indicates the difficulty of the problem.

7.5 Discussion

7.5.1 Learning with Uncertainty

Although an RL method requires an extended training stage to obtain a policy, it provides flexibility in expanding a problem domain or changing system models. The objective of a problem is implicitly included in a reward function, and an optimal policy is learned without requiring knowledge on models. While we use partial knowledge on a target model and an observation model to update beliefs using a Kalman filter, we were able to use a target model that differs from the model known to the beliefs without violating any assumption or constraint required for an algorithm. Therefore, the target tracking scenarios considered in this article are more challenging in terms of stochasticity, uncertainty, and imperfect prior knowledge, compared to scenarios presented in previous works [2, 21, 57, 112]. During learning, the learning agent is exposed to diverse and challenging training samples and is able to *learn to track with uncertain beliefs*, *learn to navigate*, and *learn to explore.*

7.5.2 Stochasticity of Tasks

Active target tracking is not a trivial task for RL. The state information of a learning agent and an environment including targets and obstacles is high-dimensional, continuous, and stochastic. The recent advancement of deep RL have shown promising results in tasks where the RL state is continuous or high-dimensional – for example, raw screen image in video games or joint angles [92, 93, 113]. Yet, the deterministic dynamics of popular deep RL benchmark environments such as the Arcade Learning Environment (ALE) and Mujoco has raised concern in the scientific community as the successfully evaluated algorithms can fail when extended to new domains [85]. In most real-world problems, a transition function and/or a reward function are likely to be stochastic, resulting in challenges applying such approaches to more realistic problems. The active target tracking task is highly stochastic. Even in its simulation setting, the innate partial observability feature brings high stochasticity in the task. A subsequent RL state is determined not only by the current agent action but also the real yet partially observable or unknown target. Therefore, the same action can result in very different RL states as the belief can be drastically updated due to a new measurement if the corresponding target is around as opposed to no target is observed. It is shown in the previous work [63] that popular benchmark deep Q-learning methods, deep Q-network and double deep Q-network, failed to achieve optimal performance in relatively simple settings. Thus, it is important to note that the environment is stochastic and an algorithm capable in such a highly stochastic domain should be considered.

7.5.3 United Policy

One of the key contributions of this study is that we expand the problem domain and tackle three major capabilities with one united policy. One might argue that we could achieve similar performance by having three different algorithms for each capability and have an additional algorithm to heuristically decide which one to use at each step. This not only can burden the computational cost but can lose the interchangeable flexibility among different capabilities.

7.6 Summary

In this chapter, we present Active Target Tracking Network (ATTN), an RL method for active target tracking that learns a unified policy capable of three main tasks - *In-sight tracking*, *Navigation*, and *Discovery.* To demonstrate, we train an RL policy in a target tracking environment described in Sec.7.3 where a mobile robot is tasked with tracking mobile targets using noisy measurements from its on-board range-bearing sensor. The learned ATTN policy shows a robust performance for agile and anomalous target motions, despite the true target model differing from the target model in the agent belief. Moreover, the policy was able to navigate through obstacles with a long path to reach the target. Finally, it learns to explore surrounding areas and reach a target when its belief on the target is inaccurate, while the existing algorithm failed to do so in all test examples.

Chapter 8

Conclusion

Reinforcement learning has attracted much attention due to its roots in psychological and neuroscientific perspectives on animal decision-making. Unlike supervised learning, it does not require an external supervisor or labeled datasets, and thus it provides a promising framework for learning tasks where we do not have an optimal solution and/or must learn from interacting with an environment. Moreover, the recent achievements in deep reinforcement learning proved promising performance in complex domains of which previous reinforcement learning methods had been incapable. Despite the impressive progress, reinforcement learning is not yet practical in most robotics and autonomous systems. To improve the limitation of reinforcement learning, we must consider sample complexity, continual learning, partial observability, uncertainty, transfer learning, reward function design, along with several other topics. This dissertation is inspired by such needs and focuses on three of the challenges – sample complexity, uncertainty, and partial observability.

Chapter 3 presents a method of using reinforcement learning for learning stand-up motion of humanoid robots with sample complexity. There have been some deep reinforcement learning methods that learn stand-up motions of a simple humanoid in the Mujoco simulator [126]. Although the simulator uses the physics of a dynamic object, it is much simpler than a simulator for a real-robot. Therefore, such methods requiring millions of samples do not easily transfer to more realistic robotic simulations. Instead of using function approximators such as a neural network, we discretize the state and action spaces by a clustering method and apply the standard Q-learning. Furthermore, we utilize bi-symmetric features of humanoid robots to reduce the state and action space and accelerate the learning. A reinforcement learning policy is trained with a DarwIn-OP humanoid robot in an open-source 3D robot simulator and evaluated both in the simulation and on a physical robot. The learned policy demonstrates successful stand-up motions in various fallen positions surpassing the manually designed motion.

The second part of this dissertation (Chapter 4 and 5) addresses the uncertainty of learning parameters in Q-learning. We introduce a novel Bayesian Q-learning method – *Assumed Density Filtering Q-learning*. The overestimation of Q-learning has been addressed by many scholars and studies despite Q-learning being one of the most popular reinforcement learning algorithms. The issue is caused by the greedy behavior of the max operator in the value update. To improve the overestimation bias, some studies proposed new methods such as Double Q-learning. However, even though its estimation values are lower than that of Q-learning, it can suffer from an underestimation issue or have insufficient correction amounts. We instead provide a principled way of regularizing the greedy update based on the uncertainty of learning parameters. Unlike having point estimates on Q-values in the standard Q-learning, we suppose that the agent maintains a Gaussian belief distribution for each state and action pair. As opposed to distributional reinforcement learning, the goal is not about learning the underlying value distribution of an environment. Rather, the learning agent utilizes its belief distributions during learning, and therefore the uncertainty of the beliefs should approach 0.0 after a sufficient amount of time passed. To solve the non-linearity brought by the max operator in the Bellman optimality equation, we approximate the non-Gaussian likelihood to Gaussian using a variational inference method called assumed density filtering. We further approximate the resulting Gaussian mixture posterior distribution to a Gaussian distribution and enable an online Bayesian update. As a result, the computational complexity is only $\mathcal{O}(|A|^2)$ at each step which is significantly less than most Bayesian reinforcement learning algorithms. Moreover, it converges to Q-learning as the uncertainty of the beliefs approach zero. These features of ADFQ are demonstrated in both deterministic and stochastic domains outperforming Q-learning and an existing Bayesian Q-learning method and converges faster to optimal Q-values. These improvements are continued with a neural network function approximator. ADFQ with a neural network, called Deep ADFQ, surpasses DQN and Double DQN in continuous state domains (both deterministic and stochastic) as well as in several Atari 2600 games by a significant amount. Additionally, it shows that it improves some estimation bias of Q-learning and Double Q-learning. The improvements are more significant in stochastic domains and domains with a large number of actions. Finally, we utilize the uncertainty information in exploration through Thompson sampling, and it accelerates learning by efficiently explore the action spaces.

In the final part of this thesis (Chapter 6 and 7), we tackle a highly stochastic and partially observable environment with ADFQ utilizing the benefits discussed above. Active information acquisition is an essential task in robotics where an agent makes a sequence of decisions to gather information about targets of interests. Various approaches have been studied for this problem with different focuses. We particularly focus on cases where both the agent and the targets are mobile. We learn a unified reinforcement learning policy that can track targets near the agent with partial knowledge on a target model, navigate through different shapes of obstacles, and explore a global domain to discover targets when prior information is inaccurate. Since there is no direct observation of the targets, it is naturally a partially observable environment. Instead of approaching the problem as a POMDP, we formulate the problem as an MDP by explicitly including the uncertainty measures on target states as a part of the reinforcement learning state. The reward function is defined by a mutual information objective, and thus, a learned policy chooses an action that maximizes cumulative discounted future rewards over an effective time horizon. We first evaluate policies learned by ADFQ, DQN, and Double DQN in relatively simple environments similar to the ones used in previous works – a target model is mostly known to the agent belief, concave obstacles, slow targets with small randomness. Since the problem is highly stochastic and partially observable, DQN and Double DQN struggle to achieve near-optimal performance in less than 100 trajectories. On the other hand, ADFQ reaches or sometimes outperforms the existing search-based planning algorithm in all domains promising its potential benefits. Therefore, we further extend this approach to a much more complex and uncertain scenario. Particularly, we are interested in a target tracking problem where a target is agile and anomalous as well as the agent has only partial knowledge about the target. In order to track targets in practice, not only you need an ability to track the target near your sensing range, it is required to have navigation as well as exploration abilities. Each ability has been studied separately, but no method has been presented to do all three tasks with one policy. The presented method with ADFQ demonstrated its capabilities in all three tasks while a comparing algorithm often failed to discover targets or to keep tracking them.

In summary, we have explored how to utilize the uncertainty of learning parameters of an off-policy TD learning and its capabilities to be effectively applied in challenging real-world problems. We also discussed how we can use reinforcement learning for a task where acquiring a large number of samples is extremely demanding. Potential extensions and future works follows:

- As ADFQ is a direct Bayesian counterpart of Q-learning, many existing variation of Q-learning can be combined with the Bayesian approach. Note that the main idea of ADFQ is to provide a principled non-greedy value update and to replace the greedy max operator. Therefore, algorithms that do not use the max operator would not obtain the same contributions by being combined with ADFQ.

- Uncertainty propagation from state uncertainty to value uncertainty is another interesting future direction that can be extended from ADFQ. As studied in Chapter 6 and 7, a state often contains considerable amount of uncertainty in partially observable environments. Additionally, we can consider learning a state representation and its uncertainty using a unsupervised learning method such as variational autoencoder [70] and utilize the state uncertainty in the value update.

- Adversarial target tracking is an exciting extension of the active target tracking work.

We have observed that when target was not allowed to approach the agent within a certain range (therefore the target motion was not completely independent of an agent path), without any changes or additions, the agent learns to block the target so that it can easily keep the target in its sensing range. Similar to Generative Adversarial Network [41], we can train an intelligent target that adapts its behavior over time to avoid being tracked by the agent, and the agent should continuously improve its policy according to the updated target behavior. Such self-supervised reinforcement learning is an emerging field and has had an increasing interest [8].

- Another natural extension of the active target tracking work is multi-agent target tracking. As pointed out in Chapter 7, it is physically impossible for a single agent to track both targets while maximizing overall mutual information when targets are quickly diverging from each other. As the number of targets increases, either the required maximum speed of the agent should increase or targets should move slowly or within a certain range (e.g. a distance constraint among targets). Multi-agent approaches for target tracking problems have been studied recognizing such limitations of single-target tracking [3, 16, 79, 90, 112]. Separately, multi-agent reinforcement learning has been widely studied for various applications including robotics, telecommunications, and distributed control [15, 77, 98]. Combining advancements from these two fields of literature would present an interesting and more practical result in scalable target tracking problems.

Appendices

Appendix A

Mathematical Derivation of Posterior Distribution of Q-beliefs

A.1 Derivation of the Posterior Distribution of Q

In the section 4.4 of the main paper, we have shown that

$$\hat{p}_{Q_{s,a}}(q|\theta, r, s') = \frac{1}{Z} p_{V_{s'}}\left(\frac{q-r}{\gamma}\middle| q, s', \theta\right) p_{Q_{s,a}}(q|\theta)$$

where Z is a normalization constant. Applying the distributions over $V_{s'}$ and $Q_{s,a}$, the posterior is derived as:

$$\begin{aligned}
\hat{p}_{Q_{s,a}}(q) &= \frac{1}{Z}\sum_{b\in\mathcal{A}} \frac{1}{\sigma_{s',b}}\phi\left(\frac{q-(r+\gamma\mu_{s',b})}{\gamma\sigma_{s',b}}\right) \prod_{b'\neq b, b'\in\mathcal{A}} \Phi\left(\frac{q-(r+\gamma\mu_{s',b'})}{\gamma\sigma_{s',b'}}\right) \frac{1}{\sigma_{s,a}}\phi\left(\frac{q-\mu_{s,a}}{\sigma_{s,a}}\right) \\
&= \frac{1}{Z\sqrt{2\pi}\sigma_{s,a}} \sum_{b\in\mathcal{A}} \frac{1}{\sigma_{s',b}} \exp\left\{-\frac{1}{2}\frac{(\mu_{s,a}-(r+\gamma\mu_{s',b}))^2}{\sigma_{s,a}^2+\gamma^2\sigma_{s',b}^2}\right\} \phi\left(\frac{q-\bar{\mu}_{\tau,b}}{\bar{\sigma}_{\tau,b}}\right) \\
&\qquad \times \prod_{b'\neq b, b'\in\mathcal{A}} \Phi\left(\frac{q-(r+\gamma\mu_{s',b'})}{\gamma\sigma_{s',b'}}\right) \\
&= \frac{1}{Z}\sum_{b\in\mathcal{A}} \frac{c_{\tau,b}}{\bar{\sigma}_{\tau,b}}\phi\left(\frac{q-\bar{\mu}_{\tau,b}}{\bar{\sigma}_{\tau,b}}\right) \prod_{b'\neq b, b'\in\mathcal{A}} \Phi\left(\frac{q-(r+\gamma\mu_{s',b'})}{\gamma\sigma_{s',b'}}\right)
\end{aligned}$$

where Z is a normalization constant and

$$c_{\tau,b} = \frac{1}{\sqrt{\sigma_{s,a}^2 + \gamma^2\sigma_{s',b}^2}}\phi\left(\frac{(r+\gamma\mu_{s',b}) - \mu_{s,a}}{\sqrt{\sigma_{s,a}^2 + \gamma^2\sigma_{s',b}^2}}\right)$$

$$\bar{\mu}_{\tau,b} = \bar{\sigma}_{\tau,b}^2\Big(\frac{\mu_{s,a}}{\sigma_{s,a}^2} + \frac{r+\gamma\mu_{s',b}}{\gamma^2\sigma_{s',b}^2}\Big) \qquad \bar{\sigma}_{\tau,b}^2 = \Big(\frac{1}{\sigma_{s,a}^2} + \frac{1}{\gamma^2\sigma_{s',b}^2}\Big)^{-1}$$

A.2 Mean and Variance of the Posterior Distribution of Q

A.2.1 Moment Generating Function

The mean and variance of the posterior distribution (Eq.4.2) can be analytically found when $|\mathcal{A}| = 2$. Consider a random variable $X_M = \max_{1\leq k\leq N} X_k$ which density function (Eq.4.1) has a similar form to the posterior distribution. The moment generating function of X_M is:

$$\begin{aligned} M(t) &= \int_{-\infty}^{\infty} e^{tx}\sum_i \frac{1}{\sigma_i}\phi\Big(\frac{x-\mu_i}{\sigma_i}\Big)\prod_{i\neq j}\Phi\Big(\frac{x-\mu_j}{\sigma_j}\Big)dx \\ &= \sum_i \eta_i(t)\int_{-\infty}^{\infty}\frac{1}{\sigma_i}\phi\left(\frac{x-\mu_i'}{\sigma_i}\right)\prod_{i\neq j}\Phi\left(\frac{x-\mu_j}{\sigma_j}\right)dx \end{aligned}$$

where

$$\eta_i(t) = \exp\left\{\mu_i t + \frac{t^2\sigma_i^2}{2}\right\} \qquad \text{and} \qquad \mu_i' = \mu_i + t\sigma_i^2$$

When $N = 2$,

$$M(t) = \int_{-\infty}^{\infty} e^{tx}\left(\frac{1}{\sigma_1}\phi\Big(\frac{x-\mu_1}{\sigma_1}\Big)\Phi\Big(\frac{x-\mu_2}{\sigma_2}\Big) + \frac{1}{\sigma_2}\phi\Big(\frac{x-\mu_2}{\sigma_2}\Big)\Phi\Big(\frac{x-\mu_1}{\sigma_1}\Big)\right)dx \tag{A.1}$$

Since the two terms are symmetric, let $M(t) = M_1(t) + M_2(t)$ and differentiate each term with respect to μ_2 and μ_1, respectively. For the first term,

$$\begin{aligned}
\frac{\partial M_1(t)}{\partial \mu_2} &= -\frac{\eta_1(t)}{\sigma_1\sigma_2}\int_{-\infty}^{\infty}\phi\Big(\frac{x-\mu_1'}{\sigma_1}\Big)\phi\Big(\frac{x-\mu_2}{\sigma_2}\Big)dx \\
&= -\frac{\eta_1(t)\sigma_{12}}{\sqrt{2\pi}\sigma_1\sigma_2}\exp\left\{-\frac{1}{2}\frac{(\mu_1'-\mu_2)^2}{\sigma_1^2+\sigma_2^2}\right\}\int_{-\infty}^{\infty}\frac{1}{\sigma_{12}}\phi\left(\frac{x-\mu_{12}}{\sigma_{12}}\right)dx \\
&= -\frac{\eta_1(t)\sigma_{12}}{\sqrt{2\pi}\sigma_1\sigma_2}\exp\left\{-\frac{1}{2}\frac{(\mu_1'-\mu_2)^2}{\sigma_1^2+\sigma_2^2}\right\} \qquad \text{(A.2)}
\end{aligned}$$

where

$$\mu_{12} = \sigma_{12}^2\left(\frac{\mu_1'}{\sigma_1^2}+\frac{\mu_2}{\sigma_2^2}\right) \qquad\qquad \frac{1}{\sigma_{12}^2} = \frac{1}{\sigma_1^2}+\frac{1}{\sigma_2^2}$$

Then, we integrate Eq.A.2 with respect to μ_2,

$$\begin{aligned}
M_1(t) &= \int \frac{\partial M_1(t)}{\partial \mu_2}d\mu_2 \\
&= -\frac{\eta_1(t)\sigma_{12}}{\sigma_1\sigma_2}\sqrt{\sigma_1^2+\sigma_2^2}\int\frac{1}{\sqrt{2\pi(\sigma_1^2+\sigma_2^2)}}\exp\left\{-\frac{(\mu_1'-\mu_2)^2}{2(\sigma_1^2+\sigma_2^2)}d\mu_2\right\} \\
&= \eta_1(t)\Phi\left(\frac{\mu_1'-\mu_2}{\sqrt{\sigma_1^2+\sigma_2^2}}\right) \qquad \text{(A.3)}
\end{aligned}$$

A.2.2 Moments of the Posterior Distribution

We apply the result in Eq.A.3 to the posterior distribution by replacing the variables in $M_1(t)$ as:

$$\mu_1 \to \bar{\mu}_{\tau,1} \quad \sigma_1 \to \bar{\sigma}_{\tau,1} \quad \mu_2 \to r+\gamma\mu_2 \quad \sigma_2 \to \gamma\sigma_2$$

and replacing the variables in $M_2(t)$ similarly. Then, we obtain the normalizing factor:

$$Z = c_{\tau,1}\Phi_{\tau,1} + c_{\tau,2}\Phi_{\tau,2} \qquad \text{(A.4)}$$

where we define the following notations for simplicity:

$$\Phi_{\tau,1} \equiv \Phi\left(\frac{\bar{\mu}_{\tau,1} - (r + \gamma\mu_{s',2})}{\sqrt{\bar{\sigma}^2_{\tau,1} + \gamma^2\sigma^2_{s',2}}}\right), \qquad \phi_{\tau,1} \equiv \frac{1}{\sqrt{\bar{\sigma}^2_{\tau,1} + \gamma^2\sigma^2_{s',2}}}\phi\left(\frac{\bar{\mu}_{\tau,1} - (r + \gamma\mu_{s',2})}{\sqrt{\bar{\sigma}^2_{\tau,1} + \gamma^2\sigma^2_{s',2}}}\right)$$

and $\Phi_{\tau,2}$ and $\phi_{\tau,2}$ are also similarly defined. The exact mean of the posterior distribution is derived by solving the first derivative of the moment generating function with respect to t at $t = 0$:

$$M'_1(t) = c_{\tau,1}\eta'_1(t)\Phi\left(\frac{\bar{\mu}'_{\tau,1} - (r + \gamma\mu_{s',2})}{\sqrt{\bar{\sigma}^2_{\tau,1} + \gamma^2\sigma^2_{s',2}}}\right) + c_{\tau,1}\eta_1(t)\frac{\bar{\sigma}^2_{\tau,1}}{\sqrt{\bar{\sigma}^2_{\tau,1} + \gamma^2\sigma^2_{s',2}}}\phi\left(\frac{\bar{\mu}'_{\tau,1} - (r + \gamma\mu_{s',2})}{\sqrt{\bar{\sigma}^2_{\tau,1} + \gamma^2\sigma^2_{s',2}}}\right)$$

$$\begin{aligned}\mathbb{E}_{q\sim\hat{p}_{Q_{s,a}}(\cdot)}[q] &= \frac{1}{Z}\left(M'_1(t=0) + M'_2(t=0)\right)\\ &= \frac{c_{\tau,1}}{Z}\left(\bar{\mu}_{\tau,1}\Phi_{\tau,1} + \bar{\sigma}^2_{\tau,1}\phi_{\tau,1}\right) + \frac{c_{\tau,2}}{Z}\left(\bar{\mu}_{\tau,2}\Phi_{\tau,2} + \bar{\sigma}^2_{\tau,2}\phi_{\tau,2}\right) \qquad \text{(A.5)}\end{aligned}$$

The variance of the posterior is also derived by solving the second derivative of the moment generating function:

$$\begin{aligned}M''_1(t) = c_{\tau,1}\eta''_1(t)\Phi\left(\frac{\bar{\mu}'_{\tau,1} - (r + \gamma\mu_{s',2})}{\sqrt{\bar{\sigma}^2_{\tau,1} + \gamma^2\sigma^2_{s',2}}}\right) + 2c_{\tau,1}\eta'_1(t)\frac{\bar{\sigma}^2_{\tau,1}}{\sqrt{\bar{\sigma}^2_{\tau,1} + \gamma^2\sigma^2_{s',2}}}\phi\left(\frac{\bar{\mu}'_{\tau,1} - (r + \gamma\mu_{s',2})}{\sqrt{\bar{\sigma}^2_{\tau,1} + \gamma^2\sigma^2_{s',2}}}\right)\\ + c_{\tau,1}\eta_1(t)\left(-\frac{\bar{\mu}'_{\tau,1} - (r + \gamma\mu_{s',2})}{(\bar{\sigma}^2_{\tau,1} + \gamma^2\sigma^2_{s',2})\bar{\sigma}^{-2}_{\tau,1}}\right)\frac{\bar{\sigma}^2_{\tau,1}}{\sqrt{\bar{\sigma}^2_{\tau,1} + \gamma^2\sigma^2_{s',2}}}\phi\left(\frac{\bar{\mu}'_{\tau,1} - (r + \gamma\mu_{s',2})}{\sqrt{\bar{\sigma}^2_{\tau,1} + \gamma^2\sigma^2_{s',2}}}\right)\end{aligned}$$

Thus, the second moment is:

$$\begin{aligned}\mathbb{E}_{q\sim\hat{p}_{Q_{s,a}}(\cdot)}[q^2] = \frac{c_{\tau,1}}{Z}\left((\bar{\mu}^2_{\tau,1} + \bar{\sigma}^2_{\tau,1})\Phi_{\tau,1} + 2\bar{\mu}_{\tau,1}\bar{\sigma}^2_{\tau,1}\phi_{\tau,1} - \frac{\bar{\sigma}^4_{\tau,1}}{\bar{\sigma}^2_{\tau,1} + \gamma^2\sigma^2_{s',2}}(\bar{\mu}_{\tau,1} - (r + \gamma\mu_{s',2}))\phi_{\tau,1}\right)\\ \frac{c_{\tau,2}}{Z}\left((\bar{\mu}^2_{\tau,2} + \bar{\sigma}^2_{\tau,2})\Phi_{\tau,2} + 2\bar{\mu}_{\tau,2}\bar{\sigma}^2_{\tau,2}\phi_{\tau,2} - \frac{\bar{\sigma}^4_{\tau,2}}{\bar{\sigma}^2_{\tau,2} + \gamma^2\sigma^2_{s',1}}(\bar{\mu}_{\tau,2} - (r + \gamma\mu_{s',1}))\phi_{\tau,2}\right) \qquad \text{(A.6)}\end{aligned}$$

and the variance is $\mathbb{E}_{q\sim\hat{p}_{Q_{s,a}}(\cdot)}[q^2] - (\mathbb{E}_{q\sim\hat{p}_{Q_{s,a}}(\cdot)}[q])^2$.

Appendix B

Q-beliefs with Gaussian White Noise

In order to incorporate stochasticity of an MDP, we add small Gaussian white noise to the likelihood, $r + \gamma V_{s'} + W$ where $W \sim \mathcal{N}(0, \sigma_w^2)$, and the likelihood distribution is obtained by solving the following integral:

$$\begin{aligned} p(r+\gamma V_{s'}|q,\theta) &= \int_{-\infty}^{\infty} \sum_{b\in\mathcal{A}} \frac{1}{\sigma_{s',b}} \phi\left(\frac{w-(q-(r+\gamma\mu_{s',b}))}{\gamma\sigma_{s',b}}\right) \\ &\qquad \times \prod_{b'\neq b} \Phi\left(-\frac{w-(q-(r+\gamma\mu_{s',b'}))}{\gamma\sigma_{s',b'}}\right) \frac{1}{\sigma_w}\phi\left(\frac{w}{\sigma_w}\right) dw \\ &= \int_{-\infty}^{\infty} \sum_{b\in\mathcal{A}} \frac{l_b}{\bar{v}_b}\phi\left(\frac{w-\bar{w}_b}{\bar{v}_b}\right) \prod_{b'\neq b}\left(1-\Phi\left(\frac{w-(q-r-\gamma\mu_{s',b'})}{\gamma\sigma_{s',b'}}\right)\right) dw \end{aligned} \tag{B.1}$$

where

$$l_b = \frac{1}{\sqrt{\sigma_w^2+\gamma^2\sigma_{s',b}^2}}\phi\left(\frac{q-(r+\gamma\mu_{s',b})}{\sqrt{\sigma_w^2+\gamma^2\sigma_{s',b}^2}}\right) \tag{B.2}$$

$$\bar{w}_b = \bar{v}_b^2\left(\frac{q-(r+\gamma\mu_{s',b})}{\gamma^2\sigma_{s',b}^2}\right) \qquad \frac{1}{\bar{v}_b^2} = \frac{1}{\gamma^2\sigma_{s',b}^2} + \frac{1}{\sigma_w^2} \tag{B.3}$$

B.1 Expected Likelihood for $|\mathcal{A}| = 2$

The distribution inside the integral in Eq.4.6 has a similar form with the posterior distribution Eq.4.2. As mentioned above, a closed form solution for its integral is not available when $|\mathcal{A}| > 2$. Therefore, we derive an analytic solution of the expected likelihood when $|\mathcal{A}| = 2$

and approximate to a simpler form so that it can be generalized to an arbitrary number of actions.

Using Eq.A.3 for finding the zeroth moment, we obtain:

$$p(r+\gamma V_{s'}|q,\theta) = l_1\Phi\left(-\frac{\bar{w}_1-(q-(r+\gamma\mu_{s',2}))}{\sqrt{\bar{v}_1^2+\gamma^2\sigma_{s',2}^2}}\right) + l_2\Phi\left(-\frac{\bar{w}_2-(q-(r+\gamma\mu_{s',1}))}{\sqrt{\bar{v}_2^2+\gamma^2\sigma_{s',1}^2}}\right) \tag{B.4}$$

Inside the CDF term is a function of q:

$$-\frac{\bar{w}_1-(q-(r+\gamma\mu_{s',2}))}{\sqrt{\bar{v}_1^2+\gamma^2\sigma_{s',2}^2}} = \frac{1}{\sqrt{\bar{v}_1^2+\gamma^2\sigma_{s',2}^2}}\left(\left(1-\frac{\bar{v}_1^2}{\gamma^2\sigma_{s',1}^2}\right)q-\left(r+\gamma\mu_{s',2}-\frac{\bar{v}_1^2}{\gamma^2\sigma_{s',1}^2}(r+\gamma\mu_{s',1})\right)\right)$$

We define

$$\mu_2^w \equiv \left(1-\frac{\bar{v}_1^2}{\gamma^2\sigma_{s',1}^2}\right)^{-1}\left(r+\gamma\mu_{s',2}-\frac{\bar{v}_1^2}{\gamma^2\sigma_{s',1}^2}(r+\gamma\mu_{s',1})\right)$$

$$\sigma_2^w \equiv \left(1-\frac{\bar{v}_1^2}{\gamma^2\sigma_{s',1}^2}\right)^{-1}\sqrt{\bar{v}_1^2+\gamma^2\sigma_{s',2}^2}$$

and express the likelihood distribution Eq.B.4 as:

$$p(r+\gamma V_{s'}|q,\theta) = l_1\Phi\left(\frac{q-\mu_2^w}{\sigma_2^w}\right) + l_2\Phi\left(\frac{q-\mu_1^w}{\sigma_1^w}\right) \tag{B.5}$$

Then, we can find the solutions of the posterior mean and variance for $\sigma_w > 0$ when $|\mathcal{A}| = 2$ by replacing $r+\gamma\mu_{s',2}$ and $\gamma\sigma_{s',2}$ with μ_2^w and σ_2^w, respectively in Eq.A.5 and Eq.A.6.

B.2 Asymptotic Limits

In one asymptotic limit of $\sigma_w/\sigma_{s',b} \to 0$,

$$\lim_{\sigma_w/\sigma_{s',b}\to 0} \bar{v}_b^2+\gamma^2\sigma_{s',b'}^2 = \lim_{\sigma_w/\sigma_{s',b}\to 0} \frac{\gamma^2\sigma_{s',b}^2\sigma_w^2}{\gamma^2\sigma_{s',b}^2+\sigma_w^2}+\gamma^2\sigma_{s',b'}^2 = \gamma^2\sigma_{s',b'}^2$$

$$\lim_{\sigma_w/\sigma_{s',b}\to 0} \frac{\bar{v}_b^2}{\gamma^2\sigma_{s',b}^2} = \lim_{\sigma_w/\sigma_{s',b}\to 0} \frac{\sigma_w^2}{\gamma^2\sigma_{s',b}^2+\sigma_w^2} = 0$$

and therefore,

$$\lim_{\sigma_w/\sigma_{s',b}\to 0} \Phi\left(-\frac{\bar{w}_b - (q - (r + \gamma\mu_{s',b'}))}{\sqrt{\bar{v}_b^2 + \gamma^2\sigma_{s',b'}^2}}\right) = \Phi\left(\frac{q - (r + \gamma\mu_{s',b'})}{\gamma\sigma_{s',b'}}\right) \tag{B.6}$$

and the likelihood distribution becomes

$$\sum_{b\in\{1,2\}} \frac{1}{\sqrt{\sigma_w^2 + \gamma^2\sigma_{s',b}^2}}\phi\left(\frac{q - (r + \gamma\mu_{s',b})}{\sqrt{\sigma_w^2 + \gamma^2\sigma_{s',b}^2}}\right)\Phi\left(\frac{q - (r + \gamma\mu_{s',b'})}{\gamma\sigma_{s',b'}}\right) \tag{B.7}$$

Therefore, the posterior distribution derived from this likelihood has the same form with Eq.4.2 but it uses $\gamma^2\sigma_{s',b}^2 + \sigma_w^2$ instead of $\gamma^2\sigma_{s',b}^2$ in $c_{\tau,b}$, $\bar{\mu}_{\tau,b}$, and $\bar{\sigma}_{\tau,b}$:

$$c_{\tau,b} = \frac{1}{\sqrt{\sigma_{s,a}^2 + \gamma^2\sigma_{s',b}^2 + \sigma_w^2}}\phi\left(\frac{(r + \gamma\mu_{s',b}) - \mu_{s,a}}{\sqrt{\sigma_{s,a}^2 + \gamma^2\sigma_{s',b}^2 + \sigma_w^2}}\right) \tag{B.8}$$

$$\bar{\mu}_{\tau,b} = \bar{\sigma}_{\tau,b}^2\Big(\frac{\mu_{s,a}}{\sigma_{s,a}^2} + \frac{r + \gamma\mu_{s',b}}{\gamma^2\sigma_{s',b}^2 + \sigma_w^2}\Big) \qquad \bar{\sigma}_{\tau,b}^2 = \Big(\frac{1}{\sigma_{s,a}^2} + \frac{1}{\gamma^2\sigma_{s',b}^2 + \sigma_w^2}\Big)^{-1} \tag{B.9}$$

It is identical to the posterior of the case when $\sigma_w = 0$.

In the other asymptotic limit,

$$\lim_{\sigma_{s',b}/\sigma_w\to 0} \bar{v}_b^2 + \gamma^2\sigma_{s',b'}^2 = \gamma^2\sigma_{s',b}^2 + \gamma^2\sigma_{s',b'}^2 \quad \text{and} \quad \lim_{\sigma_{s',b}/\sigma_w\to 0} \frac{\bar{v}_b^2}{\gamma^2\sigma_{s',b}^2} = 1$$

Since we set σ_w as a small number, $\sigma_{s',b}/\sigma_w \to 0$ infers $\sigma_{s',b} \to 0$, and therefore, the likelihood distribution becomes Gaussian :

$$\begin{aligned}\lim_{\sigma_{s',b}/\sigma_w\to 0} p(r + \gamma V_{s'}|q,\theta) &= \lim_{\sigma_{s',b}/\sigma_w\to 0} l_1\Phi\left(\frac{\mu_{s',1} - \mu_{s',2}}{\sqrt{\gamma^2\sigma_{s',1}^2 + \gamma^2\sigma_{s',2}^2}}\right) + l_2\Phi\left(\frac{\mu_{s',2} - \mu_{s',1}}{\sqrt{\gamma^2\sigma_{s',1}^2 + \gamma^2\sigma_{s',2}^2}}\right) \\ &= \frac{1}{\sqrt{\sigma_w^2 + \gamma^2\sigma_{s',b^+}^2}}\phi\left(\frac{q - (r + \gamma\mu_{s',b^+})}{\sqrt{\sigma_w^2 + \gamma^2\sigma_{s',b^+}^2}}\right)\end{aligned} \tag{B.10}$$

where $b^+ = \text{argmax}_{b\in\{1,2\}}\,\mu_{s',b}$. Therefore, the posterior distribution becomes Gaussian with mean at $\bar{\mu}_{\tau,b^+}$ and variance at $\bar{\sigma}_{\tau,b^+}^2$ in Eq.B.9.

B.3 Approximate Likelihood

In order to have closed-form expressions for the ADFQ update, we extend the asymptotic result for $|\mathcal{A}| = 2$ presented in the previous section to the general case ($|\mathcal{A}| = n$ for $n \in \mathbb{N}$) with an assumption of $\sigma_w \ll \sigma_{s',b}\ \forall b \in \mathcal{A}$. Therefore, the approximate likelihood is:

$$p(r + \gamma V_{s'}|q, \theta) = \sum_{b\in\mathcal{A}} \frac{\gamma}{\sqrt{\gamma^2\sigma_{s',b}^2 + \sigma_w^2}} \phi\left(\frac{q - (r + \gamma\mu_{s',b})}{\sqrt{\gamma^2\sigma_{s',b}^2 + \sigma_w^2}}\right) \prod_{b'\neq b, b'\in\mathcal{A}} \Phi\left(\frac{q - (r + \gamma\mu_{s',b'})}{\gamma\sigma_{s',b'}}\right)$$

Then, the posterior distribution is derived as:

$$\begin{aligned}
\hat{p}_{Q_{s,a}}(q) &= \frac{1}{Z}\sum_{b\in\mathcal{A}} \frac{\gamma}{\sqrt{\gamma^2\sigma_{s',b}^2 + \sigma_w^2}} \phi\left(\frac{q - (r + \gamma\mu_{s',b})}{\sqrt{\gamma^2\sigma_{s',b}^2 + \sigma_w^2}}\right) \\
&\qquad \times \prod_{b'\neq b, b'\in\mathcal{A}} \Phi\left(\frac{q - (r + \gamma\mu_{s',b'})}{\gamma\sigma_{s',b'}}\right) \frac{1}{\sigma_{s,a}}\phi\left(\frac{q - \mu_{s,a}}{\sigma_{s,a}}\right) \\
&= \frac{1}{Z\sqrt{2\pi}\sigma_{s,a}} \sum_{b\in\mathcal{A}} \frac{\gamma}{\sqrt{\gamma^2\sigma_{s',b}^2 + \sigma_\epsilon^2}} \exp\left\{-\frac{1}{2}\frac{(\mu_{s,a} - (r + \gamma\mu_{s',b}))^2}{\sigma_{s,a}^2 + \gamma^2\sigma_{s',b}^2 + \sigma_w^2}\right\} \\
&\qquad \times \phi\left(\frac{q - \bar{\mu}_{\tau,b}}{\bar{\sigma}_{\tau,b}}\right) \prod_{b'\neq b, b'\in\mathcal{A}} \Phi\left(\frac{q - (r + \gamma\mu_{s',b'})}{\gamma\sigma_{s',b'}}\right) \\
&= \frac{1}{Z}\sum_{b\in\mathcal{A}} \frac{c_{\tau,b}}{\bar{\sigma}_{\tau,b}} \phi\left(\frac{q - \bar{\mu}_{\tau,b}}{\bar{\sigma}_{\tau,b}}\right) \prod_{b'\neq b, b'\in\mathcal{A}} \Phi\left(\frac{q - (r + \gamma\mu_{s',b'})}{\gamma\sigma_{s',b'}}\right)
\end{aligned}$$

where Z is a normalization constant and

$$c_{\tau,b} = \frac{1}{\sqrt{\sigma_{s,a}^2 + \gamma^2\sigma_{s',b}^2 + \sigma_w^2}} \phi\left(\frac{(r + \gamma\mu_{s',b}) - \mu_{s,a}}{\sqrt{\sigma_{s,a}^2 + \gamma^2\sigma_{s',b}^2 + \sigma_w^2}}\right)$$

$$\bar{\mu}_{\tau,b} = \bar{\sigma}_{\tau,b}^2\Big(\frac{\mu_{s,a}}{\sigma_{s,a}^2} + \frac{r + \gamma\mu_{s',b}}{\gamma^2\sigma_{s',b}^2 + \sigma_w^2}\Big) \qquad \bar{\sigma}_{\tau,b}^2 = \Big(\frac{1}{\sigma_{s,a}^2} + \frac{1}{\gamma^2\sigma_{s',b}^2 + \sigma_w^2}\Big)^{-1}$$

Appendix C

Proofs

C.1 Lemma 1

Lemma 1. *Let X be a random variable following a normal distribution, $\mathcal{N}(\mu, \sigma^2)$. Then we have:*

$$\lim_{\sigma \to 0} \left[\Phi\left(\frac{x-\mu}{\sigma}\right) - \exp\left\{ -\frac{1}{2} \left[-\frac{x-\mu}{\sigma} \right]_+^2 \right\} \right] = 0 \tag{C.1}$$

where $[x]_+ = \max(0, x)$ is the ReLU *nonlinearity.*

Proof.

$$\lim_{\sigma \to 0} \frac{x-\mu}{\sigma} = -\infty$$

Let's define $y \equiv (x-\mu)/\sigma$,

$$\begin{aligned}
\Phi(y<0) &= \int_{-\infty}^{y} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}t^2} dt \\
&= \int_{-\infty}^{0} \frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{1}{2}(y+t')^2\right\} dt' \\
&= \frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{1}{2}y^2\right\} \int_{-\infty}^{0} \exp\left\{-\left(y+\frac{t'}{2}\right)t'\right\} dt' \\
&= \frac{1}{2} \exp\left\{-\frac{1}{2}y^2\right\} \int_{-\infty}^{0} \exp\left\{-yt'\right\} dt' \\
&= -\frac{1}{2y} \exp\left\{-\frac{1}{2}y^2\right\} \\
&= \frac{1}{2|y|} \exp\left\{-\frac{1}{2}y^2\right\}
\end{aligned}$$

$$\lim_{y<0, y\to-\infty} \left[\Phi(y) - \exp\left\{-\frac{1}{2}y^2\right\}\right] = \lim_{y<0, y\to-\infty} \left(1 - \frac{1}{2|y|}\right) \exp\left\{-\frac{1}{2}y^2\right\} = 0$$

For $x \geq \mu$ and $y \geq 0$ and

$$\lim_{\sigma\to 0} \frac{x-\mu}{\sigma} = \infty$$

$$\lim_{y\to\infty} \Phi(y) = \lim_{y\to\infty} \int_{-\infty}^{y} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}t^2} dt = 1 = e^0$$

Therefore,

$$\lim_{\sigma\to 0} \left[\Phi\left(\frac{x-\mu}{\sigma}\right) - \exp\left\{-\frac{1}{2}\left[-\frac{x-\mu}{\sigma}\right]_+^2\right\}\right] = 0$$

□

C.2 Theorem 1

Theorem 1. *Suppose that the mean and variance of $Q_{s,a}$ $\forall s \in \mathcal{S}, \forall a \in \mathcal{A}$ are iteratively updated by the mean and variance of $\hat{p}_{Q_{s,a}}$ after observing r and s' at every step. When $|\mathcal{A}| = 2$, the update rule of the means is equivalent to the Q-learning update if all state-*

action pairs are visited infinitely often and the variances approach 0. In other words, at the kth update on $\mu_{s,a}$:

$$\lim_{k\to\infty,\{\sigma\}\to 0} \mu_{s,a;k+1} = (1-\alpha_{\tau;k})\,\mu_{s,a;k} + \alpha_{\tau;k}\big(r + \gamma \max_{b\in\mathcal{A}} \mu_{s',b;k}\big)$$

where $\alpha_{\tau;k} = \sigma^2_{s,a;k} / \left(\sigma^2_{s,a;k} + \gamma^2\sigma^2_{s',b^+;k} + \sigma^2_w\right)$ *and* $b^+ = \mathrm{argmax}_{b\in\mathcal{A}}\,\mu_{s',b}$.

Proof. For simplicity, we first show the convergence of the algorithm for $\sigma_w = 0$ and then extend the result to the general case.

For simplicity, we define new notations as:

$$y_b \equiv r + \gamma\mu_{s',b} - \mu_{s,a}, \qquad v_0 \equiv \sigma^2_{s,a}, \qquad v_b \equiv \sigma^2_{s,a} + \gamma^2\sigma^2_{s',b}$$

In Sec.A.2, we obtained the exact solutions for the posterior mean and variance when $|\mathcal{A}| = 2$ (Eq.A.5 and Eq.A.6). When $\sigma_{s,a}, \sigma_{s',a_1}, \sigma_{s',a_2} \to 0$, the posterior mean is approximated as:

$$\frac{\bar{\mu}_{\tau,1}c_{\tau,1}\Phi_{\tau,1} + \bar{\mu}_{\tau,2}c_{\tau,2}\Phi_{\tau,2}}{c_{\tau,1}\Phi_{\tau,1} + c_{\tau,2}\Phi_{\tau,2}} \tag{C.2}$$

Then, using the Lemma.1, $c_{\tau,1}\Phi_{\tau,1}$ is approximated as:

$$\begin{aligned}
&\frac{1}{\sqrt{2\pi(\sigma^2_{s,a} + \gamma^2\sigma^2_{s',1})}} \exp\left\{ -\frac{(r+\gamma\mu_{s',1} - \mu_{s,a})^2}{2(\sigma^2_{s,a} + \gamma^2\sigma^2_{s',1})} - \frac{\left[r + \gamma\mu_{s',2} - \bar{\mu}_{\tau,1}\right]_+^2}{2(\bar{\sigma}^2_{\tau,1} + \gamma^2\sigma^2_{s',2})} \right\} \\
&= \frac{1}{\sqrt{2\pi v_1}} \exp\left\{ -\frac{y_1^2}{2v_1} - \frac{[y_2 - \alpha_1 y_1]_+^2}{2v_1^{-1}(v_1 v_2 - v_0^2)} \right\} \qquad \text{where } \alpha_b \equiv \frac{\sigma^2_{s,a}}{\sigma^2_{s,a} + \gamma^2\sigma_{s',b}} = \frac{v_0}{v_b}
\end{aligned} \tag{C.3}$$

Since the RHS of the equation is a sum of exponential function with the denominator of the inside term is proportional to a negative inverse variance, $\mathbb{E}_{q\sim\hat{p}_{Q_{s,a}}(\cdot)}[q]$ is approximated to $\bar{\mu}_{\tau,2} = (1-\alpha_2)\mu_{s,a} + \alpha_2(r + \gamma\mu_{s',a_2})$ if $c_{\tau,1}\Phi_{\tau,1} \ll c_{\tau,2}\Phi_{\tau,2}$ which is identical with the Q-learning update. Therefore, proving Theorem 2 is equivalent to proving the following statement. If $\mu_{s',2} > \mu_{s',1}$, and $\sigma_{s,a}, \sigma_{s',1}$, and $\sigma_{s',2}$ approach to 0, then $c_{\tau,1}\Phi_{\tau,1}/c_{\tau,2}\Phi_{\tau,2}$

approaches to 0. From the Eq.C.2 and Eq.C.3,

$$\log\left(\sqrt{\frac{v_1}{v_2}}\frac{c_1\Phi_1}{c_2\Phi_2}\right) = -\frac{y_1^2}{2v_1} - \frac{[y_2-\alpha_1 y_1]_+^2}{2v_1^{-1}(v_1v_2-v_0^2)} + \frac{y_2^2}{2v_2} + \frac{[y_1-\alpha_2 y_2]_+^2}{2v_2^{-1}(v_1v_2-v_0^2)} \tag{C.4}$$

Here, $[y_2-\alpha_1 y_1]_+^2$ is 0 if $\bar{\mu}_{\tau,1} \geq r+\gamma\mu_{s',2}$. Likewise, $[y_1-\alpha_2 y_2]_+^2$ is 0 if $\bar{\mu}_{\tau,2} \geq r+\gamma\mu_{s',1}$. We consider the following three cases which determine whether the max function terms are 0 or not.

(i) For $\mu_{s,a} < r+\gamma\mu_{s',1}$ and $\bar{\mu}_{\tau,2} < r+\gamma\mu_{s',1}$,

$$\begin{aligned}(RHS) =& -\frac{y_1^2}{2v_1}\left(1+\frac{v_0^2}{v_1v_2-v_0^2}-\frac{v_1v_2}{v_1v_2-v_0^2}\right)+\frac{y_2^2}{2v_2}\left(1+\frac{v_0^2}{v_1v_2-v_0^2}-\frac{v_1v_2}{v_1v_2-v_0^2}\right)\\ =& \left(-\frac{y_1^2}{2}+\frac{y_2^2}{2}\right)\cdot 0\end{aligned}$$

Therefore,

$$\frac{c_1\Phi_1}{c_2\Phi_2} = \sqrt{\frac{v_2}{v_1}} \qquad \text{and} \qquad \mu_{s,a}^{(new)} = \frac{\bar{\mu}_{\tau,1}\sqrt{v_2}+\bar{\mu}_{\tau,2}\sqrt{v_1}}{\sqrt{v_1}+\sqrt{v_2}}$$

Since $\bar{\mu}_{\tau,1} \geq \mu_{s,a}$ and $\bar{\mu}_{\tau,2} \geq \mu_{s,a}$, the newly updated mean is located somewhere between $\bar{\mu}_{\tau,1}$ and $\bar{\mu}_{\tau,2}$ and always $\mu_{s,a}^{(new)} \geq \mu_{s,a}$. Therefore, if $\mu_{s,a} < r+\gamma\mu_{s',1}$ and $\bar{\mu}_{\tau,2} \leq r+\gamma\mu_{s',1}$, then $\mu_{s,a}^{(new)} > \mu_{s,a}$ until $\bar{\mu}_{\tau,2}$ becomes larger than $r+\gamma\mu_{s',1}$.

(ii) For $r+\gamma\mu_{s',1} \leq \bar{\mu}_{\tau,1} < r+\gamma\mu_{s',2}$ ($\bar{\mu}_{\tau,2} > r+\gamma\mu_{s',1}$ from this condition),

$$\begin{aligned}(RHS) =& -\frac{y_1^2}{2v_1} - \frac{(y_2-\alpha_1 y_1)^2}{2v_1^{-1}(v_1v_2-v_0^2)} + \frac{y_2^2}{2v_2}\\ =& -\frac{(y_1-\alpha_2 y_2)^2}{2v_2^{-1}(v_1v_2-v_0^2)}\end{aligned}$$

Therefore, $(RHS) < 0$ and

$$\lim_{\sigma_{s,a},\sigma_{s',1},\sigma_{s',2}\to 0}\frac{c_1\Phi_1}{c_2\Phi_2} = \lim_{v_0,v_1,v_2\to 0}\left[\sqrt{\frac{v_2}{v_1}}\exp\left\{-\frac{(y_1-\alpha_2 y_2)^2}{2v_2^{-1}(v_1v_2-v_0^2)}\right\}\right] = 0$$

(iii) For $\mu_{s,a} > r + \mu_{s',2}$ and $\bar{\mu}_{\tau,1} \geq r + \gamma\mu_{s',2}$ ($\bar{\mu}_{\tau,2} > r + \gamma\mu_{s',1}$ from this condition),

$$\begin{aligned}(RHS) = & -\frac{y_1^2}{2v_1} + \frac{y_2^2}{2v_2} \\ = & -\frac{y_1^2}{2v_1}\left(1 - \frac{v_1}{v_2}\frac{y_2^2}{y_1^2}\right)\end{aligned}$$

If $y_2^2/v_2 < y_1^2/v_1$, then $(RHS) < 0$ with $\sigma_{s,a}, \sigma_{s',1}, \sigma_{s',2} \to 0$, and thus $c_{\tau,1}\Phi_{\tau,1}/c_{\tau,2}\Phi_{\tau,2}$ approaches to 0 as the previous case. If $y_2^2/v_2 \geq y_1^2/v_1$,

$$\frac{c_1\Phi_1}{c_2\Phi_2} = C\sqrt{\frac{v_2}{v_1}} \qquad \text{for some constant } C$$

Therefore,

$$\mu_{s,a}^{(new)} = \frac{\bar{\mu}_{\tau,1}C + \bar{\mu}_{\tau,2}}{C+1}$$

Similar to the first case, $\mu_{s,a}^{(new)}$ will be located somewhere between $\bar{\mu}_{\tau,1}$ and $\bar{\mu}_{\tau,2}$ and always $\mu_{s,a}^{(new)} < \mu_{s,a}$ until $\bar{\mu}_{\tau,1}$ becomes smaller than or equal to $r + \gamma\mu_{s',2}$.

In conclusion, when the variables satisfy either (i) or (iii), the mean value is contracted to the range corresponding to (ii) which is identical to the Q-learning update.

For $\sigma_w > 0$, $r + \gamma\mu_{s',b'}$ and $\gamma\sigma_{s',b'}$ in the CDF terms are replaced by $\mu_{b'}^w$ and $\sigma_{b'}^w$, respectively as Eq.B.5. $\sigma_{b'}^w$ approaches 0 as $\sigma_{s,a}, \sigma_{s',1}, \sigma_{s',2} \to 0$ and therefore, the above proofs are applied. However, the proofs are invalid when $\sigma_{s',b'}/\sigma_w = 0$ since the CDF terms in the likelihood distribution are no longer functions of q. As we have shown in Sec.B.2, the posterior mean is:

$$\mu_{s,a}^{(new)} = \bar{\mu}_{\tau,b^+} = \bar{\sigma}_{\tau,b^+}^2\left(\frac{\mu_{s,a}}{\sigma_{s,a}^2} + \frac{r + \gamma\mu_{s',b^+}}{\gamma^2\sigma_{s',b^+}^2 + \sigma_w^2}\right)$$

where $b^+ = \operatorname{argmax}_{b\in\mathcal{A}}\mu_{s',b}$. Thus, the update rule is still identical to the Q-learning update rule with the following learning rate, α_τ:

$$\alpha_\tau = \frac{\sigma_{s,a}^2}{\sigma_{s,a}^2 + \gamma^2\sigma_{s',b^+}^2 + \sigma_w^2}$$

□

C.3 Theorem 2: Convergence of ADFQ

Theorem 2. *The ADFQ update on the mean $\mu_{s,a}$ $\forall s \in \mathcal{S}$, $\forall a \in \mathcal{A}$ for $|\mathcal{A}| = 2$ is equivalent to the Q-learning update if the variances approach 0 and if all state-action pairs are visited infinitely often. In other words, we have :*

$$\lim_{k\to\infty,\{\sigma\}\to 0} \mu_{s,a;k+1} = (1-\alpha_{\tau;k})\,\mu_{s,a;k} + \alpha_{\tau;k}\left(r + \gamma \max_{b\in\mathcal{A}} \mu_{s',b;k}\right)$$

where $\alpha_{\tau;k} = \sigma^2_{s,a;k} / \left(\sigma^2_{s,a;k} + \gamma^2\sigma^2_{s',b^+;k} + \sigma^2_w\right)$ *and* $b^+ = \operatorname{argmax}_{b\in\mathcal{A}} \mu_{s',b}$.

Proof. Similar to the proof for the exact update case, we will show that the ratios of the coefficients, $k^*_b/k^*_{b_{max}}$ becomes 0 $\forall b \in \mathcal{A}$, $b \neq b_{max}$ where $b_{max} = \operatorname{argmax}_b \mu_{s',b}$, and $\mu^+_b \to \bar{\mu}_b$ as $\sigma_{s,a}, \sigma_{s',b}$ $\forall b \in \mathcal{A}$ goes to 0. When $|\mathcal{A}| = 2$ and $\mu_{s',2} > \mu_{s',1}$,

$$\frac{k^*_1}{k^*_2} = \frac{\sigma^*_1}{\sigma^*_2}\frac{\sigma_{s',2}}{\sigma_{s',1}} \exp\left\{ -\frac{y_1^2}{2v_1} + \frac{y_2^2}{2v_2} - \frac{(\mu^*_1 - \bar{\mu}_{\tau,1})^2}{2\bar{\sigma}^2_{\tau,1}} + \frac{(\mu^*_2 - \bar{\mu}_{\tau,2})^2}{2\bar{\sigma}^2_{\tau,2}} \right.$$
$$\left. - \frac{[r+\gamma\mu_{s',2} - \mu^*_1]^2_+}{2\gamma^2\sigma^2_{s',2}} + \frac{[r+\gamma\mu_{s',1} - \mu^*_2]^2_+}{2\gamma^2\sigma^2_{s',1}} \right\}$$

According to the definition of μ^+_b,

$$\mu^*_1 - \bar{\mu}_{\tau,1} = \frac{\bar{\sigma}_{\tau,1}}{\gamma^2\sigma^2_{s',2}}[r + \gamma\mu_{s,'2} - \mu^*_1]_+$$

and $\mu^+_b \geq \bar{\mu}_{\tau,b}$. Therefore,

$$\log\left(\frac{k^*_1\sigma_{s',1}\sigma^*_2}{k^*_2\sigma_{s',2}\sigma^*_1}\right) = -\frac{y_1^2}{2v_1} + \frac{y_2^2}{2v_2} - \frac{[r+\gamma\mu_{s',2} - \mu^*_1]^2_+}{2\gamma^2\sigma^2_{s',2}}\left(1 + \frac{\bar{\sigma}^2_{\tau,1}}{\gamma^2\sigma^2_{s',2}}\right)$$
$$+ \frac{[r+\gamma\mu_{s',1} - \mu^*_2]^2_+}{2\gamma^2\sigma^2_{s',1}}\left(1 + \frac{\bar{\sigma}^2_{\tau,2}}{\gamma^2\sigma^2_{s',1}}\right)$$

When $\mu_b^+ < r + \gamma \mu_{s',b'}$

$$\mu_b^+ = \left(\frac{1}{\bar{\sigma}_{\tau,b}} + \frac{1}{\gamma^2 \sigma_{s',b'}^2} \right)^{-1} \left(\frac{\bar{\mu}_{\tau,b}}{\bar{\sigma}_{\tau,b}} + \frac{r + \gamma \mu_{s',b'}}{\gamma^2 \sigma_{s',b'}^2} \right) \tag{C.5}$$

When $\mu_b^+ \geq r + \gamma \mu_{s',b'}$, $\mu_b^+ = \bar{\mu}_{\tau,b}$.

For $\mu_1^* < r + \gamma \mu_{s',2}$ and $\mu_2^* < r + \gamma \mu_{s',1}$, it is also, $\mu_{s,a} \leq \bar{\mu}_{\tau,1} \leq \bar{\mu}_{\tau,2} < r + \gamma \mu_{s',1} < r + \gamma \mu_{s',2}$. Then, using Eq.C.5, we have

$$\log \left(\frac{k_1^* \sigma_{s',1} \sigma_2^*}{k_2^* \sigma_{s',2} \sigma_1^*} \right) = -\frac{y_1^2}{2v_1} + -\frac{(y_2 - \alpha_1 y_1)^2}{2v_1^{-1}(v_1 v_2 - v_0^2)} + \frac{y_2^2}{2v_2} + \frac{(y_1 - \alpha_2 y_2)^2}{2v_2^{-1}(v_1 v_2 - v_0^2)}$$

which is same with (i) of the proof of Theorem 1. The new mean will be weighted sum of μ_1^*, μ_2^*. Since $\mu_{s,a}$ is smaller than both $\bar{\mu}_{\tau,1}$ and $\bar{\mu}_{\tau,2}$, $\mu_{s,a}^{(new)} > \mu_{s,a}$ until $r + \gamma_{s',1} < \bar{\mu}_{\tau,2}$. For the other cases, the same directions in the proof of Theorem 1 are applied.

We can apply the same proof procedures to the case of $\sigma_w > 0$ using $\gamma^2 \sigma_{s',b}^2 + \sigma_w^2$ instead of $\gamma^2 \sigma_{s',b}^2$ in $\bar{\mu}_{\tau,b}$, $\bar{\sigma}_{\tau,b}$, and $c_{\tau,b}$. Therefore, the mean update rule converges to the Q-learning update and the corresponding learning rate is:

$$\alpha_\tau = \frac{\sigma_{s,a}^2}{\sigma_{s,a}^2 + \gamma^2 \sigma_{s',b^+}^2 + \sigma_w^2}$$

$\square$

Appendix D

Mathematical Derivation of Fast ADFQ

In this chapter, The function $f(\cdot)$ is defined in the Sec.4.8 as the approximation of the below term when the term of the product of the Gaussian CDFs approaches to 0.

$$\frac{1}{\bar{\sigma}_{\tau,b}}\phi\Big(\frac{q-\bar{\mu}_{\tau,b}}{\bar{\sigma}_{\tau,b}}\Big)\cdot\prod_{b'\neq b}\Phi\Big(\frac{q-(r+\gamma\mu_{s',b'})}{\gamma\sigma_{s',b'}}\Big)$$

$$f(q;\mu,\sigma)=\begin{cases}\frac{\epsilon}{\sigma}\phi\Big(\frac{q-\mu}{\sigma}\Big) & \text{for } q\in[\mu-\epsilon,\mu+\epsilon], \epsilon\ll 1\\ 0 & \text{otherwise}\end{cases} \tag{D.1}$$

In order to simplify notations, we define following variables:

- The best action : $b^{+}\doteq \text{argmax}_{b\in\mathcal{A}}\,\mu_{s',b}$
- The second best action : $b^{2+}\doteq \text{argmax}_{b\in\mathcal{A},b\neq b^{+}}\,\mu_{s',b}$
- A truncated Gaussian CDF : $\Phi_a^b\left(\frac{q-\mu}{\sigma}\right)\doteq\int_a^b\frac{1}{\sigma}\phi\left(\frac{q-\mu}{\sigma}\right)$
- TD target : $t_{\tau,b}\doteq r+\gamma\mu_{s',b}$

D.1 Normalization

First, we solve the analytic expression of the normalization factor, Z:

$$Z = \sum_{b \neq b^+} c_{\tau,b} \left\{ \int_{-\infty}^{t_{\tau,b^+}} f(q; \bar{\mu}_{\tau,b}, \bar{\sigma}_{\tau,b}) dq + \Phi_{t_{\tau,b^+}}^{\infty} \left(\frac{q - \bar{\mu}_{\tau,b}}{\bar{\sigma}_{\tau,b}} \right) \right\}$$
$$+ c_{\tau,b^+} \left\{ \int_{-\infty}^{t_{\tau,b^{2+}}} f(q; \bar{\mu}_{\tau,b^+}, \bar{\sigma}_{\tau,b^+}) dq + \Phi_{t_{\tau,b^{2+}}}^{\infty} \left(\frac{q - \bar{\mu}_{\tau,b^+}}{\bar{\sigma}_{\tau,b^+}} \right) \right\}$$

$$\begin{aligned} Z = & \sum_{b \neq b^+} c_{\tau,b} \left\{ H(t_{\tau,b^+} - (\bar{\mu}_{\tau,b} + \epsilon)) \cdot \epsilon \cdot \Phi_{\bar{\mu}_{\tau,b} - \epsilon}^{\bar{\mu}_{\tau,b} + \epsilon} \left(\frac{q - \bar{\mu}_{\tau,b}}{\bar{\sigma}_{\tau,b}} \right) + \Phi_{t_{\tau,b^+}}^{\infty} \left(\frac{q - \bar{\mu}_{\tau,b}}{\bar{\sigma}_{\tau,b}} \right) \right\} \\ & + c_{\tau,b^+} \left\{ H(t_{\tau,b^{2+}} - (\bar{\mu}_{\tau,b^+} + \epsilon)) \cdot \epsilon \cdot \Phi_{\bar{\mu}_{\tau,b^+} - \epsilon}^{\bar{\mu}_{\tau,b^+} + \epsilon} \left(\frac{q - \bar{\mu}_{\tau,b^+}}{\bar{\sigma}_{\tau,b^+}} \right) + \Phi_{t_{\tau,b^{2+}}}^{\infty} \left(\frac{q - \bar{\mu}_{\tau,b^+}}{\bar{\sigma}_{\tau,b^+}} \right) \right\} \\ = & \sum_{b \neq b^+} c_{\tau,b} \left\{ \epsilon H(t_{\tau,b^+} - (\bar{\mu}_{\tau,b} + \epsilon)) \left(\Phi\left(\frac{\epsilon}{\bar{\sigma}_{\tau,b}} \right) - \Phi\left(\frac{-\epsilon}{\bar{\sigma}_{\tau,b}} \right) \right) + 1 - \Phi\left(\frac{t_{\tau,b^+} - \bar{\mu}_{\tau,b}}{\bar{\sigma}_{\tau,b}} \right) \right\} \\ & + c_{\tau,b^+} \left\{ \epsilon H(t_{\tau,b^{2+}} - (\bar{\mu}_{\tau,b^+} + \epsilon)) \left(\Phi\left(\frac{\epsilon}{\bar{\sigma}_{\tau,b^+}} \right) - \Phi\left(\frac{-\epsilon}{\bar{\sigma}_{\tau,b^+}} \right) \right) + 1 - \Phi\left(\frac{t_{\tau,b^{2+}} - \bar{\mu}_{\tau,b^+}}{\bar{\sigma}_{\tau,b^+}} \right) \right\} \end{aligned}$$

where $H(\cdot)$ is a Heaviside step function which $H(x) = 1$ if $x \geq 0$ and $H(x) = 0$ otherwise. Since $\bar{\sigma}_{\tau,b} \ll 1 \ \forall b \in \mathcal{A}$ and $\epsilon > 0$

$$\Phi\left(\frac{t_{\tau,b^+} - \bar{\mu}_{\tau,b}}{\bar{\sigma}_{\tau,b}} \right) \approx H(t_{\tau,b^+} - \bar{\mu}_{\tau,b}) \qquad \Phi\left(\frac{\epsilon}{\bar{\sigma}_{\tau,b}} \right) \approx 1$$

$$
\begin{aligned}
Z \approx & \sum_{b \neq b^+} c_{\tau,b} \Big\{ \epsilon H(t_{\tau,b^+} - (\bar{\mu}_{\tau,b} + \epsilon)) - H(t_{\tau,b^+} - \bar{\mu}_{\tau,b}) \Big\} \\
& \quad + c_{\tau,b^+} \Big\{ \epsilon H(t_{\tau,b^{2+}} - (\bar{\mu}_{b^+} + \epsilon)) - H(t_{\tau,b^{2+}} - \bar{\mu}_{b^+}) \Big\} \\
\approx & \sum_{b \neq b^+} c_{\tau,b} \left(1 - (1-\epsilon) H(t_{\tau,b^+} - \bar{\mu}_{\tau,b})\right) + c_{\tau,b^+} \left(1 - (1-\epsilon) H(t_{b^{2+}} - \bar{\mu}_{\tau,b^+})\right) \\
\approx & \sum_{b} c_{\tau,b} \nu_{\tau,b}
\end{aligned}
$$

where

$$
\nu_{\tau,b} \doteq \begin{cases} 1 - (1-\epsilon) H(t_{\tau,b^+} - \bar{\mu}_{\tau,b}) & \text{for } b \neq b^+ \\ 1 - (1-\epsilon) H(t_{\tau,b^{2+}} - \bar{\mu}_{\tau,b}) & \text{for } b = b^+ \end{cases}
$$

.

D.2 Mean

$$
\begin{aligned}
\mathbb{E}_{\tilde{p}_{Q_{s,a}}}[q] = & \frac{1}{Z} \sum_{b \neq b^+} c_{\tau,b} \Bigg\{ \epsilon H(t_{\tau,b^+} - (\bar{\mu}_{\tau,b} + \epsilon)) \int_{\bar{\mu}_{\tau,b} - \epsilon}^{\bar{\mu}_{\tau,b} + \epsilon} \frac{q}{\bar{\sigma}_{\tau,b}} \phi\bigg(\frac{q - \bar{\mu}_{\tau,b}}{\bar{\sigma}_{\tau,b}} \bigg) dq \\
& \qquad + \int_{t_{\tau,b^+}}^{\infty} \frac{q}{\bar{\sigma}_{\tau,b}} \phi\bigg(\frac{q - \bar{\mu}_{\tau,b}}{\bar{\sigma}_{\tau,b}} \bigg) dq \Bigg\} \\
& + \frac{1}{Z} c_{\tau,b^+} \Bigg\{ \epsilon H(t_{\tau,b^{2+}} - (\bar{\mu}_{\tau,b^+} + \epsilon)) \int_{\bar{\mu}_{\tau,b^+} - \epsilon}^{\bar{\mu}_{\tau,b^+} + \epsilon} \frac{q}{\bar{\sigma}_{\tau,b^+}} \phi\bigg(\frac{q - \bar{\mu}_{\tau,b^+}}{\bar{\sigma}_{\tau,b^+}} \bigg) dq \\
& \qquad + \int_{t_{\tau,b^{2+}}}^{\infty} \frac{q}{\bar{\sigma}_{\tau,b^+}} \phi\bigg(\frac{q - \bar{\mu}_{\tau,b^+}}{\bar{\sigma}_{\tau,b^+}} \bigg) dq \Bigg\}
\end{aligned}
$$

The mean of the two sided truncated normal distribution with the original normal distribution $\mathcal{N}(\mu, \sigma^2)$ is :

$$
\mathbb{E}[X | a < X < b] = \mu + \sigma \frac{\phi\big(\frac{a-\mu}{\sigma}\big) - \phi\big(\frac{b-\mu}{\sigma}\big)}{\Phi\big(\frac{b-\mu}{\sigma}\big) - \Phi\big(\frac{a-\mu}{\sigma}\big)}
$$

With a normalization constant, Z_E,

$$
\mathbb{E}[X | a < X < b] \cdot Z_E = \mu \left(\Phi\left(\frac{b-\mu}{\sigma}\right) - \Phi\left(\frac{a-\mu}{\sigma}\right) \right) + \sigma \left(\phi\left(\frac{a-\mu}{\sigma}\right) - \phi\left(\frac{b-\mu}{\sigma}\right) \right)
$$

Therefore,

$$\int_{\bar{\mu}_{\tau,b}-\epsilon}^{\bar{\mu}_{\tau,b}+\epsilon} \frac{q}{\bar{\sigma}_{\tau,b}} \phi\left(\frac{q-\bar{\mu}_{\tau,b}}{\bar{\sigma}_{\tau,b}}\right) dq = \bar{\mu}_{\tau,b}\left(2\Phi\left(\frac{\epsilon}{\bar{\sigma}_{\tau,b}}\right)-1\right) + \bar{\sigma}_{\tau,b}\cdot 0 \approx \bar{\mu}_{\tau,b}$$

and

$$\begin{aligned}
\mathbb{E}_{\tilde{p}_{Q_{s,a}}}[q] &= \frac{1}{Z}\sum_{b\neq b^+} c_{\tau,b}\left\{\epsilon H\left(t_{\tau,b^+}-(\bar{\mu}_{\tau,b}+\epsilon)\right)\bar{\mu}_{\tau,b}\right. \\
&\qquad \left. + \bar{\mu}_{\tau,b}\left(1-\Phi\left(\frac{t_{\tau,b^+}-\bar{\mu}_{\tau,b}}{\bar{\sigma}_{\tau,b}}\right)\right) + \bar{\sigma}_{\tau,b}\phi\left(\frac{t_{\tau,b^+}-\bar{\mu}_{\tau,b}}{\bar{\sigma}_{\tau,b}}\right)\right\} \\
&+ \frac{1}{Z}c_{\tau,b^+}\left\{\epsilon H\left(t_{\tau,b^{2+}}-(\bar{\mu}_{\tau,b^+}+\epsilon)\right)\bar{\mu}_{\tau,b^+}\right. \\
&\qquad \left. + \bar{\mu}_{\tau,b^+}\left(1-\Phi\left(\frac{t_{\tau,b^{2+}}-\bar{\mu}_{\tau,b^+}}{\bar{\sigma}_{\tau,b^+}}\right)\right) + \bar{\sigma}_{\tau,b^+}\phi\left(\frac{t_{\tau,b^{2+}}-\bar{\mu}_{\tau,b^+}}{\bar{\sigma}_{\tau,b^+}}\right)\right\} \\
&\approx \frac{1}{Z}\sum_{b} c_{\tau,b}\nu_{\tau,b}\bar{\mu}_{\tau,b}
\end{aligned}$$

D.3 Variance

First of all, the second moment is :

$$\begin{aligned}
\mathbb{E}_{\tilde{p}_{Q_{s,a}}}[q^2] = &\frac{1}{Z}\sum_{b\neq b^+} c_{\tau,b}\left\{\epsilon H(t_{\tau,b^+}-(\bar{\mu}_{\tau,b}+\epsilon))\int_{\bar{\mu}_{\tau,b}-\epsilon}^{\bar{\mu}_{\tau,b}+\epsilon}\frac{q^2}{\bar{\sigma}_{\tau,b}}\phi\left(\frac{q-\bar{\mu}_{\tau,b}}{\bar{\sigma}_{\tau,b}}\right)dq + \int_{t_{\tau,b^+}}^{\infty}\frac{q^2}{\bar{\sigma}_{\tau,b}}\phi\left(\frac{q-\bar{\mu}_{\tau,b}}{\bar{\sigma}_{\tau,b}}\right)dq\right\} \\
&+ \frac{1}{Z}c_{\tau,b^+}\left\{\epsilon H(t_{\tau,b^{2+}}-(\bar{\mu}_{b^+}+\epsilon))\int_{\bar{\mu}_{b^+}-\epsilon}^{\bar{\mu}_{b^+}+\epsilon}\frac{q^2}{\bar{\sigma}_{b^+}}\phi\left(\frac{q-\bar{\mu}_{b^+}}{\bar{\sigma}_{b^+}}\right)dq + \int_{t_{\tau,b^{2+}}}^{\infty}\frac{q^2}{\bar{\sigma}_{b^+}}\phi\left(\frac{q-\bar{\mu}_{b^+}}{\bar{\sigma}_{b^+}}\right)dq\right\}
\end{aligned}$$

The variance of the truncated normal distribution is:

$$\mathrm{Var}[X|a<X<b] = \sigma^2\left[1+\frac{\frac{a-\mu}{\sigma}\phi(\frac{a-\mu}{\sigma})-\frac{b-\mu}{\sigma}\phi(\frac{b-\mu}{\sigma})}{\Phi(\frac{b-\mu}{\sigma})-\Phi(\frac{a-\mu}{\sigma})} - \left(\frac{\phi(\frac{a-\mu}{\sigma})-\phi(\frac{b-\mu}{\sigma})}{\Phi(\frac{b-\mu}{\sigma})-\Phi(\frac{a-\mu}{\sigma})}\right)^2\right]$$

With a normalization constant, Z_V,

$$\begin{aligned}\mathbb{E}[X^2|a<X<b]\cdot Z_V &= (\mu^2+\sigma^2)\left(\Phi\left(\frac{b-\mu}{\sigma}\right)-\Phi\left(\frac{a-\mu}{\sigma}\right)\right)\\ &\quad +\sigma^2\left(\frac{a-\mu}{\sigma}\phi\left(\frac{a-\mu}{\sigma}\right)-\frac{b-\mu}{\sigma}\phi\left(\frac{b-\mu}{\sigma}\right)\right)+2\mu\sigma\left(\phi\left(\frac{a-\mu}{\sigma}\right)-\phi\left(\frac{b-\mu}{\sigma}\right)\right)\end{aligned}$$

Thus, when $t_{\tau,b^+}\geq\bar{\mu}_{\tau,b}+\epsilon$,

$$\begin{aligned}\int_{\bar{\mu}_{\tau,b}-\epsilon}^{\bar{\mu}_{\tau,b}+\epsilon}\frac{q^2}{\bar{\sigma}_{\tau,b}}\phi\left(\frac{q-\bar{\mu}_{\tau,b}}{\bar{\sigma}_{\tau,b}}\right)dq &= (\bar{\mu}_{\tau,b}^2+\bar{\sigma}_{\tau,b}^2)\left(2\Phi(\frac{\epsilon}{\bar{\sigma}_{\tau,b}})-1\right)-2\epsilon\bar{\sigma}_{\tau,b}\phi(\frac{\epsilon}{\bar{\sigma}_{\tau,b}})\\ &\approx \bar{\mu}_{\tau,b}^2+\bar{\sigma}_{\tau,b}^2\end{aligned}$$

For the second term,

$$\begin{aligned}\int_{t_{\tau,b^+}}^{\infty}\frac{q^2}{\bar{\sigma}_{\tau,b}}\phi\left(\frac{q-\bar{\mu}_{\tau,b}}{\bar{\sigma}_{\tau,b}}\right)dq &= (\bar{\mu}_{\tau,b}^2+\bar{\sigma}_{\tau,b}^2)\left(1-\Phi\left(\frac{t_{\tau,b^+}-\bar{\mu}_{\tau,b}}{\bar{\sigma}_{\tau,b}}\right)\right)\\ &\quad +\bar{\sigma}_{\tau,b}(t_{\tau,b^+}-\bar{\mu}_{\tau,b})\phi\left(\frac{t_{\tau,b^+}-\bar{\mu}_{\tau,b}}{\bar{\sigma}_{\tau,b}}\right)+2\bar{\mu}_{\tau,b}\bar{\sigma}_{\tau,b}\phi\left(\frac{t_{\tau,b^+}-\bar{\mu}_{\tau,b}}{\bar{\sigma}_{\tau,b}}\right)\\ &\approx (\bar{\mu}_{\tau,b}^2+\bar{\sigma}_{\tau,b}^2)\left(1-H(t_{\tau,b^+}-\bar{\mu}_{\tau,b})\right)\end{aligned}$$

Therefore,

$$\begin{aligned}\mathbb{E}_{\tilde{p}_{Q_{s,a}}}[q^2] &\approx \frac{1}{Z}\sum_{b\neq b^+}c_{\tau,b}(\bar{\mu}_{\tau,b}^2+\bar{\sigma}_{\tau,b}^2)\left(\epsilon H(t_{\tau,b^+}-(\bar{\mu}_{\tau,b}+\epsilon))+\left(1-H(t_{\tau,b^+}-\bar{\mu}_{\tau,b})\right)\right)\\ &\quad +\frac{1}{Z}c_{\tau,b^+}(\bar{\mu}_{\tau,b^+}^2+\bar{\sigma}_{\tau,b^+}^2)\left(\epsilon H(t_{\tau,b^{2+}}-(\bar{\mu}_{\tau,b^+}+\epsilon))+\left(1-H(t_{\tau,b^{2+}}-\bar{\mu}_{\tau,b^+})\right)\right)\\ &\approx \frac{1}{Z}\sum_{b\neq b^+}c_{\tau,b}(\bar{\mu}_{\tau,b}^2+\bar{\sigma}_{\tau,b}^2)\left(1-(1-\epsilon)H(t_{\tau,b^+}-\bar{\mu}_{\tau,b})\right)\\ &\quad +\frac{1}{Z}c_{\tau,b^+}(\bar{\mu}_{\tau,b^+}^2+\bar{\sigma}_{\tau,b^+}^2)\left(1-(1-\epsilon)H(t_{\tau,b^{2+}}-\bar{\mu}_{\tau,b^+})\right)\\ &\approx \frac{1}{Z}\sum_{b}c_{\tau,b}\nu_{\tau,b}(\bar{\mu}_{\tau,b}^2+\bar{\sigma}_{\tau,b}^2)\end{aligned}$$

$\mathrm{Var}_{\tilde{p}_{Q_{s,a}}}[q]=E_{\tilde{p}_{Q_{s,a}}}[q^2]-(E_{\tilde{p}_{Q_{s,a}}}[q])^2$

Appendix E

Experimental Details

E.1 Deep ADFQ in Atari games

E.1.1 Neural Network Architecture and Details

In the all domains, we use the default settings of the OpenAI baselines [29] for DQN and Double DQN, and made minimal changes for ADFQ. Each network consists of three convolution layers followed by a 256 neuron linear layer. The first convolution layer contains 32 filters of size 8 with stride 4. The second convolution layer contains 64 filters of size 4 with stride 2. The final convolution layer contains 64 filters of size 3 with stride 1. We used ReLU nonlinearities and the Adam optimizer with mini-batches size of 32.

E.1.2 Initialization

Xavier initialization is used for weight variables, and all bias variables are initialized to zero in all networks. Additionally, for ADFQ, the weights of the final hidden layer are initialized with 0.0 and its bias variables are initialized with two constant values which correspond to μ_0 and $-\log(\sigma_0)$ where μ_0 is an initial mean and σ_0^2 is an initial variance (e.g. an initial bias vector of the final layer is $\vec{b} = [\mu_0, \cdots, \mu_0, -\log(\sigma_0), \cdots, -\log(\sigma_0)]^T$. We set $\sigma_0 = 50.0$ for the Atari games.

E.2 Deep ADFQ in cart-pole balancing tasks

E.2.1 Neural Network Architecture and Details

In the all domains, we use the default settings of the OpenAI baselines [29] for all algorithms. Each network consists of 1 hidden layer with 64 hidden unites.We used ReLU nonlinearities and the Adam optimizer with mini-batches size of 32.

E.2.2 Initialization

The initialization method described above (Sec.E.1) is used. We set $\sigma_0 = 30.0$ for these tasks.

Bibliography

[1] Francesco Amigoni and Vincenzo Caglioti. An information-based exploration strategy for environment mapping with mobile robots. *Robotics and Autonomous Systems*, 58(5):684–699, 2010.

[2] Nikolay Atanasov, Jerome Le Ny, Kostas Daniilidis, and George J Pappas. Information acquisition with sensing robots: Algorithms and error bounds. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6447–6454. IEEE, 2014.

[3] Nikolay Atanasov, Jerome Le Ny, Kostas Daniilidis, and George J Pappas. Decentralized active information acquisition: Theory and application to multi-robot slam. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4775–4782. IEEE, 2015.

[4] Nikolay Atanasov, Bharath Sankaran, Jerome Le Ny, Thomas Koletschka, George J Pappas, and Kostas Daniilidis. Hypothesis testing framework for active object detection. In *2013 IEEE International Conference on Robotics and Automation*, pages 4216–4222. IEEE, 2013.

[5] Kamyar Azizzadenesheli, Emma Brunskill, and Animashree Anandkumar. Efficient exploration through bayesian deep q-networks. *arXiv preprint arXiv:1802.04412*, 2018.

[6] J Andrew Bagnell, Sham M Kakade, Jeff G Schneider, and Andrew Y Ng. Policy search by dynamic programming. In *Advances in neural information processing systems*, pages 831–838, 2004.

[7] Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning Proceedings 1995*, pages 30–37. Elsevier, 1995.

[8] Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. *arXiv preprint arXiv:1909.07528*, 2019.

[9] Marc G Bellemare, Will Dabney, and Remi Munos. A distributional perspective on reinforcement learning. *arXiv preprint arXiv:1707.06887*, 2017.

[10] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[11] C. M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2007.

[12] Frederic Bourgault, Alexei A Makarenko, Stefan B Williams, Ben Grocholsky, and Hugh F Durrant-Whyte. Information based adaptive robotic exploration. In *IEEE/RSJ international conference on intelligent robots and systems*, volume 1, pages 540–545. IEEE, 2002.

[13] Xavier Boyen and Daphne Koller. Tractable inference for complex stochastic processes. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, Berkeley, CA, 1998.

[14] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[15] Lucian Bu, Robert Babu, Bart De Schutter, et al. A comprehensive survey of multi-agent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

[16] Benjamin Charrow, Vijay Kumar, and Nathan Michael. Approximate representations for multi-robot control policies that maximize mutual information. *Autonomous Robots*, 37(4):383–400, 2014.

[17] Tao Chen, Saurabh Gupta, and Abhinav Gupta. Learning exploration policies for navigation. In *7th International Conference on Learning Representations, ICLR 2019*, 2019.

[18] Han-Lim Choi. *Adaptive sampling and forecasting with mobile sensor networks.* PhD thesis, Massachusetts Institute of Technology, 2009.

[19] Sanjiban Choudhury, Ashish Kapoor, Gireeja Ranade, and Debadeepta Dey. Learning to gather information via imitation. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 908–915. IEEE, 2017.

[20] Girish Chowdhary, Miao Liu, Robert Grande, Thomas Walsh, Jonathan How, and Lawrence Carin. Off-policy reinforcement learning with gaussian process. *IEEE/CAA Journal of Automatica Sinica*, 1(3):227–238, 2014.

[21] Timothy H Chung, Joel W Burdick, and Richard M Murray. A decentralized motion coordination strategy for dynamic target tracking. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 2416–2422. IEEE, 2006.

[22] Lehel Csató and Manfred Opper. Sparse on-line gaussian processes. *Neural computation*, 14(3):641–668, 2002.

[23] Nathaniel D Daw, Yael Niv, and Peter Dayan. Uncertainty-based competition between prefrontal and dorsolateral striatal systems for behavioral control. *Cerebral Cortex*, 13:400–408, 2003.

[24] Peter Dayan and Nathaniel D Daw. Decision theory, reinforcement learning, and the brain. *Cognitive, Affective, & Behavioral Neuroscience*, 8(4):429–453, 2008.

[25] Richard Dearden, Nir Friedman, and David Andre. Model based bayesian exploration. In *Proceedings of the 15th conference on Uncertainty in artificial intelligence*, pages 150–159. Morgan Kaufmann Publishers Inc., 1999.

[26] Richard Dearden, Nir Friedman, and Stuart Russell. Bayesian q-learning. In *AAAI/IAAI*, pages 761–768, 1998.

[27] Morris H Degroot. *Probability and Statistics (4th Edition)*. Pearson, 2011.

[28] Marc Peter Deisenroth, Gerhard Neumann, and Jan Peters. A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1-2):1–142, 2011.

[29] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Openai baselines. `https://github.com/openai/baselines`, 2017.

[30] Michael Duff. Optimal learning: Computational procedures for bayes-adaptive markov decision processes. *PhD thesis, University of Massachusetts, Amherst*, 2002.

[31] Matthew Dunbabin and Lino Marques. Robots for environmental monitoring: Significant advancements and applications. *IEEE Robotics & Automation Magazine*, 19(1):24–39, 2012.

[32] Yaakov Engel, Shie Mannor, and Ron Meir. Bayes meets bellman: The gaussian process approach to temporal difference learning. In *Proceedings of the 20th International Conference on Machine Learning*, volume 20, 2003.

[33] Yaakov Engel, Shie Mannor, and Ron Meir. Reinforcement learning with gaussian processes. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 201–208, 2005.

[34] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

[35] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot visual imitation learning via meta-learning. In *Conference on Robot Learning*, pages 357–368, 2017.

[36] Matthieu Geist and Olivier Pietquin. Kalman temporal differences. *Journal of artificial intelligence research*, 39:483–532, 2010.

[37] Mohammad Ghavamzadeh and Yaakov Engel. Bayesian policy gradient algorithms. In *Advances in neural information processing systems*, pages 457–464, 2007.

[38] Mohammad Ghavamzadeh, Shie Mannor, Joelle Pineau, and Aviv Tamar. Bayesian reinforcement learning: A survey. *Foundation and Trends in Machine Learning*, 8(5-6):359–483, 2015.

[39] Alessandro Giusti, Jérôme Guzzi, Dan C Cireşan, Fang-Lin He, Juan P Rodríguez, Flavio Fontana, Matthias Faessler, Christian Forster, Jürgen Schmidhuber, Gianni Di Caro, et al. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters*, 1(2):661–667, 2015.

[40] Héctor H González-Banos and Jean-Claude Latombe. Navigation strategies for exploring indoor environments. *The International Journal of Robotics Research*, 21(10-11):829–848, 2002.

[41] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[42] Shane Griffith, Kaushik ubramanian, Jonathan Scholz, Charles Isbell, and Andrea L. Thomaz. Policy shaping: Integrating human feedback with reinforcement learning. In *Advances in neural information processing systems*, pages 2625–2633, 2013.

[43] 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.

[44] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838, 2016.

[45] Florent Guenter, Micha Hersch, Sylvain Calinon, and Aude Billard. Reinforcement learning for imitating constrained reaching movements. *Advanced Robotics*, 21(13):1521–1544, 2007.

[46] Arthur Guez, David Silver, and Peter Dayan. Efficient bayes-adaptive reinforcement learning using sample-based search. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1071–1079, 2012.

[47] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2616–2625, 2017.

[48] William H Guss, Cayden Codel, Katja Hofmann, Brandon Houghton, Noboru Kuno, Stephanie Milani, Sharada Mohanty, Diego Perez Liebana, Ruslan Salakhutdinov, Nicholay Topin, et al. The minerl competition on sample efficient reinforcement learning using human priors. *arXiv preprint arXiv:1904.10079*, 2019.

[49] Tuomas Haarnoja, Anurag Ajay, Sergey Levine, and Pieter Abbeel. Backprop kf: Learning discriminative deterministic state estimators. In *Advances in Neural Information Processing Systems*, pages 4376–4384, 2016.

[50] Anna Harutyunyan, Marc G Bellemare, Tom Stepleton, and Rémi Munos. Q (λ) with off-policy corrections. In *International Conference on Algorithmic Learning Theory*, pages 305–320. Springer, 2016.

[51] Hado V Hasselt. Double q-learning. In *Advances in neural information processing systems*, pages 2613–2621, 2010.

[52] He He, Paul Mineiro, and Nikos Karampatziakis. Active information acquisition. *arXiv preprint arXiv:1602.02181*, 2016.

[53] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[54] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[55] Allaa Hilal. *An Intelligent Sensor Management Framework for Pervasive Surveillance.* PhD thesis, University of Waterloo, 2013.

[56] Hirohisa Hirukawa, Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, and Takakatsu Isozumi. The human-size humanoid robot that can walk, lie down and get up. *The International Journal of Robotics Research*, 24(9):755–769, 2005.

[57] Gabriel M Hoffmann, Steven L Waslander, and Claire J Tomlin. Mutual information methods with particle filters for mobile sensor network control. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 1019–1024. IEEE, 2006.

[58] Geoffrey A Hollinger and Gaurav S Sukhatme. Sampling-based motion planning for robotic information gathering. In *Robotics: Science and Systems*, volume 3. Citeseer, 2013.

[59] Marco Huber. *Probabilistic framework for sensor management*, volume 7. KIT Scientific Publishing, 2009.

[60] Dinesh Jayaraman and Kristen Grauman. Learning to look around: Intelligently exploring unseen environments for unknown tasks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1238–1247, 2018.

[61] Heejin Jeong and Daniel D Lee. Efficient learning of stand-up motion for humanoid robots with bilateral symmetry. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1544–1549. IEEE, 2016.

[62] Heejin Jeong and Daniel D Lee. Learning complex stand-up motion for humanoid robots. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[63] Heejin Jeong, Brent Schlotfeldt, Hamed Hassani, Manfred Morari, Daniel D Lee, and George J Pappas. Learning q-network for active information acquisition. In *2019*

IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 6822–6827. IEEE, 2019.

[64] Heejin Jeong, Clark Zhang, George J Pappas, and Daniel D Lee. Assumed density filtering q-learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 2607–2613. AAAI Press, 2019.

[65] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

[66] Gregory Kahn, Tianhao Zhang, Sergey Levine, and Pieter Abbeel. Plato: Policy learning using adaptive trajectory optimization. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3342–3349. IEEE, 2017.

[67] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*, pages 651–673, 2018.

[68] Vasiliy Karasev, Alessandro Chiuso, and Stefano Soatto. Controlled recognition bounds for visual learning and exploration. In *Advances in neural information processing systems*, pages 2915–2923, 2012.

[69] Simon Killcross and Etienne Coutureau. Coordination of actions and habits in the medial prefrontal cortex of rats. *Nature Neuroscience*, 8:1704–1711, 2005.

[70] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[71] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[72] Thomas Kollar and Nicholas Roy. Trajectory optimization using reinforcement learning for map exploration. *The International Journal of Robotics Research*, 27(2):175–196, 2008.

[73] George Konidaris, Sarah Osentoski, and Philip Thomas. Value function approximation in reinforcement learning using the fourier basis. In *Twenty-fifth AAAI conference on artificial intelligence*, 2011.

[74] Christopher M Kreucher. *An information-based approach to sensor resource allocation.* University of Michigan, 2005.

[75] Vijay Kumar, Daniela Rus, and Sanjiv Singh. Robot and sensor networks for first responders. *IEEE Pervasive computing*, 3(4):24–33, 2004.

[76] Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of machine learning research*, 4(Dec):1107–1149, 2003.

[77] Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4190–4203, 2017.

[78] Mikko Lauri and Risto Ritala. Planning for robotic exploration based on forward simulation. *Robotics and Autonomous Systems*, 83:15–31, 2016.

[79] Laura Leal-Taixé, Anton Milan, Ian Reid, Stefan Roth, and Konrad Schindler. Motchallenge 2015: Towards a benchmark for multi-target tracking. *arXiv preprint arXiv:1504.01942*, 2015.

[80] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.

[81] Jiwei Li, Will Monroe, Tianlin Shi, Sébastien Jean, Alan Ritter, and Dan Jurafsky. Adversarial learning for neural dialogue generation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2157–2169, 2017.

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

[83] Long-Ji Lin. Reinforcement learning for robots using neural networks. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.

[84] Wenhan Luo, Peng Sun, Fangwei Zhong, Wei Liu, Tong Zhang, and Yizhou Wang. End-to-end active object tracking via reinforcement learning. *arXiv preprint arXiv:1705.10561*, 2017.

[85] Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

[86] Tiago V Maia. Reinforcement learning, conditioning, and the brain: Successes and challenges. *Cognitive, Affective, & Behavioral Neuroscience*, 9(4):343–364, 2009.

[87] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pages 50–56, 2016.

[88] Tim Matthews, Sarvapali D. Ramchurn, and Ceorgios Chalkiadakis. Competing with humans at fantasy football: Team formation in large partially-observable domains. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, pages 1394–1400, 2012.

[89] Peter S Maybeck. *Stochastic models, estimation, and control.* Academic press, 1982.

[90] Anton Milan, S Hamid Rezatofighi, Anthony Dick, Ian Reid, and Konrad Schindler. Online multi-target tracking using recurrent neural networks. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[91] 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, 2016.

[92] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS) Deep Learning Workshop*, 2013.

[93] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7549):529–533, 2015.

[94] Igor Mordatch, Nikhil Mishra, Clemens Eppner, and Pieter Abbeel. Combining model-based policy search with online model learning for control of physical humanoids. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 242–248. IEEE, 2016.

[95] Jun Morimoto and Kenji Doya. Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning. *Robotics and Autonomous Systems*, 36(1):37–51, 2001.

[96] Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. Bridging the gap between value and policy based reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2775–2785, 2017.

[97] Brendan O'Donoghue, Ian Osband, Remi Munos, and Volodymyr Mnih. The uncertainty bellman equation and exploration. *arXiv preprint arXiv:1709.05380*, 2017.

[98] Shayegan Omidshafiei, Jason Pazis, Christopher Amato, Jonathan P How, and John Vian. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2681–2690. JMLR. org, 2017.

[99] Peter Ondruska and Ingmar Posner. Deep tracking: Seeing beyond seeing using recurrent neural networks. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[100] OpenAI. Openai five. `https://blog.openai.com/openai-five/`, 2018.

[101] Manfred Opper. A bayesian approach to online learning. *On-Line Learning in Neural Networks*, 1999.

[102] Charles Blundell Alexander Pritzel Osband, Ian and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In *Advances in Neural Information Processing Systems*, pages 4026–4034, 2016.

[103] Christos H Papadimitriou and John N Tsitsiklis. The complexity of markov decision processes. *Mathematics of operations research*, 12(3):441–450, 1987.

[104] Ramesh S Patil, Peter Szolovits, and William B Schwartz. Information acquisition in diagnosis. In *AAAI*, pages 345–348, 1982.

[105] Jan Peters, Sethu Vijayakumar, and Stefan Schaal. Reinforcement learning for humanoid robotics. In *Proceedings of the third IEEE-RAS international conference on humanoid robots*, pages 1–20, 2003.

[106] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 3406–3413. IEEE, 2016.

[107] Pascal Poupart, Nikos Vlassis, Jesse Hoey, and Kevin Regan. An analytic solution to discrete bayesian reinforcement learning. In *Proceedings of the 23rd International Conference on Machine Learning*, volume 20, pages 697–704, 2006.

[108] Robert A Rescorla, Allan R Wagner, et al. A theory of pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement. *Classical conditioning II: Current research and theory*, 2:64–99, 1972.

[109] Martin Riedmiller. Neural fitted q-iteration-first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning.* Springer Berlin Heidelberg, 2005.

[110] Paul E Rybski, Sascha A Stoeter, Michael D Erickson, Maria Gini, Dean F Hougen, and Nikolaos Papanikolopoulos. A team of robotic agents for surveillance. In *Proceedings of the fourth international conference on autonomous agents*, pages 9–16, 2000.

[111] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[112] Brent Schlotfeldt, Dinesh Thakur, Nikolay Atanasov, Vijay Kumar, and George J Pappas. Anytime planning for decentralized multirobot active information gathering. *IEEE Robotics and Automation Letters*, 3(2):1025–1032, 2018.

[113] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

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

[115] Wolfram Schultz, Peter Dayan, and P. Read Montague. A neural substrate of prediction and reward. *Science*, 275(5306):1593–1599, 1997.

[116] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

[117] Robert Sim and Nicholas Roy. Global a-optimal robot exploration in slam. In *Proceedings of the 2005 IEEE international conference on robotics and automation*, pages 661–666. IEEE, 2005.

[118] Bruno Sinopoli, Luca Schenato, Massimo Franceschetti, Kameshwar Poolla, Michael I Jordan, and Shankar S Sastry. Kalman filtering with intermittent observations. *IEEE transactions on Automatic Control*, 49(9):1453–1464, 2004.

[119] Stefano Soatto. Steps towards a theory of visual information: Active perception, signal-to-symbol conversion and the interplay between sensing and control. *arXiv preprint arXiv:1110.2053*, 2011.

[120] Malcome Strens. A bayesian framework for reinforcement learning. In *Proceedings of the 17th International Conference on Machine Learning*, pages 943–950, 2000.

[121] Jörg Stückler, Johannes Schwenk, and Sven Behnke. Getting back on two feet: Reliable standing-up routines for a humanoid robot. In *IAS*, pages 676–685, 2006.

[122] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* MIT press, 2018.

[123] Gerald Tesauro. Temporal difference leaerning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[124] William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

[125] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*, volume 1. MIT press Cambridge, 2000.

[126] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[127] John N. Tsitsiklis. On the convergence of optimistic policy iteration. *Journal of Machine Learning Research*, 3(Jul):59–72, 2002.

[128] Tsuyoshi Ueno, Yutaka Nakamura, Takashi Takuma, Tomohiro Shibata, Koh Hosoda, and Shin Ishii. Fast and stable learning of quasi-passive dynamic walking by an unstable biped robot based on off-policy natural actor-critic. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5226–5231. IEEE, 2006.

[129] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[130] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1995–2003, 2016.

[131] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[132] Steven D. Whitehead and Dana H. Ballard. Active perception and reinforcement learning. In *Machine Learning Proceedings 1990*, pages 179–188. Elsevier, 1990.

[133] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[134] Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Multi-task reinforcement learning: A hierarchical bayesian approach. In *Proceedings of the 24th International Conference on Machine Learning*, Corvallis, OR, 2007.

[135] Brian Yamauchi. A frontier-based approach for autonomous exploration. In *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97.'Towards New Computational Principles for Robotics and Automation'*, pages 146–151. IEEE, 1997.

[136] Seung-Joon Yi, Stephen McGill, Dennis Hong, and Daniel Lee. Hierarchical motion control for a team of humanoid soccer robots. *International Journal of Advanced Robotic Systems*, 13(1):32, 2016.

[137] Da Zhang, Hamid Maei, Xin Wang, and Yuan-Fang Wang. Deep reinforcement learning for visual object tracking in videos. *arXiv preprint arXiv:1701.08936*, 2017.

[138] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3357–3364. IEEE, 2017.

[139] Brian D Ziebart. Modeling purposeful adaptive behavior with the principle of maximum causal entropy. *Ph.D. Thesis*, 2010.