UNIVERSITY OF CALIFORNIA

Los Angeles

Trustable Deep Reinforcement Learning with Efficient Data Utilization

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Electrical and Computer Engineering

by

Zahra Mahmoodzadeh Poornaki

2020

ABSTRACT OF THE DISSERTATION

Trustable Deep Reinforcement Learning with Efficient Data Utilization

by

Zahra Mahmoodzadeh Poornaki
Doctor of Philosophy in Electrical and Computer Engineering
University of California, Los Angeles, 2020
Professor Ali Mosleh, Chair

We live in the era of big data in which the advancement of sensor and monitoring technologies, data storage and management, and computer processing power enable us to acquire, store and process over 2.5 Quintilian bytes of data daily. This massive data brings the necessity of using trustable and high-performance data-driven models that extract knowledge out of data. This dissertation focuses on learning to solve highly risk-averse and complex sequential decision-making problems from retrospective data sets by deep Reinforcement Learning (RL).

Deep RL has gained remarkable breakthroughs in many applications. It achieved superhuman performance in video and Atari games, defeated the world champion in game of Go, gained competent autonomy in simulated self-driving cars, and successfully learned to perform some robotic tasks. Despite all the notable advancements in deep RL, its application to real-world problems such as clinical treatment policy or industrial asset maintenance management is insignificant. Studies are underway to investigate deep RL use in realistic problems; however, none has been deployed in real-world settings. Several limitations hinder the deep RL application to real-world problems, among which trustability and excessive thirst for data are the main issues. This research is an effort to smooth the way of applying deep RL to real-world problems by addressing the above two limitations.

We first provide a concrete definition for trust in RL algorithms, and then we propose a sample efficient deep RL agent that computes a trustable solution to real-world sequential decision-making problems. The agent tackles the trust problem from two aspects. It imposes risk barriers to the RL agent's policy improvement process and provides off-policy performance estimation with a confidence bound prior to putting the agent in interaction with the actual system or environment. We address the RL significant demand for data by implementing the most advanced efficient data utilization techniques as well as deploying new techniques that improve the Trustable Deep RL sample efficiency.

The proposed methodology is tested and evaluated on a novel pipeline corrosion maintenance test bench that mimics the real system restrictions. The results witness that the Trustable Deep RL algorithm efficiently digests a retrospective data set from the pipeline environment and gains a superior and trustable interaction policy.

The dissertation of Zahra Mahmoodzadeh Poornaki is approved.

Jonathan Kao

Gregory Pottie

Alex Bui

Ali Mosleh, Committee Chair

University of California, Los Angeles

2020

To my parents

for giving me the opportunity to follow my dreams.

To my beloved husband, Ali,

for his endless care, support, and patience.

To my daughter, Nadia,

for reminding me what matters most every time I look at her.

and

to Professor B. John Garrick

founder of the UCLA Risk Sciences Institute.

May his generous and compassionate soul rest in peace.

Table of Contents

List of Figures

List of Tables

Acknowledgment

I would like to first express my sincere gratitude to my advisor, Professor Ali Mosleh, for his mentorship and support throughout my Ph.D. journey. He has taught me how to prioritize the big picture of the research prior to diving into the details. He always encourages us to practice critical thinking for every detail we come across in the research. I will forever cherish the lessons learned from him and continue to follow the example he sets as a scientist.

I also would like to thank other members of my committee, Professor Jonathan Kao, Professor Gregory Pottie, and Professor Alex Bui, who generously lent their time and expertise to my research. Their advice and feedback greatly enriched my work.

I am also indebted to my great collaborator, Professor Enrique López Droguett, for his invaluable suggestions on my research, his advice on paper writing, and his dedication to our shared projects.

Finally, I would like to thank my fellow doctoral students, lab members, postdocs, and friends at the Garrick Institute for the Risk Sciences, Joselyne Saldana, Dr. Keo-Yuan Wu, Dr. Elaheh Rabiei, Dr. Hassan Masoomi, Dr. Samaneh Balali, Professor Ramin Ramezani, Dr. Marilia Ramos, Dr. Tarannom Parhizkar, Anna Dong, Noor Nakhaei, Wadie Chalgham, Theresa Stewart, Lixian Huang, and Amruth Varshinee, with whom I shared most of my Ph.D. journey, lots of pleasant memories, and an energetic work environment.

Curriculum Vitae

2006 – 2011	B.Sc. Electrical Engineering, Sharif University of Technology, Tehran, Iran.
2011 – 2013	M.Sc. Electrical Engineering, Sharif University of Technology, Tehran, Iran.
2016 – Present	Ph.D. student in Electrical and Computer Engineering, University of California, Los Angeles (UCLA).

Publications

Z. Mahmoodzadeh, A. Mosleh, ”Trustable Deep Reinforcement Learning with Efficient Data Utilization,” in preparation

Z. Mahmoodzadeh, A. Mosleh, ”Development and Application of a Trustability Metric for Reinforcement Learning Approaches in Asset Integrity Management of Industrial Installations,” Annual Reliability and Maintainability Symposium (RAMS), Orlando, FL, US, 2021.

Z. Mahmoodzadeh, K-Y. Wu, E. Lopez Droguett, A. Mosleh, ”Condition-Based Maintenance with Reinforcement Learning for Dry Gas Pipeline Subject to Internal Corrosion,” Sensors, 2020, 20(19):5708.

Z. Mahmoodzadeh, K-Y. Wu, E. Lopez Droguett, A. Mosleh, ”Smart Condition-Based Maintenance for Internal Corrosion in Dry Gas Pipeline: a Reinforcement Learning Approach.” 30th European Safety and Reliability Conference (ESREL) and the 15th Probabilistic Safety Assessment and Management (PSAM) Conference, Venice, Italy. 2020.

Z. Mahmoodzadeh, S. Balali, A. Mosleh, “Entropy Based Method for Identification of Leading Risk Indicators,” Reliability and Maintainability Symposium (RAMS) 2018, Reno, NV, US, 2018.

Z. Mahmoodzadeh, N. Ghanbari, M. Ehsan, A. Mehrizi-Sani “Energy Loss Estimation in Distribution Networks Using Stochastic Simulation,” 2015 IEEE PES General Meeting, Denver, Colorado, US, 2015.

Z. Ghofrani-Jahromi, **Z. Mahmoodzadeh**, M. Ehsan, “Distribution Loss Allocation for Radial Systems Including DGs,” Power Delivery, IEEE Transactions on, vol.29, no.1, pp.72,80, 2014.

Z. Mahmoodzadeh, A. Mosleh, “Model-Based Identification of Gas Pipeline Risk Leading Indicator,” Technical report prepared for DNV GL and California Energy Commission, 2017.

M. Diaconeasa, **Z. Mahmoodzadeh**, A. Mosleh, “Commercial Off-The-Shelf (COTS) Usage in Space Systems: System Reliability Analysis Methodology and Software Platform,” Technical report prepared for prepared for JPL, August 2017.

CHAPTER 1

Introduction

1.1 Problem Statement

Within the next generation, autonomous decision-making agents could be part of our everyday lives in autonomous vehicles, assistive robots, personal-health guide applications, smart homes, intelligent stock trading, etc. These autonomous agents learn how to perform by either analyzing the previous information and data from their target environment or interacting with a virtual environment that mimics the target environment. In either case, some variant of Artificial Intelligence (AI) will teach the autonomous agents how to perform effectively.

Reinforcement Learning (RL) is an area of AI concerned with data-driven modeling and solving sequential decision-making problems under uncertainty. RL is recently attracting increasing attention from various research communities to some extent because of the same reasons that all the data-driven techniques are getting more attention, among which the following three reasons are playing the leading roles:

- Advances in monitoring technology: Advancements in sensors and monitoring technology enable us to gain high dimensional and high-resolution data from systems and environments [KS14]. Today's monitoring systems can measure previously known as inaccessible values and instantaneously transfer the high volume of measurement to storage facilities. Besides, sensors are becoming more accurate while cheaper every day. As a result, these days, even a small system is equipped with many sensors that can produce highly valuable data about the system's condition and operation.

- Advances in data storage and management systems: Recent advances in data storage and management systems capacitates us to store the abundance of historical data with lower cost more than ever. As a result, extremely large data sets become available from years of system or environment operation that provide great potentials to gain more knowledge about various aspects of the system or the environment [OBLB18].

- Advances in data processing power: Ever-increasing data processing speed and capabilities made the digestion of these overwhelming amounts of data computationally tractable [YHL+17]. Although today's data sources are richer both in breadth (abundance of data) and depth (higher resolution and dimensional), the prosperity of data-driven models would not be feasible without the great progress in computational processing power.

Aside from the recent popularity of all data-driven approaches, RL has been in the center of AI research attention for the past fifty years because of the distinguishing characteristics that make it well suited to formulate real-world sequential decision-making problems. The capability of dealing with delayed effects as well as interacting and learning in a stochastic environment are the main two among many superior characteristics of RL.

Significant progress has been achieved in applying the RL, and particularly deep RL, to video and Atari games to the extent that the recent deep RL algorithms can achieve superhuman performance in those games [New18]. Other outstanding achievements of deep RL algorithms include defeating the world's champion in game of Go [New17], skillful performance in robot table tennis [MJL+18], and an adequate level of autonomy in simulated self-driving cars.

Despite all the outstanding achievements of RL, its application to real-world problems is still very limited. Most RL success is in problems with virtual environments or accurate simulators that have the following unrealistic characteristics. First, these simulated environments have provided the researchers with the luxury of unlimited data, which is never accessible in real-world problems. Second, playing wrong actions in the simulated environments is inconsequential and imposes no risk at all. Third, the data generation speed in the simulated environments is in the order of microseconds, whereas, in real-world problems, it might be several orders of magnitude larger.

There are many other issues in applying RL to real-world problems such as state definition and state visibility, credit assignment, and no online interaction. However, we believe its excessive thirst for data and the RL agent's trustability are the main obstacles towards the real-world deployment. It is known that RL is the most data-hungry machine learning algorithm; therefore, data inefficiency is a significant barrier in implementing RL algorithms to real-world problems. Moreover, most real applications, such as medical applications, are highly risk-averse, i.e., the decision-maker is always obligated to pick the best actions. In such environments, the risks of taking sub-optimal actions are intolerable; therefore, deployment of an RL-based decision-maker is impossible unless its trustability is guaranteed.

In this research, we specifically focus on the two aforementioned problems. We address the significant demand for data by implementing most of the techniques recommended in the literature to increase the sample efficiency of deep RL. These techniques are elaborated extensively in chapters 2 and 4. Besides, in chapter 6, we propose three novel techniques for increasing the data efficiency.

We address the trustability issue by first providing a concrete definition for the RL agent's trustability, i.e., it must act safely with performance guarantees. Safety and performance guarantees have been discussed separately in the literature. We brought the related background work in chapter 3. Our contribution is to unify them and tie them together and bring novelty to both, as presented in chapter 6.

This dissertation is at the intersection of three distinct arenas of research, namely deep RL's sample efficiency, RL's safety, and RL's off-policy performance evaluation. Hence, it requires a balance presentation of the state of the art in each arena. Chapters 2 to 4 mostly focus on detailed assessment of the existing approaches to each of the three aforementioned lines of research. The main contribution of the dissertation, the Trustable Deep RL algorithm, is presented and explained in chapter 6. This algorithm encompasses the data efficiency, safety, and performance guarantees' novelties of the dissertation. It efficiently digests a retrospective data set from the environment transitions to gain an improved interaction policy.

Afterward, the new policy passes through the safety constraints to ensure the policy would take no risky actions. Finally, the resulting policy's performance is evaluated and estimated to give the users performance guarantees prior to any level of deployment.

The proposed algorithm is tested on a pipeline corrosion environment to determine a trustable and optimal corrosion maintenance management policy. This environment is developed as part of this research and presented in chapter 7. Although the test environment is a simulated environment, we used that with all the constraints that a real environment would pose to data generation and algorithm testing.

1.2 Contributions

The contributions of this dissertation are summarized as follows:

1. Conceptualizing the RL agents' trustability phenomena.
2. Proposing the first trustable deep RL algorithm.
3. Implementing a deep RL method on maintenance optimization of pipeline for the first time.
4. Proposing a methodology for initializing the actor network and the critic network weights separately for increasing data efficiency of deep actor-critic networks.
5. Proposing a batch estimator for the lower confidence bound of the off-policy performance evaluation based on concentration inequalities.
6. Implementing a test bench for interaction evaluation between a maintenance decision-maker and a corrosive pipeline.

1.3 Dissertation Organization

The rest of the dissertation is organized as follows:

- Chapter 2 introduces the RL's preliminaries and key concepts, through which the notations used throughout this dissertation is presented.
- Chapter 3 provides a literature review on two aspects of RL trustability, i.e., RL safety and off-policy evaluation methods.
- Chapter 4 studies the recent advances in deep RL and explains the data efficiency techniques used in the deep RL agent of the proposed Trustable Deep RL algorithm.
- Chapter 5 demonstrates the architecture and the implementation details of the utilized deep RL agent.
- Chapter 6 presents the main contribution of this research, i.e., the Trustable Deep RL algorithm.
- Chapter 7 introduces the testing environment of our methodology and how a decision-making agent would be able to interact with the environment.
- Chapter 8 concentrates on presenting and interpreting the results of applying the methodology in the test environment.
- Chapter 9 summarizes the work and proposes avenues for future work.

CHAPTER 2

Preliminaries and Notation

Reinforcement Learning (RL) is a computational approach to learning from experience in a setting that a goal-oriented learner is sequentially interacting with an environment to achieve its goal. The objective of applying RL is to optimize (or improve) the interaction policy of the learner. The underlying idea of RL theory comes from behavioral sciences stating that in our interactions with the surrounding environment, we always adjust our behavior based on the feedback we receive from that. For example, when trying to hold a conversation with a new colleague or attending a job interview, we continuously observe the environment or another person responds to what we do and say, and we strive to have better performance or behavior by analyzing the responses. These examples serve as evidence of why learning from experience is a fundamental idea behind all theories of learning and intelligence.

In artificial intelligence, RL is a class of methods for modeling and solving sequential decision-making problems with delayed effects. In the general RL setting, a decision-maker (often called the agent) interacts with an uncertain environment to achieve a goal. The actions influence the environment, and the agent can observe the influence through some observation signals as well as a reward signal. The reward signal should reflect the aptness of the previous actions, and the agent should improve and optimize its decisions by learning from the previous rewards. In this chapter, first we formally describe RL and the mathematical representations of its main elements. Afterward, we review and explain the two significant RL algorithms, namely Q-learning and actor-critic, that would build the foundation for chapters 4, 5, and 6.

2.1 Notation Standardization

Prior to getting into the mathematical details, it is necessary to standardize the notation for this chapter and the entire dissertation.

- Sets are denoted by calligraphic capital letters, e.g., $\mathcal{X}$.
- Elements of sets are written in corresponding lowercase letters similar to the set they belong to, e.g., $x \in \mathcal{X}$.
- Random variables are shown with capital letters, e.g., X.
- Particular realization of random variables are written in corresponding lowercase letters, e.g., x.
- Vector quantities are denoted by bold characters, e.g., $\boldsymbol{\theta}$.
- The estimate of a value is indicated by a circumflex above the value. e.g., $\hat{J}$ is an estimate for J.
- The support set of a probability distribution function f is denoted by $suppf$, i.e., $\{x : f(x) \neq 0\}$.
- $\mathbb{E}_{\pi,E}\left[.\right]$ denotes the expectation operator over action selection according to the policy, π, and the reward generation and the state transition according to the environment, E.

2.2 Markov Decision Process

Markov Decision Process (MDP) formally describes a framework for RL. MDP is a mathematical abstraction that is defined by a tuple $< \mathcal{S}, \mathcal{A}, R, T >$ in which:

1. $\mathcal{S}$ is a set of states. The state is some representation of the environment that is perceived by the agent, and the agent chooses the next action on that basis. s denotes an element of $\mathcal{S}$, and S_t is a random variable representing the state that occurs at time t.

2. $\mathcal{A}$ is the set of possible actions that the agent can select actions from that. a denotes an element of $\mathcal{A}$, and A_t is a random variable representing the action that is taken at time t.

3. R is the reward function that governs how the scalar reward R_t is produced at time t. In MDP, we assume that the reward depends only on S_t, A_t, and S_{t+1}:

$$R(r|s, a, s') := Pr\big(R_{t+1} = r|S_t = s, A_t = a, S_{t+1} = s'\big). \tag{2.1}$$

4. T is the transition function that governs how the environment transitions between states. In MDP, we assume that the next state depends only on S_t and A_t,

$$T(s'|s, a) := Pr\big(S_{t+1} = s'|S_t = s, A_t = a\big). \tag{2.2}$$

At each time step, the agent arrives at the state S_t. Then according to a policy, it decides to perform the action A_t which takes it to the next state S_{t+1} with a probability derived from $T(S_t, A_t)$. Following this transition, the agent receives a reward derived from $R(S_t, A_t, S_{t+1})$. The sequence of states, actions, rewards up until time t is called the history till time t and is denoted as

$$H_t := \big(S_0, A_0, R_1, S_1, A_1, R_2, S_2..., S_{t-1}, A_{t-1}, R_t, S_t\big). \tag{2.3}$$

The two Markov assumptions in MDP are:

- The next state is independent of the full past history and only depends on the last state and the last action.

- The next reward is independent of the full past history and only depends on the last state and the last action and the arriving state.

Often MDPs are used as discounted MDP (DMDP) for RL problems. DMPD is defined with the same tuple except that it has one extra element, which is the discount factor $\gamma \in [0, 1]$. By discounting future rewards with γ, we can determine the present value of future rewards. In this dissertation, we use DMDP and MDP interchangeably, and both are referring to DMPD.

2.3 RL Agent Objective and Policy

The reward signal is a scalar value emitted from the environment that tells the agent how good or bad its action was. While the reward signal indicates the immediate sense of worthiness of an action, the agent endeavor is to maximize its cumulative reward. To better understand the agent's objective, consider a physician who knows a particular medication has an immediate adverse effect on the general well-being of her patient while, in the long run, improves his or her health condition. In this situation, the physician prescribes the medication, knowing that although there is no immediate reward for her action, this will pay off in the long run.

In RL literature, the cumulative discounted reward is called return, and the return for a completed trajectory in an episodic task is formally defined as

$$G(H_L) := \sum_{t=0}^{L} \gamma^t R_{t+1}, \tag{2.4}$$

where H_L is a trajectory terminated after L time steps and can be represented as follows:

$$H_L := \big(S_0, A_0, R_1, S_1, A_1, R_2, ..., S_{L-1}, A_{L-1}, R_L, S_L\big). \tag{2.5}$$

The agent's objective is to obtain the largest expected return. There is a philosophical dilemma in this way of defining the RL agent's objective, which is considering the purpose of the agent from interacting with the environment is to achieve a goal, is it always possible to express any goal as the expected cumulative discounted reward? In this research, we assume that this statement is true, and no matter what the goal of agent-environment interaction is, it can always be mathematically represented as the expected return.

How the agent chooses between actions is defined in its policy. The policy is the agent's decision-making mechanism that can adequately define its behavior. Formally, it is a mapping from states to probabilities of selecting each possible action.

$$\pi(a|s) = Pr\big(A_t = a|S_t = s\big) \tag{2.6}$$

A policy is known to be better than another policy if the expected return of the episodes when using the first policy is larger than the expected return of the episodes when using the second policy. Hence, we can say that the agent's goal is to find a policy that maximizes its expected return. Therefore, the expected return is the agent's objective function and is formalized as follows:

$$J(\pi) := \mathbb{E}[G(H_L)|H_L \sim \pi], \;\; J : \Pi \to \mathbb{R}. \tag{2.7}$$

It is shown in [SB18] that there is always at least one policy that performs better than or equal to all other policies. This policy is known as the optimal policy, and here we denote it as π_*.

2.4 Value-Based Methods

There are two approaches to solve the RL problems. The first approach aims to learn the values of actions (or states) and then select actions based on their estimated value functions. In this approach, the policy is typically defined implicitly through the value functions. On the other hand, the second approach, the policy-based approach, aims at learning the policy directly without consulting a value function, which is the topic of Section 2.5.

The value-based methods are pioneer methods of solving RL problems. The core idea in all value-based methods is the agent is performing optimally if it always takes actions that yield the highest long term returns. Therefore, the pivotal question shifts from "what is the optimal policy" to "what is the best action at each step." Value functions are defined to answer the latter question quantitatively.

2.4.1 Value Functions

The *state-value function* is a function from the state space $\mathcal{S}$ to $\mathbb{R}$ that estimates how good it is for the agent to be in a state s, and thereafter following policy π. The mathematical representation of the notation "good" is defined in terms of the cumulative discounted future rewards that can be expected if the agent starts from a state s and continues choosing actions according to the policy π until it hits the terminal state.

$$V_\pi(s) = \mathbb{E}_{\pi,E}\Bigg[\sum_{k=0}^{terminal} \gamma^k R_{t+k+1}|S_t = s\Bigg], \;\; for\ all\ s \in \mathcal{S}, \tag{2.8}$$

in which the expectation is performed over action selection according to the policy as well as the reward generation and the state transition according to the environment. We use the brief notation for $\mathbb{E}_{A_i\sim\pi;S_i,R_i\sim E}[.]$ as $\mathbb{E}_{\pi,E}[.]$ in equation 2.8 and the entire dissertation.

Similarly, *action-value function* is a function from $\mathcal{S} \times \mathcal{A}$ to $\mathbb{R}$ that estimates how good it is for the agent to be in state s, performing action a, and thereafter following policy π. The definition of the action-value function is

$$Q_\pi(s,a) = \mathbb{E}_{\pi,E}\Bigg[\sum_{k=0}^{terminal} \gamma^k R_{t+k+1}|S_t = s, A_t = a\Bigg], \;\; for\ all\ s \in \mathcal{S},\ a \in \mathcal{A}. \tag{2.9}$$

It is apparent that the state-value function and the action-value function are distinct for different policies. When the agent follows the optimal policy, the respective state-value function and action-value function are also optimal, i.e.,

$$V_*(s) = \max_\pi V_\pi(s),\ Q_*(s,a) = \max_\pi Q_\pi(s,a), \;\; for\ all\ s \in \mathcal{S},\ a \in \mathcal{A}. \tag{2.10}$$

2.4.2 Q-Learning

Q-learning is a value-based RL solution method that is simple yet highly efficient in solving RL problems. Although Q-learning is one of the most vanilla RL algorithms, it is very popular in various fields spanning from healthcare [ZKZ09] to financial markets [NFK06]. The popularity of Q-learning is due to its simplicity in formulation, implementation, and analysis, as well as having reasonable computational and memory costs. But more importantly, the Q-learning algorithm is model-free, which makes it highly favorable for using in data-driven approaches, and consequently opens up the opportunity for researchers in various domains to use Q-learning on retrospective data sets.

Q-learning is first proposed by Watkin [Wat89], and over the years, evolved from a simple algorithm that can only solve problems with small and discreet state and action spaces, to more complex algorithms, e.g., deep Q-networks that is capable of learning how to play multiple video games from the raw video frames. It aims at approximating the optimal policy from estimates of Q_*. To better convey the Q-learning algorithm, we first need to explain the Bellman optimality equations. The Bellman optimality equation for the optimal value function states that the value of a state under an optimal policy is equal to the expected return for the best action from that state. Knowing that the best action at state s is $\underset{a}{\operatorname{argmax}}\, Q_*(s,a)$, we can formally state the Bellman optimality equation for the optimal value function as

$$V_*(s) = \max_a Q_*(s,a). \tag{2.11}$$

The Bellman optimality equation for the action-value function can be driven by representing the action-value function in terms of state-value function. For any policy π, one can calculate the $Q_\pi(s,a)$ by

$$Q_\pi(s,a) = \mathbb{E}_{\pi,E}\big[R_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s, A_t = a\big]. \tag{2.12}$$

By replacing the policy π with the optimal policy and substituting $V_*(S_{t+1})$ from 2.11, we arrive at the Bellman optimality equation for the optimal action-value function.

$$Q_*(s,a) = \mathbb{E}_{\pi,E}\big[R_{t+1} + \gamma \max_{a'} Q_*(S_{t+1},a') | S_t = s, A_t = a\big] \tag{2.13}$$

Equation 2.13 is the pivotal equation used to update the estimate of optimal action-value functions at each iteration of the Q-learning algorithm, which is presented in the following.

Assuming that Q_* is known for all the state-action pairs, the optimal policy is to always pick the action with the best value

$$\pi_*(a|s) = \begin{cases} 1 & if \;\; a = \underset{a}{\operatorname{argmax}}\, Q_*(s,a) \\ 0 & otherwise \end{cases} \qquad for\ all\ s \in \mathcal{S},\ a \in \mathcal{A}. \tag{2.14}$$

Q-learning is one of the temporal difference (TD) algorithms in which the estimate of an action-value function in each step is updated toward a better estimate. In Q-learning, we start from an initial estimate of Q_* for all state-action pairs. Then at each step of the episode, we take an action according to the optimal policy in equation 2.14 and our current estimate of Q_*s. Afterward, we receive a reward and observe the new state S_{t+1}. By incorporating these two new pieces of information into the $Q_*(S_t, A_t)$ estimate, we can arrive at a better estimate of that. The new estimate of $Q_*(S_t, A_t)$ is called the TD_{target} and is directly derived from 2.13. The difference between the TD_{target} value and the previous estimate of the $Q_*(S_t, A_t)$ is known as the TD_{error}.

$$TD_{target} = R_{t+1} + \gamma \max_{a'} Q_*(S_{t+1}, a') \tag{2.15}$$

$$TD_{error} = TD_{target} - Q_*(S_t, A_t) = R_{t+1} + \gamma \max_{a'} Q_*(S_{t+1}, a') - Q_*(S_t, A_t) \tag{2.16}$$

After arriving at the TD_{error}, the value of the $Q_*(S_t, A_t)$ gets updated in the direction of TD error with a step size α.

This implementation of the Q-learning is known as greedy Q-learning, which does not address the exploration-exploitation dilemma. When the Q-learning chooses the optimal action according to the current estimate of Q_* values, it exploits the current knowledge about the actions' worthiness. However, there might be other actions with higher actual value functions that are not chosen because the current estimates of the value functions are not accurate. Therefore, the agent needs to balance between exploiting the current knowledge about the best actions and exploring the other actions to gain more accurate estimates. The exploration-exploitation trade-off is a very well known dilemma in RL literature and has been the subject of many research studies for the past 50 years [Tok10, ALL+09, SYH+18].

The exploration-exploitation trade-off in Q-learning is addressed by altering the policy from an absolute greedy policy to an epsilon-greedy policy, i.e., at each step, the agent chooses the current best action by $1 - \epsilon$ probability, and chooses randomly among the other actions by ϵ probability. The pseudocode for the epsilon-greedy Q-learning is shown in algorithm 1.

$$\pi_\epsilon(a|s) = \begin{cases} 1 - \epsilon \quad if \;\; a = \underset{a}{\operatorname{argmax}}\, Q_*(s, a) \\ \frac{\epsilon}{|\mathcal{A}_s| - 1} \quad otherwise \end{cases} \qquad for\ all\ s \in \mathcal{S},\ a \in \mathcal{A}. \tag{2.17}$$

Algorithm 1 On-policy epsilon-greedy Q-learning

```
Require: Step size $\alpha \in [0, 1]$, reward discount factor, $\gamma$
 1: Initialize $Q_*(s, a)\ \ for\ \forall s \in \mathcal{S},\ \forall a \in \mathcal{A}$
 2: for each episode do
 3:     t=1
 4:     Observe $S_t$
 5:     while $S_t \neq$terminal do
 6:         Choose $A_t$ from $\mathcal{A}(S_t)$ according to equation 2.17
 7:         Take action $A_t$ and observe $R_{t+1}$ and $S_{t+1}$
 8:         $Q_*(S_t, A_t) \leftarrow Q_*(S_t, A_t) + \alpha\big[R_{t+1} + \gamma \max_{a'} Q_*(S_{t+1}, a') - Q_*(S_t, A_t)\big]$
 9:         $t \leftarrow t + 1$
10:     end while
11: end for
```

In RL taxonomy, algorithm 1 is categorized as an on-policy algorithm because the agent is following its own policy when choosing which action to take in response to the environment state transition and reward. The on-policy algorithms do not apply to many problems because of various restrictions and limitations. For example, in many risk-sensitive applications, the interaction of a naive RL agent with the environment is impermissible. In other words, we should assure that the agent's policy is not risky, if not optimal, to allow its interaction with the environment. In some other applications, the agent is required to learn from historical data, and therefore, there is no real-time interaction at all to obey the agent's policy.

Apart from the type of application and its restrictions, off-policy learning has the additional advantage of enabling experience replay. The idea of experience replay is to store and reuse past experiences. In practice, the state of the art RL algorithms implement experience replay to enable learning from batches of transition pairs instead of only one transition sample, which consequently reduces the sequence of samples correlations and improves sample efficiency. This concept is discussed further in chapter 4.

Another important competence of Q-learning that capacitates it to serve as the backbone of the new advanced RL algorithms is that it is adaptable to an off-policy setting. The only difference between the on-policy and off-policy Q-learning algorithms is that in off-policy Q-learning, the agent does not have the freedom to chose the next action. Instead, it just reads the next action from the history of interactions. However, the update rule is still similar to the on-policy Q-learning, which is to update the Q-value in the TD_{error} direction by α step size. The pseudocode for the off-policy Q-learning is shown in algorithm 2.

Algorithm 2 Off-policy Q-learning

```
Require: Step size $\alpha \in [0,1]$, reward discount factor, $\gamma$
Require: A set of retrospective trajectories, $\mathcal{H}$
 1: Initialize $Q_*(s,a)\ \ for\, \forall s \in \mathcal{S},\ \forall a \in \mathcal{A}$
 2: for each trajectory in $\mathcal{H}$ do
 3:     t=1
 4:     Read $S_t$
 5:     while $S_t \neq$terminal do
 6:         Read $A_t$, $R_{t+1}$, $S_{t+1}$
 7:         $Q_*(S_t,A_t) \leftarrow Q_*(S_t,A_t) + \alpha\big[R_{t+1} + \gamma \max_{a'} Q_*(S_{t+1},a') - Q_*(S_t,A_t)\big]$
 8:         $t \leftarrow t+1$
 9:     end while
10: end for
```

In this research, the off-policy Q-learning is used because we target the real-world sequential decision-making problems, and in this class of problems the agent cannot have online interaction with the environment to learn the policy. The biggest caveat of Q-learning is that the convergence of Q-learning to the optimal policy is subject to the condition that all state-action pairs continue to be visited. Although this condition is not feasible when dealing with an off-line interaction data set, we empirically observe that Q-learning mostly arrives at fairly good policies.

2.5 Policy-Based Methods

The goal of all RL solution methods is to find an optimal decision-making strategy for the agent to achieve the maximum long-run return. The policy-based methods target modeling and optimizing the decision making strategy directly. They tackle this problem by defining a parameterized policy, $\pi(a|s;\boldsymbol{\theta})$, and to find the vector of $\boldsymbol{\theta}$ parameters that yield to a policy with maximum return.

Most policy-based methods use the gradient of the return (or some other performance measures) with respect to the policy parameters to optimize the policy. Therefore, they are repeatedly referred to as policy gradient methods. Here in this thesis, we defined the agent's objective function as the expected return over episodic trajectories ($J(\pi)$ in equation 2.7), and at each step of the gradient ascent, we update the policy parameters, $\boldsymbol{\theta}$, toward the gradient of $J(\pi)$ with step size α'. (We choose α' to distinguish between the temporal difference learning step size, α, and the gradient ascent step size.)

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha' \, \nabla_{\boldsymbol{\theta}} J(\pi(a|s;\boldsymbol{\theta})) \tag{2.18}$$

Assuming there is an initial value for $\boldsymbol{\theta}$ and a differentiable distribution function for the policy, the problem would narrow down to approximate the gradient of the expected return, $J(\pi)$, with respect to $\boldsymbol{\theta}$. In essence, this is the purpose of all the policy gradient algorithms.

There are numerous advantages to applying policy gradient methods. One advantage is that the policy may be a more straightforward function to approximate. Problems have different structures in complexity. For some, the value functions could be simpler to approximate, and for others, the policy might be more straightforward [SB18]. Another advantage is in case of the existence of substantial prior knowledge; the policy parametrization could be a good way of injecting the prior knowledge into the reinforcement learning system.

Presumably, the most important advantage is policy gradient methods are easily extendable to large action spaces or even continuous action spaces with an infinite number of actions. In this research, we are dealing with a finite set of actions, and therefore, we are not fully exploiting this property of policy gradient methods. However, structuring our methodology on a policy gradient method enables the effortless extension of the methodology to applications with continuous action spaces, e.g., medication dosage administration.

A reasonable and common way of policy parameterization for discreet action spaces is first to define a numerical preference for each action at any state, and then assign probabilities of taking each action on the grounds of its preference value. We denote the action preference as $H(s, a; \boldsymbol{\theta}) \in \mathbb{R}$. Later, the actions with the highest preferences are given the highest probabilities according to a distribution function, e.g., softmax.

$$\pi(a|s, \boldsymbol{\theta}) = \frac{e^{H(s,a;\boldsymbol{\theta})}}{\sum\limits_{a_i \in \mathcal{A}(s)} e^{H(s,a_i;\boldsymbol{\theta})}} \tag{2.19}$$

One may claim that actions could be selected according to a softmax distribution which inputs the action-values instead of the action preferences. This approach is taken in some studies, e.g., [PSB+18, STK18, SML+15]. However, it should be noted that the action-value function is not defined to express the preference among the actions. In other words, they may implicitly show the advantage of actions over the others, but depending on the type of the problem, their numeric values might not be a proper input for the action probability assignment.

For example, in a case that the Q-values of various actions are scattered around a large value with small variation, the softmax of Q-values would result in misleading probabilities that are not capable of showing the true advantage of actions over each other. Table 2.1 demonstrates an example of the pointed issue. Assuming that Q_* values are the optimal action-values that are provided by an oracle, the optimal policy is always selecting a_2, which translates to probability 1 for a_2 and 0 for a_1 and a_3. However, the softmax output assigned more even probabilities to the actions.

In many applications, the low absolute difference between various action-value functions in a state conveys that the future reward collection at that specific state is mostly associated with the state, and the choice of action could not drastically change the future rewards. Therefore, several studies, for example [CW18, WSH+16, IGW18, ESM+18], define the preference function (also known as advantage function) as the state-value function subtracted from the Q-values,

Table 2.1: An example of Q-values limitation for using with softmax function.

Actions	Q_* **value**	**Softmax Probabilities**	π_* **Probabilities**
a_1	1000	0.307	0
a_2	1000.5	0.506	1
a_3	999.5	0.186	0

equation 2.20.

$$H(s,a) = Q(s,a) - V(s) \tag{2.20}$$

A highly important theoretical advantage of using policy gradient methods is that stronger convergence guarantees are available for them than the value function methods. With continuous policy parametrization, the probabilities of actions alter smoothly as a function of the iteratively learning parameters. Whereas in action selection based on action-values, the probabilities could change drastically for an arbitrary small change in estimated values. More precisely, when the i^{th} action has the highest value, it is picked with probability 1 (or $1-\epsilon$). If in the next iteration the j^{th} action achieves the maximal value, the probability of action i abruptly decreases to 0 (or $\frac{\epsilon}{n}$), and probability of action j increases to 1 (or $1-\epsilon$). This drives the attention to another practical advantage of policy gradient that finding the optimal policy with policy gradient methods is much less costly than epsilon-greedy exploration for problems with large action spaces.

To leverage several advantages of policy-based methods and practically implement them, we need to find a way to approximate the gradient of the objective function, $J(\pi)$ with respect to $\boldsymbol{\theta}$. This could be challenging because the expected return depends on both the action selection policy and the environment's transitions among the states. While we can easily parameterize the policy and derive the required gradient, the environment's dynamics are unrevealed to the agent. Fortunately, the policy gradient theorem provides an excellent theoretical answer to this challenge.

2.5.1 The Policy Gradient Theorem

The policy gradient theorem provides a useful reformation of the objective function's derivative to involve only the gradient of the policy. It greatly simplifies the gradient computation because it does not require the derivative of the state distribution. The policy gradient theorem for problems with episodic trajectories states that

$$\nabla_{\boldsymbol{\theta}}\, J\big(\pi(a|s;\boldsymbol{\theta})\big) \;\propto\; \sum_{s} \mu_\pi(s) \,\sum_{a} Q_\pi(s,a) \nabla_{\boldsymbol{\theta}} \pi(a|s;\boldsymbol{\theta}), \tag{2.21}$$

in which the symbol $\propto$ means "proportional to". In this equation, $\mu_\pi(s)$ is the stationary distribution of states under $\pi_{\boldsymbol{\theta}}$, which is defined as

$$\mu_\pi(s) = \frac{\eta_\pi(s)}{\sum\limits_{s'} \eta_\pi(s')}, \tag{2.22}$$

where $\eta_\pi(s)$ is the expected number of state s's visits under the policy. The detailed proof of the policy gradient theorem (equation 2.21) is provided in [SB18], and we refer the interested reader to study through the proof in chapter 13 of the 2018 edition.

Equation 2.21 offers an exact expression proportional to the gradient. However, it is still one step away from providing a practical estimation of the gradients from samples. It includes the states' distribution, but the sum over actions is not weighted by $\pi(a|s;\boldsymbol{\theta})$, as it is needed for an expectation under π. By multiplying and then dividing the terms inside the summation by $\pi(a|s;\boldsymbol{\theta})$, we introduce the necessary weighting without changing the proportionality.

$$\nabla_{\boldsymbol{\theta}}\, J\big(\pi(a|s;\boldsymbol{\theta})\big) \;\propto\; \sum_{s} \mu_\pi(s) \,\sum_{a} \frac{\pi(a|s;\boldsymbol{\theta})}{\pi(a|s;\boldsymbol{\theta})}\, Q_\pi(s,a) \nabla_{\boldsymbol{\theta}} \pi(a|s;\boldsymbol{\theta}) \tag{2.23}$$

$$\propto \sum_{s,a} \mu_\pi(s)\, \pi(a|s;\boldsymbol{\theta})\, Q_\pi(s,a)\, \frac{\nabla_{\boldsymbol{\theta}} \pi(a|s;\boldsymbol{\theta})}{\pi(a|s;\boldsymbol{\theta})} \tag{2.24}$$

$$\propto \mathbb{E}_{\pi,E}\left[Q_\pi(s,a)\, \nabla_{\boldsymbol{\theta}} \big[\ln\, \pi(a|s;\boldsymbol{\theta}) \big] \right] \tag{2.25}$$

Expression 2.25 is a quantity that can be sampled on each time step with expectation proportional to the gradient. Consequently, at each iteration of the gradient ascent,

$\nabla_{\boldsymbol{\theta}}\, J\big(\pi(a|s;\boldsymbol{\theta})\big)$ could be replaced with equation 2.25 since it provides an approximation for the gradient directly from one or a set of transition samples. The proportionality constant is not a concern because it could be absorbed into the gradient ascent step size, α'. Finally, equation 2.26 represents an update rule in a stochastic policy gradient ascent algorithm.

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha' \Bigg[Q_{\pi}(S_t, A_t)\, \nabla_{\boldsymbol{\theta}} \big[\ln\, \pi(A_t|S_t;\boldsymbol{\theta})\big] \Bigg] \tag{2.26}$$

The intuitive interpretation of equation 2.26 is the update rule increases the parameter vector in the direction of $\nabla_{\boldsymbol{\theta}}\pi(A_t|S_t;\boldsymbol{\theta})$ proportional to $\frac{Q(S_t,A_t)}{\pi(A_t|S_t;\boldsymbol{\theta})}$. $\nabla_{\boldsymbol{\theta}}\pi(A_t|S_t;\boldsymbol{\theta})$ is the direction in the parameters' space that most increases the probability of repeating the action A_t in the future visits of S_t. The increment is proportional to $Q(S_t, A_t)$ (or as we see later, some other representation of the return) since it causes the parameter to move most in the direction that favors actions with the highest return. The increment is also inversely proportional to the action probability because otherwise, actions that are selected frequently are at an advantage.

2.5.2 REINFORCE

REINFORCE [Wil92] is the first and the most vanilla policy gradient learning algorithm that finds an unbiased estimate of the gradient without the assistance of a learned value function. It is a Monte Carlo algorithm because it uses the whole trajectory's rewards from time t to the end to compute the parameters vector update. To derive REINFORCE update rule, we start from approximating $Q(S_t, A_t)$ in equation 2.26 by G_t because $Q(S_t, A_t) = \mathbb{E}_{\pi,E}\big[G_t|A_t, S_t\big]$. Algorithm 3 presents the pseudocode of REINFORCE [SB18].

As a stochastic gradient algorithm, REINFORCE enjoys good theoretical convergence properties. However, as a Monte Carlo algorithm, it is prone to high variance because the return is the sum of several random variables, $R_1, ..., R_{terminal}$. Moreover, each step's reward depends on the policy's choice of action and the state transition dynamics by the environment. In fact, REINFORCE learns much more slowly than value-based methods and has received relatively little attention [SMSM00].

Algorithm 3 REINFORCE

```
Require: gradient ascent step size $\alpha' > 0$
Require: reward discount factor $\gamma$
Require: a differentiable policy parameterization $\pi(A_t|S_t;\boldsymbol{\theta})$
Require: a set of retrospective trajectories $\mathcal{H}$
1: Initialize $\boldsymbol{\theta}$
2: for each trajectory in $\mathcal{H}$ do
3:     t=1
4:     while $S_t \neq$terminal do
5:         $G \leftarrow \sum_{k=t}^{terminal} \gamma^{k-t} R_{k+1}$
6:         $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha' \, \gamma^t \, G \, \nabla_{\boldsymbol{\theta}} \ln \, \pi_{\boldsymbol{\theta}}(A_t|S_t)$
7:         $t \leftarrow t + 1$
8:     end while
9: end for
```

2.5.3 The Policy Gradient Theorem with Baseline

The iterative update rule of equation 2.26 is susceptible to high variance in the policy gradient estimation because it entails a return-based factor. As pointed out in the REINFORCE algorithm, the return is a summation of several random variables and could result in high variance gradient estimation. In the literature, it is suggested to subtract a baseline value from the action-value function, Q_π, to reduce the variance of the gradient estimation while keeping the bias unchanged. We refer the interested reader to [WT01] and [GBB04] for the detailed proof of why the baseline could help reduce the variance and find the optimal baseline. However, here we demonstrate why the policy gradient theorem, equation 2.21, remains valid. We assume the baseline to be any function, even a random variable, that is not a function of action, $b(s)$. As long as $b(s)$ does not depend on action, we can rewrite the policy gradient theorem as follows:

$$\sum_s \mu_\pi(s) \sum_a \Big(Q_\pi(s,a) - b(s)\Big) \nabla_{\boldsymbol{\theta}} \pi(a|s;\boldsymbol{\theta}), \tag{2.27}$$

$$= \sum_s \mu_\pi(s) \Big(\sum_a Q_\pi(s,a) \nabla_{\boldsymbol{\theta}} \pi(a|s;\boldsymbol{\theta}) - \sum_a b(s) \nabla_{\boldsymbol{\theta}} \pi(a|s;\boldsymbol{\theta}) \Big). \tag{2.28}$$

Since $b(s)$ is not a function of action, we can rewrite the second term inside the large parenthesis as:

$$\begin{aligned} \sum_a b(s) \nabla_{\boldsymbol{\theta}} \pi(a|s;\boldsymbol{\theta}) &= b(s)\ \nabla_{\boldsymbol{\theta}} \sum_a \pi(a|s;\boldsymbol{\theta}) \\ &= b(s)\ \nabla_{\boldsymbol{\theta}} 1 \\ &= 0. \end{aligned} \tag{2.29}$$

Consequently, equation 2.27 is equal to the policy gradient theorem, equation 2.21. The general form of the iterative update rule of any policy gradient algorithm with baseline is presented in equation 2.30.

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha' \Big[\big(Q_\pi(S_t, A_t) - b(S_t) \big)\ \nabla_{\boldsymbol{\theta}} \big[\ln\ \pi(A_t|S_t;\boldsymbol{\theta}) \big] \Big] \tag{2.30}$$

The choice of the baseline $b(S_t) = V_\pi(S_t)$ yields almost the lowest possible variance [SML+15]. By substituting $b(S_t)$ with $V_\pi(S_t)$, the coefficient of the gradient term turns into the advantage function $A_\pi(S_t, A_t) = Q_\pi(S_t, A_t) - V_\pi(S_t)$. The advantage function measures if the action is better or worse than the policy's default behavior (that led to V_π). Consequently, this choice of the baseline can be intuitively justified by the following interpretation of the policy gradient: "a step in the policy gradient direction should increase the probability of better-than-average actions and decrease the probability of worse-than-average actions" [SML+15]. Hence, multiplying the gradient term by the advantage function results in updating the parameters proportional to the relative advantage of the actions. Equation 2.31 presents the iterative update rule of policy gradient with advantage function.

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha' \Big[A_\pi(S_t, A_t)\ \nabla_{\boldsymbol{\theta}} \big[\ln\ \pi(A_t|S_t;\boldsymbol{\theta}) \big] \Big] \tag{2.31}$$

The introduction of the baseline resulted in several fruitful variations and innovations in the policy gradient algorithms. Schulman et al. presented a concise summary of different forms of the policy gradient theorem equation in [SML+15]. In this research, we are not aiming at reviewing all of them. Instead, we direct the focus toward a category of policy-based algorithms, namely actor-critic methods, that demonstrates superior performance and data utilization in episodic tasks [WBH+16, CW18, IGW18, ZBW19]. The proposed Trustable Deep RL algorithm leverages upon this class of policy-based methods. Consequently, the rest of the chapter is devoted to reviewing the basics of the actor-critic methods.

2.5.4 Actor-Critic Method

Two main components in the policy gradient methods (equation 2.30) are the policy model and the value function. Learning the value function and using it to reduce the gradient estimate's variance appears to be essential for rapid learning[SMSM00]. Methods that learn estimates of both policy and value function are referred to as actor-critic methods.

The intuition behind the actor-critic methods is the actor chooses actions according to a parametrized policy, and the critic judges the aptness of the actions using the value functions. Whenever the actor picks an action, the critic observes the environment's transition and reward following the action. Afterward, the critic updates its value function estimation, and the updated value function is then utilized to update and improve the policy. Therefore, the actor-critic methods consist of two estimators, which may or may not share parameters:

- Actor that estimates the optimal policy, $\pi(a|s;\boldsymbol{\theta})$, by updating the policy parameters, $\boldsymbol{\theta}$, in the direction suggested by the critic.
- Critic that estimates the value functions by updating the value function parameters $\boldsymbol{w}$. Depending on the algorithm, it could use various representations of the return including the action-value function $Q(s, a; \boldsymbol{w})$, the state-value function $V(s; \boldsymbol{w})$, or the advantage function $A(s, a; \boldsymbol{w})$.

The pseudocode of a vanilla actor-critic algorithm is presented in algorithm 4. It encodes a policy model with parameters vector $\boldsymbol{\theta}$ and a value function estimator with parameters vector $\boldsymbol{w}$. This algorithm along with the other actor-critic algorithms presented in this chapter and chapter 4, are based on the policy gradient with advantage function, equation

2.31. The parameter δ in line 8 of algorithm 4 is the TD error which in fact is a one-step estimator of $A(S_t, A_t)$ because according to equation 2.12, the expected value of the term $R_{t+1}+\gamma V(S_{t+1})$ is equal to $Q(S_t, A_t)$. However, δ provides a heavily biased and noisy estimate of the advantage function because the value function $V(s)$ is also approximated. The issue of finding a better estimator for the advantage function has been addressed by many studies in the policy gradient literature [SML+15, Waw09], and is discussed further in chapter 4.

Algorithm 4 Online episodic actor-critic

```
Require: Temporal difference step size, $\alpha > 0$
Require: Policy gradient ascent step size, $\alpha' > 0$
Require: Reward discount factor, $\gamma$
Require: A differentiable policy parameterization, $\pi(a|s;\boldsymbol{\theta})$
Require: A differentiable value function parameterization $V(s;\boldsymbol{w})$
 1: Initialize $\boldsymbol{\theta}$ and $\boldsymbol{w}$
 2: while true do
 3:     Initialize state $S_0$
 4:     t=0
 5:     while $S_t \neq$terminal do
 6:         Sample action $A_t \sim \pi(.|S_t, \boldsymbol{\theta})$
 7:         Take action $A_t$, observe $S_{t+1}$ and $R_{t+1}$
 8:         $\delta \leftarrow R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$
 9:         $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha' \, \delta \, \nabla_{\boldsymbol{\theta}} \ln \, \pi(A_t|S_t;\boldsymbol{\theta})$
10:         $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \, \delta \, \nabla_{\boldsymbol{w}} V(S_t;\boldsymbol{w})$
11:         $t \leftarrow t + 1$
12:     end while
13: end while
```

Algorithm 4 is an online actor-critic algorithm because at each time step, the action is sampled from the policy with the most updated parameters. As stated in section 2.4.2, the online algorithms have limitations; for example, in many risk-sensitive applications, the direct interaction of a naive RL agent with the environment is impermissible. In some other applications, the agent is required to learn from historical data, and therefore, there is no real-time interaction. Most importantly, online learning restricts the batch implementation of the algorithm. Consequently, we elaborate on a sample off-line actor-critic algorithm in the following section.

2.5.5 Off-Policy Actor-critic

This section presents an off-policy actor-critic that is one of the building blocks of the proposed deep RL algorithm in chapter 6. In an off-policy learning algorithm, the agent does not have the choice of action because it is not trusted or cannot physically perform online interactions with the environment. In either case, the agent ought to learn from other agents' experience with the environment. The off-policy learning has the additional advantages that (a) it enables experience replay, which results in much better sample efficiency, (b) the sample collection follows a policy different from the learning policy prompts a better exploration.

The pivotal question is how to compute the policy gradients using samples collected from a different policy than the learning policy. Looking back at equation 2.21, the expected return gradient is in the direction of the policy gradient, and $Q_\pi(s, a)$ appears as a coefficient. Knowing that the Q-values are a function of the policy, and consequently, a function of the policy parameters, one would expect that the Q-values gradients, $\nabla_{\boldsymbol{\theta}} Q_{\pi_{\boldsymbol{\theta}}}(s, a)$, would also appear in the return's gradient. However, the gradient policy theorem provides a solution for avoiding $\nabla_{\boldsymbol{\theta}} Q_{\pi_{\boldsymbol{\theta}}}(s, a)$, as rigorously shown in chapter 13 of [SB18]. Unfortunately, the proof provided at [SB18] does not hold for an off-policy setting, and because $\nabla_{\boldsymbol{\theta}} Q_{\pi_{\boldsymbol{\theta}}}(s, a)$ is very complicated to compute, there has been no exact expression for the off-policy gradient so far.

However, Degris et al. [DWS12] derived an approximation for the return gradient with $\nabla_{\boldsymbol{\theta}} Q_{\pi_{\boldsymbol{\theta}}}(s, a)$ ignored. Although they presented an approximated off-policy policy gradient theorem, they still provided policy improvement guarantees for algorithms that apply this theorem. Degris et al. off-policy version of the policy gradient theorem is presented in equation 2.32. The policy that the samples are collected from is known as the behavior policy,

π_b, and we pick π_l as the notation for the learning policy. Looking back at equation 2.21, the only difference between the on-policy and off-policy policy gradient theorems is that the stationary distribution, $\mu(s)$, is not according to π_l anymore. Instead, it is according to π_b, as shown in equation 2.32.

$$\nabla_{\boldsymbol{\theta}} J\big(\pi_l(a|s;\boldsymbol{\theta})\big) \propto \sum_s \mu_{\pi_b}(s) \sum_a Q_{\pi_l}(s,a)\nabla_{\boldsymbol{\theta}}\pi_l(a|s;\boldsymbol{\theta}) \tag{2.32}$$

Accordingly, the expectation estimation of the gradient should be performed over π_b rather than π_l. To derive an expectation expression similar to expression 2.25, equation 2.32 is multiplied and divided by $[\pi_l \times \pi_b]$, and then, similar to expressions 2.23 - 2.25, the expectation over π_b and the environment is derived as follows.

$$\nabla_{\boldsymbol{\theta}} J\big(\pi_l(a|s;\boldsymbol{\theta})\big) \propto \sum_s \mu_{\pi_b}(s) \sum_a \frac{\pi_l(a|s;\boldsymbol{\theta})\pi_b(a|s)}{\pi_l(a|s;\boldsymbol{\theta})\pi_b(a|s)}\, Q_{\pi_l}(s,a)\nabla_{\boldsymbol{\theta}}\pi_l(a|s;\boldsymbol{\theta}) \tag{2.33}$$

$$\propto \sum_{s,a} \mu_{\pi_b}(s)\, \pi_b(a|s)\frac{\pi_l(a|s;\boldsymbol{\theta})}{\pi_b(a|s)}\, Q_{\pi_l}(s,a)\, \frac{\nabla_{\boldsymbol{\theta}}\pi_l(a|s;\boldsymbol{\theta})}{\pi_l(a|s;\boldsymbol{\theta})} \tag{2.34}$$

$$\propto \mathbb{E}_{\pi_b,E}\left[\frac{\pi_l(a|s;\boldsymbol{\theta})}{\pi_b(a|s)} Q_{\pi_l}(s,a)\, \nabla_{\boldsymbol{\theta}}\big[\ln\, \pi_l(a|s;\boldsymbol{\theta})\big]\right] \tag{2.35}$$

In summary, when estimating policy gradient with the expectation estimation in the off-policy learning setting, 2.35, the samples need to be adjusted with the weight $\frac{\pi_l(a|s;\boldsymbol{\theta})}{\pi_b(a|s)}$, which is the ratio of the actions probabilities in the learning policy to the probabilities in the behavior policy. This weight is also known as the importance weight in statistics. Equation 2.36 presents the incremental policy parameters learning equation for off-policy stochastic policy gradient ascent algorithm.

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha' \left[\frac{\pi_l(a|s;\boldsymbol{\theta})}{\pi_b(a|s)}\, Q_{\pi_l}(S_t,A_t)\, \nabla_{\boldsymbol{\theta}}\big[\ln\, \pi_l(A_t|S_t;\boldsymbol{\theta})\big]\right] \tag{2.36}$$

Equation 2.36 sets the ground for the off-policy episodic actor-critic, algorithm 5. The added requirements for algorithm 5 compared with 4 are the off-policy version must be provided with a set of retrospective trajectories of samples as well as the behavioral policy that the samples are derived from.

Algorithm 5 Off-policy episodic actor-critic

```
Require: Temporal difference step size, $\alpha > 0$
Require: Policy gradient ascent step size, $\alpha' > 0$
Require: Reward discount factor, $\gamma$
Require: A differentiable learning policy parameterization, $\pi_l(a|s;\boldsymbol{\theta})$
Require: A differentiable value function parameterization, $V(s;\boldsymbol{w})$
Require: A set of retrospective trajectories, $\mathcal{H}$
Require: The behavior policy, $\pi_b(a|s)$
 1: Initialize $\boldsymbol{\theta}$ and $\boldsymbol{w}$
 2: for each trajectory in $\mathcal{H}$ do
 3:     t=1
 4:     while $S_t \neq$terminal do
 5:         Read $S_t$, $A_t$, $R_{t+1}$, $S_{t+1}$
 6:         $\delta \leftarrow R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$
 7:         $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha' \, \delta \, \frac{\pi_l(a|s;\boldsymbol{\theta})}{\pi_b(a|s)} \, \nabla_{\boldsymbol{\theta}} \ln \, \pi_l(A_t|S_t;\boldsymbol{\theta})$
 8:         $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \, \delta \, \nabla_{\boldsymbol{w}} V(S_t;\boldsymbol{w})$
 9:         $t \leftarrow t + 1$
10:     end while
11: end for
```

CHAPTER 3

Background and Related Work

With all its substantial accomplishments, RL is still not well-accepted for solving real-world problems. Researchers are hesitant about applying RL to real-world sequential decision-making problems, such as medical recommendation, asset integrity management, inventory management, and financial advising. In this research, we propose and define a new terminology, trustable RL, that encompasses the researchers' concerns when applying RL solutions to these highly risk-averse problems.

As implies by the name, we define trustable RL as a set of RL algorithms that are trustable from two aspects:

1. It acts safely, i.e., it does not demonstrate undesirable risky behavior.
2. The solution policy by the RL agent behaves at least as good as the previously deployed policy.

These two aspects have been addressed separately in the RL community. We unify them and tie them together by introducing the RL agent's trustability concept. The issue of avoiding undesirable or risky actions by the agent has been studied under the term safe RL in the literature. We brought the background work on safe RL in section 3.1. The problem of providing performance guarantees for a new recommended policy prior to deployment is tackled under the term off-policy evaluation, which is the topic of section 3.2.

3.1 Literature Review on Safe RL

Safe Reinforcement Learning is a broad category of RL solutions in which the agent takes into account some form of risk or safety notation in its endeavor to find the optimal policy. These RL solutions are pertinent to problems that the solution policy is required to respect a set of safety constraints and ensure reasonable system performance. For example, agents designed for self-driving cars must keep a safe distance from the surrounding vehicles and objects as well as obeying the traffic rules. Autonomous controllers of expensive robotic platforms have to ensure the robot's safety while fulfilling their objective task. Medicine recommender systems and healthcare decision-makers must assure well being of the patient. Expensive asset integrity management systems must optimize the operation and maintenance cost while minimizing the downtime and keeping the asset and the surrounding environment safe from catastrophic failure hazard.

There are two sources of risk, or its opposite safety, in RL problems. The first is the inherent stochasticity of the environment that comprised of the state transition and the reward emission processes by the environment. The second is the exploration process by the agent. Accordingly, safe RL approaches can be segmented into two categories:

1. Approaches that transform the optimization criteria of the RL agent to include a numeric notation of risk into the optimization cost function.

2. Approaches that modify the exploration process to avoid high-risk states.

This taxonomy to categorize and organize the existing safe RL literature is first introduced in [GF15]. Later this classification has been criticized for not being mutually exclusive because the first category is included in the second category, i.e., changing the optimization criteria would consequently change the exploration process.

Although this is a valid criticism, this classification is opening its way into safe RL literature because with this taxonomy the risk-averse RL bifurcates into two streams of thought. The first group believes that RL, as a feedback-based learning method, can achieve the desired level of performance and safety given it has proper optimization function and criteria. In contrast, the second group believes that it can achieve the desired performance level but not safety. The second group argues that feedback-based learning has an intrinsic delay that cannot immediately deflect the agent from risky actions, including entering an unrecoverable state [PSB+18].

This thesis's target is those applications of RL that are regarded as highly risky such that any poor decision could potentially cause a catastrophic event. For example, in medical applications, it could cause the death of a patient; in asset integrity management, it could lead to the burst of a gas pipeline, or in self-driving cars, it could trigger a multiple-vehicle collision. The second approach of safe RL is more favorable for these types of applications since it is generally more successful and quicker in avoiding risky situations. Also, they usually need less knowledge about the environment. This preference will be elaborated more in this chapter.

Resulting from the above considerations, here we review both approaches with more emphasis on the second approach, modifying the exploration process. Moreover, we give more attention to recent works. We encourage the interested reader to refer to [GF15] for a more detailed review of the safe RL literature prior to 2015.

3.1.1 Safe RL through Modifying the Optimization

The objective of a risk-neutral RL agent is to perform a sequence of actions that maximize the expected return, J, as formally stated in equation 2.7. However, this objective function is not adequate for risky applications that require a risk-aware agent. A wealth of studies has been performed to modify the agent's objective function to turn it into a more risk-sensitive criterion.

Variance is a popular notation of risk as higher variance implies more instability and risk. A considerable body of literature assumed the variance of the return as the risk measure. Markowitz in his paper [Mar52], suggested a weighted sum of mean and variance of the return as the objective function, equation 3.1. Therefore, the agent is forced to balance between higher mean and lower variance of the return. This criterion is also known as variance-penalized criterion [Gos09], expected-value-variance criterion [Tah13, Heg94], and expected-value-minus-variance-criterion [GW05] in the literature.

$$\max_{\pi} \left(\left[\mathbb{E}[G(H)] - Var[G(H)] | H \sim \pi \right] \right) \tag{3.1}$$

[HM72, CS87] proposed an exponential utility function that intrinsically considers all moments of the return including the variance, rather than only the mean. The exponential criterion is more prevalent in optimal control where optimal policies are time-dependent, and the system model is present. Therefore, except for [Bor01] and [BBB08], less work has been done in model-free RL using the exponential utility function.

All approaches that use variance as the risk notation in the agent's objective function share the same drawbacks. First, the variance is not an adequate metric of risk in problems where the chance of a catastrophic event is small, but it has highly severe consequences when it happens. Also, variance underestimates the risk in problems where the reward distribution is highly skewed or does not follow a Gaussian distribution. Second, variance does not differentiate between positive and negative risks as it penalizes both equally. Third, the reward function might not accurately reflect the risk associated with states and actions because reward shaping is a complicated task in complex environments. Hence, the reward function might not adequately be designed to reflect the actual cost of entering an undesirable state. Additionally, optimizing mean minus variance in the MDP framework is prone to resulting counter-intuitive policies[MT11].

In the light of variance limitations, several other risk notations are proposed in the literature. [MJL$^+$18] and [SCMHL19] replaced variance in equation 3.1 by a measure of risk based on temporal difference error. The idea is to underweight taking actions where the action-value function varies drastically from the expectation, i.e., the temporal difference error is high in those actions. [MJL$^+$18] proposed substituting variance with the probability that a trajectory H, which is generated by the implementation of policy π, terminates in an undesirable absorbing state. [SCMHL19] proposed risk mapping by similarity which associates the risk of a newly discovered state with its similarity to the previously known risky states. These are two examples of heuristic approaches. Heuristic approaches are successful in their specific targeted applications yet not guaranteed to fit all applications.

Some studies tackled the safe RL problem from another angle. Instead of including an additional numeric risk notation into the objective function, they suggest maximizing the worst-case return scenarios. A policy is regarded as optimal under this criterion if its worst-case return is superior. Equation 3.2 states the worst-case criterion.

$$\max_{\pi} min \left(\mathbb{E}[G(H)|H \sim \pi]\right) \tag{3.2}$$

The worst-case criterion, also known as the minimax criterion, is discussed at length in the literature. Chakravorty and Hyland, in their paper [CH03], introduced minimax to the actor-critic solution method. Heger proposed Q̂-learning, the Q-learning counterpart related to the minimax criterion, in [Heg94]. Jiang et al. presented the convergence theory for Q̂-learing in [JWC98]. In Q̂-learning, the new estimate of $Q(S_t, A_t)$ is the minimum between the TD target and the previous estimate of $Q(S_t, A_t)$, as stated in equation 3.3.

$$\hat{Q}(S_t, A_t) = min\Big(\hat{Q}(S_t, A_t), R_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(S_{t+1}, a_{t+1})\Big) \tag{3.3}$$

In general, the worst-case criterion is known for behaving too restrictive since the agent's objective function mainly considers highly sever yet extremely rare events [MN02, Gas03]. β-pessimistic Q̂-learning is an extension to Q̂-learning which has adjustable pessimism [Gas03]. This algorithm compromises between absolute optimism of standard Q-learning and extreme pessimism of Q̂-learning by a rendering β factor.

$$Q_\beta(S_t, A_t) = Q_\beta(S_t, A_t) + \alpha\Big(R_{t+1} + \gamma\big((1-\beta)\max_{a_{t+1}} Q_\beta(S_{t+1}, a_{t+1}) + \beta \min_{a_{t+1}} Q_\beta(S_{t+1}, a_{t+1})\big)\Big) \tag{3.4}$$

Another popular approach of formally expressing the safe RL problem is through constrained Markov decision processes [Alt99]. This approach puts the safety concerns into one or more constraints and maximizes the return subject to those constraints. For example, Abe et al. [AMP+10] applied constrained RL to the problem of the tax collection optimization system in which the safety constraints are consist of legal, business, and resource constraints.

In a different domain, Li et al. [LWH19] deployed their proposed constrained RL algorithm to EV charging scheduling. In this problem, the safety constraint is to ensure the electric vehicle is fully charged upon departure. [CNDGG18] proposed a Lyapunov approach to solve the constrained Markov decision processes and implemented their methodology on a problem form the gird world domain with safety considerations.

There are two main shortcomings of utilizing constrained RL for safety-critical RL applications. First, many safety restrictions cannot be expressed with the mathematical formalization of constraint Markov decision process. In cases which it is doable, it could be as challenging as assigning the reward function and would need painstaking trial and error effort. Second, the constraints do not prevent fatal consequences in the short term.

3.1.2 Safe RL through Modifying the Exploration Process

RL is an experience learning method with inherent delay. In other words, if learning from scratch, the agent would not know how desirable a state-action pair is until it performs the action at the state. In environments that traps or unrecoverable states exist, the RL agent cannot guarantee safety as it is blind to the risk of traps until it falls into that. For this reason, many AI scholars believe learning solely by classic random exploration/exploitation strategies cannot ensure safety [GW05, GF12, PSB$^+$18]. In addition, random exploration without incorporation of any previously known knowledge about the environment wastes a great deal of time as the agent would explore irrelevant regions of the state-action space in which the optimal policy would not set foot in.

Consequently, lots of effort have been devoted to modifying the exploration to use either some external knowledge or a risk metric to direct the exploration and mitigate the described above difficulties. In the following sections, we introduce three main approaches of modifying the exploration process which consists of:

1. to learn from a set of teachers' behavioral policy samples,

2. to guide the exploration through teachers' advice,

3. to guide the exploration by using a risk measure.

3.1.2.1 Modifying the Exploration Process through Samples of Teacher's Behavior

One approach for integrating the external knowledge into the RL agent is through learning from a finite set of teacher's behavioral policy samples. These samples might be acquired from a variety of sources. A ubiquitous source is the historical data of an expert's behaviors, or a group of experts' behaviors, on performing the task. For example, [MB05] used existing demonstrations by a teacher in a grid word domain to derive high quality initial value functions. The obtained initial values are more informative than any random initialization, resulting in a safer and faster future learning process. Zheng et al. [ZZZ+18] applied deep RL techniques on a history data of commercial news recommendation to learn a more effective personalized news recommender system.

Similarly, a teacher or expert could provide some demonstrations of how to perform the task only for the matter of teaching to the RL agent. For example, Martin and Lope in their RL competition helicopter hovering task winner paper [dL+09], used sample behaviors from several teachers to assign the initial weights of a neural network in their deep RL agent. Koppejan and Whiteson in their papers [KW09, KW11] also took a deep RL with an evolutionary algorithm for the task of hovering a helicopter in a simulated environment. The algorithm starts from a neural network whose weights correspond to the teacher's behavior and then evolved into a more mature agent.

Another source of teacher's behavioral policy samples could be the instances of the teacher's behavior on a different yet similar task. This is known as transfer learning in the literature [TS09]. Taylor and Stone are one of the pioneers in introducing transfer learning to RL. In [TSL07], they trained an RL agent on the source task of playing Keepaway, an isolated sub-problem of RoboCup, and then the agent uses this experience in the target task of playing Knight Joust, a sub-problem in the grid world. They translated the Q-values learned in the source task to the Q-values of the target task by a hand-coded inter-task mapping.

The transfer learning line of research is getting more attention in robotics and gaming fields [BCJS+15, BSDS+18, CJMDMB10]. Although it is effective in speeding up the learning process and directing the learning process to the more promising action-state region, it has shortcomings that limit its application to safe RL. First, not many risky applications have a twin or similar application that the teacher could interact with it safely. Second, translating the knowledge from the source task to the target task is not trivial.

The teacher could also be a non-human expert. For example, the behavior from a mature RL agent trained in a simulated environment could be used by a second agent, which is learning how to perform in a real-world version of the simulated environment. Ramakrishnan et al. [RKD+18] studied the safety of the second agent from a different perspective. They focused on the second agent's blind spots in its understanding of the real world due to mismatches between the simulated and the real environments. They took a supervised learning approach to detect the blind spots, and followed the detection with an inquiry from an oracle, a human teacher, for advice.

Looking from another angle, the way the sample policies from an expert's behavior are integrated into the RL agent learning process also matters. One very well established approach is to use the samples to bootstrap the value functions. For example, [DD04] used a set of demonstrations recorder from a human teacher to build a partial initial Q-function, which are used later to guide further exploration of the state space. [SK00] took a Q-learning approach for steering a real robot down a corridor towards a dead-end. They bootstrapped the value functions using supplied initial policies and gained a much faster value function convergence.

Inarguably, the bias introduced to the value functions to bootstrap the learning process improves safety in the exploration process and speeds up the convergence; therefore, it makes this approach advisable for risk-averse RL. But clearly, it is not sufficient for safe exploration. Following the initialization step, the agent needs to explore, which could result in visiting new states that the agent does not have much information about them.

A less popular approach to take advantage of the sample policies from the teacher's behavior is to directly deploy the teacher's policy as the RL agent's initial baseline policy. [ZBL19] took this approach to quickly and safely learn a control policy for OpenAI gym environment games. They assumed a reasonably good policy exists as a supervisor during training, and the RL agent learns from both the RL training signals and the supervisor policy. They observed significant acceleration and safety improvement of the learning process comparing to learning without a supervisor. The main drawback of this approach that extensively limits its application to real-world environments is that the teachers' behavior policy is not explicitly available. It needs to be extracted from the sample trajectories of teacher and environment interactions.

Another common approach is to take advantage of the teacher's behavior samples to build a model for the environment, and then derive the safe policy with a model-based RL solution. This approach is known as apprenticeship learning and is first named by Abbeel and Ng [AN04, AN09]. Abbeel et al. [ACN10] applied apprenticeship learning to autonomous helicopter control, a problem that is regarded as a highly challenging control problem. [HBW19] exercised apprenticeship learning to driving an autonomous car.

Despite all the great potentials of apprenticeship learning, it does not fit into safe RL approach requirements. The reason is apprenticeship learning does not provide any information on the situations where no teacher demonstration exists. In other words, if the agent does not explore beyond the teacher demonstration, the performance will be heavily limited by the quality of the demonstration. If it does explore, it has no means of avoiding high-risk situations. More generally, this criticism is valid for any safe RL solution that tends to ensure the safety merely by incorporating the sample behaviors of a teacher. One possible way to overcome this shortcoming is to perform exploration under the careful surveillance of a supervisor, which is the topic of the next two sections.

3.1.2.2 Directing the Exploration Process through Teacher's Guidance

The benefit of using a teacher's guidance is twofold. First, it could reduce the sample complexity of the learning algorithm since the teacher could guide the agent in promising parts of the stat-action space. Second, following the teacher's advice in situations that are regarded as high risk by either the agent or the teacher could prevent catastrophic events. The methodologies for integrating the teacher's guidance into the RL agent exploration could be divided into two categories based on who determines the need for teacher aid. The two categories are (i) the learner (the RL agent) asks for advice when it considers necessary, (ii) the teacher provides guidance when it identifies the need for outside intervention.

In the learner asks for advice approach, the learner has an inner mechanism for measuring the confidence of its actions. When this measure is low in a state, the learner would ask for the teacher's advice. This approach is also known as Ask for Help and is first introduced by Clouse in his paper [Clo97]. In this work, Clouse defined a metric for actions' confidence based on Q-values similarity, i.e., when the Q-values of the actions pertinent to a state are very close, the agent is unsure about taking actions. He made the difference between the minimum and maximum as the basis for the confidence parameter. However, this metric is not a befitting uncertainty metric for the states that have similar Q-values in nature. Sometimes the states are desirable or undesirable intrinsically, and similar Q-values for various actions in those states simply means no matter what action is performed, the outcome would be the same.

Hans et al. [HSSU08] presented another confidence measure to detect high-risk situations by the agent. They defined a metric based on fatal transitions that correspond to transitions that bear a reward less than a predefined threshold. An action is risky if performing the action in a given state leads to a fatal transition. Garcia et al. [GF11, GF12] proposed a new metric of confidence based on unknown parts of the state-action space. Intuitively, the agent would ask for help when it is in an unfamiliar situation. To identify the unfamiliar situation, they proposed a case base that stores the previously experienced state-action pairs. A risk function that inputs the similarity between a new pair and the previously visited pairs in the case base determines the new state's familiarity. If the new state is determined as unfamiliar, then the query state is labeled as unknown or risky.

The risk function applied in [GF11, GF12] is a step function that outputs one when the distance to the closest pair in the case base is larger than a hyperparameter θ, and outputs zero when the distance is less than θ. However, in a subsequent work [GAF13], Garcia et al. replaced the step risk function by a continuous risk function because risk advances gradually in essence. In other words, it might be too late to wait for the risk function to hit the threshold and then ask for the teacher's advice. The proposed continuous risk function in [GAF13] is a sigmoid function that gradually increases the probability of asking for teacher's advice as the distance between the query state-action pair and the closest pair in the case base gets larger.

Baheir et al. took a novel approach to detect risk in the future states accruing after an action. In their study of safety in autonomous highway driving [BNT+19], they trained an RNN in a supervised learning framework to predict if the upcoming states are safe or risky. If one of the future states are unsafe, i.e., leads to a collision, according to the RNN and a set of expert provided safety rules, then a negative reward will be assigned to the action's Q-value to avoid collision and accelerate the learning process. They tested the capabilities of their proposed methodology in a simulated autonomous deriving environment with varying traffic density and concluded that it has superior performance than the case where only the rule-based safety measures are taken.

In some proposed algorithms for Ask for Help, the agent is not equipped with a risk or confidence measure. Instead, a subset of states is labeled as unsafe. The learner is required to ask for advice when it arrives at any of those states. For example, Vleugel and Gelens [VHG11] took this approach to avoid unsafe regions in the agent exploration.

Geramifared et al. [GRRH11, Ger12, GRH13] defined risk as the probability of visiting any of the sates in a constrained function. This constrained function maps the state space into a binary $\{0, 1\}$ set in which 1 is assigned to the permitted states. If performing an action in a particular state results in landing on one of the banned states, the learner invokes the teacher's action that is assumed to be safe. The main obstacle of using this method in real-world applications is that the environment model, and subsequently the states' transition function, are considered to be available.

In the second approach for integrating the teacher's guidance into the RL agent exploration, the teacher decides when to provide advice. In this category of algorithms, there is no add-on mechanism inside the agent to express and recognize its need for advice. Instead, there is an interface between the teacher and the RL agent that enables the teacher to monitor the agent's behavior constantly. The interface usually allows the teacher to directly modify the agent's decisions or to manipulate the reward signals to indirectly provide advice to the agent. For example, [Clo97, TB^{+}06, TB08, SC11, QVIRRGVR13] are some of the studies in the robotic field that placed an interface between the RL agent and the human teacher to help the teacher monitor the RL agent actions and enforce his decisions when necessary either by reward or punishment or direct advising.

The interface could also be a set of IF-THEN rules. Maclin et al. implemented a set of rules as the teacher's guidance. In their first research [MSWT05], the teacher could manipulate the Q-values of the actions when the if condition is satisfied. However, in their following work [MST^{+}05] the teacher directly changes the policy rather than the value functions. Torry et al. [TWSM05] also took the set of rules approach for providing the teacher's guidance. In their proposed methodology, the rules are extracted from a similar task performed previously by inductive logic programming. Walsh et al. [WHM11] applied a set of IF-THEN rules to help the agent learns concepts that it cannot learn effectively by itself. In their work, the teacher observes the return of each episode, and if the return of the agent in that episode is smaller than a certain value, the teacher performs a demonstration of that episode.

In general, directing the exploration through teacher's advice is one of the highly effective approaches to ensuring safety in RL problems. However, there are some limitations when using this class of safe RL methods. In ask for help algorithms, the learner must be armed with a risk detection mechanism. Mostly, the mechanisms cannot detect long-term risks, and as a result, the algorithms are sensitive only to short-term risks. In the second category, which the teacher decides when to provide the advice, an interface between the teacher and the learner is required to monitor the learner's actions and also to facilitate the teacher for imposing the advice. Implementing this interface is costly or not feasible in many real applications.

3.1.2.3 Directing the Exploration Process through Risk Metrics Guidance

Despite all the significant advancements in RL, the exploration remains a critical problem, especially in safety-critical applications. Many scholars tried to tackle this problem by introducing some notation of the risk in the exploration process. Gehring and Precup [GP13] proposed directing the exploration with a risk metric, which is defined based on a new notation of state controllability, i.e., a particular state is less controllable if its temporal difference signals show considerable variability. The intuition is if an agent wants to stay safe, it should avoid states where predicting the effect of its actions is harder. They formally defined controllability as the temporal difference error expected value.

$$C_\pi(s,a) = -\mathbb{E}_\pi\big[|\delta_t| | S_t = s, A_t = a\big] \tag{3.5}$$

Considering equation 3.5 definition, a state with higher variability (temporal difference error) has lower controllability. They also provide an iterative temporal update rule for the controllability, as shown in equation 3.6.

$$C(S_t, A_t) \leftarrow C(S_t, A_t) - \alpha'\big(|\delta_t| + C(S_t, A_t)\big) \tag{3.6}$$

Later, the exploration process is performed using this notation of controllability as an exploration bonus, choosing actions in a greedy manner according to $Q(S_t, A_t) + wC(S_t, A_t)$. Therefore, the agent is motivated to explore in more controllable regions of the environment. The w coefficient in the above utility function is a trade-off parameter that adjusts between the desire to gain higher returns or tighter controllability.

Law et al. [LCPR05] suggested directing the exploration process with an entropy-based risk metric. They defined the risk metric as the weighted sum of the entropy and normalized expected reward of the action.

$$Risk(S_t, A_t) = wH(S_t, A_t) - (1-w)\frac{\mathbb{E}[R_{t+1}]}{\max\limits_a\ \big[\mathbb{E}[R_{t+1}]\big]} \tag{3.7}$$

In equation 3.7, $H(S_t, A_t)$ is the entropy term that accounts for the stochasticity of the outcomes of a given action in a state, and the normalized expected reward accounts for how much this action is worse than the best action in this state. The described risk metric is combined linearly with the Q-values to form a risk-adjusted utility function.

$$U(S_t, A_t) = p\ (1 - Risk(S_t, A_t)) + (1 - p)\ Q(S_t, A_t) \tag{3.8}$$

In the end, they drove the policy, i.e., assigning probabilities to the actions, through a Boltzman distribution.

In a recent work from Safarian et al. [STK18], they proposed to take the evaluation errors of Q-values as a risk consideration in the learner exploration process. This type of error is inevitable when the Q-values are earned from a finite set of historical data. Theoretically, gaining the optimal policy is possible if the Q-values are estimated from a data set with an infinite number of visitations in each state-action pair. However, this condition cannot be met practically, and therefore, exploration should be constrained. Intuitively, the Q-values have higher errors for actions that were taken less frequently. Consequently, they suggested enforcing a constraint, named reroute constraint, for any policy improvement. They evaluated their methodology on Atari games and concluded that their algorithm is a safe and data-efficient learning approach.

Dai et al. [DXWC19] took a neural network approach for risk level estimation of the agent's actions. In their proposed deep RL algorithm with safe exploration, a convolutional neural network estimates the Q-values as well as the long term risk level of each state-action pair. Afterward, the probability of selecting each action is determined according to both the estimated risk level and the Q-values. They proposed a modified Boltzmann distribution for the policy parametrization. The Boltzmann distribution uses the estimated risk level as the temperature parameter that adjusts the decisions' randomness. Therefore, the action with a higher Q-value and a lower risk level will be selected with a higher probability. The modified Boltzman distribution is presented in equation 3.9 in which $E(s, a)$ is the risk level.

$$\pi(s, a) = \frac{exp\big(\frac{Q(s,a)}{E(s,a)+1}\big) I\big((s, a) \in L\big)}{\sum\limits_{a' \in A_s} \Big[exp\big(\frac{Q(s,a')}{E(s,a')+1}\big) I\big((s, a') \in L\big)\Big]} \tag{3.9}$$

In this equation, L is a blacklist of forbidden state-action pairs. Accordingly, if taking an action in a state is prohibited, then the probability of taking that action would be set to zero by the indicator function $I(.)$. Dai et al. tested the proposed methodology in a network security problem and concluded that the algorithm gained a superior performance in terms of higher returns and lower number of outages.

The main downside of the algorithms that direct the exploration process through a risk measure guidance is that the risk metric values are learned through the learning process. In other words, similar to the value functions, the risk metric values are not accurate unless the agent interacts sufficiently with the environment. However, it is more desirable to avoid risky situations from the early steps in the learning process. Specifically, in applications concerned with traps or absorbing unsafe states, the agent demands accurate risk values from the beginning of its interaction with the environment.

3.2 Literature Review on Off-Policy Evaluation with Confidence Bounds

Off-policy evaluation is the problem of estimating the performance of a new and never implemented policy in an environment from episodes of interactions generated under one or multiple previously deployed policies in the same environment. Here we refer to the new policy as the evaluation policy, π_e, and the previous policy as the behavior policy, π_b.

The quantitative metric for policy's performance is the expected return, $J(\pi)$, as defined in equation 2.7. In cases where the policy could be tested by a trial deployment, it is reasonable to estimate the expected return by empirical mean of the returns gained in episodes of policy-environment interactions. However, in critical applications, a new policy cannot be implemented even for trial without providing proof and confidence about its superior performance. In this case, we ought to estimate the new policy's expected return from previous behavioral policies' history of interactions.

To formally define the problem, we assume that a data set $\mathcal{D}$ consisting of $n_{\mathcal{D}}$ episodes of interactions generated under π_b is provided,

$$\mathcal{D} = \{H_{L_1}, ..., H_{L_{n_\mathcal{D}}} : H_i \sim \pi_b\}. \tag{3.10}$$

We aim at estimating the evaluation policy's expected return from trajectories in $\mathcal{D}$.

$$\hat{J}(\pi_e) = f(\mathcal{D}, \pi_b, \pi_e) \tag{3.11}$$

This thesis's target applications are highly critical environments where the deployment of bad policy can be illegal, inhuman, costly, or dangerous. In such applications, we cannot gain trust in the new policy by solely providing a point estimate of the expected return. We further need to express quantitatively how confident is the proposed estimated value. For example, if the current policy's performance is available, then our method must produce statements of the form "with confidence 90%, the new policy will perform at least as well as the current policy."

This problem is known as off-policy evaluation with confidence bounds in the RL taxonomy and is gaining increasing attention among the RL community. The approaches to this problem could be divided into two categories. If the expected return's estimate and the confidence bounds must be evaluated only from the historical data $\mathcal{D}$, then Importance Sampling (IS) approaches, section 3.2.2, are recommended. However, if the user is willing to apply some function approximation along with the historical data, then section 3.2.1 approaches are recommended.

3.2.1 Value Function Approximation Based Approaches for Off-Policy Evaluation

This section presents function approximation based approaches for off-policy evaluation. Although this is not the approach taken in the Trustable Deep RL algorithm proposed in chapter 6, we here provide a brief review and a high-level introduction to these approaches for the sake of completeness of the off-policy evaluation literature review.

looking back at the definition of the state-value function and the action-value function, equations 2.8 and 2.9 in chapter 2, reveals that the expected return is in fact equal to either of the value functions at the initial state

$$V_{\pi_e}(s) = \mathbb{E}\Bigg[\sum_{t=0}^{terminal} \gamma^t R_{t+1} | \pi_e, S_0 = s\Bigg], \tag{3.12}$$

$$Q_{\pi_e}(s,a) = \mathbb{E}\Bigg[\sum_{t=0}^{terminal} \gamma^t R_{t+1} | \pi_e, S_0 = s, A_0 = a\Bigg]. \tag{3.13}$$

Consequently, an approach for estimating the expected return is to estimate either of the value functions at the initial state. In a general setting where the state space or the action space could be large or even continuous, a parameterized function approximation is used to estimate the value function, $\hat{V}_{\pi_e}$.

Liu et al. [LLG+15] showed that the algorithms developed for estimating $\hat{V}_{\pi_e}$ using off-policy data are true stochastic gradient algorithms for primal-dual saddle-point objective functions; and therefore, finite-sample bounds on the error of the value functions approximations could be defined. The bound quantifies how close the off-policy estimate of the value function is to the true value function as the number of iterations increases. Equation 3.14 demonstrates the general form of the bounds. It states that with probability $1 - \delta$, the error of the estimation is less than $f(\boldsymbol{\theta}, n, \boldsymbol{\zeta}, \delta)$.

$$Pr\Bigg(||V_{\pi_e} - \hat{V}_{\pi_e}|| \leq f(\boldsymbol{\theta}, n, \boldsymbol{\zeta}, \delta)\Bigg) \geq 1 - \delta, \tag{3.14}$$

where the left side of the inner inequality is a measure of the error, and the right side is the bound that usually depends on n (the number of iterations in the estimator algorithm), $\boldsymbol{\theta}$ (the parameters of the function approximator), δ, and several unknown parameters and properties of the MDP and the choice of function approximator, which we show here with the vector $\boldsymbol{\zeta}$.

Since the parameters in $\boldsymbol{\zeta}$ are unknown, It is not possible to compute an exact error bound for the estimation. However, one could assign high-confidence limits to all the unknown parameters and compute a confidence bound for the off-policy evaluation of π_e performance.

The major drawback of this approach is that combining several confidence bounds for the unknown parameters with the union bound leads to over conservative evaluation policy's performance bound. Hence, instead of taking the function approximation approach for off-policy evaluation, we take the IS approach that uses IS estimators and concentration inequalities to produce confidence bounds on the evaluation policy's estimated performance. We would like to still emphasize that finite-sample bounds in off-policy evaluation with function approximation is a promising avenue of research, and an interested reader could follow the current works of Liu et al. [LLTZ18], Uehara et al. [UJ19] , and Nechum et al[NDK+19, NCDL19].

3.2.2 Importance Sampling Based Approaches for Off-Policy Evaluation

IS is a statistical method for estimating the expected value of a function when samples are collected from a distribution that is different from the desired distribution. More concretely, let q and p be two probability mass functions of the random variable X where $x \in \mathcal{X}$, and f is a function of X, $f : \mathcal{X} \to \mathbb{R}$. The goal is to estimate the expected value of $f(X)$ according to p; however, samples of X are available only from the q distribution [Jia10]. We call p the target distribution, and q the sampling distribution.

Given that $\mathit{supp}\, p \subseteq \mathit{supp}\, q$, we can find $\mathbb{E}[f(X)|X \sim p]$ by multiplying the definition of expected value by $1 = \frac{q(x)}{q(x)}$, and derive the IS estimator:

$$\begin{aligned} \mathbb{E}[f(X)|X \sim p] &= \sum_{x \in \mathit{supp}\, p} p(x) f(x) \\ &= \sum_{x \in \mathit{supp}\, q} p(x) \frac{q(x)}{q(x)} f(x) \\ &= \sum_{x \in \mathit{supp}\, q} q(x) \frac{p(x)}{q(x)} f(x) \\ &= \mathbb{E}[\frac{p(X)}{q(X)} f(X)|X \sim q]. \end{aligned} \tag{3.15}$$

The last line of the above equation states that $\frac{p(X)}{q(X)} f(X)$ is an estimator of $f(X)$'s expected value under p when X is sampled from q. The intuitive explanation of the IS estimator is when a specific x is more likely under q than the target distribution p, then sampling from q generates x too many times. Hence, it should be given a weight that is < 1 to account for its more frequent occurrence. In the same way, if a specific x is less likely under the sampling distribution, q, than the target distribution, p, then sampling from q generates x too few times. Therefore, it should be given a weight that is > 1 to compensate for its less frequent occurrence.

When the IS estimator is applied to the RL context, $\mathcal{X}$ is the set of possible episodes, p is the episodes distribution according to the evaluation policy, q is the episodes distribution according to the behavior policy, and $f(X)$ is the return, $G(H_i)$. To ensure that $supp\, p \subseteq supp\, q$, we could assume the condition that:

$$if\ \ \pi_e(a|s) \neq 0\ \ then\ \ \pi_b(a|s) \neq 0\ \ for\ \ all\ \ a \in \mathcal{A},\ \ and\ \ s \in \mathcal{S}. \tag{3.16}$$

If assumption 3.16 holds, then the IS estimator for the expected return of the evaluation policy could be derived as follows:

$$\hat{J}^{IS}(\pi_e|H_i, \pi_b) = G(H_i|H_i \sim \pi_b)\frac{Pr(H_i|\pi_e)}{Pr(H_i|\pi_b)}, \tag{3.17}$$

where $Pr(H_i|\pi_b)$ can be decomposed into

$$Pr(H_i|\pi_b) = Pr\big(s_0^i, a_0^i, r_1^i, ..., s_{L_i-1}^i, a_{L_i-1}^i, r_{L_i}^i, s_{L_i}^i|\pi_b\big). \tag{3.18}$$

The environment, and consequently the transition and reward functions, are the same whether the episode is sampled from π_e or π_b. Hence, the probabilities of all the elements in H_i are equal under the two policies except for the actions. As a result, the ratio $\frac{Pr(H_i|\pi_e)}{Pr(H_i|\pi_b)}$ can be expressed as the ratio of the actions' probability in the evaluation policy to the actions' probability in the behavior policy. Accordingly, the estimated expected return of the evaluation policy according to one episode in $\mathcal{D}$ can be expressed as:

$$\hat{J}^{IS}(\pi_e|H_i, \pi_b) = G(H_i|H_i \sim \pi_b)\prod_{t=0}^{L_i-1}\frac{\pi_e(a_t^i|s_t^i)}{\pi_b(a_t^i|s_t^i)}. \tag{3.19}$$

where $\displaystyle\prod_{t=0}^{L_i-1}\frac{\pi_e(a_t^i|s_t^i)}{\pi_b(a_t^i|s_t^i)}$ is known as the IS weight of H_i episode,

$$W_{H_i}^{IS} = \prod_{t=0}^{L_i-1}\frac{\pi_e(a_t^i|s_t^i)}{\pi_b(a_t^i|s_t^i)}. \tag{3.20}$$

Equation 3.19 could be simply extended to include all $n_\mathcal{D}$ episodes in $\mathcal{D}$. The batch estimator is presented in the following.

$$\hat{J}^{IS}(\pi_e|\mathcal{D}, \pi_b) = \frac{1}{n_\mathcal{D}} \sum_{i=1}^{n_D} \left[G(H_i|H_i \sim \pi_b) \prod_{t=0}^{L_i-1} \frac{\pi_e(a_t^i|s_t^i)}{\pi_b(a_t^i|s_t^i)} \right]. \tag{3.21}$$

In RL off-policy evaluation literature, equation 3.21 is known as the most basic IS estimator of the expected return [Pre00, TTG15]. It is trivial to show if assumption 3.16 holds, the basic IS estimator is an unbiased and consistent estimator of $J(\pi_e)$.

One major drawback of the basic IS estimator is it tends to produce very high variance estimates in sequential decision-making problems [WAD17]. Since it is the product of actions' importance weights, and each of the importance weights could have a large value. The variance issue grows exponentially and gets worse as the length of the episode gets larger [JL15]. Per-decision importance sampling (PDIS) is another IS-based estimator specific to sequential problems and often has a lower variance than the basic IS estimator [TTG15, JL15]. It can be derived by applying the importance weights to each step's reward rather than one importance weight for the entire return. The weight of each step's reward is the product of the probabilities' ratio up to that step.

$$\hat{J}^{PDIS}(\pi_e|\mathcal{D}, \pi_b) = \frac{1}{n_\mathcal{D}} \sum_{i=1}^{n_D} \sum_{t=1}^{L_i} \left[\gamma^{t-1} R_t^i \prod_{k=0}^{t-1} \frac{\pi_e(a_k^i|s_k^i)}{\pi_b(a_k^i|s_k^i)} \right] \tag{3.22}$$

PDIS estimator is first derived by Precup et al. in [Pre00]. They also proved PDIS's unbiasedness and consistency. What PDIS estimator does is to use ordinary IS to estimate the expected reward at time t under π_e, and then to apply equation 2.7 to calculate the total episode's return under π_e. It worth mentioning that PDIS is equivalent to IS if the rewards are all zero except for the final reward (which is very common in real-world problems).

Another variant of the basic IS estimator is the weighted importance sampling (WIS), which is originally proposed to overcome the high variance issue of IS [Jia10, WMMY11]. The WIS estimator is the basic IS estimator, equation 3.21, divided by the sum of the importance weights:

$$\hat{J}^{WIS}(\pi_e|\mathcal{D},\pi_b) = \frac{\frac{1}{n_\mathcal{D}}\sum_{i=1}^{n_D}\left[G(H_i|H_i\sim\pi_b)\prod_{t=0}^{L_i-1}\frac{\pi_e(a_t^i|s_t^i)}{\pi_b(a_t^i|s_t^i)}\right]}{\frac{1}{n_\mathcal{D}}\sum_{i=1}^{n_D}\left[\prod_{t=0}^{L_i-1}\frac{\pi_e(a_t^i|s_t^i)}{\pi_b(a_t^i|s_t^i)}\right]}. \tag{3.23}$$

The basic IS estimator is prone to over and under estimation when the behavior and evaluation policies are much different. Because when the two policies are quite different, the IS weights become either too large or too small, which consecutively cause overestimation and underestimation. However, in the WIS estimator, the denominator will compensate for too small or too large weights causing the estimator to maintain a reasonable value.

Although WIS is a biased estimator, [Tho15] showed that it is consistent, which means it becomes less biased as the number of samples, $n_\mathcal{D}$, increases. A distinctive characteristic of WIS estimator is it enjoys a much smaller range comparing to IS and PDIS. In fact, the $\hat{J}^{WIS}$ is always bounded by the actual return's range. As we show later, this property is very critical in defining tight confidence bounds for the estimators.

Similar to the IS estimator that is modified to produce the WIS estimator, we can modify the PDIS estimator to derive a weighted per-decision importance sampling (WPDIS) estimator. This modification is handled by applying WIS to estimate the expected reward at each time t under π_e, and then applying equation 2.7 to calculate the total episode's return under π_e.

$$\hat{J}^{WPDIS}(\pi_e|\mathcal{D},\pi_b) = \sum_{t=1}^{L}\left[\gamma^{t-1}\frac{\sum_{i=1}^{n_D} R_t^i\prod_{k=0}^{t-1}\frac{\pi_e(a_k^i|s_k^i)}{\pi_b(a_k^i|s_k^i)}}{\sum_{i=1}^{n_D}\prod_{k=0}^{t-1}\frac{\pi_e(a_k^i|s_k^i)}{\pi_b(a_k^i|s_k^i)}}\right]. \tag{3.24}$$

The idea of applying WIS to the MDP setting in a step-wise manner is first proposed by Precup et al. in [Pre00]. However, their derivation of WPDIS is biased and not consistent. Later, Thomas in his work [Tho15], suggested equation 3.24 derivation of WPDIS which is biased but consistent. The consistency property ensures that the estimator becomes less biased as the number of samples, $n_\mathcal{D}$ increases. WPDIS has the lowest variance among all IS estimators, and it is considered the most practical point estimator in the IS family [JL15].

Up to this point, we reviewed the IS-based estimators for the expected return. The presented estimators compute only a point estimate of the expected return and do not provide confidence bounds to quantify how accurate their estimates are. This is a concerning issue, mainly because the off-policy evaluation methods are known to produce estimates that have high variance. Meaning that although the new policy might have a desirable estimated expected return, the estimate has high variance and is not reliable. Consequently, the rest of this section is dedicated to the methods of computing high confidence lower bounds on the IS-based estimators.

Concentration inequalities (CIs) offer the solution for computing the confidence bounds. They provide bounds on how a random variable deviates from some value (usually the mean or median value), and therefore, how the probability mass or density of the random variable is concentrated. The type of CI that is of interest in off-policy evaluation is in the form presented by equation 3.25. Note that not all the CIs are in this form. Consider n independent random samples, $\{X_i\}_{i=1}^{n}$, and $(1-\delta)100\%$ as the desired confidence level, then the X's density around its mean can be written as:

$$Pr\left(\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} X_i\right] \geq f(X_1, ..., X_n, \delta)\right) \geq 1-\delta, \tag{3.25}$$

where $\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} X_i\right]$ is the true mean, f is some function related to the specific CI, and δ is any real number in $[0, 1]$. Various CIs might have different additional restrictions on X_is, such that they are bounded or identically distributed. Often f is in the form of the sample mean minus a positive term that goes to zero as $n \to \infty$.

$$Pr\left(\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} X_i\right] \geq \frac{1}{n}\sum_{i=1}^{n} X_i \;-\; g(X_1, ..., X_n, \delta)\right) \geq 1-\delta, \tag{3.26}$$

in which $\frac{1}{n}\sum_{i=1}^{n} X_i$ is the sample mean.

In off-policy evaluation problem, the random variable X is the estimated expected return from each episode, $\hat{J}(\pi_e|H_i, \pi_b)$, and therefore, the true mean is $\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} \hat{J}(\pi_e|H_i, \pi_b)\right]$ which is equal to $J(\pi_e)$ if $\hat{J}$ is unbiased.

$$Pr\bigg(J(\pi_e) \geq f(\hat{J}(\pi_e|H_1, \pi_b), ..., \hat{J}(\pi_e|H_n, \pi_b), \delta)\bigg) \geq 1 - \delta \tag{3.27}$$

Hence, the right side of the inner inequality is a $1 - \delta$ confidence lower bound on $J(\pi_e)$.

Along with $\hat{J}$ unbiasedness, some CIs might put some other restrictions on X_is, such as that they are independent, bounded, or identically distributed. In the following, we review some of the CIs that are commonly used for the MDP problem settings [Tho15, TTG15, JL15, Jia10, HSN17, WAD17].

The most well known CI is probably the The Chernoff-Hoeffding (CH) inequality [Mas07]. It requires the X_is to be independent, and each be bounded such that $Pr(X_i \in [a_i, b_i]) = 1$, where $a_i \in \mathbb{R}$ and $b_i \in \mathbb{R}$ for all $i \in \{1, ..., n\}$. If these conditions hold, then

$$Pr\left(\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} X_i\right] \geq \frac{1}{n}\sum_{i=1}^{n} X_i - \underbrace{\sqrt{\frac{ln(\frac{1}{\delta})\sum_{i=1}^{n}(b_i - a_i)^2}{2n^2}}}_{g}\right) \geq 1 - \delta. \tag{3.28}$$

The term under the square root is the g function from equation 3.26, and it is crucial to see how quickly it goes to zero as $n \to \infty$. For CH inequality, the rate is asymptotically proportional to $\frac{1}{\sqrt{n}}$.

The CH inequality uses only the sample mean of the random variable. Later, Maurer and Pontil [MP09] derived a better bound by using both the sample mean and the sample variance. They named it as the Empirical Bernstein bound, and we refer to it as the Maurer and Pontil's Empirical Bernstein (MPeB) CI. The MPeB requires the X_i random variables to be independent and all be bounded in the same range such that $Pr(X_i \in [a, b]) = 1$, where $a, b \in \mathbb{R}$. The mathematical representation of MPeB is:

$$\begin{aligned} Pr\Bigg(\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} X_i\right] \geq \frac{1}{n}\sum_{i=1}^{n} X_i \\ - \frac{7(b-a)ln(\frac{2}{\delta})}{3(n-1)} \\ - \sqrt{\frac{2ln(\frac{2}{\delta})}{n}\frac{1}{n(n-1)}\sum_{i,j=1}^{n}\frac{(X_i - X_j)^2}{2}}\Bigg) \geq 1 - \delta. \end{aligned} \tag{3.29}$$

For MPeB inequality, the rate of g going to zero as $n \to \infty$ is asymptotically proportional to $\frac{1}{n}$, which is much faster than the g term in the CH inequality. However, it requires that all X_is having the same range. If X_is are in different ranges, we can trivially select $a = min(a_1, ..., a_n)$ and $b = max(b_1, ..., b_n)$. In cases where the ranges are loose for some X_is and tight for other majority of the random variables, then MPeB might give looser lower confidence bound than CH.

Anderson [And69] proposed another CI that improved upon CH and MPeB by taking into account even more statistics from the empirical cumulative distribution of X. This inequality had an unspecified parameter that was later derived by Massart [Mas90]; therefore, it is known as the Anderson and Massart's (AM) inequality. AM inequality has more strict requirement that all X_is be identically distributed. However, regarding the X range, it only requires the lower range such that $Pr(X_i \geq a) = 1$ for all $i \in \{1, ..., n\}$. Equation 3.30 presents the AM inequality.

$$Pr\Bigg(\mathbb{E}\Bigg[\frac{1}{n}\sum_{i=1}^{n} X_i\Bigg] \geq Z_n - \sum_{i=0}^{n-1}(Z_{i+1} - Z_i)\min\Bigg\{1, \frac{i}{n} + \sqrt{\frac{ln(\frac{2}{\delta})}{2n}}\Bigg\}\Bigg) \geq 1 - \delta, \tag{3.30}$$

where, $Z_0 = a$, and $\{Z_i\}_{i=1}^{n}$ are sorted version of $\{X_i\}_{i=1}^{n}$ such that $Z_1 \leq Z_2 \leq ... \leq Z_n$.

Unlike CH and MPeB inequalities, AM does not directly depend on the range of the random variable. Instead, it heavily relies on the value of the largest observation, Z_n. In the applications where a tight upper range is not available, and the best estimate of the upper range is several orders of magnitude larger than the largest observed sample of $\hat{J}(\pi_e|H_1, \pi_b)$, the AM inequality provides significantly tighter lower confidence bounds. However, when the range of the random variable X is not large, i.e., the random variable does not have a heavy upper tail, the AM inequality tends to provide looser lower bound than the other CIs.

The CIs presented so far do not make any assumption about the type of random variable's distribution. In the following, we present two approximate CIs that usually provide tighter lower bounds with the cost of making assumptions about the random variable's distribution.

According to the central limit theorem, the sample mean $\overline{X}_n = \frac{1}{n}\sum_{i=0}^{n} X_i$ becomes normally distributed as $n \to \infty$. Given the n samples are independent and each drawn from a distribution with mean μ_i and finite variance σ_i^2, then $\overline{X}_n$ distribution converges to a normal distribution with mean $\frac{1}{n}\sum_{i=0}^{n} \mu_i$ and variance $\frac{\sum_{i=0}^{n} \sigma_i^2}{n}$ [SS94]. If we assume that the central limit theorem is valid for our $\overline{X}_n$, then we can apply the one-tailed Student's t-test to find the lower confidence bound of the estimator.

$$Pr\Bigg(\mathbb{E}\Bigg[\frac{1}{n}\sum_{i=1}^{n} X_i\Bigg] \geq \frac{1}{n}\sum_{i=0}^{n} X_i - \frac{\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X}_n)^2}}{\sqrt{n}} t_{1-\delta,n-1}\Bigg) \geq 1-\delta, \tag{3.31}$$

where $t_{1-\delta,n-1}$ is the $100(1-\delta)$ percentile of the Student's t distribution with $n-1$ degrees of freedom. The t-test is known to be to be overly conservative when the X_i's distribution has a heavy upper tail [TTG15]. Heavy upper tail is likely in the distribution of the return in MDP problems.

Inspired from the t-test, one could first estimate the distribution of $\overline{X}_n$ from the available samples, and then produce a confidence bound specialized to that distribution. This is the main idea of bootstrap confidence bounds [CB00], and it was first described by Efron [ET94]. The advantage of this approach is it offers tight confidence bounds if the underlying assumption about the $\overline{X}_n$ distribution is valid. However, its main drawback is it cannot guarantee at most a δ error rate because of the assumption it makes. In fact, they are sometimes referred to as trying to produce an error rate of approximately δ, as opposed to upper bounding the error rate by δ.

Although the bootstrap confidence bounds are approximate, some of them are used frequently in medical applications. For example, bias corrected and accelerated (BCa) bootstrap proposed by Davison and Hinkley [DH97] is popular in medical research. This might infer that the lower bounds calculated by bootstrap methods may be sufficiently reliable even for risk-averse applications. It is easier to understand BCa if expressed as an algorithm than a single equation. Algorithm 6 presents the pseudocode of the BCa.

Algorithm 6 Bias corrected and accelerated (BCa) bootstrap

```
Require: $\{X_1, ..., X_n\}$ independent samples of the random variable X
Require: The error rate $\delta$
Require: The number of bootstrap resamplings $B$
 1: Compute the sample mean $\bar{X}_n \leftarrow \frac{1}{n}\sum_{i=1}^{n} X_i$
 2: for $k = 1 : B$ do
 3:     Take n random samples with replacement from $\{X_1, ..., X_n\}$
 4:     Set $\xi_k$ to be the sample mean of these $n$ resamples $\xi_k =\leftarrow \frac{1}{n}\sum_{i=1}^{n} X_i^k$
 5: end for
 6: Sort $\xi_k$s and put them in the vector $\boldsymbol{\xi} = (\xi_1, \xi_2, ..., \xi_B)$ such that $\xi_j \leq \xi_w$ for all $1 \leq j \leq w \leq B$
 7: Compute the bias constant $z_0 \leftarrow \Phi^{-1}\left(\frac{\#\{\xi_k < \bar{X}_n\}}{B}\right)$
 8: for $i = 1 : n$ do
 9:     Set $y_i$ to be the sample mean, excluding the $i^{th}$ element: $y_i \leftarrow \frac{1}{n-1}\sum_{j=1, j \neq i}^{n} X_j$
10: end for
11: $\bar{y} \leftarrow \frac{1}{n}\sum_{i=1}^{n} y_i$
12: Compute the skewness $a \leftarrow \frac{\sum_{i=1}^{n}(\bar{y}-y_i)^3}{6[\sum_{i=1}^{n}(\bar{y}-y_i)^2]^{3/2}}$
13: $z_L \leftarrow z_0 - \frac{\Phi^{-1}(1-\delta)-z_0}{1+a(\Phi^{-1}(1-\delta)-z_0)}$
14: $Q \leftarrow (B+1)\Phi(z_L)$
15: $l \leftarrow min\{\lfloor Q \rfloor, B-1\}$
16: return The $(1-\delta)$ confidence lower bound as $\bar{X}_n + \frac{\phi^{-1}(\frac{Q}{B+1})-\phi^{-1}(\frac{l}{B+1})}{\phi^{-1}(\frac{l+1}{B+1})-\phi^{-1}(\frac{l}{B+1})}(\xi_{\lfloor Q \rfloor+1} - \xi_{\lfloor Q \rfloor})$
```

Bootstrap methods work by creating multiple resamples with replacement from a single set of observations, and then computing the statistic of interest's distribution from the resample sets, lines 2 to 6. Here, the statistic of interest is the mean value, and if following the most basic bootstrap method, one could calculate the $(1-\delta)$-confident lower bound as $\xi_{\lfloor B(1-\delta)\rfloor}$, where B is the number of bootstrap resamplings. However, BCa brings more accuracy to the confidence intervals by first assuming a normal distribution for the mean value (according to the central limit theorem), and then correcting this assumption by counting for the data's bias and skewness. The bias constant is computed in line 7, and the skewness is computed in lines 8 to 12. The adjustments of the confidence bound to the bias constant and skewness are carried out on lines 13-16.

In summary, in this section, we reviewed the most popular IS-based estimators for off-policy evaluation of the π_e's performance and brought several CIs that are suggested in the literature to compute confidence bounds for the mentioned estimators. The choice of the IS estimator and the CI highly depends on the specifications of the particular problem, and to the best of the author's knowledge, there is no one-size-fits-all solution. For example, in problems where the horizon of the MDP is large, it is recommended to avoid applying IS and PDIS. Or when the upper limit for the π_e's return is high but rarely happens, CH and MPeB inequalities are not recommended. In chapter 6, we explain the specifications of the environments that are targeted by this thesis and demonstrate the off-policy evaluation method with confidence bounds that matches our target environments, and we implemented for the test environment of chapter 7.

CHAPTER 4

Data Efficient Deep Reinforcement Learning Algorithms

In this chapter, we study recent advances in deep RL and explain the exercised techniques in the data-efficient deep RL agent implemented in the proposed trustable RL algorithm in chapter 6. We should mention that deep RL is a very active field of research with a high publication rate, and therefore, this chapter does not provide a comprehensive literature review on deep RL. Instead, it aims at presenting recent advances and techniques in deep RL algorithms that are pertinent to our problem setting.

More precisely, our problem setting requires a highly data-efficient agent with off-policy learning capability designed to deal with episodic tasks. Among various RL approaches, actor-critic methods are better aligned with our problem specification [WBH+16], as mentioned in section 2.5. Hence, the overall direction of this chapter is toward presenting the most recent and effective advances in off-policy deep actor-critic algorithms.

In the following, we first present the deep Q-network and its advancements. Because not only is it the most common deep RL agent, but also deep Q-networks (or variants of them like value networks) serve at the core of the latest actor-critic algorithms as the critic value function approximator. After establishing a data-efficient Q-network, we present advancements in deep policy gradient, actor-critic algorithms that make them more practical by reducing their variance and enabling off-policy learning. At the end, we demonstrate the deep RL agent's architecture that is implemented in the proposed Trustable Deep RL algorithm in chapter 6.

4.1 Vanilla Deep Q-Network and Its Pitfalls

In a conventional categorization of value-based RL algorithms, they are divided into two main classes, which are tabular solution methods and approximate solution methods. In tabular solution methods, the value functions are usually represented in tables or arrays; therefore, they implicitly impose the condition that the state and action spaces must be small for the value functions to be representable as arrays or tables. For example, the Q-learning algorithm presented in chapter 2 is a tabular solution method.

It is obvious that the tabular methods are not naturally extendable to arbitrary large state and action spaces because there are too many state-action pairs to store in the memory. Besides, the learning process for very large state and action spaces is too slow. In such problems, we ought to generalize from experienced encounters. The approximate solution methods are any combination of value-based reinforcement learning methods with function generalization methods. The type of generalization that is of interest here is called function approximation since it attempts to generalize the entire function from examples of that.

Undoubtedly, neural networks are one of the best function approximators we currently know. Any algorithm that uses a neural network to approximates value functions (or as we see later, the agent's policy) in an RL setting can be classified as deep RL. However, the most commonly used deep RL algorithms are designed to approximate action-value functions. These networks are known as deep Q-networks, and they approximate the Q-values of different actions through separate output nodes. Figure 4.1 shows a simplified demonstration of deep Q-network for an MDP with the action space of size 2.

The deep Q-network is trained with a variant of the Q-learning algorithm. Similar to temporal difference Q-learning, deep Q-network updates the $Q(s, a)$ estimate towards the TD_{target}. However, unlike the temporal difference Q-learning that uses temporal difference update regime, deep Q-network uses stochastic gradient descent as the update regime of the network parameters. The deep Q-network loss function for time step t is

$$Loss = \big(Q_{target}(S_t, A_t) - \hat{Q}(S_t, A_t)\big)^2, \tag{4.1}$$

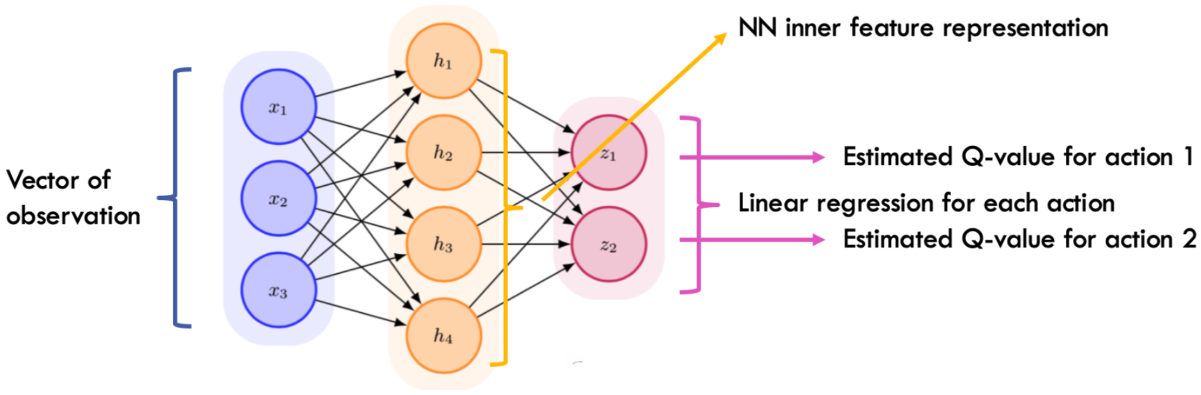

NN inner feature representation
Vector of observation
x_1
x_2
x_3
h_1
h_2
h_3
h_4
z_1
z_2
Estimated Q-value for action 1
Linear regression for each action
Estimated Q-value for action 2

Figure 4.1: A simplified demonstration of deep Q-network for an MDP with action space size of 2.

where $Q_{target}(S_t, A_t) = R_{t+1} + \gamma \max_{a'} \hat{Q}(S_{t+1}, a')$. Here we assumed that at each time step, the agent encounters a new experience tuple $< S_t, A_t, R_{t+1}, S_{t+1} >$, and we update the network parameters by one iteration of gradient descent, accordingly. This approach is known as online Q-network first presented by [Lin93] and could outperform the best human backgammon players 27 years ago. However, it was not successful in playing other Atari games.

The vanilla Q-network suffers from large oscillations in the learning curve and often catastrophic divergence. The instability of Q-network is rooting in two sources. First, learning the control policy from consecutive experiences is insufficient due to the high correlation between experiences. When learning on-policy, the current parameters directly affect the control policy, which consecutively determines the next data sample. For example, if the maximizing action is to move left, then the training samples will be dominated by samples from the left-hand side; if the maximizing action then switches to the right, then the training distribution will also switch. Second, in a supervised learning setting of neural network training, we always have the ground truth as a fixed value, and we try to minimize the loss value to get closer to the ground truth. However, in the RL setting, the target values are moving as we update the Q-values, causing instability and confusion in the agent's policy training.

4.2 Advances in Deep Q-networks

4.2.1 Experience Replay

In 2013, Mnih et al. in their highly noted paper [MKS+13], presented a successful integration of Q-network with modern deep neural network architectures and advanced hardware technology toward achieving a scalable RL algorithm. The main idea in their proposed methodology is to store agent's experiences at each time step $e_t =< S_t, A_t, R_{t+1}, S_{t+1} >$ in a replay memory, and then at each iteration of network update, perform minibatch gradient descent (instead of stochastic gradient descent) by randomly drawing a batch of experiences from the memory.

This approach has several benefits over the standard online Q-learning. First, each step of the experience is potentially used in many weight updates, which leads to greater data efficiency. Second, learning directly from consecutive samples is inefficient because of the strong correlations between the samples. Randomizing the samples breaks these correlations, and therefore, reduces the variance of the updates. Third, when learning on-policy, the current parameters determine the next data sample that the parameters are trained on, which causes unwanted feedback loops that may lead to the parameters getting stuck in a poor local minimum. By using experience replay, the behavior distribution is averaged over many of its previous states, smoothing out learning and avoiding oscillations or divergence in the parameters.

The second contribution of [MKS+13] is to benefit from 2D convolutional layers in their neural network architecture. Convolutional neural networks are well-known for their outstanding ability to extract features from images; therefore, they greatly enhance learning successful control policies from raw video data in video and Atari games.

4.2.2 Separate Target Network

In 2015, Mnih and his colleagues at DeepMind presented another paper for stabilizing deep RL [MKS+15]. This time they addressed the Q_{target} variation. Their idea is to separate the target network from the Q-network and update the target network parameters periodically, thereby reducing the correlation of the target value with the Q-values. In their methodology implementation, the Q-network and the target network are the same, except that the Q-network parameters get updated at every iteration with mini batches from the experience replay memory approach presented in [MKS+13]. While the target network freezes and is updated periodically only at every $\tau = 10000$ iteration.

The loss function after implementing the experience replay memory and freezing target network can be expressed as follows.

$$Loss = \frac{1}{n}\sum_{i=i}^{n}\left[Q_{target}(s_i, a_i) - \hat{Q}(s_i, a_i; w_Q)\right]^2, \tag{4.2}$$

$$Q_{target}(s_i, a_i) = r_i + \gamma \max_{a'} \hat{Q}(s'_i, a'; w_{target}), \tag{4.3}$$

where $< s_i, a_i, r_i, s'_i >$ are random samples of experience tuples $< S_t, A_t, R_{t+1}, S_{t+1} >$ stored in the replay memory, n is the size of the minibatch, w_Q is the Q-network parameters, and w_{target} is the target network parameters.

4.2.3 Double Deep Q-network

Another great advance in deep Q-network is the adaptation of Double Q-learning with DQN [VHGS16]. In both Q-learning and deep Q-network, the target is the summation of the reward and the best action-value function in the next step. But how we can make sure that the next state's best action is $\underset{a'}{\operatorname{argmax}}\, Q(S_{t+1}, a')$ when $Q(s, a)$ is just an estimate of actual worthiness of actions. In other words, the maximum over the estimated values is implicitly used as an estimate of the maximum value, which can lead to significant positive bias. In fact, Hasselt et al. showed in their paper [VHGS16] that this positive bias caused overestimation in deep Q-network, and implementing the Double Q-learning leads to much better performance and more accurate value estimation.

Double Q-learning aims at reducing the maximization bias by choosing the best action from one estimator and to estimate the best action-value by another estimator. Thanks to the separate target network proposed by [MKS+15], the deep Q-network already has two estimators. Accordingly, we can decompose the max operation in the target into action selection from the Q-network and action evaluation from the target network as follows:

$$Q_{target}(s_i, a_i) = r_i + \gamma \hat{Q}\big(s'_i, \underset{a'}{\text{argmax}}\, \hat{Q}(s'_i, a'; w_Q); w_{target}\big). \tag{4.4}$$

This simple to implement yet highly intelligent adaptation of double Q-learning leads to striking improvements in several Atari games, in some cases bringing scores much closer to human performance, or even surpassing that.

4.2.4 Dueling Double Deep Q-network

The idea of dueling network architecture comes from the very early notation of state value function and state-dependent action advantage function separation in Q-values [BI93],

$$Q(s, a) = V(s) + A(s, a). \tag{4.5}$$

Decomposing the Q-values into two separate streams intuitively can lead to learning which states are valuable or not valuable regardless of the actions. This is particularly useful in states where its action does not affect the environment in any perceivable way for the agent. For example, consider a student that unfortunately had a poor performance during the quarter and the final exam. Regardless of whether the student studies or not after the final exam, he can no longer improve his final grade. However, studying prior to the final exam could have changed his final grade. Therefore, studying in an earlier state could bring a significant advantage while it does not have an advantage and would not affect his grade after the final exam. This fact is very obvious to the human mind that no student would spend any effort in a course after the final exam, but this is not clear for an RL agent and might lead to unnecessary policy search in those states. In fact, implementing the advantage updating in the Q-learning algorithm was shown to speed up the convergence [HBIK95].

The dueling deep Q-network is proposed by Wang et al. [WSH+16] in which they explicitly separate the representation of state values and action advantage values in the architecture of the Q-network. As presented in 4.2, the dueling architecture comprises two streams representing the state value and the advantage functions, while sharing common feature

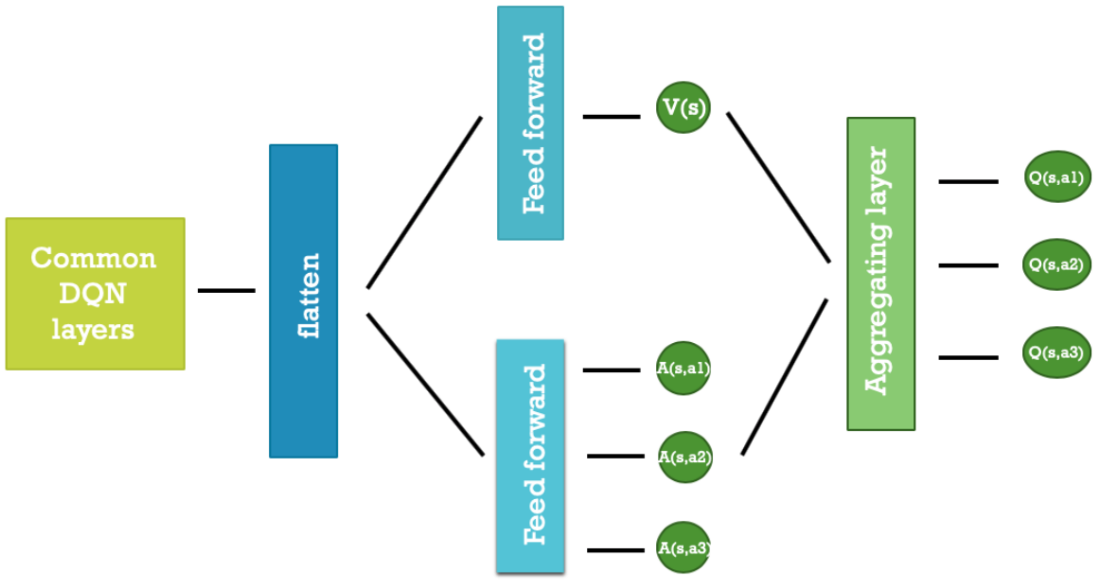


Figure 4.2: Demonstration of the dueling deep Q-network for an action space size of 3.

learning layers. The two streams are combined back together via a special aggregating layer, presented in equation 4.6, to produce an estimate of the action-value function. Wang et al. in [WSH+16] showed that the dueling architecture converges at about the same speed of Q-network in environments with a few actions while converges much faster in environments with larger action spaces.

$$Q(s,a) = V(s) + A(s,a) - \frac{1}{|\mathcal{A}|}\Sigma_i A(s,a_i) \tag{4.6}$$

4.2.5 Prioritized Experience Replay

The experiences were uniformly sampled from the replay memory in the basic deep Q-network with experience replay memory. However, this approach simply replays transitions at the same frequency as experienced originally regardless of their significance. Consequently, the rich experiences that occur rarely have almost no chance to be selected. Schual et al. in their paper [SQAS15], proposed a methodology for prioritizing experiences to replay important transitions more frequently, and therefore, learn more efficiently. With prioritize experience replay, we try to change the sampling distribution by using a criterion to define the priority of each tuple of experience.

The proposed method is to prioritize experiences when there is a significant difference between the predicted Q-value and the target value. The priority value is defined as

$$p_i = |\delta_i| + e, \tag{4.7}$$

where δ_i in a double deep Q-network with separate target network is as follows:

$$\delta_i = \hat{Q}(s_i, a_i; w_Q) - \left(r_i + \gamma \hat{Q}\big(s_i', \operatorname*{argmax}_{a'} \hat{Q}(s_i', a'; w_Q); w_{target}\big) \right), \tag{4.8}$$

and $e \in [0, 1]$ is a small constant value to ensure no experience has 0 probability. Concretely, the sampling probability of each experience tuple is defined as

$$P_i = \frac{p_i^\alpha}{\sum\limits_{k=1}^{N} p_k^\alpha}. \tag{4.9}$$

The exponent α determines how much prioritization is used, with $\alpha = 0$ corresponding to the uniform case, and N is the total size of the experience replay buffer.

Introducing prioritized experience replay causes bias in the estimation because the estimation of the action-value functions relies on the expectation of the sampled experiences. The bias is due to changed distribution of the samples; and therefore, changes the solution that the estimates will converge to. To correct this bias, the importance-sampling weights are applied to the minibatch lost function.

$$IS\ factor_i = \big(\frac{1}{N} \cdot \frac{1}{P_i}\big)^\beta. \tag{4.10}$$

In equation 4.10, β controls how much these importance sampling factors affect learning. In the beginning it is around 0 and at the end of learning when our Q-values begin to converge, it becomes close to 1.

4.3 Deep Actor-Critic

The dueling double deep Q-network with prioritized experience replay was used to be the most sample efficient agent by a significant margin on Atari games simulated environments of

[BNVB13], [SQAS15, WSH+16]. However, we need to do better than that because finding the best policy by greedy action selection with respect to the Q-functions is costly for large action spaces and not possible for continuous action spaces. This need for better sample efficiency is even more compelling when deploying the RL solutions in real-world environments.

Policy gradient methods have been at the center of highly notable advancements in RL [SLH+14, LHP+15, SLA+15, MBM+16, SHM+16]. Actor-critic methods are the dominant class of policy gradient methods that are applicable to both continuous and discreet state and action spaces. These methods are dominant because they have superior learning speed owing to their lower variance in gradient estimation. The lower variance is the result of integrating the value function learning with policy learning. Therefore, actor-critic consists of two models; the model that learns the policy is known as the actor, and the model that learns the value function is known as the critic.

As presented in section 2.5.4, the actor model is based on a parametrized policy and an estimator for the parameters. This estimator leverages upon the policy gradient theorem to optimize the parameters via gradient ascent. The critic model is an estimator of the value functions whose estimations are utilized in policy gradient calculation. All the advancements and improvements upon vanilla actor-critic, algorithms 4 and 5, are either (1) to improve the optimal policy parameters estimator by reducing the gradient's variance and bias, (2) to improve the value function estimator through reducing its bias and variance. In the following, we present the most recent and effective techniques and advancements that are proposed in the literature to improve the actor-critic learner.

4.3.1 Deep Q-network Critic

In the actor-critic algorithms, the role of the critic is to estimate the value functions. Then the estimated value functions weight the policy parameters' updates such that the policy parameters increase most in the directions that favors the actions with the highest value functions, equation 2.26. Consequently, the critic's evaluation of the actor's actions affects the future update direction of the actor's policy.

The quality of the critic's value function estimations directly influences the performance of the whole actor-critic algorithm. As discussed in section 4.2, the recent advancements of the deep Q-networks highly improve the variance and bias of the value function estimation, and therefore, make it a compelling choice for deployment in the critic formalization.

The deployment of the deep Q-networks in actor-critic agents is first proposed by Mnih et al. [MBM$^+$16] at deep mind. They used a deep Q-network as the critic in their popular asynchronous advantage actor-critic (A3C) algorithm. Unlike the deep Q-networks we introduced so far, this deep Q-network is not off-policy and does not hold an experience replay memory. Instead, it relies on parallel on-policy actor-learners. To compensate for the experience samples' correlation in on-policy settings, they run different exploration policies in different threads of learners. This provides a viable alternative to experience replay, since different exploration diversifies and decorrelates the data, and consequently, stabilize the network.

It should be noted that multiple online interactions of the learner and the environment is not feasible in many real-world RL problems. Consequently, the deep Q-network critic of A3C does not apply to our problem setting. Because in our target environment, there is no online access to the environment, and the agent is limited to learn from a history of interaction data. The critic we implement in chapter 5 is an off-policy counterpart of the A3C's and leverages upon most advancements presented in section 4.2.

4.3.2 Off-Policy Actor's Gradient Estimation

In policy gradient actor-critic algorithms, the actor's main role is to provide a reasonably good estimation of the objective function's gradient with respect to the policy parameters. Controlling the variance in the off-policy gradient estimators without causing bias error is notoriously hard [WBH$^+$16]. Equation 2.35 by Degris et al. [DWS12] provides a starting point for deriving a competent gradient estimator. For ease of reading, this equation is repeated hereunder.

$$\nabla_{\boldsymbol{\theta}}\, J\big(\pi_l(a|s;\boldsymbol{\theta})\big) \propto\, \mathbb{E}_{\pi_b,E}\left[\frac{\pi_l(a|s;\boldsymbol{\theta})}{\pi_b(a|s)} Q_{\pi_l}(s,a)\, \nabla_{\boldsymbol{\theta}}\big[\ln\, \pi_l(a|s;\boldsymbol{\theta})\big]\right] \tag{4.11}$$

Equation 4.11 is itself an approximation of the gradient direction, and as shown by Degris et al. [DWS12], it has proven policy improvement grantees. To improve upon equation 4.11, one could (1) enhance Q_{π_l} estimation, (2) control the variance that the importance weight introduces to the gradient estimation. In the following, we further discuss both improvement possibilities.

4.3.2.1 Improved Estimation of the Action-Value Function

Perhaps the most basic action-value estimator, which is derived directly from the action-value definition in equation 2.9, is the return-based Monte Carlo estimator, equation 4.12. It is straightforward to show from the action-value definition that the Monte Carlo estimator is unbiased; however, it suffers from high variance as it includes the summation of many random variables.

$$\hat{Q}_{\pi_l}(S_t, A_t) = \sum_{k=0}^{terminal} \gamma^k R_{t+k+1} \tag{4.12}$$

Another trivial action-value estimator is the popular TD(0) estimator, equation 4.13. TD(0) is low-variance and easy to compute but highly biased because the exact value of the next state's value function, $V(S_{t+1})$, is unknown, and therefore, we ought to estimate $Q(S_t, A_t)$ from estimated $V(S_{t+1})$. This estimation towards another estimate leads to considerable bias.

$$\hat{Q}_{\pi_l}(S_t, A_t) = R_{t+1} + \gamma \hat{V}_{\pi_l}(S_{t+1}) \tag{4.13}$$

To trade-off between bias and variance, Degris et al. [DWS12] used off-policy lambda return. This estimator applies a weighted sum of the high-bias, low-variance TD(0) estimator and the low-bias, high-variance return-based Monte Carlo estimator. Therefore, it provides a way of control over the bias and variance trade-off. The off-policy lambda return is presented in equation 4.14.

$$R_t^\lambda = R_{t+1} + (1-\lambda)\,\gamma\,\hat{V}_{\pi_l}(S_{t+1}) + \lambda\,\gamma\,\rho(S_{t+1}, A_{t+1})\,R_{t+1}^\lambda \tag{4.14}$$

The variable $\rho(S_{t+1}, A_{t+1})$ in this equation is the importance weight and is equal to $\frac{\pi_l(A_{t+1}|S_{t+1})}{\pi_b(A_{t+1}|S_{t+1})}$. Degris et al estimated the action-value function $\hat{Q}_{\pi_l}(S_t, A_t)$ with the lambda return at time t. The downside of this estimator is it requires the user to know how to choose lambda ahead of time to trade-off bias and variance. Moreover, when using small values of lambda to reduce variance, occasional large importance weights can still cause instability [WBH+16].

Munos et al. [MSHB16] proposed a more efficient and lower variance off-policy estimator for the action-value function called the Retrace estimator. This estimator and the related algorithm are very recent but fundamental developments in RL. We refer the interested reader to [MSHB16] in which Munos et al. proved that the retrace algorithm converges (in the tabular case) to the action-value function of the target policy for any behavior policy. The Retrace estimator can be expressed recursively as follows:

$$Q^{ret}(S_t, A_t) = R_{t+1} + \gamma\bar{\rho}(S_{t+1}, A_{t+1})\Big[Q^{ret}(S_{t+1}, A_{t+1}) - \hat{Q}_{\pi_l}(S_{t+1}, A_{t+1})\Big] + \gamma\hat{V}_{\pi_l}(S_{t+1}), \tag{4.15}$$

where $\bar{\rho}(S_{t+1}, A_{t+1})$ is the truncated importance weight that is defined as $\bar{\rho}(S_{t+1}, A_{t+1}) = min\{c, \rho(S_{t+1}, A_{t+1})\}$, and $\hat{V}_{\pi_l}(S_{t+1})$ is the expected value of the current estimates of the action values over the learning policy, $\mathbb{E}_{\pi_l}\left[\hat{Q}_{\pi_l}(S_{t+1}, a)\right]$.

The recursive Retrace equation, 4.15, requires an estimate of the action values, $\hat{Q}_{\pi_l}$. Here, we leverage upon the critic's deep Q-network of section 4.3.1 and directly take its outputs as the estimated action values. Finally, to approximate the gradient via equation 4.11, we use Q^{ret} as an estimate of Q_{π_l}.

The Retrace estimator has a significantly lower bias because it benefits from multi-step returns. Consequently, applying the Retrace estimator to gradient computation results in better gradient estimations. Wang et al. [WBH+16] took one step further and utilized the Retrace estimator as the target of the mean squared loss of the critic's neural network.

Because Retrace is return-based, it enables faster learning of the critic. In this research, we follow Wang et al. and use Q^{ret} as the deep Q-network critic's target, equation 4.16. Therefore, the benefits of Q^{ret} are twofold: reduces bias in the gradient estimations, and enables faster learning of the critic.

$$Loss = \frac{1}{n} \sum_{i=i}^{n} \left[Q^{ret}(s_i, a_i) - \hat{Q}(s_i, a_i; w_Q) \right]^2, \tag{4.16}$$

4.3.2.2 Importance Weight Truncation with Corrections

Looking back at 4.11, we so far enhanced our estimation of the Q_{π_l}. In this section, we take measures to safeguard against large values of the importance weights that could cause instability in gradient estimation. One trivial solution is to truncate the importance weight, similar to equation 4.15. However, truncation causes bias error in the estimation. Wang et al. [WBH+16] suggest a bias correction term for the truncated importance weight. The off-policy gradient estimator with truncated importance weight and the bias correction term is presented in equation 4.17. For more readable notation, hereon we use $\hat{g}(s, a)$ to denote the estimated expected return's gradient with respect to the policy parameters, $\widehat{\nabla_{\boldsymbol{\theta}} J}\big(\pi_l(a|s; \boldsymbol{\theta})\big)$.

$$\begin{aligned} \hat{g}(s, a) = \mathbb{E}_E \Bigg(& \mathbb{E}_{\pi_b} \Big[\bar{\rho}(s, a) Q_{\pi_l}(s, a) \; \nabla_{\boldsymbol{\theta}} \big[\ln \; \pi_l(a|s; \boldsymbol{\theta}) \big] \Big] + \\ & \mathbb{E}_{a_i \sim \pi_l} \Big[max\{0, \frac{\rho(s, a_i) - c}{\rho(s, a_i)}\} \; Q_{\pi_l}(s, a_i) \; \nabla_{\boldsymbol{\theta}} \big[\ln \; \pi_l(a_i|s; \boldsymbol{\theta}) \big] \Big] \Bigg) \end{aligned} \tag{4.17}$$

By clipping the importance weight in the first term of equation 4.17, we make sure the variance is kept bounded, and by adding the second term, we ensure that we compensate the bias caused by gradient clipping and the gradient estimate remains unbiased. Note that the second term involves an expectation over the learning policy rather than the behavior policy. Therefore, at each state, the correction term is active for all the actions that have importance weights larger than c. In particular, when choosing a large value for c, the correction term only becomes effective when the variance of the original off-policy estimator of equation 4.11 is very high. If this happens, the truncated importance weight in the first term is at most c, while the correction weights in the correction term are at most 1.

Equation 4.17 expresses the gradient estimator in terms of expected values. To use that with retrospective episodes from the behavior policy, we can approximate it by the following equation:

$$\begin{aligned} \hat{g}(S_t, A_t) = \bar{\rho}(S_t, A_t)Q^{ret}(S_t, A_t)\, \nabla_{\boldsymbol{\theta}}\big[\ln\, \pi_l(A_t|S_t;\boldsymbol{\theta})\big]+ \\ \mathbb{E}_{a_i\sim\pi_l}\Big[max\{0, \frac{\rho(S_t, a_i) - c}{\rho(S_t, a_i)}\}\, \hat{Q}(S_t, a_i; w_Q)\, \nabla_{\boldsymbol{\theta}}\big[\ln\, \pi_l(a_i|S_t;\boldsymbol{\theta})\big]\Big]. \end{aligned} \tag{4.18}$$

Note that $Q_{\pi_l}(S_t, A_t)$ in the first term of equation 4.18 is modeled by $Q^{ret}(S_t, A_t)$, while in the second term is modeled by the outputs of the critic's deep network, $\hat{Q}(S_t, a_i; w_Q)$. It is due to the fact that the Q^{ret} value is not available for all actions at state S_t but the critic does compute all actions' Q-values.

4.3.2.3 Baseline Subtraction

To further reduce the variance of the gradient estimation, we subtract a baseline from the action values, as previously presented in section 2.5.3. Here we hold to the classical baseline $V(S_t)$ because, as shown by schulman et al. [SML+15], it yields almost the lowest possible variance. To estimate the state-value functions, we simply take expectation of $\hat{Q}(S_t, a_i; w_Q)$ under the learning policy π_l. The gradient estimator with baseline is presented as follows:

$$\begin{aligned} \hat{g}(S_t, A_t) = \bar{\rho}(S_t, A_t)\big[Q^{ret}(S_t, A_t) - \hat{V}_{\pi_l}(S_t)\big]\, \nabla_{\boldsymbol{\theta}}\big[\ln\, \pi_l(A_t|S_t;\boldsymbol{\theta})\big]+ \\ \mathbb{E}_{a_i\sim\pi_l}\Big[max\{0, \frac{\rho(S_t, a_i) - c}{\rho(S_t, a_i)}\}\, \big[\hat{Q}(S_t, a_i; w_Q) - \hat{V}_{\pi_l}(S_t)\big]\, \nabla_{\boldsymbol{\theta}}\big[\ln\, \pi_l(a_i|S_t;\boldsymbol{\theta})\big]\Big]. \end{aligned} \tag{4.19}$$

Equation 4.19 presents the final gradient estimator that is utilized to update the parameters of the actor's policy.

CHAPTER 5

Architecture of the Network Used in Trustable Deep RL

Some of the current advancements in deep RL that brought great new achievements in the field were reviewed in the previous chapter. We should emphasize that this review is not exhaustive, and the purpose is to introduce only the techniques relevant to our problem setting, i.e., data-efficient, off-policy learning for episodic tasks with continuous state space and discrete, finite action space. In this chapter, we demonstrate the architectures as well as the implementation details of the deep RL agent utilized in the proposed data-efficient Trustable Deep RL methodology.

The agent is an actor-critic algorithm with two separate deep neural networks, i.e., the actor network and the critic network. The actor network outputs the actions' probability distribution vector, and the critic network outputs the actions' value function vector. The state is the input to both the actor and the critic networks. Since the size of the state variables in the test environment, chapter 7, is small, we use fully connected layers rather than convolutional or recurrent layers.

The agent also enjoys a replay memory. However, unlike the experience replay memory of sections 4.2.1 that stores experience tuples at each time step separately, this one stores the whole episode of the interactions from the initial state to the terminal state. For example, it could store the episodes generated when human experts were interacting with the environment. The neural network training is performed through mini-batches of samples from the episode replay memory. In the following, we first elaborate upon the actor network, then we demonstrate the critic network. At the end, we provide more details about the replay memory and the sampling strategy.

5.1 The Actor Network

The actor network consists of three fully connected layers. The size of the first and the second hidden layers are 6 and 12, respectively, each followed by ELU activation. The output layer has 5 nodes, each node representing one of the 5 possible actions. The output layer has Softmax activation to turn the logits into meaningful probabilities. We must remark a subtle difference between the purpose of applying Softmax function in multi-class classification networks versus the actor networks. In classification networks, the Softmax calculates the probability of input belonging to each class. However, in the actor networks, it represents the action taking policy.

The actor network's loss calculation is performed according to equation 5.1. This equation requires Q^{ret} value. Since Q^{ret} is computed recursively, equation 4.15, we need to perform the loss calculation from the trajectory's last step and move backward to the first step. To provide a more concrete example, assume $H := \big(S_1, A_1, R_2, S_2, A_2, R_3, ..., S_{L-1}, A_{L-1}, R_L, S_L\big)$ is sampled from the replay memory. If S_L is a terminal state, then $Q^{ret}(S_{L-1}, A_{L-1}) = R_L$. Otherwise, it is equal to $R_L + \gamma\hat{V}_{\pi_l}(S_L)$. Since $\hat{V}_{\pi_l}(S_L)$ is not known, we take the expectation of $Q(S_L, a_i; w_{critic})$ under the actor network's policy $\pi_l(S_L, a_i; \boldsymbol{\theta})$.

$$\begin{aligned} Loss_{actor}(S_t, A_t) = \bar{\rho}(S_t, A_t)\big[Q^{ret}(S_t, A_t) - \hat{V}_{\pi_l}(S_t)\big]\ \big[\ln\ \pi_l(A_t|S_t; \boldsymbol{\theta})\big] + \\ \mathbb{E}_{a_i\sim\pi_l}\Bigg[max\{0, \frac{\rho(S_t, a_i) - c}{\rho(S_t, a_i)}\} \\ \big[Q(S_t, a_i; w_Q) - \hat{V}_{\pi_l}(S_t)\big]\ \big[\ln\ \pi_l(a_i|S_t; \boldsymbol{\theta})\big]\Bigg]. \end{aligned} \tag{5.1}$$

Continuing backward from the trajectory's last step, we compute the loss of each step according to 5.1 and then add it to the trajectory's total loss. One step of the gradient descent is performed for the actor network when the total trajectory's loss is computed.

To regenerate the gradient as presented in equation 4.19, the loss's gradient is taken assuming only the policy inside the natural logarithm depends on $\boldsymbol{\theta}$. Finally, the network is trained with mini-batch gradient descent, and Adam is chosen as the optimizer.

5.2 The Critic Network

The critic network consists of three parts, the common feature learning layers, the state value function stream, and the advantage function stream. The two separate value function and advantage function streams embody the dueling network implementation of section 4.2.4. The common feature learning layers include two feed forward layers with ELU activation. Since the input of the network is a small vector of the size three, we designed the two common layers with 6 and 12 nodes, respectively. Moreover, the output of the common layers is already flat; hence there is no need for additional flattening layers, as shown in figure 5.1.

The state value function stream is formed of one feed forward layer of size 6 with ELU activation connected to a single-node layer with no activation function. This single node layer represents the value function of the input state. In parallel with the state value function stream, there is the advantage function stream that includes a 6-node layer with ELU activation and a 5-node layer without activation. The test environment has 5 possible actions; therefore, the output of the advantage stream must have 5 nodes. Finally, the outputs of the state value stream and the advantage stream are aggregated, as explained in equation 4.6.

The loss function is the mean value of the squared differences between the target values and the estimated Q-values, as brought in Equation 4.16. Following [WBH+16], we chose $Q^{ret}(S_t, A_t)$ as the target value. Therefore there is no justification for using a separate target network for the critic. Moreover, retrace is a multi-step return estimator of the Q-values; therefore, it does not suffer from the maximization bias like the TD estimator of equation 4.3. Consequently, we abandon the use of double learning.

Knowing that the Q^{ret} value could be computed recursively, the same recursive process of actor's loss computation is adopted for the critic, i.e., at each step first we compute the Q^{ret}, and then we compute the step's loss by equation 4.16. We continue recursively until we get to the first step. One step of the gradient descent is performed for the critic network when

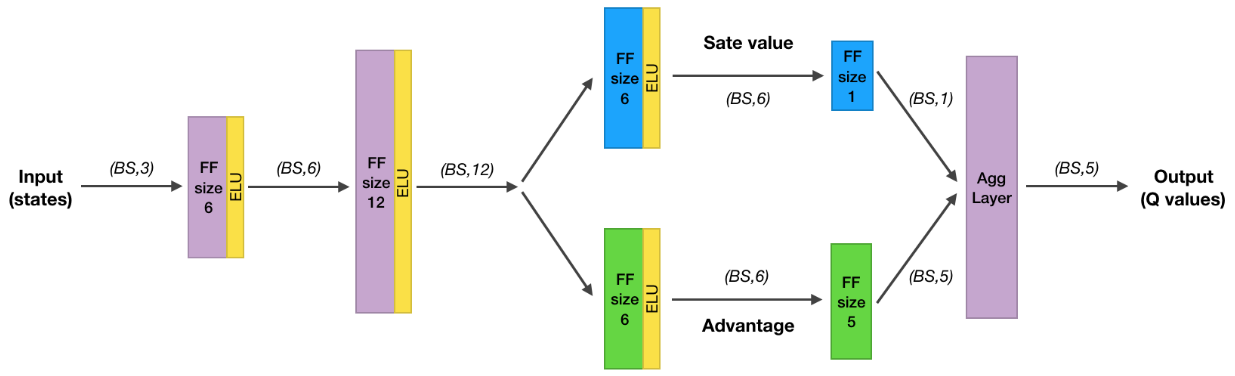


Figure 5.1: Architecture of the critic network.

the total trajectory's loss is computed. Like the actor network, the critic network is trained by mini-batch gradient descent with an Adam optimizer.

5.3 The Replay Memory

The replay memory is a set of episodes of interactions with the environment which are recorded in the past. As a reminder, we define the episode as a sequence of interactions from an initial state to a terminal state, and the trajectory is any consecutive steps of an episode that could be cut out from any part of it with any desired length. Each episode consists of a series of environment's transitions. To comput the actor network and the critic network losses, the following information is required from each transition:

- the state S_t,
- the action A_t,
- the reward received after performing A_t and arriving at the next state,
- the policy that led to choosing A_t,
- a flag that shows if the next state is a terminal state.

Comparing the above list with the experience tuple of section 4.2.1, $e_t =< S_t, A_t, R_{t+1}, S_{t+1} >$, reveals that we do not save the next state mainly because we do not perform one step temporal difference. However, we keep the policy because the policy is needed for computing the importance weights, and we keep the terminal flag because it is required to compute the Q^{ret}.

For each iteration of the deep networks update, we take mini-batch-sized samples of the episodes from the replay memory. These episodes must have the same length to accommodate the tensor calculation of the deep network packages and the system's GPU. Consequently, they are randomly cut to the size of the batch's minimum length. Further talk about prioritized sampling of trajectoris if you have any

CHAPTER 6

Proposed Trustable Deep RL

In the light of all the notable researches that have been done to address the limitations of RL solutions, we propose a methodology that tackles the two main issues of the current deep RL solutions, which are the data efficiency and the trustability. We believe that these two issues are the main barriers against deploying the state of the art deep RL algorithms to real-life sequential decision-making problems.

Along with the trust and the data efficiency issues in deep RL, there are several assumptions in the current benchmark environments that are not valid in real-world problems and highly restrict deployment of deep RL in unsimulated, real cases. Hence, in the first section of this chapter, we discuss the characteristics of the real environments, and we clearly state the assumptions we made about the target environments. Then in section 6.2, we demonstrate the proposed solutions for increasing data efficiency in the deep actor-critic training procedure. Afterward, we introduce layers of safety-assurance procedures in the algorithms in section 6.3. Section 6.4 presents how we evaluate the performance of the deep RL agent's policy prior to deployment with confidence bounds. Finally, Section 6.5 sums up the entire Trustable Deep RL algorithm by presenting the whole algorithm's training and evaluation pseudocode, as well as the deployment pseudocode.

6.1 Characteristics of the Target Environments

While most deep RL advancements are deployed and evaluated on the Atari 2600 games emulator environment [BNVB13], the games are quite different from the real environments. First, in simulated environments, we have no limit on the number of interactions with the environment. We can generate as much data as needed to train the RL agent adequately. On the contrary, real environments do not possess this attribute, and often the number of acquired experience samples from real environments is very limited. That is the reason why data efficiency is highly critical in real-world applications of RL.

Second, in real-world environments, we are not authorized to explore any action at any state, due to the risks and unaffordable costs they might have. However, in simulated environments, there is no actual risk associated with any action. As mentioned in chapter 2, one of the conditions for achieving the optimal policy in the Q-learning class of algorithms is that the RL agent must continuously visit all state-action pairs. This condition cannot be met in real-world environments with limited and constrained agent-environment interactions. As a result, we cannot claim that RL's outcome policy in real environments is optimal. Yet it might be better than the previously in-service policy.

Third, in simulated environments, the agent usually has online interaction with the environment, which is almost never valid in the real world. Generally, in real environments, a relatively good policy has been in place for several years without any variation, and data set of interaction history according to that policy is available. Most of the time, this retrospective data set is the only source of information to find an improved policy. As a result, one important requirement of applying deep RL to real environments is to shift from on-policy to off-policy learning algorithms. Fortunately, off-policy deep RL algorithms are gaining more attention because they can learn from batches of data, in contrast with online learning that could ingest data one observation at a time.

Forth, the real-world problems are risk-averse, i.e., we cannot deploy any new policy in the environment unless we guarantee the superior performance of the new policy with high confidence. This restriction includes testing the nwe policy on the the environment for evaluation purposes. Inevitably, we are limited to the history of previous experiences generated under the old policy for the evaluation phase. This concern never appears in simulated environments; therefore, their policy evaluation metrics are mostly inapplicable in real environments.

Considering the above fundamental differences between the simulated environments and the real environments, we must frame our methodology within the boundaries of the real environments because we assume that our proposed methodology's potential applications are high-risk real enviroments. In summary, the following assumptions are made for the target environments of the proposed Trustabl Deep RL algorithm. It is required that any potential application of the proposed methodology meets these conditions:

1. A decision-making mechanism, called the behavior policy, is already in use in the system. This behavior policy makes decisions, and the outcomes of those decisions are observed and recorded. RL algorithms can analyze historical data from the behavior policy and the environment interactions to propose a new policy that may be better than the current policy. This new policy is known as the evaluation policy.

2. The new policy cannot be directly tested on the system for evaluation. At the same time, it needs to be proven as safe and trustable prior to deployment.

3. The under study environment must be fully observable because the underlying learning techniques are based on the Markov decision process.

4. The state variables can belong to either continuous or discreet infinite state spaces, but the action space is assumed to be discreet and finite.

5. The agent–environment interaction is episodic,i.e., the interactions last a finite number of time steps and end in a terminal state. Thus, the interactions can naturally break down into a set of separate episodes that each starts with an initial state and ends in one terminal state.

6. The environment is stationary. Our methodology uses the data collected in the past to determine how to act in the future. Without some assumption that the future will resemble the past, the retrospective data cannot be used to inform future decisions.

6.2 Data Efficiency

When applying deep RL to problems beyond the simulated environments, data efficiency is key to the success of the algorithm. The current state of the art deep RL algorithms such as rainbow deep Q-network [HMVH+18], A3C [MBM+16], and ACER [WBH+16] can achieve human-level performance after more than 10 million steps of interactions with the game environment. This number of interaction steps is far from achievable in real-world applications. To provide a more tangible example, assume a medication recommendation tracking that takes $10,000$ patients under investigation for three years with monthly decision points for medication adjustment. This study, at best, would have $360,000$ steps of interaction. Although a study with $10,000$ patients is considered large medical research, it has about two orders of magnitude less number of steps than 10 million interactions of the simulated environment.

The above example emphasizes the importance of data efficiency in applying deep RL to real-world problems. In this research, we take advantage of most of the techniques recommended in the literature to increase the sample efficiency of deep RL which are relevant to off-policy learning. These techniques are elaborated extensively in chapters 2 and 4, and are listed as follows:

- We use a deep actor-critic agent which is more sample efficient than deep Q-networks because it learns the policy and the value functions jointly.
- We use Retrace that is an off-policy action-value function estimator for approximating the gradient in the actor network. Retrace benefits from multi-step return and demonstrates low bias and variance. Hence, it could speed up the learning process.
- We use the Retrace Q-value estimations as the target of the critic network, which results in faster critic learning.
- We implement the importance weight truncation in the off-policy gradient estimation of the actor network. Truncation reduces the variance significantly, thus improves sample

efficiency.

- We exercise an episode replay memory, which uses each step of the past experience many times.
- We implement the dueling architecture in the critic network. The dueling architecture speeds up convergence in environments with large action spaces.

In addition to the above list, we propose three more techniques for increasing the data efficiency. We observed that these techniques greatly improved our algorithm's data efficiency in the test environment.

1. Prior to training the actor-network in the deep actor-critic setting, we initialize the actor network's weights such that the network mimics the previous behavioral policy. The initial weight adjustment is carried out by training only the actor network in a multi-class classification setting while considering the performed actions as the class labels. This network is also needful in the other parts of the proposed Trustable deep RL algorithm to estimate the behavioral policy when the explicit behavioral policy is not attainable. Hence, we keep a copy of the trained network and name it as the behavioral policy estimator network.
2. We initialize the critic network before training the actor-critic structure, as well. A Dueling Double Deep Q-network with the same architecture as the critic network is trained following section 4.2. Then, the wights of this deep Q-network, we name it as the Q-val network, are copied into the weights of the critic network. Furthermore, the estimated Q-values by this network are used as the initial target Q-values of the critic network.
3. To better stabilize the joint actor-critic network training, we update the critic's mean squared loss target value every $\tau = 100$ iterations. Looking back at equation 4.16, we update the Q^{ret} in this equation every τ iterations. We also choose the estimated Q-values by the Q-val network as the initial target of the critic network.

Algorithm 7 presents the high-level psudocode of the actor-critic network training including the above three novel techniques. For more details on the network structure, please refer to chapter 5.

Algorithm 7 Training steps for the deep actor-critic network

```
Require: If available, the explicit behavior policy, $\pi_b$
Require: $\mathcal{D}_{train} = \{H_{L_1}, H_{L_2}, ..., H_{L_{n_{train}}} | H_{L_i} \sim \pi_b\}$
Require: The reward discount factor, $\gamma$
 1: Save the $\mathcal{D}_{train}$ episodes in an episode replay memory (section 5.3).
 2: Break the episodes into experience tuples and save in an experience replay memory (section 4.2.1).
 3: Build the behavior policy network with the same structure as the actor network.
 4: Train the behavior policy network in a multi-class classification setting considering $S_t$ and $A_t$ from the
    experience tuples as the input features and the labels, respectively.
 5: Copy weights of the behavior policy network into the actor network.
 6: Build the Q-val network with the same structure as the critic network.
 7: Train the Q-val network in a Dueling Double Deep Q-network setting using the experience replay memory
    (section 4.2).
 8: Copy weights of the Q-val network into the critic network.
 9: if explicit $\pi_b$ is not available then
10:     $\pi_b \leftarrow$ the trained behavior policy network
11: end if
12: Train the actor-critic network by the episode replay memory (section 4.3). Update the target of the
    critic's mean square loss (equation 4.16) every $\tau = 100$ iterations.
13: return The trained actor-critic network and the trained behavior policy network
```

6.3 Safety

This section and the next section focus on the trustability of the proposed algorithm. Trust is known as the greatest challenge of applying artificial intelligence to real-world problems such as asset maintenance management or medical treatment policy [Ins15]. We offer a methodology that tackles this issue from various aspects. As our target environments are highly risk-averse, we enforce multiple protection layers by integrating safe RL approaches together with off-policy evaluation. Consequently, the algorithm could be trustable by decision-makers in risk-averse domains.

Consider there is a history of agent-environment interaction episodes according to some behavior policy. An RL agent analyzes these retrospective episodes and suggests a new policy. This new policy is regarded as trustable if:

1. it does not demonstrate undesirable behavior to achieve a better policy, and

2. it is at least as "good" as the behavioral policy with high confidence.

We address the first condition in this section, and the second condition is the topic of the next section, section 6.4.

The first step is to define the notation of "undesirable behavior". One might claim that it is unnecessary to take into account the desirability of the agent's behavior if the reward function is designed accurately because undesirable behaviors can be avoided by assigning negative rewards to them. This statement is true; however, one should note that the reward design is a long-lasting challenge in the RL community. There is no scientific solution for it other than optimizing the reward function by trial and error [NBS10]. This results in a safety concern that even if the RL agent correctly optimizes the specified objective function (the expected return), the objective function's designer may not foresee undesirable ways that the objective function can be optimized until after the agent has produced undesirable behavior [Wie64, FF19]. Therefore, it is safer to explicitly exclude behaviors that are known to be risky for the policy learning search space.

Two types of undesirable behaviors are identified in this proposal. Type one is the set of behaviors that are known to be high-risk by common sense or by an expert, and type two is the set of behaviors that are not explored under the behavior policy, which means there is no information about the safety of those actions. Each type of undesirable behavior is associated with a set of constraints in the policy learning search space, as depicted in figure 6.1. The subset of the policy learning search space that leads to high-risk behaviors is named as the Hazard Zone, and the subset that leads to previously unexplored behaviors is named as the Unknown Zone.

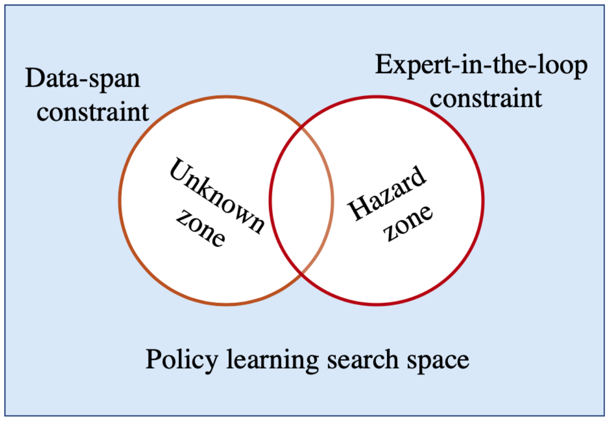


Figure 6.1: Schematic of the two sets of constraints that are associated with the two types of undesirable behaviors.

To formally impose the hazard zone's constraints, we identify a subset of the state space for each action in which performing the action in that part of the state space is endorsed by an expert as unsafe. We regard this type of constraints as the Expert-in-the-Loop policy search constraints and mathematically define it as

$$\pi_e\big(a_i|s \in S^{HZ}(a_i)\big) = 0; \quad \forall a_i \in A, \tag{6.1}$$

where $S^{HZ}(a_i)$ is the notation for the a_i's hazard zone. Similarly, the unknown zone constraints are imposed by identifying a subset of the state space for each action that the action has never been tried in that part of the state space. We name this type of constraints as the Data-Span policy search constraints.

$$\pi_e\Bigg(a_i|s \notin Supp\big(\pi_b(a_i)\big)\Bigg) = 0; \quad \forall a_i \in A. \tag{6.2}$$

Often, it might not be straightforward to find the support set for the various actions in the behavior policy, for example, when we do not have access to the explicit representation of the behavior policy. In those cases, we need to estimate the actions' support sets from a retrospective data set. Estimating the actions' support sets might be hard, particularly if the problem we are dealing with is in a continuous state space.

We suggest a similarity check between the experienced and the new state-action pairs, instead of the support set, for the problems where the explicit policy is not available. To perform the similarity check, we first divide all the experienced state-action pairs in the data set based on the actions. Then we make a case-base from the state vectors of each group. When searching for the best policy in state s, we measure the similarity of s to the states in each group separately. If the similarity is less than a threshold in any case-bases, we assign zero probability to the action of that case-base in state s.

We pick the similarity measure as the distance to the nearest neighbor in the related action's case-base. Accordingly, if $s_q^{a_i}$ is the nearest neighbor of s in ith action's case-base, the similarity measure of s to that case-base would be the Euclidean distance between s and $s_q^{a_i}$:

$$d^{a_i}\Big(s\Big) = \sqrt{\sum_{k=1}^{state\_size} (s_q^{a_i}[k] - s[k])^2}. \tag{6.3}$$

The Data-Span policy constraint in the problems where the explicit policy is not available is defined as:

$$\pi_e\Bigg(a_i|d^{a_i}\Big(s\Big) > \theta\Bigg) = 0; \;\; \forall a_i \in A. \tag{6.4}$$

The threshold θ is highly problem specific, and the user could assign it considering how the states numeric representations are chosen.

In addition to the two proposed safety constraints, Expert-in-the-Loop and Data-Span policy constraints, that are placed to avoid undesirable risky behaviors, our problem setting has additional inherent safety enforcement. Since the agent is learning the new policy from stored samples of expert experiences, the data has an intrinsic bias. The bias is passed to the agent during the training process and greatly improves the agent's safety.

6.4 Off-Policy Evaluation and Confidence Bounds

The second trustability condition is the new policy must be at least as "good" as the behavioral policy with high confidence. For this condition, we need to formalize the "good" notation. The return value $G(H)$ introduced in equation 2.4 is chosen to quantify "goodness" of an episode. However, the environment is stochastic, and a better value of return in one episode cannot prove the superior performance of a policy. Therefore, we use the notation of the expected return to quantify a policy's performance [GF15], as shown in equation 2.7. We can formally say an evaluation policy π_e is better than a behavior policy π_b if and only if the evaluation policy's expected return is greater than the behavior policy's expected return.

$$\pi_e > \pi_b \Leftrightarrow J(\pi_e) > J(\pi_b) \tag{6.5}$$

where $J(\pi_e) = \mathbb{E}[G(H_L)|H_L \sim \pi_e]$ and $J(\pi_b) = \mathbb{E}[G(H_L)|H_L \sim \pi_b]$.

In the absence of an accurate model of the system, we cannot formally derive the expected return; therefore, we use the empirical mean of the returns to estimate the expected return. Consider $\mathcal{D}$ is a set of episodes generated by the interaction of the behavioral policy with the system.

$$\mathcal{D} = \{H_{L_1}, H_{L_2}, ..., H_{L_{n_D}} | H_{L_i} \sim \pi_b\} \tag{6.6}$$

The estimated expected return of the behavior policy according to $\mathcal{D}$ is as follows

$$\hat{J}(\pi_b) = \frac{1}{n_{\mathcal{D}}} \sum_{i=1}^{n_D} G(H_{L_i} | H_{L_i} \sim \pi_b). \tag{6.7}$$

Noting that the evaluation policy cannot be deployed without a performance guarantee, it is not possible to possess any episode from the evaluation policy to estimate $\hat{J}(\pi_e)$, and we ought to use the off-policy evaluation methods.

In chapter 3, we reviewed the off-policy evaluation with confidence bounds methods under section 3.2. We discussed two lines of research, one based on value function approximation and another based on importance sampling (IS) and concentration inequalities (CIs). We saw that often the first approach leads to over conservative bounds for the evaluation policy's performance. Consequently, we chose the second approach, IS and CIs approach.

6.4.1 Estimating the Expected Return under the Evaluation Policy

One important consideration when applying the IS approaches to off-policy evaluation is that it is a valid estimation where $Pr(H_{L_i}|\pi_b) \neq 0$. Or if $Pr(H_{L_i}|\pi_b) = 0$ then $Pr(H_{L_i}|\pi_e) = 0$. This consideration does not impose any concern in the Trustable Deep RL algorithm because the Data-Span policy search constraint already enforces this consideration. The Data-Span policy search constraint states that if an action's probability is zero in a particular state under the behavior policy, the evaluation policy must assign zero probability to that action in that particular state. Consequently, if an episode's probability is zero under the behavior policy, it is zero under the evaluation policy, as well.

Four versions of the IS-based point estimators are developed in the literature for the off-policy expected return estimation, basic IS, PDIS, WIS, WPDIS. These estimators are extensively elaborated in section 3.2. Basic IS and PDIS are prone to high variance because of possible large values of importance weights. WIS and WPDIS estimators suffer less because in their formulation, the importance weights in the denominator counteract the numerator's large variance.

WIS and WPDIS are biased but consistent estimators. Hence, their estimates are biased if the number of samples is small, but they become less biased as the number of samples increases. Another drawback of WIS and WPDIS is that they are batch estimators, i.e., they cannot estimate the evaluation policy's expected return from one episode. This is a challenge because conventionally CIs input estimated expected return from single episodes.

Our test environment (and potentially other target applications of the proposed algorithm) is lengthy in horizon. The test environment could have up to 360 steps of agent-environment interaction. The lengthy interaction episodes cause drastically large importance weights, leading to very noisy estimations if using the IS and PDIS estimators. Consequently, we ought to choose WIS and WPDIS over IS and PDIS. Between WIS and WPDIS, we prefer WPDIS because it demonstrates less variance according to section 3.2. However, the challenge of utilizing batch estimators with CIs still remains. The pseudocode of the WPDIS estimator for the expected return under the evaluation policy is brought in algorithm 8. This algorithm is based on equation 3.24.

Algorithm 8 WPDIS estimator for π_e's expected return

```
Require: The behavior and the evaluation policies, $\pi_b$ and $\pi_e$
Require: Batch of $K$ episodes generated by $\pi_b$, $\mathcal{D} = \{H_{L_1}, H_{L_2}, ..., H_{L_K} | H_{L_i} \sim \pi_b\}$
Require: The reward discount factor, $\gamma$
 1: $\hat{J}^{WPDIS} = 0$
 2: for $t = 1 : max\{L_1, ..., L_K\}$ do
 3:     $reward\_sum = 0$, $ISW\_sum = 0$
 4:     for $i = 1 : K$ do
 5:         if $t < L_i$ then
 6:             $ISW = 1$
 7:             for $j = 0 : t - 1$ do
 8:                 $ISW = ISW \times \frac{\pi_e(a_j^i | s_j^i)}{\pi_b(a_j^i | s_j^i)}$
 9:             end for
10:             $reward\_sum = reward\_sum + R_t^i \times ISW$
11:             $ISW\_sum = ISW\_sum + ISW$
12:         end if
13:     end for
14:     $\hat{J}^{WPDIS} = \hat{J}^{WPDIS} + \gamma^{t-1} \times \frac{reward\_sum}{ISW\_sum}$
15: end for
16: return $\hat{J}^{WPDIS}$
```

6.4.2 Estimating Lower Confidence Bound for the Expected Return

CIs provide confidence bounds for the estimations of the evaluation policy's expected return when a group of point estimates is available. We reviewed five popular CIs for off-policy evaluation in section 3.2. First, we introduced CH inequality, and we observed that it is an apt option when the samples are independent (but not necessarily identically distributed), and each has individual upper and lower limits. Otherwise, it does not provide very tight confidence bounds. In our problem setting, we are dealing with i.i.d. samples. Hence, although CH could provide reasonable bounds, the other CIs could provide tighter bounds.

CH uses only the sample mean to find confidence bounds. MPeB is another CI introduced in section 3.2 that uses the sample mean and the sample variance, thus providing better bounds. MPeB requires independent samples with the same variation range, which is met by our i.i.d. samples. AM is another introduced inequality that improves upon MPeB by considering even more statistics from the empirical estimated return's distribution. It has a strict condition that the samples must be i.i.d. ; however, it requires only a lower range of random variables. The formulation of AM, brought in 3.30, heavily relies on the largest observed value. Therefore, it could provide tight confidence bound when the random variable has a heavy upper tail, but the largest observation is relatively small. This criterion is very application-specific. For example, the estimated return of the evaluation policy in our test environment is not a heavy-tailed distribution so that MPeB might calculate a better lower bound on the expected return.

The Bootstrap confidence bound is another approach to computing the confidence bounds. The advantage of this approach is it offers tight confidence bounds if the underlying assumption about the estimated return's distribution is valid. Among bootstrap confidence bound methods, we introduced the Student's t-test and BCa. BCa is more advanced and reliable that is often used in medical research.

The above summary of the various CIs' merits infers that there is no one-size-fits-all CI choice. The best CI for each application might vary depending on the reward and the environment properties. We aim at providing a generic trustable algorithm; hence, we implement the three best CIs and take the maximum from their lower bounds. We are interested only in the lower bound of the estimated return because if the actual expected return is bigger than the estimated value, it won't raise any concern; therefore, the upper bound is not needed.

The i.i.d. assumption is valid in our problem setting because all the samples are derived from the same behavior policy. Consequently, we won't implement CH and Student's t-test because with i.i.d. assumption, MPeB provides tighter bounds than CH, and BCa provides tighter bounds than Student's t-test. However, we keep AM because it works better than MPeB if the estimated return distribution has a heavy upper tail (this is common in RL problems), or the user cannot provide reasonable bounds for the return under the evaluation policy. In conclusion, we select MPeB, AM, and BCa for deployment in the proposed Trustable Deep RL agent.

Section 6.4.1 concluded that WPDIS is the best estimator for the evaluation policies' expected return. We also mentioned that WPDIS is a batch estimator, and it is a challenge to use it with CIs because conventionally CIs are applied to single samples of episodes' return. We tackle this challenge by assuming the random variable X in equation 3.25 is an estimated expected return from a K-sized batch of episodes rather than a single episode. Consequently, X is $\hat{J}^{WPDIS}(\pi_e|\{H_i\}_{i=1}^{K}, \pi_b)$, and equation 3.25 could be expressed as follows for the evaluation policies expected return:

$$Pr\Bigg(\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}\hat{J}^{WPDIS}(\pi_e|D_i, \pi_b)\right] \geq f(\hat{J}^{WPDIS}(\pi_e|D_1, \pi_b), ..., \hat{J}^{WPDIS}(\pi_e|D_n, \pi_b), \delta)\Bigg) \geq 1 - \delta, \tag{6.8}$$

where D_is are all in the same size, K. Since WPDIS is a consistent estimator, $\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}\hat{J}_i^{WPDIS}\right]$ would converge to the true expected return, $J(\pi_e)$, when the batch size grows to infinity, $K \to \infty$. This means that the CIs could provide an approximate $1-\delta$confidence lower bound.

A subtle but crucial point about this new derivation of the CIs for $J(\pi_e)$'s lower confidence bound is the bound comes from batch estimators of the expected return. The batch estimators are less noisy, resulting in tighter confidence bounds than single episode estimators. When comparing π_e's performance with another policy, we need to account for this effect of the batch estimators and express the other policy's performance in the same way as π_e's performance. The performance comparison is a critical step in the proposed Trustable Deep RL algorithm. In order to have a fair comparison, we derive a lower confidence bound for the behavior policy from batch estimators in the same way as the evaluation policy. Algorithm 9 handles this task and computes four values that could be used by the user for comparing the evaluation and the behavior policies' performances. These values are:

- π_e's expected return according to algorithm 8,
- π_b's expected return according to equation 4.3,
- δ-error confidence lower bound of the π_e's expected return,
- δ-error confidence lower bound of the π_b's expected return.

6.5 Trustable Deep RL Algorithm

This section combines deep RL, data efficiency techniques, safety measures, and the off-policy evaluation into the united Trustable Deep RL algorithm. The integration is carried out in two stages, policy improvement and policy evaluation. The new improved policy comes from the deep actor-critic network, then it passes through the safety measures, and at the end, will be approved for deployment if the policy evaluation results are satisfactory. In the deployment phase, the agent behaves according to the improved and trustable policy in the previously explored areas of the state-action space. In unexplored areas, it hands the decision making to the expert (or the user). However, if requested by the user, it could guide future data gathering to help the decision-makers explore more informative and promising parts of the state-action space.

Prior to getting into the policy improvement and policy evaluation details, we need to clarify the train and test sets preparation steps. Assuming that $\mathcal{D}$ is a set of episodes

Algorithm 9 Lower confidence bound estimation for π_e and π_b's expected returns

```
Require: The behavior and the evaluation policies, $\pi_b$ and $\pi_e$
Require: $\mathcal{D}_{test} = \{H_{L_1}, H_{L_2}, ..., H_{L_{n_{D_{test}}}} | H_{L_i} \sim \pi_b\}$
Require: The error rate, $\delta$
Require: The reward discount factor, $\gamma$
Require: The batch size of the WPDIS estimator, $K$
Require: The number of BCa's bootstrap resamplings, $B$
Require: The best approximate for lower and upper range of $\pi_e$'s expected return, $a_e$ and $b_e$; and $\pi_b$'s
    expected return, $a_b$ and $b_b$
 1: $n = \lfloor \frac{n_{D_{test}}}{K} \rfloor$
 2: for $w = 1 : n$ do
 3:     Take $K$ random episodes without replacement from $\mathcal{D}_{test}$, $\{H_{L_1}, ..., H_{L_K}\}$
 4:     Compute $\hat{J}^e_w$ by algorithm 8 $\big(\gamma, \pi_b, \pi_e, \{H_{L_1}, ..., H_{L_K}\}\big)$
 5:     Compute $\hat{J}^b_w$ by $\frac{1}{K}\sum_{i=1}^{K} G(H_{L_i})$
 6: end for
 7: $\pi_e\_MPeB\_bound = \frac{1}{n}\sum_{w=1}^{n} \hat{J}^e_w - \frac{7(b_e - a_e)ln(\frac{2}{\delta})}{3(n-1)} - \sqrt{\frac{2ln(\frac{2}{\delta})}{n} \frac{1}{n(n-1)} \sum_{i,j=1}^{n} \frac{(\hat{J}^e_i - \hat{J}^e_j)^2}{2}}$
 8: $\pi_b\_MPeB\_bound = \frac{1}{n}\sum_{w=1}^{n} \hat{J}^b_w - \frac{7(b_e - a_e)ln(\frac{2}{\delta})}{3(n-1)} - \sqrt{\frac{2ln(\frac{2}{\delta})}{n} \frac{1}{n(n-1)} \sum_{i,j=1}^{n} \frac{(\hat{J}^b_i - \hat{J}^b_j)^2}{2}}$
 9: Compute $\pi_e\_BCa\_bound$ by algorithm 6 $\big(\delta, B, (\hat{J}^e_1, ..., \hat{J}^e_n)\big)$
10: Compute $\pi_b\_BCa\_bound$ by algorithm 6 $\big(\delta, B, (\hat{J}^b_1, ..., \hat{J}^b_n)\big)$
11: Sort $\hat{J}^e_w$s and $\hat{J}^b_w$s, and put them in the vectors $\boldsymbol{Z^e} = (Z^e_1, ..., Z^e_n)$ and $\boldsymbol{Z^b} = (Z^b_1, ..., Z^b_n)$
12: Add $Z^e_0 = a_e$ and $Z^b_0 = a_b$ to the sorted vectors $\boldsymbol{Z^e}$ and $\boldsymbol{Z^b}$
13: $\pi_e\_AM\_bound = Z^e_n - \sum_{i=0}^{n-1}(Z^e_{i+1} - Z^e_i) \min\left\{1, \frac{i}{n} + \sqrt{\frac{ln(\frac{2}{\delta})}{2n}}\right\}$
14: $\pi_b\_AM\_bound = Z^b_n - \sum_{i=0}^{n-1}(Z^b_{i+1} - Z^b_i) \min\left\{1, \frac{i}{n} + \sqrt{\frac{ln(\frac{2}{\delta})}{2n}}\right\}$
15: $\pi_e\_lower\_conf\_bound = max\{\pi_e\_MPeB\_bound, \pi_e\_AM\_bound, \pi_e\_BCa\_bound\}$
16: $\pi_b\_lower\_conf\_bound = max\{\pi_b\_MPeB\_bound, \pi_b\_AM\_bound, \pi_b\_BCa\_bound\}$
17: Compute $\hat{J}^e$ by algorithm 8 $\big(\gamma, \pi_b, \pi_e, D_{test}\big)$
18: Compute $\hat{J}^b$ by $\frac{1}{K}\sum_{i=1}^{n_{D_{test}}} G(H_{L_i})$
19: return $\hat{J}^e$, $\hat{J}^b$, $\pi_e\_lower\_conf\_bound$, and $\pi_b\_lower\_conf\_bound$
```

generated according to the behavior policy, equation 6.6, the episodes are first partitioned into the test and the train sets with 30% to 70% ratio. The train set is meant only for training the deep actor-critic network, and the test set is for off-policy evaluation of the new policy. The test set is kept as it is while the train set is stored into two memories, the experience replay memory and the episode replay memory. The experience relay memory is needed to train the Q-val network and initialize the critic network's weights; therefore, the episodes are broken into experience tuples similar to $< S_t, A_t, R_{t+1}, S_{t+1} >$ and then saved in an experience replay memory. The episode replay memory is used to train the deep actor-critic network, and the episodes are saved in an episode replay memory according to section 5.3.

In the policy improvement stage, a deep actor-critic network is trained to estimate the best policy according to the train set. Afterward, states of the test episodes are fed into the trained network sequentially, and the output policy for each state passes through the following steps for safety and trustability adjustments:

1. Compute the policy from the trained actor-critic network,

$$\pi_e(a_i|s) = \pi(a_i|s, w_{actor-critic}); \;\; \forall a_i \in A \tag{6.9}$$

2. Apply the expert-in-the-loop policy constraints.

$$if\ s \in S^{HZ}(a_i) \rightarrow \pi_e(a_i|s) = 0; \;\; \forall a_i \in A \tag{6.10}$$

3. Apply the data-span policy constraints.

$$\begin{cases} \pi_e\big(a_i|s \notin Supp\big(\pi_b(a_i)\big)\big) = 0; \;\; \forall a_i \in A & \text{if } \pi_b \text{ is explicit} \\ \pi_e\big(a_i|d^{a_i}(s) > \theta\big) = 0; \;\; \forall a_i \in A & \text{if } \pi_b \text{ is not explicit} \end{cases} \tag{6.11}$$

4. Adjust the policy's probabilities to the constraints.

$$\pi_e(a_i|s) \leftarrow \frac{\pi_e(a_i|s)}{\sum_{i=1}^{|A|} \pi_e(a_i|s)}; \;\; \forall a_i \in A \tag{6.12}$$

In the evaluation stage, we provide π_b, final π_e, and the test episodes to algorithm 9 to estimate π_e's expected return and provide statistics on both policies' performance. This algorithm first makes multiple estimations of π_e's and π_b's expected returns, and then computes δ-error confidence lower bounds for the estimations. Therefore, the user could ensure that the Trustable Deep RL algorithm's new policy would, on average, perform better than the lower confidence bound with probability $1 - \delta$. Finally, the evaluation policy will be approved for deployment in the system if:

1. π_e's estimated expected return is greater than or equal to the π_b's estimated expected return, and

2. δ-error confidence lower bound on π_e's expected return is greater than or equal to δ-error confidence lower bound on π_b's expected return .

The pseudocode of training and evaluation phases of the Trustable Deep RL agent is brought in Algorithm 10.

When the Trustable Deep RL's performance is approved, it could be deployed according to algorithm 11. Due to the unknown zone and the hazard zone constraints, section 6.3, it is possible that for a new and never explored state, the probabilities of all the actions are zero. In these situations, the Trustable Deep RL agent would raise the "unprecedented state" error and hand the decision-making to the expert or the user. However, if the user is interested in knowing which parts of the unexplored state-action space are more promising, the algorithm could advise where to explore and gather new data. The advice is made based on recommendations from the actor-critic deep network, i.e., we input the new state to the network and then pass the output policy through the expert-in-the-loop constraints. Finally, we adjust the policy's probabilities and recommend it to the user.

Algorithm 10 Training and evaluation of the Trustable Deep RL agent

```
Require: Set of episodes generated by $\pi_b$, $\mathcal{D} = \{H_{L_1}, H_{L_2}, ..., H_{L_N} | H_{L_i} \sim \pi_b\}$
Require: The error rate, $\delta$
Require: The reward discount factor, $\gamma$
Require: The batch size of the WPDIS estimator, $K$
Require: The number of BCa's bootstrap resamplings, $B$
Require: The best approximate for lower and upper ranges of $\pi_e$ and $\pi_b$ expected returns, $a_e$, $b_e$, $a_b$, $b_b$
Require: The threshold for the states similarity measure, $\theta$
Require: The hazard zone for each action $S^{HZ}(a_i),\ \forall a_i \in A$
Require: If available, the explicit behavior policy, $\pi_b$
 1: Partition $\mathcal{D}$ into a train and a test set with ratio 70% to 30%.
 2: Get the trained actor-critic network and the trained behavior policy network by calling algorithm 7; input
    $\mathcal{D}_{train}$ and $\pi_b$ (if available).
 3: if the explicit $\pi_b$ is not available then
 4:     Estimate $\pi_b$ by the trained behavior policy network, $\pi_b(a_i|s) = \pi(a_i|s, w_{behavior-network})$; $\forall a_i \in A$.
 5: end if
 6: Compute the evaluation policy from the trained actor-critic network, $\pi_e(a_i|s) = \pi(a_i|s, w_{actor-critic})$;
    $\forall a_i \in A$.
    #Apply expert-in-the-loop policy constraints
 7: $if\ s \in S^{HZ}(a_i) \rightarrow \pi_e(a_i|s) = 0;\ \forall a_i \in A$
    #Apply data-span policy constraints
 8: if the explicit $\pi_b$ is not available then
 9:     $if\ d^{a_i}(s) > \theta \rightarrow \pi_e(a_i|s) = 0;\ \forall a_i \in A$
10: else
11:     $if\ s \notin Supp(\pi_b(a_i)) \rightarrow \pi_e(a_i|s) = 0;\ \forall a_i \in A$
12: end if
    #Adjust the policy's probabilities to the constraints.
13: $\pi_e(a_i|s) \leftarrow \frac{\pi_e(a_i|s)}{\sum_{i=1}^{|A|} \pi_e(a_i|s)};\ \forall a_i \in A$
14: Compute $\hat{J}^e$, $\hat{J}^b$, $\pi_e\_lower\_conf\_bound$, and $\pi_b\_lower\_conf\_bound$ by calling algorithm 9; input $\mathcal{D}_{test}$,
    $\pi_e$, $\pi_b$, $\delta$, $\gamma$, $K$, $B$, $a_e$, $b_e$, $a_b$, $b_b$.
15: if $(\hat{J}^e \geq \hat{J}^b)$ and $(\pi_e\_lower\_conf\_bound \geq \pi_b\_lower\_conf\_bound)$ then
16:     return $\pi_e$
17: else
18:     return "no better policy found"
19: end if
```

Algorithm 11 Deployment of the Trustable Deep RL Agent

```
Require: The trained actor-critic network
Require: The hazard zone for each action $S^{HZ}(a_j), \ \forall a_i \in A$
Require: If available, the explicit behavior policy, $\pi_b$
 1: Read the encountered state, $s$.
 2: Compute the Trustable deep RL agent's policy from the trained actor-critic network, $\pi(a_i|s, w_{actor-critic})$;
    $\forall a_i \in A$.
    #Apply expert-in-the-loop policy constraints
 3: $if \ s \in S^{HZ}(a_i) \rightarrow \pi(a_i|s) = 0; \ \forall a_i \in A$
 4: Save $\pi$ as the future data gathering guidance policy $\pi_{guide}$
    #Apply data-span policy constraints
 5: if the explicit $\pi_b$ is not available then
 6:     $if \ d^{a_i}(s) > \theta \rightarrow \pi(a_i|s) = 0; \ \forall a_i \in A$
 7: else
 8:     $if \ s \notin Supp(\pi_b(a_i)) \rightarrow \pi(a_i|s) = 0; \ \forall a_i \in A$
 9: end if
10: if $\pi(a_i|s) = 0; \forall a_i \in A$ then
11:     $\pi_{guide}(a_i|s) \leftarrow \frac{\pi_{guide}(a_i|s)}{\sum_{i=1}^{|A|} \pi_{guide}(a_i|s)}; \ \forall a_i \in A$
12:     return "Unprecedented-state! the recommended policy is $\pi_{guide}$"
13: else
14:     $\pi(a_i|s) \leftarrow \frac{\pi(a_i|s)}{\sum_{i=1}^{|A|} \pi(a_i|s)}; \ \forall a_i \in A$
15:     return $\pi$
16: end if
```

CHAPTER 7

The Test Environment

This chapter presents the test environment used for evaluating the proposed Trustable Deep RL methodology's performance. The environment is an underground dry gas pipeline subject to internal corrosion. The sequential decision-making problem we aim at solving by the proposed methodology is to find the best maintenance strategy for mitigating the corrosion-related costs in the pipeline's life cycle while securing the pipeline reliability.

In this chapter, we first introduce the environment, and then, we elaborate upon how to frame the corrosion-related maintenance management problem into an MDP structure. The corrosion model presented in the following is developed at the Garrick Institute of Risk Sciences. This chapter's contribution is building a decision-making test bench out of the corrosion model that is capable of:

1. Interacting with a decision-maker, receiving action orders from the decision-maker, and reporting the corrosion status to that.
2. Adjusting its internal corrosion model to multiple concurrent maintenance actions.
3. Approximating the cost associated to the maintenance actions and the pipeline health state.

7.1 Corrosion Management Test Bench

The natural gas pipeline network transports natural gas from the wellhead to the customers throughout the entire USA. Corrosion is a very severe and costly issue in natural gas pipeline. It is specified as the gradual reduction of the pipeline wall thickness that might lead to substantial environmental and economic consequences as a result of catastrophic events, including leak and burst. Therefore, pipeline corrosion management has become highly essential [YKTA17].

In the absence of real data from the gas pipeline, we developed and used a simulated test bench to mimic the real system (the pipeline corrosion) behavior. Prior to getting into the test bench details, we state the assumptions made in the test bench implementation.

1. The system is one section of the pipeline, which is presumed to be new at time t = 0, and a corrosion defect gradually develops after that depending on the time-varying operating conditions.

2. It is assumed that the pipeline section suffers from only one corrosion defect at each time.

3. The pipeline section is assumed to corrode by two types of corrosion, namely pitting and uniform corrosions. A linear combination of the two is considered to aggregate the two corrosion processes, as suggested by Wu and Castro [WC20]. When the combined degradation reaches threshold values, the pipeline system fails due to a leak or burst.

4. Two types of maintenance are considered, namely, preventive maintenance and a replacement of the system. The preventive maintenance includes batch corrosion inhibitor, internal coating, and pigging. The replacement renews the system completely.

5. Determining in line inspection interval is always a challenging problem for pipeline operators, which depends on the conditions of the pipeline systems [G$^+$19]. We simplify the problem by assuming that the RL maintenance scheduler receives signals from the pipeline's health state through monthly inspections. It can then decide whether to do nothing, do one of the preventive maintenance, or do a replacement. The scheduler's maintenance order is disclosed to the system at the beginning of each month.

6. Inspection and maintenance times are neglected.

The following subsections describe the implementation details of the proposed test bench. First, an overview of various components of the test bench and their interactions is presented, then the details of each particular component are further elaborated.

7.1.1 Components of the Test Bench

The test bench is composed of three components. First, a set of environmental and operational parameters that mimic the daily variation of a real pipeline's environmental and operational parameters are simulated. Afterward, the parameters are fed into a pipeline model along with the order of the month's maintenance action. Then, the pipeline corrosion model adjusts itself to the received maintenance order and then simulates the corrosion progress over a month period. At the end of the month, the total corrosion depth and length are reported to the pipeline reliability model that simulates the failure events, namely leak and burst. Eventually, a cost approximator estimates the month's expenditure according to the exitance of any failure as well as any maintenance actions.

Figure 7.1 shows an overview of the developed test bench. It should be emphasized that the maintenance scheduler is not part of the test bench. It is a separate entity that interacts with the test bench by reading in the end of the month corrosion condition and the cost and then instructing a maintenance order for the next month. The choice of the maintenance actions is limited to the set of actions that the corrosion model could adjust to them.

7.1.2 The Environmental and Operational Parameters Simulator

The complex operating condition of the pipeline system is simulated by modeling a variety of environmental parameters. The most common environmental parameters related to the pipeline corrosion include temperature, total pressure, partial pressure of CO2, partial pressure of H2S, flow velocity, pH value, chloride ion concentration, sulfate ion concentration, and the presence of solids. The environmental parameters are stochastic in nature; therefore, the Poisson Square Wave Process (PSWP) [ZZ13, WM19] is used to model them and generate different time-varying environmental parameters. We refer the interested reader to [MWM]

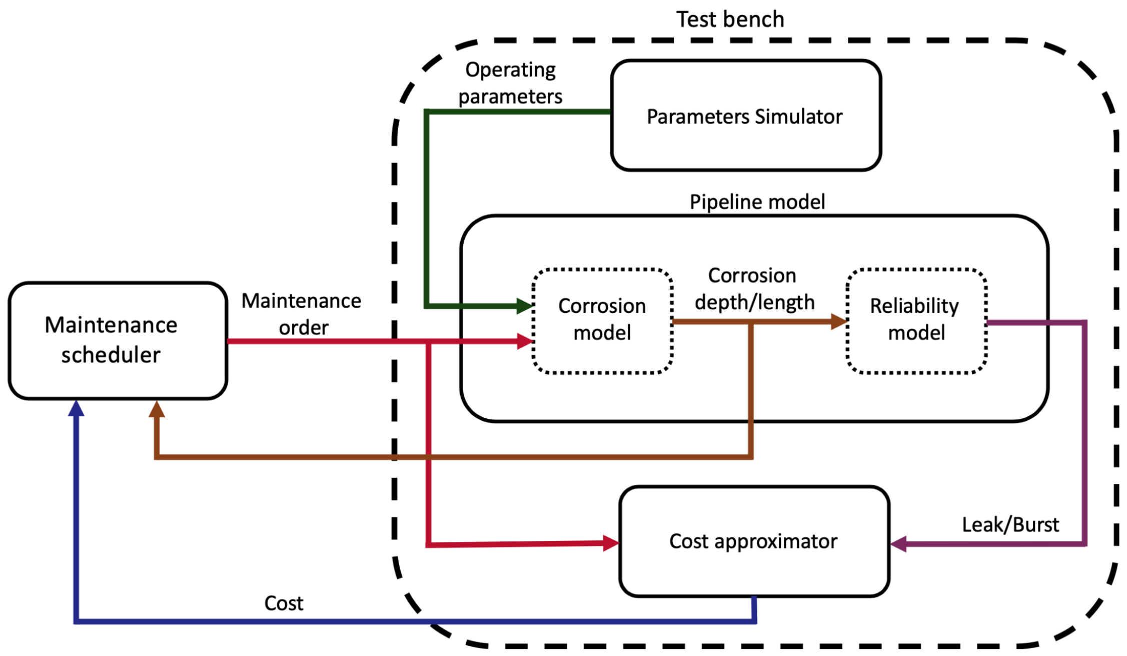


Figure 7.1: The block diagram of the test bench for interaction evaluation between a maintenance decision-maker and a corrosive pipeline.

for further details about parameters simulation with PSWP.

7.1.3 The Pipeline Model

The simulated system is a dry natural gas pipeline, which is 1.6 km in length, 509 mm in outer diameter, and 492 mm in inner diameter. The pipeline is considered for transporting natural gas (i.e., CH4) with a small amount of corrosive gases (i.e., CO2 and H2S) and elements (i.e., Cl-). The degradation is the result of internal corrosion, including uniform and pitting corrosions. Degradation is a function of the operating conditions, which are considered and simulated as time-varying parameters. When the linear combination of the two degradation processes reaches the threshold value, the pipeline fails either due to leak or burst. In summary, the pipeline model takes the environmental parameters as inputs and outputs two Boolean signals representing whether the pipeline fails by either leaks or bursts. The pipe model consists of a corrosion model and a reliability model, each of which will be

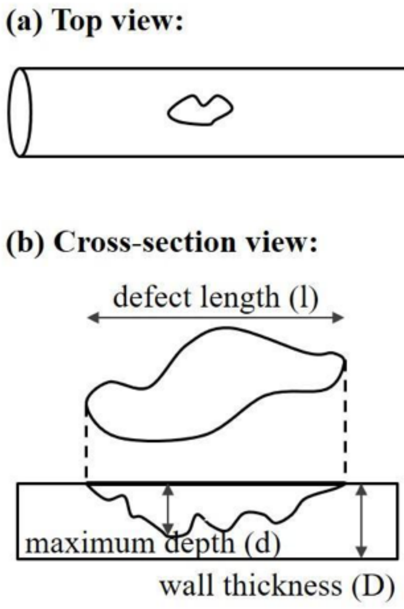


Figure 7.2: A rectangular-like shaped corrosion defect in the pipeline in: (a) top view and (b) cross-section view.

described in the following subsections.

7.1.3.1 Corrosion Model

The pipeline system degrades as a result of internal corrosion, including uniform and pitting corrosions. The corrosion defect is assumed to have a rectangular-like shape, as shown in figure 7.2, and can be described by corrosion depth and corrosion length. To simulate the overall degradation, a linear combination of the uniform and pitting corrosion is adopted as expressed in the following equations [WC20],

$$d_C(t) = d_{UC}(t) + d_{PC}(t), \tag{7.1}$$

$$l_C(t) = l_{UC}(t) + l_{PC}(t), \tag{7.2}$$

where d_C and l_C are the total corrosion depth and length, d_{UC} and l_{UC} are the uniform corrosion depth and length, d_{PC} and l_{PC} are the pitting corrosion depth and length. The depth of each type of the corrosions is calculated by accumulating the related corrosion rate.

For example, assuming $CR_{UC}(i)$ is the uniform corrosion rate in the ith day, the corrosion depth in day t from the uniform corrosion is as follows:

$$d_{UC}(t) = \sum_{i=1}^{t} CR_{UC}(i). \tag{7.3}$$

The uniform corrosion rate, CR_{UC}, is simulated by the two-stage model proposed in [WM19], and the pitting corrosion rate, CR_{PC}, is simulated by the model presented in [PDR10]. These models take various environmental parameters and predict the corrosion exacerbation in one day.

Unlike modeling of the corrosion depth that has been studied for a long time, there are no widely accepted models to calculate corrosion length [MPMY+14]. Therefore, a linear growth model [Zho10] with an expert suggested corrosion length growth rate (LGR) is assumed. The same LGR is considered for uniform and pitting corrosion; therefore, the following represents both the uniform corrosion length, l_{UC}, and the pitting corrosion length, l_{PC}:

$$l_{UC}(t) = l_{PC}(t) = LGR \times t. \tag{7.4}$$

The simulated overall corrosion depth $d_C(t)$ and length $l_C(t)$ are inputs to the reliability model, which will be introduced in the following subsection.

7.1.3.2 Reliability Model

A gas pipeline may fail by leak or burst, given the presence of corrosion defects. Our simulator considers these two failure modes, and the system reliability is modeled by the load and strength interference technique by defining their limit state functions at a given corrosion defect. The limit state function of the leak can be written as $g_1 = \lambda - d_{max}$ where λ is the corrosion allowance (i.e., 0.8×wall thickness) and d_{max} is the maximum defect depth. This limit state function shows that the leak happens when the maximum defect depth is larger than the corrosion allowance ($g_1 \leq 0$).

On the other hand, the limit state function of the burst g_2 can be written as $g_2 = P_b - P_{op}$ where P_b is burst pressure of the corroded pipeline, and P_{op} is the operating pressure. Burst occurs when the operating pressure is larger than the burst pressure ($g_2 \leq 0$). The burst pressure (or remaining strength) of the pipeline decrease due to corrosion damage, and therefore, the pipeline becomes weaker with increasing time. The burst pressure can be calculated via a number of developed models, all of which are related to the geometry of the corrosion defect and the material property of the pipeline. In this simulator, we select ASME B31G standard [oMECCfPP91] to do the calculation. Once the probabilities of failure for the failure modes are calculated, Binomial distributions are chosen to determine whether the scenario ends with a failure event or not. Specifically, we derive two samples from two Binomial distributions with the leak and burst probabilities. If either failure happens, the pipeline is regarded as failed.

7.1.3.3 Adjustment of the Corrosion Model to Maintenance Actions

Two types of maintenance are considered, namely preventive maintenance and replacement. Preventive maintenance includes batch corrosion inhibitor, internal coating, and pigging, all of which are common ways of mitigating the internal corrosion in dry natural gas pipelines [oPP18]. Although the internal coating is usually applied during the manufacturing process, it is not uncommon for pipeline operators to do coating repair if corrosion damages the coating [Sin17, PJP00]. Therefore, for comparison purposes, the internal coating is considered as one maintenance action here. Do nothing is also an option to provide the agent with the flexibility of applying no maintenance action. The influence of these actions on the corrosion are described as follows:

1. Do nothing: The corrosion continues with no mitigation.

2. Batch corrosion inhibitor: It provides a protective barrier on the inside wall of the pipeline and reduces the corrosion rate on the basis of inhibitor efficiency.

3. Internal coating: It isolates the pipeline from its environment to prevent water from reaching the pipeline surface; therefore, no corrosion will happen during the coating lifetime.

4. Pigging: It can remove liquids, solids, and debris; and no corrosion will happen until the water accumulates again.

5. Replacement: The damaged section is removed and then replaced with a new one.

Failure intensity is a common metric used to quantify the effect of repair models in asset management problems. For example, Doyen and Gaudoin [DG04] proposed reduction of the intensity model to express the repair effect on the system failure. Inspired by this paper, we propose a discount factor (df) ranging between zero and one to account for the influences of maintenance actions on the pipeline corrosion. The corrosion rate is multiplied by the discount factor to model the updated corrosion rate. Each maintenance action that directly influences the corrosion rate has its lifetime beyond which the discount factor becomes one. The discount factors and the lifetime with respect to different maintenance actions are listed in table 7.1.

Table 7.1: Discount factors and lifetime of different maintenance actions.

Actions	**Discount factors**	**Lifetime**
Batch corrosion inhibition	$df = 1 - 0.95e^{-0.023\times t}$	1 month
Internal coating	$df_{coating} = 0$	5 years
Pigging	$df_{pigging} = 0$	2 weeks

The variable t in the batch corrosion inhibitor's df represents time from the inhibitor appliance. According to table 7.1, the batch corrosion inhibitor's df is represented with an exponentially increasing function, i.e., immediately following a batch corrosion inhibitor appliance, the corrosion rate significantly lessens, and afterward gradually increases to its original value by an exponential function [YSKS15]. The coating effect and the pigging effect are to set the corrosion rate to zero but with different time scales. The coating effect is more permanent and remains for a more extended period of time; therefore, the corrosion rate is set to zero for five years. However, following a pigging action, the corrosion rate is set to zero only for two weeks.

7.1.4 Cost Approximator

Defining the costs is a critical task in any decision-making problem because the reward/cost is the only mean of the decision-maker to know the aptness of its actions. The goal of the maintenance decision-maker in the corrosion maintenance scheduling problem is to:

1. avoid catastrophic failures, namely leakage and burst,
2. extend the life of the asset,
3. reduce the maintenance costs.

To include these three goals in the reward function, we define three types of rewards as follows. Here we use cost to address the negative reward, and the total reward after each month is the algebraic summation of these three values:

1. Cost of failure: the cost that is associated with the situations where the system has failed. This cost is non-zero only when the pipeline enters a failure state. Upon the pipeline's arrival in a failed state, a high cost is emited to the decision-maker to penalize its decision-making policy. This cost should be considerably higher than the other types of costs because of the massive hazard the failures impose on people's lives and the pipeline's surrounding environment. Besides, failures cause an interruption in the service that might have significant economic consequences.
2. Life extension reward: the reward of delaying the failure. At the end of each month that the pipeline is not in one of the terminal states, namely failure and replacement, the agent receives a positive reward for extending the life of the pipeline for that month. This value is small in comparison to other types of cost.
3. Cost of maintenance: the cost that is associated with each maintenance action. The cost of the month's maintenance action is emitted to the agent at the end of each month, along with the failure delaying reward and the failure cost.

One of the challenges in designing the test bench is assigning values to the reward function since the cost-sensitive decision-making mechanisms highly rely on the cost values. The approach taken in this research is to assign values to the maintenance costs from general information appertaining to the gas pipeline maintenance and then specify the remaining reward values with respect to the maintenance costs. It should be noted that the values assigned here could be updated for different application scenarios. In addition, at this stage, the problem is simplified by considering only a segment of the pipeline, which is 1-mile long. By doing that, we can ignore the spatial effect on our studied system.

Table 7.2 lists the values of the three types of reward functions considered in this study. Regarding the maintenance cost, first of all, "Do nothing" costs \$0 in general if the cost of the inspection is neglected for all the other maintenance actions. Second, the "Pigging" operation starts from inlet to outlet. Therefore, the cost is determined based on the distance between the launcher and the receiver. Based on the data from pipeline operators, the cost is roughly \$35000/mile [Mit13]. Third, "Batch-treatment of corrosion inhibitor" is to pump the corrosion inhibitors from the inlet, and the volume of the pipeline determines the cost. Here it is assumed that one-third of the tubing volume is needed for the inhibition efficiency to be over 90%, and the cost is estimated to be \$130000/mile [Gar94]. In addition, "Internal coating" is a costly option as it requires the pipeline excavation. The cost is approximately \$800000/mile, according to [Lar20]. Finally, the cost of a corroded pipeline replacement varies based on the pipeline size, which is hard to estimate, but in general, it is more expensive than any rehabilitation actions. Here, we assign it to be \$1600000/mile, two times the coating cost.

Two types of failure costs are considered in this research. The leak cost is assumed to be three times the replacement cost because a leaked pipeline needs to be replaced, but the replacement follows with an unplanned interruption of the service. The burst cost also includes the replacement cost, but along with that, it has a significantly larger cost reflecting the highly catastrophic hazards it might cause. The life extension reward per month is set to \$50000 based on the sensitivity analysis results performed on the value of the life extension reward.

Table 7.2: Values of reward functions for different types of costs in the unit of $10,000.

Type	**Cases**	**Value($10,000)**
Maintenance cost	Do nothing	0
	Pigging	-3.5
	Inhibitor	-13
	Coating	-80
	Replacement	-160
Failure cost	Leakage	-480
	Burst	-1760
Life extension reward	Reward per month	+5

7.2 Framing the Test Bench in an MDP Format

The first step to frame the pipeline corrosion maintenance as a sequential decision-making problem is to define it in an MDP format. As stated before, MDP is the mathematical formulation of RL and consists of a state space, an action space, a reward function, and a state transition function. The transition function is not required here because the techniques employed in this research are data-driven (model-free) RL that does not need any knowledge about the pipeline's health state transition. In the following, we define the state space, the action space, and the reward function for the developed test bench.

7.2.1 State Space

The state in an MDP must be defined in a way that they have the Markov property. In the case of an MDP, the Markov property requires that the future state depends only on the current state and the current action. In other words, the state is a concise summary of the environment's history that includes all the necessary information to predict the next state given the action. Because this study's scope is the pipeline's corrosion, the state definition should include all the essential information to predict the next corrosion status given the action. Therefore, the state definition consists of the depth, length, and rate of the corrosion. Instead of directly taking the value of the depth and length, we considered the max-normalized representation of them to remove the agent dependency on the pipeline's parameters. Equations 7.5 and 7.6 define the corrosion depth and length as utilized in the pipeline's health state definition where the maximum corrosion depth is the wall thickness, and the maximum corrosion length is estimated by running the model for 40 years without maintenance; and equation 7.7 defines the corrosion rate.

$$NCD = \frac{corrosion\ depth}{max\ corrosion\ depth}\ , \qquad NCD \in S_{NCD} = [0, 1], \tag{7.5}$$

$$NCL = \frac{corrosion\ length}{max\ corrosion\ length}\ , \qquad NCL \in S_{NCL} = [0, 1], \tag{7.6}$$

$$CR = NCD(t) - NCD(t-1)\ , \qquad CR \in S_{CR} = [0, 1], \tag{7.7}$$

In the above equations, NCD is the normalized corrosion depth, NCL is the normalized corrosion length, and CR is the corrosion rate. The whole state space is defined as

$$s = (NCD, NCL, CR)\ , \qquad s \in S = S_{NCD} \times S_{NCL} \times S_{CR}. \tag{7.8}$$

7.2.2 Action Space

The action space is defined by available actions that the maintenance scheduler can take in response to the system's state. As presented in section 7.1.3.3, the test bench provides the scheduler with five actions. Therefore, a discrete action space of size 5 is considered for the agent as follows,

$$\text{A} = \{\text{do nothing, pigging, batch corrosion inhibitor, internal coating, replacement}\}. \tag{7.9}$$

7.2.3 Reward Function

The reward is a feedback signal from the environment to the agent that reflects the aptness of the RL agent's actions from the environment's prospect. In real-world RL problems, the reward function is usually a handcrafted variable by an expert derived from the environment's conditions. Assigning the reward function is a longtime challenge in RL [NR+00]. In this research, the test bench provides the agent with approximated cost values on a monthly basis. Therefore, the agent can use the negative cost as the reward signal. As stated in section 7.1.4, the total cost signal is the algebraic summation of the three types of costs, each of which reflects one of the expectations from the maintenance scheduler. Equation 7.10 formulates the reward signal. The bold italic terms are binary variables that represent the presence or absence of the term. For example, if pigging is the maintenance order in the past month, then the pigging value is set to 1; otherwise, it is 0. Also, $\vee$ signifies the Boolean "or".

$$\begin{aligned} \textit{total monthly cost} = & [480 \times \mathbf{leak} + 1760 \times \mathbf{burst}] + \\ & [3.5 \times \mathbf{pigging} + 13 \times \mathbf{inhibitor} + 80 \times \mathbf{coating} + \\ & 160 \times \mathbf{replacement} + [(1 - (\mathbf{leak} \vee \mathbf{burst})) \times (-5)] \end{aligned} \tag{7.10}$$

Finally, the maintenance scheduler makes decisions about which one of the five maintenance actions are the most appropriate and cost-effective to perform in a timely manner. Here we make the assumption that the corrosion depth and length are inspected and provided to the scheduler on a monthly basis, and the maintenance scheduler can choose among the maintenance actions every month after the inspection. In addition to the monthly corrosion inspection, the maintenance scheduler receives the cost of the previous month and incorporates that into its current month's decision.

CHAPTER 8

Results

In the previous chapters, we presented the proposed trustable deep RL algorithm and our test environment. This chapter concentrates on presenting and interpreting the results of applying the methodology to the test environment. Prior to delivering the results, we introduce a behavior policy that is deployed in the test environment to generate data in section 8.1.

The results section, section 8.2, follows the same structure as chapter 6 since this structure better conveys the merits of each part of the methodology's contributions. First, we present the results of applying the deep actor-critic network with weights initialization on data-efficiency. We then show how the safety constraints successfully avoid high-risk actions and improve the trustability of the deep agent's policy. Afterward, we evaluate the safety-adjusted policy's performance by estimating its expected return and a lower confidence bound on the estimation. In the last part of this chapter, section 8.3, we perform sensitivity analysis on how variations in the behavior policy would affect the proposed Trustable Deep RL's methodology's performance and accuracy.

8.1 Behavior Policy

The Trustable Deep RL methodology is developed for the class of sequential decision-making problems that requires an improved decision-making policy upon one or more previously applied policies. The Trustable Deep RL agent could arrive at a better policy by digesting a data set generated under the old policies. Therefore, the methodology's first requirements are the test and the train sets derived from the old policies. As stated in the previous chapters, we refer to the old policy as the behavior policy. The behavior policy utilized for data generation

Table 8.1: The stochastic behavior policy suggested by an environment expert and used for data generation.

CR	**NCD\NCL**	**0-0.2**	**0.2-0.4**	**0.4-0.6**	**0.6-1**
0	**0-1**	[1,0,0,0,0]	[1,0,0,0,0]	[1,0,0,0,0]	[1,0,0,0,0]
>0	**0-0.33**	[0.9,0.025, 0.025,0.025,0.025]	[0.04,0.75, 0.04,0.15,0.02]	[0.04,0.75, 0.04,0.15,0.02]	[0.025,0.025, 0.9,0.025,0.025]
>0	**0.33-0.66**	[0.04,0.75, 0.04,0.15,0.02]	[0.04,0.75, 0.04,0.15,0.02]	[0.04,0.75, 0.04,0.15,0.02]	[0.025,0.025, 0.9,0.025,0.025]
>0	**0.66-1+**	[0.025,0.025, 0.9,0.025,0.025]	[0.025,0.025, 0.9,0.025,0.025]	[0.025,0.025, 0.9,0.025,0.025]	[0.025,0.025, 0.9,0.025,0.025]

in this chapter is presented in table 8.1. This is a relatively good policy suggested by the experts of the simulated pipeline corrosion environment, and as it can be realized from the table, it is a stochastic policy that associates a probability mass function to each one of the five actions at each state. The sequence of the actions are do nothing, batch corrosion inhibitor, internal coating, pigging, and replacement.

By closely looking at the behavior policy, it can be referred that the expert choice of action for the states in which the corrosion rate is zero is to do nothing deterministically. The zero corrosion rate happens when the coating is applied, and therefore, there is no corrosion exacerbation and no need for any other maintenance actions. The behavior policy shifts from do nothing to pigging to inhibitor to coating as the corrosion gets worse.

In order to generate the training set, 1050 episodes with maximum length of 30 years (360 steps) are generated while the behavioral policy was in place. Also, another set of 450 episodes with maximum length of 30 years are generated for the test set. The episodes terminate if the agent reaches the maximum simulation time step, or if the pipeline goes through replacement, or if the pipeline enters a failed state.

The generated train set is preprocessed by removing the too short episodes, i.e., episodes with length less than 10 steps. Although these short episodes might be valuable experiences, they are removed because they adversely affect the training process. During the training, random batches of episodes with the same length are sampled from the episode replay memory. Therefore, when the episodes' lengths are different, we truncate all the episodes to have the same length as the shortest one. Consequently, too short episodes result in short training batches that leads to an agent with poor long-run performance.

In the next step of the preprocessing, the train episodes are saved in an experience replay memory as experience tuples and in an episodic replay memory as complete episodes. Later, the experience memory is used for the Q-val network training and the episode replay memory used for the actor-critic network training. The total number of experience tuples in the training set is 121109, and the total number of episodes in the training set is 994. The test set's episodes are saved as complete episodes and later used for evaluating the new policy's performance.

8.2 Applying the Proposed Trustable Deep RL to the Pipeline Maintenance Management

The training and evaluation phases of the Trustable Deep RL agent is brought in algorithm 10. This algorithm consists of three parts, each highlights one of the three main ideas of this research, namely, computing the deep actor-critic agent's policy (lines 1-6), adjusting the deep agent's policy according to the safety constraints (lines 7-13), evaluating the safety-adjusted policy's performance with a lower confidence bound (lines 14-19). The results of each part are brought in the following.

8.2.1 Data Efficiency

The data efficiency of the proposed actor-critic network with weight initialization is evaluated in comparison with a Dueling Double Deep Q-network (DDDQN, section 4.2.4) and an actor-critic network with no initialization. First, we follow algorithm 7 to train the actor-critic network with weight initialization. Then, we show the learning curves of the three deep RL agents and compare them.

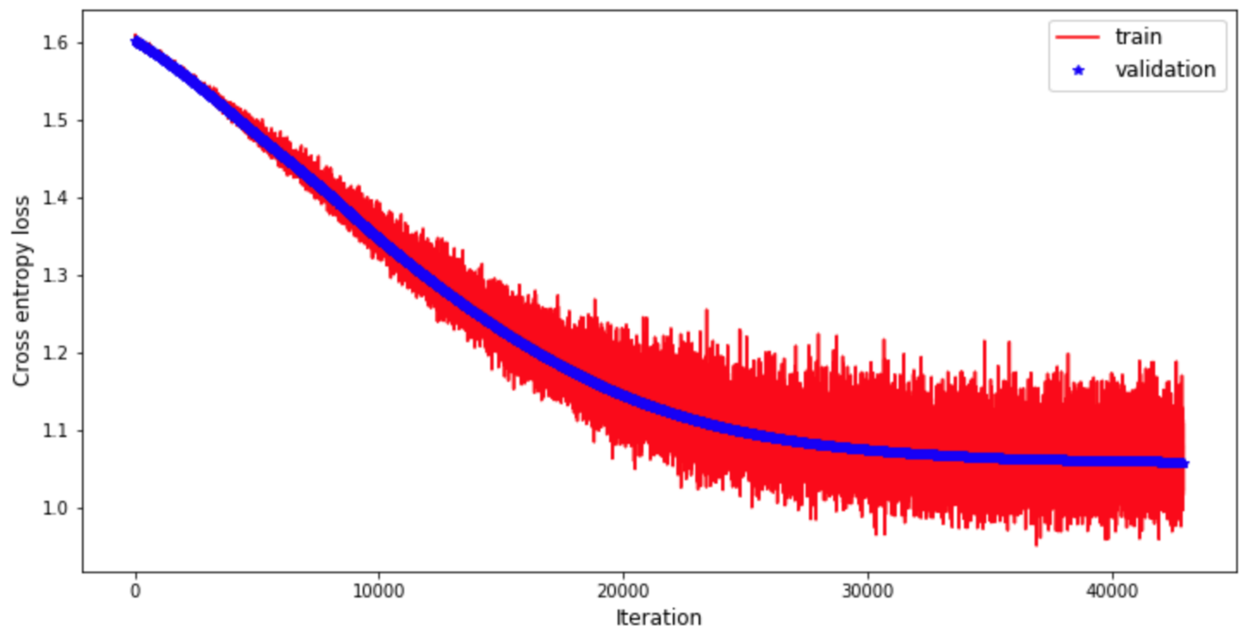


Figure 8.1: The cross-entropy loss of the train and the validation sets during the behavior network's training process.

We built the behavior policy network with the same structure as the actor network (section 5.1) and trained that in a multi-class classification setting considering S_t and A_t from the experience tuples as the input features and the labels, respectively. Figure 8.1 presents the behavior network loss curve. Unlike classification problems that widely choose accuracy as the learning evaluation indicator, we picked the cross-entropy loss because our goal is to estimate each action's probabilities, not the true labels. Figure 8.1 shows the behavior network training reached a stable value and does not suffer from high bias or variance. Therefore, we copy the weights into the actor network.

In the next step, a DDDQN with the same structure as the critic network (section 8.2) is trained using the experience replay memory. This network is called the Q-val network, and its loss curve during the training is plotted in figure 8.2.

This plot is misleading for evaluating the Q-val network's training quality because it shows constantly increasing loss with lots of variations. However, we cannot rely on the loss curve for deep RL agents' training evaluation because the target values are not fixed in deep RL networks

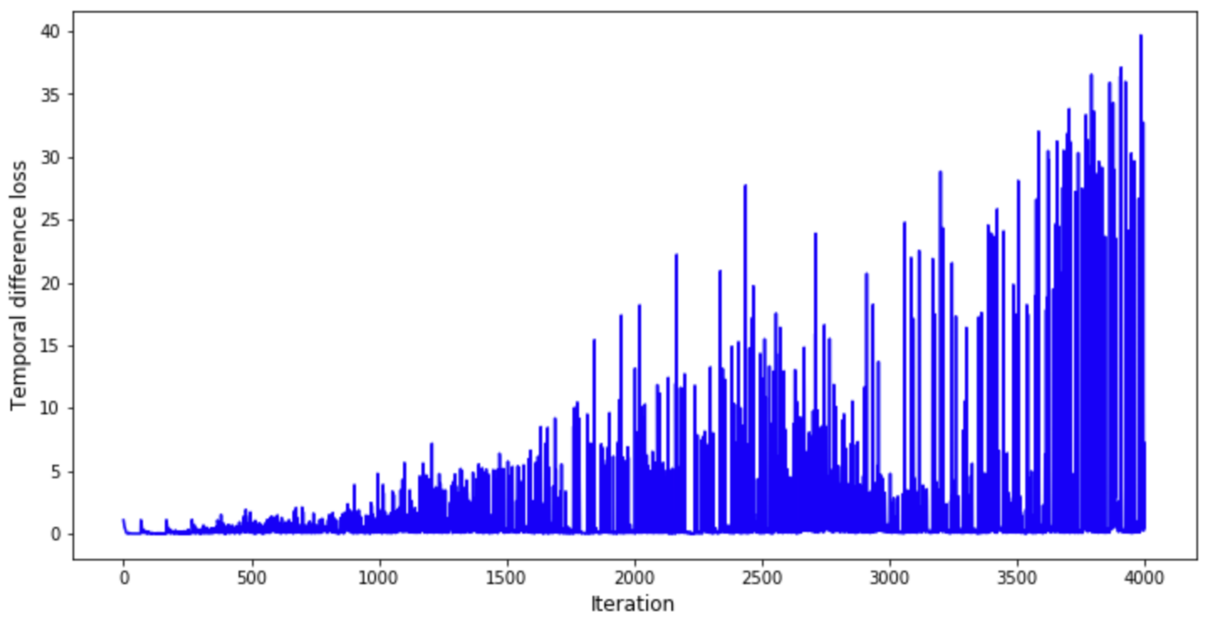


Figure 8.2: The temporal difference loss during the Q-val network training process.

training (please refer to equations 5.1 and 4.4, and their related sections.) The standard training evaluation practice in the deep RL literature is to plot an indicator of the agent's performance in various training stages [HMVH+18, MKS+13, MKS+13, DRBM18, SQAS15]. One major difference between our deep RL algorithm and the studies mentioned above is in our problem framework, learning happens in an offline setting while those studies work in a fully or partially online settings. Online training means the agent's choice of action is implemented in the system, and the reward is visible at each iteration. Hence, it is straightforward to compute the total reward at the end of each episode.

Here we take the same approach of plotting the agent's performance versus iterations to evaluate the agent's training with one distinction that we cannot implement the agent's choice of action in every iteration of offline training. Instead, we propose testing the agent's performance by implementing the most updated policy in the simulated pipeline system every 50 iterations. To account for the environment and the policy stochasticity, we simulate three episodes with the most updated policy and report the average return over the three episodes. Figure 8.3 shows the average return of the agent over the three test episodes in every 50 iterations for the Q-val network.

We should emphasize that testing in the target environment is not possible when dealing with real, risk-sensitive settings, and this step is not part of the Trustable Deep RL algorithm. Here we take advantage of our simulated environment only to demonstrate the quality of the training process.

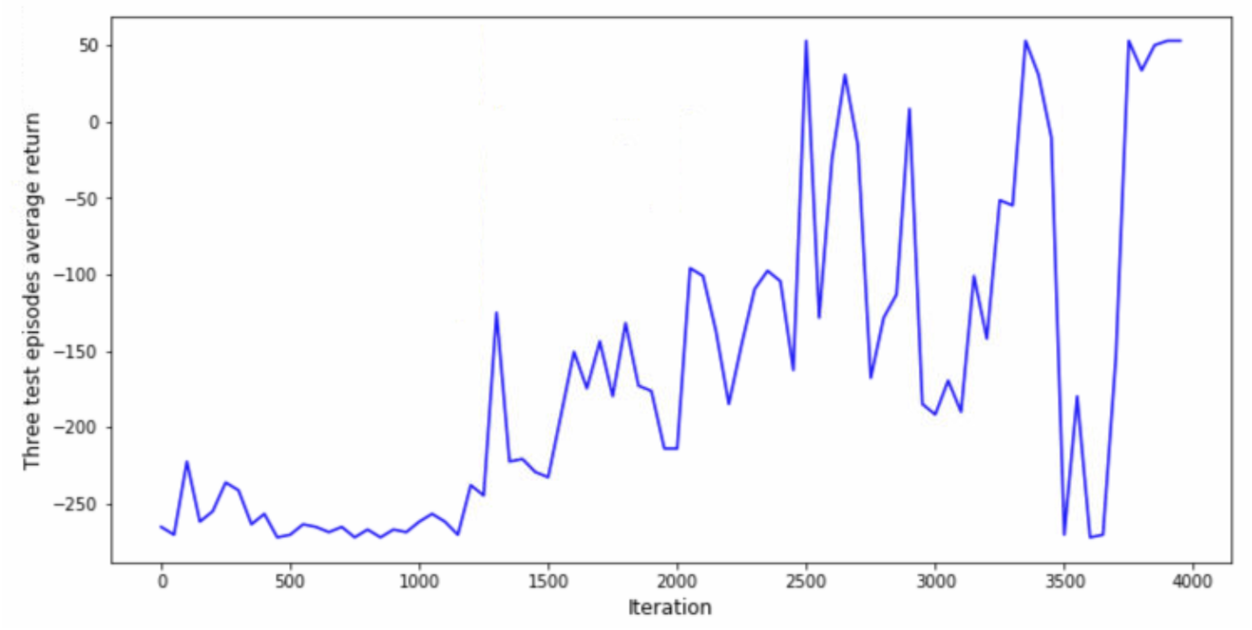


Figure 8.3: The average return over three test episodes during the Q-val network training process.

The Q-val network starts training with randomly initialized weights, and therefore, the network's policy performs poorly at the beginning, with three episodes average return around −250. After 1200 iterations, the network begins to improve the policy and gain higher average returns; however, the improvement is not stable, and we observe variations in the three-episode average return value. Finally, after 4000 iterations, the Q-val network's return reaches 53.

According to algorithm 7, in the next step, we copy the Q-val network's weights into the critic network, which has the same architecture as the Q-val network. Afterward, we train the actor-critic network with initialized weights copied from the behavior network and the Q-val network, following section 4.3. To evaluate the training process, we implement the most updated policy every 50 iterations in the simulated pipeline environment and run three test episodes. Figure 8.4 shows the average return over the three test episodes versus the

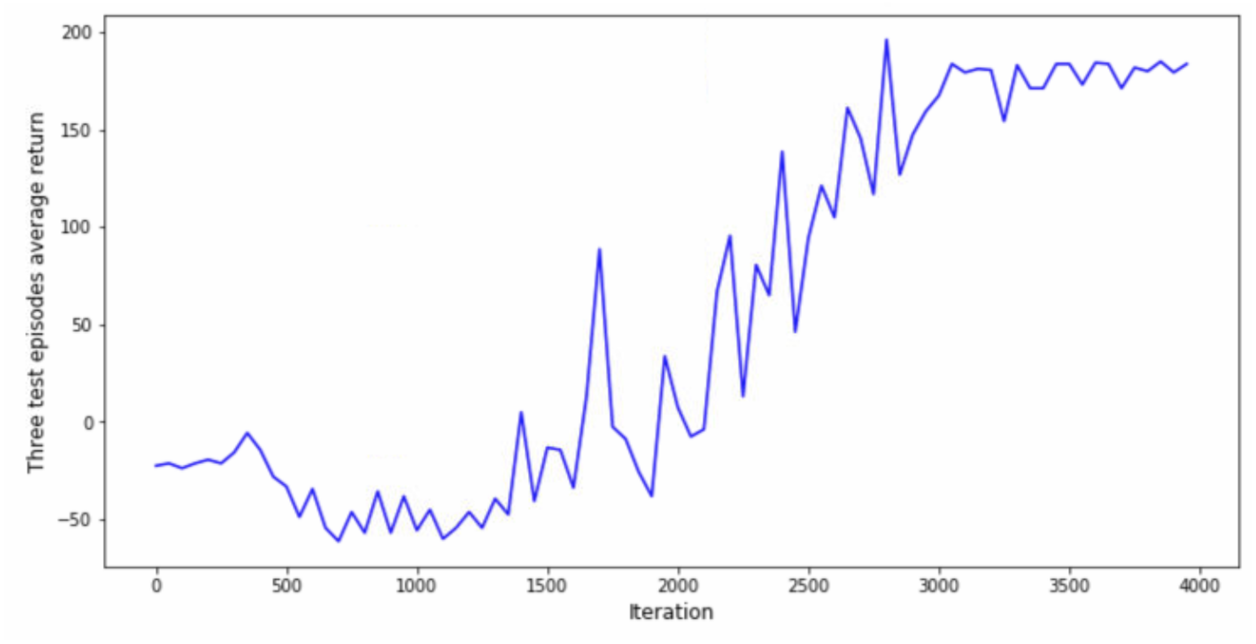


Figure 8.4: The average return over three test episodes during the actor-critic network with weight initialization.

iterations.

The actor-critic network with initialized weights starts off the training from a higher return value than the DDDQN, figures 8.3 and 8.4. This could be associated with the initial weights of the two networks. The actor network's initial weights mimic the behavior policy, while the initial weights of the DDDQN are random, resulting in a random initial policy. The training of the actor-critic network with initialized weights continues with a drop, and then it generally increases until it reaches the return value of 196 at iteration 3000. In the last 1000 iterations, the return value does not vary much and stays at around 190 with small variations. By comparing figure 8.4 with figure 8.3, we can conclude that the proposed actor-critic network with weights initialization outperforms the DDDQN because it reaches a higher return and demonstrates more stable training; therefore, it has superior data efficiency than the DDDQN network.

To evaluate the proposed initialization merits on the actor-critic structure's data efficiency, we train an actor-critic network in the same way as section 4.3 with random Xavier weight initialization [GB10]. This network's training is evaluated in the same way as the previous

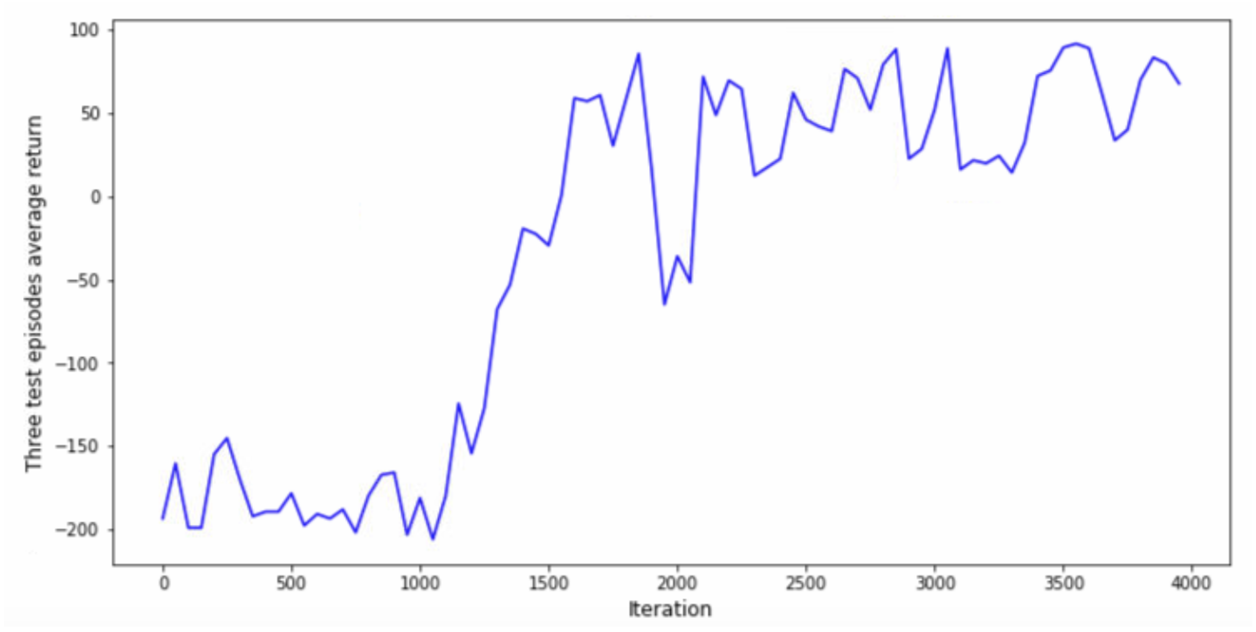


Figure 8.5: The average return over three test episodes during the actor-critic network with no weight initialization.

two networks, and the result is plotted in figure 8.5.

According to the above graph, the actor-critic network starts the training with a low return value at around -200, and does not make improvement until after 1000 iterations. The plots' overall trend shows the policy improves from 1000 iterations to 1700 iterations, and till the end of the training at 4000 iterations, it swings between 91 and 24.

Comparing the three deep RL networks' training evaluation graphs (Figures 8.3, 8.4, and 8.5) shows that the proposed actor-critic network with weights initialization achieved the highest return and the most stable training. Since the training set and the number of training iterations are the same for the three deep RL networks, the higher return value infers that the proposed actor-critic network with weights initialization has better data efficiency than DDDQN and actor-critic without initialization.

8.2.2 Safety

The first trustability criterion defined in section 6.3 is safety. The proposed algorithm ensures the safety of the deep actor-critic RL agent by eliminating undesirable behaviors. This section quantitatively investigates how eliminating undesirable behaviors improves the deep RL outcome policy's safety.

Following the Trustable Deep RL agent methodology, algorithm 10, after training the deep actor-critic network and the behavior network, the user must apply expert-in-the-loop constraints to the deep actor-critic network's output policy. The actions "do nothing" and "pigging" shall not be performed by common sense when the corrosion is severe. We consider a corrosion defect as severe when 60% of the wall thickness is corroded. On this ground, we define the following hazard zones for the two aforementioned actions:

$$\mathbf{S}^{HZ}\ (a_i = \textit{do nothing}) = \{CR \neq 0,\ \ NCD \geq 0.6, NCL \geq 0.6\}, \tag{8.1}$$

$$\mathbf{S}^{HZ}\ (a_i = \textit{pigging}) = \{CR \neq 0,\ \ NCD \geq 0.6, NCL \geq 0.6\}. \tag{8.2}$$

Other actions do not have expert-in-the-loop constraints,

$$\mathbf{S}^{HZ}\,(a_i) = \emptyset\ ;\ \ a_i \in \{\textit{inhibitor}, \textit{coating}, \textit{replacement}\}. \tag{8.3}$$

The next step is to set the data-span constraints that require the unknown zone for each action. We have access to the explicit behavior policy of table 8.1, and it is straightforward to derive the unknown zone of each action from their support sets in the explicit policy. However, we take the similarity check approach because knowing the explicit policy is rare in real problems.

The case-bases are formed by first separating the training set experience tuples based on the actions and then storing each group's state vectors in a separate case-base. Consequently, we build five case-bases of state vectors, each related to one of the five actions. When a new state arrives, the distances between the new state and the nearest neighbors in each of the five case-bases are measured according to equation 6.3. Any action that its corresponding nearest neighbor distance is larger than θ would have zero probability in the new state's policy, equation 6.4.

To assign θ, we first measure the greatest distance within the state space. In the test environment, the state variables are max-normalized; hence the state space is a $1 \times 1 \times 1$ cube, and the line between diagonally opposite corners is the greatest distance in the cube. The length of the diagonal is $\sqrt{3}$, hence we pick $\theta = \frac{\sqrt{3}}{2}$. We would like to emphasize here that θ is one of the algorithm's hyperparameters, and it might take several experiments to find the best value for that. When the explicit policy is not available, but some outside information exists on the unknown zones of the actions, we recommend integrating that with the similarity check approach to gain more accurate data-span constraints.

After setting up the policy search constraints, we need to define a metric to measure whether the constraints improve the RL agent's safety. Our test environment is an industrial installation, namely a gas pipeline system, in which equipment failure causes extensive downtime, environment and facility damage, and on-site injuries. Therefore, a safer maintenance policy is the one that leads to lower failure risk. To evaluate a maintenance policy's failure risk, we simulate 50 pipeline life episodes while the maintenance policy is in place. The fraction of the episodes that terminates by failure reflects the failure risk of the in-place maintenance policy.

Here we call the deep actor-critic network's policy before passing through the safety measures as the network policy and after passing through the safety measures as the evaluation policy. Table 8.2 reports the number of failed episodes under the behavior policy, the network policy, and the evaluation policy.

Table 8.2: The number of the failed episodes in 50 runs of the pipeline simulation while the behavior policy, the network policy, and the evaluation policy were in place.

		Behivor policy	Network policy	Evaluation policy
Failed episodes in 50 runs	Leak	5	2	0
	Burst	1	0	0

According to table 8.2, 6 out of 50 simulated pipeline life episodes ended by failure when the behavior policy was in place. That was foreseeable because we intentionally designed the behavior policy imprudent. The network policy performed much safer because the actor-critic deep agent has learned from the behavior policy experiences and made improvements upon that. However, 2 cases of leakage failure are still observed in 50 simulated pipeline life episodes. Finally, the evaluation policy, which is the network policy after passing through the hazard zone and unknown zone constraints, performed completely safe without any failed episodes.

We further looked into the average returns of the network and evaluation policies to check how much of the actor-critic output policy's optimality is lost by imposing the policy constraints. The evaluation policy average return is 187 while the network policy average return is 193. Accordingly, we can conclude that undesirable behavior elimination successfully improved the safety of the deep RL agent's maintenance policy without a significant reduction in the network policy's expected return (optimality).

8.2.3 Off-Policy Evaluation with Lower Confidence Bound

The second trustability criterion is the new policy suggested by the agent must perform at least as good as the behavior policy. Following algorithm 10, we compute the estimated expected returns of the two policies and their lower confidence bounds to evaluate how the new policy performs comparing to the behavior policy. Algorithm 9, which is invoked within algorithm 10, handles this computation. It applies equation 6.7 on the whole $\mathcal{D}_{test}$ to estimate the behavior policy's expected return, $\hat{J}^b$, and the WPDIS estimator (algorithm 8) on the whole $\mathcal{D}_{test}$ to estimate the evaluation policy's expected return, $\hat{J}^e$. The estimated expected returns under the two policies are brought in table 8.3. These values are computed for the 450 episodes in the test set.

We observe that the evaluation policy's expected return is estimated to be 226.56, which is larger than the estimated expected return for the behavior policy, 44.31. Hence, on average, the evaluation policy is estimated to perform much better than the behavior policy.

Table 8.3: The estimated expected returns by the Trustable Deep RL algorithm for the behavior and evaluation policies on the $\mathcal{D}_{test}$.

Estimated expected return	$\hat{J}^b$	$\hat{J}^e$
Value	44.31	226.56

Comparing only the point estimates of the expected returns might not gain trust of the highly risk-averse domains' users. The proposed Trustable Deep RL algorithm provides lower confidence bounds on the expected returns to bring more confidence about the new policy's performance with respect to the behavior policy. Algorithm 9 calculates three values for the lower confidence bounds and report the maximum of the three values to make the bounds tighter.

The three confidence bounds are computed leveraging the MPeB, AM, and BCa concentration inequalities, which are extensively discussed in chapters 3 and 6. The first requirement for using these CIs is to acquire a set of samples of the estimated expected returns. Algorithm 9 partitions the test set into batches of size $K = 5$ and computes the estimated expected returns for the behavior and the evaluation policies, in each batch. Consequently, there are $\frac{450}{5} = 90$ samples for the behavior policy's expected return estimated by the empirical mean, equation 6.7, and 90 samples for the evaluation policy's expected returns estimated by WPDIS, algorithm 8.

We need the lower and upper ranges of the estimated expected returns to compute the MPeB bounds. In the gas pipeline test environment, the largest return happens when the maximum reward of +5 receives in every step, and the length of the episode is the largest, i.e., 360 steps. This condition is impossible to happen because the reward of +5 is gained when no maintenance action has been applied, and the pipeline would not operate for 30 years without maintenance. In fact, it would fail due to severe corrosion after approximately five years. However, this unrealistic scenario provides us with an approximate upper range for the return. If we assign $R_t = +5$, $t_{max} = 360$, and $\gamma = 0.99$, we get the return upper limit of +486. We assume all the episodes in equation 6.7 have the maximum return of +486;

therefore, we get an upper range of $+486$ for the estimated expected return of the behavior policy. The upper range of the evaluation policy's estimated expected return is the same as the behavior's because the importance weights in the numerator and denominator of equation 3.24 cancel out each other's effects.

We investigate a few scenarios to find a tight estimate for the lower range of the return under the behavior policy. The reward discount factor in all the scenarios is kept at 0.99, a typical value used in the RL literature that deals with long episodes [MBM+16, WAD17, LHP+15]. The first scenario is when the most costly maintenance happens every month. Among the pipeline's four maintenance actions, replacement is the most expensive action. However, the episode would end as soon as a replacement is performed. In the worst case, if we assume the replacement happens at the first step of the episode, it would result in the return value of -160. The second most expensive maintenance action is coating. When coating is performed, the corrosion rate would be zero for five consecutive years; hence, according to the behavior policy, table 8.1, no other maintenance action would be performed. Therefore, this scenario does not provide a low return value.

The next scenario is to apply the batch corrosion inhibitor every month. This scenario would result in the return value of -778, assuming the pipeline survives for 30 years. We do not consider a scenario with pigging because it is a less costly action than the inhibitor. The last scenario is when the pipeline life episode terminates by a burst failure. If no maintenance action is performed, the pipeline will fail after approximately five years by burst failure. To get a lower range for the return, we assume it survives for four years with no maintenance and afterward fails due to burst. In this scenario, we get the return value of -898. The reward and cost values in all the above scenarios are taken from table 7.2.

The minimum return among the above scenarios belongs to the last scenario, pipeline failure by burst after four years of operation without maintenance actions. Therefore, we pick -898 as the return lower limit. Similar to the upper range, the same value, -898, holds as the lower range of the behavior and evaluation policies' estimated expected return.

The confidence level selected for this experiment is 90%; therefore, the final lower confidence

Table 8.4: The input parameters used in algorithm 9.

Parameter	$\mathcal{D}_{test}$	δ	γ	K	B	a_e	b_e	a_b	b_b
Value	450	10%	0.99	5	200	-898	486	-898	486

bounds estimate values that with a 90% chance, the true expected returns are larger than those values. The results of applying the three confidence bounds to the test set are brought in table 8.5.

Table 8.5: Lower confidence bounds estimated for the policies' expected returns.

Policy \ Bound type	MPeB	AM	BCa	Maximum
π_e	20.69	26.37	223.37	223.37
π_b	-181.41	-139.91	36.15	36.15

According to the above table, the MPeB and AM inequalities computed very close lower bounds, with AM performing slightly better. Whereas the BCa performed much better and computed much tighter bounds. Consequently, we pick BCa bounds as the 90% confidence lower bounds of the returns. The evaluation policy's performance has 90% confidence lower bound of approximately 223.37, and the behavior policy has 90% confidence lower bound of approximately 36.15. Looking jointly at tables 8.5 and 8.3 endorses that the estimated expected return of the evaluation policy is much larger than the behavior policy. Besides, its 90% confidence lower bound is much larger. This result indicates that the evaluation policy is strongly a better policy than the behavior policy, and therefore, it is safe to be implemented in real pipeline maintenance management.

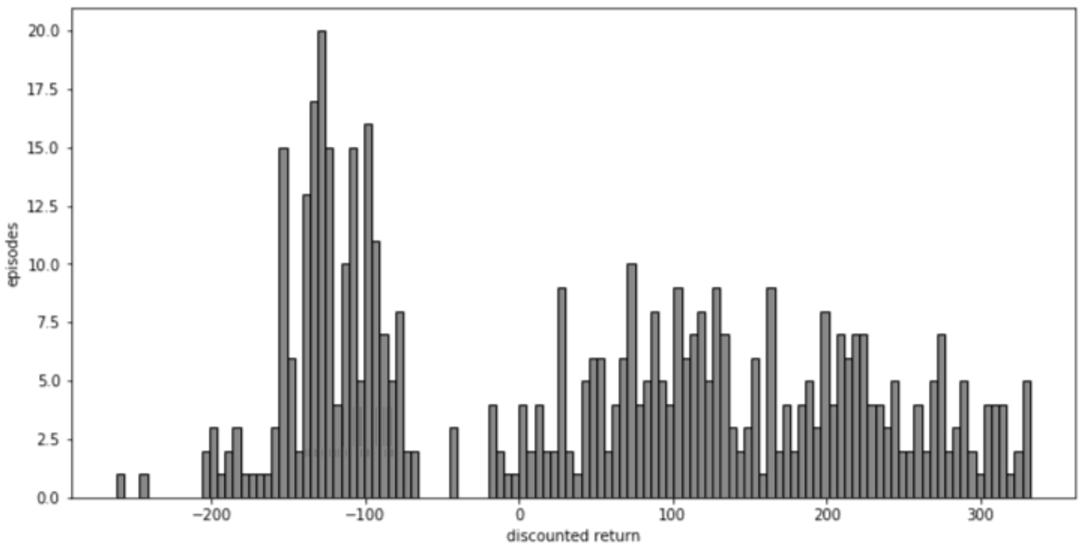
After the new policy's performance is endorsed by the off-policy evaluation steps in the Trustable Deep RL algorithm, it is allowed to be deployed as the decision-making mechanism of the pipeline maintenance management. Thus, we can assess the off-policy evaluation steps' accuracy by on-policy evaluation, i.e., directly implementing the evaluation policy in the test environment. We generate 450 episodes of the pipeline life when the new policy is in place

Table 8.6: Empirical and estimated expected returns of the behavior and the evaluation policies.

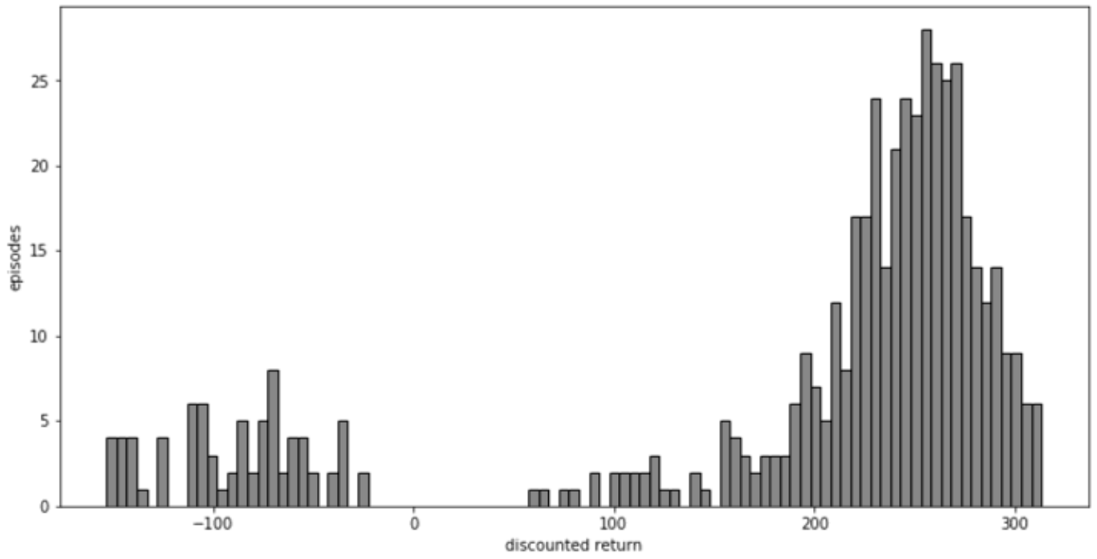
Average return	**Value**
Empirical $\hat{J}^b$ over 450 test episodes	44.31
Estimated $\hat{J}^e$ over 450 test episodes	226.56
Empirical $\hat{J}^e$ over 450 deployed episodes	190.02

and compute each episode's return. According to table 8.6, the empirical mean of these 450 episodes' returns is 190.02, which was estimated to be 226.56 by the WPDIS algorithm. This result proofs that the adopted methodology for off-policy evaluation provides an estimate with reasonably good accuracy. We should emphasize that implementing the new policy for off-policy accuracy assessment is not part of the Trustable Deep RL algorithm and might not be doable in most real environments' problems.

The histograms of the returns under both policies for 450 episodes are brought in figures 8.6a and 8.6b to better compare the performances of the evaluation policy with the behavior policy. Figure 8.6a shows that there are two peaks in the frequency distribution of the return with the behavior policy, one around -120 and one around $+120$. The peak around -120 is sharp and narrow while the one near $+120$ is wide and flat. The return histogram of the evaluation policy is also a bimodal distribution; however, the right peak is dominant in 8.6b showing the new policy is more confident about taking better actions. We conclude that the new policy suggested by the Trustable Deep RL algorithm successfully shifted the return toward higher values. Moreover, these figures endorse the considerable difference between the 90% confidence lower bounds of the behavior and evaluation policies.



(a)



(b)

Figure 8.6: Histogram of the returns for on-policy evaluation of: 8.6a behavior policy and 8.6b evaluation policy.

8.3 Sensitivity Analysis

This section performs sensitivity analysis on the behavior policy to investigate how variations in that would affect the proposed Trustable Deep RL algorithm's performance and accuracy. Particularly, we study how the algorithm responds when using a more optimal or less optimal policy for data generation.

The first sensitivity scenario is to use a more optimal policy for data generation. To design a more optimal policy, we closely studied some of the pipeline life episodes under the previous section's evaluation policy, which histograms of figures 8.6a and 8.6b proved its superior performance. We observed that the previous section's evaluation policy performs all the maintenance actions less frequently and prefers pigging rather than applying inhibitor. Hence, in the first sensitivity scenario's behavior policy, table 8.7, we assigned higher probability to "do nothing" in all the stages of the corrosion, higher probability of "pigging" in the earlier stages of the corrosion, and higher probability of "inhibitor" in the later stages of the corrosion comparing with the policy of table 8.1.

In the next step, we generate 1050 episodes with maximum length of 30 years as the training set and another 450 episodes as the test set while the first scenario's behavior policy is in place. The generated data set is processed similar to section 8.1. Afterward, we follow algorithm 10 for training and evaluation of the Trustable Deep RL algorithm. All the algorithm's requirements, such as the hazard zones, lower and upper ranges of the returns, and the WPDIS batch size, are same as the previous sections, except for the behavior policy. The evaluation results of the new policy suggested by the Trustable Deep RL algorithm are brought in tables 8.8 and 8.9.

The following tables indicate that the new policy is estimated to have the expected return of 304 with 90% confidence lower bound of approximately 296.92. Whereas the behavior policy is estimated to have the expected return of 210 with 90% confidence lower bound of approximately 208.83. Hence, the new policy is strongly a better policy than the behavior policy and trustable for implantation in the pipeline test bench. In the next step, to examine

Table 8.7: The stochastic behavior policy used for data generation in the first sensitivity scenario.

CR	NCD\NCL	0-0.2	0.2-0.4	0.4-0.6	0.6-1
0	0-1	[1,0,0,0,0]	[1,0,0,0,0]	[1,0,0,0,0]	[1,0,0,0,0]
>0	0-0.33	[0.74,0.1, 0.005,0.15,0.005]	[0.45,0.1, 0.005,0.44,0.005]	[0.45,0.1, 0.05,0.44,0.005]	[0.45,0.1, 0.005,0.44,0.005]
>0	0.33-0.66	[0.45,0.1, 0.005,0.44,0.005]	[0.45,0.1, 0.005,0.44,0.005]	[0.325,0.42, 0.1,0.15,0.005]	[0.325,0.4, 0.16,0.11,0.005]
>0	0.66-1+	[0.225,0.47, 0.2,0.1,0.005]	[0.322,0.4, 0.2,0.1,0.005]	[0.325,0.4, 0.16,0.11,0.005]	[0.325,0.1, 0.5,0.025,0.05]

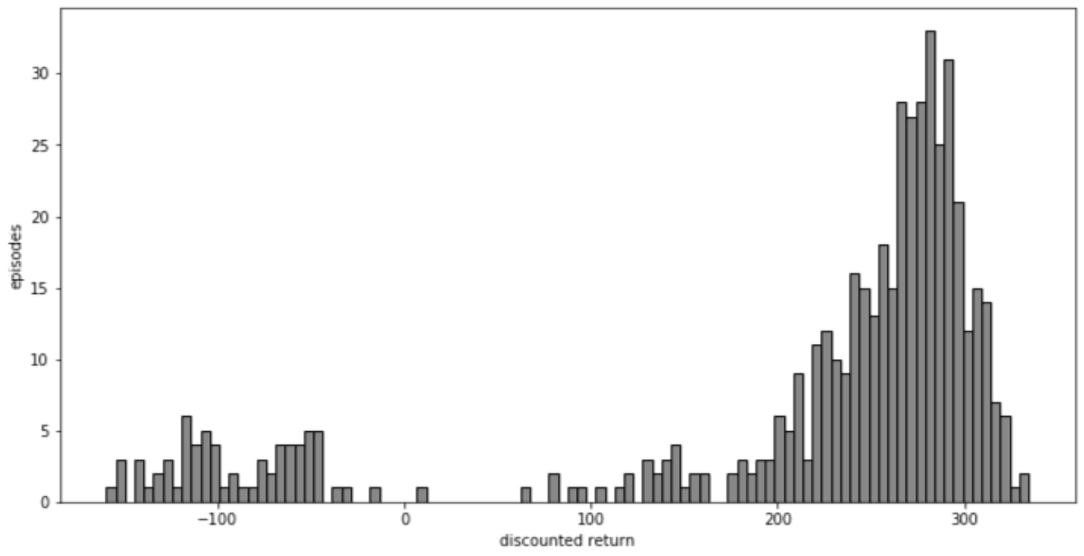
Table 8.8: The estimated expected returns by the Trustable Deep RL algorithm for the behavior and evaluation policies in the first sensitivity scenario.

Estimated expected return	$\hat{J}^b$	$\hat{J}^e$
Value	210	304

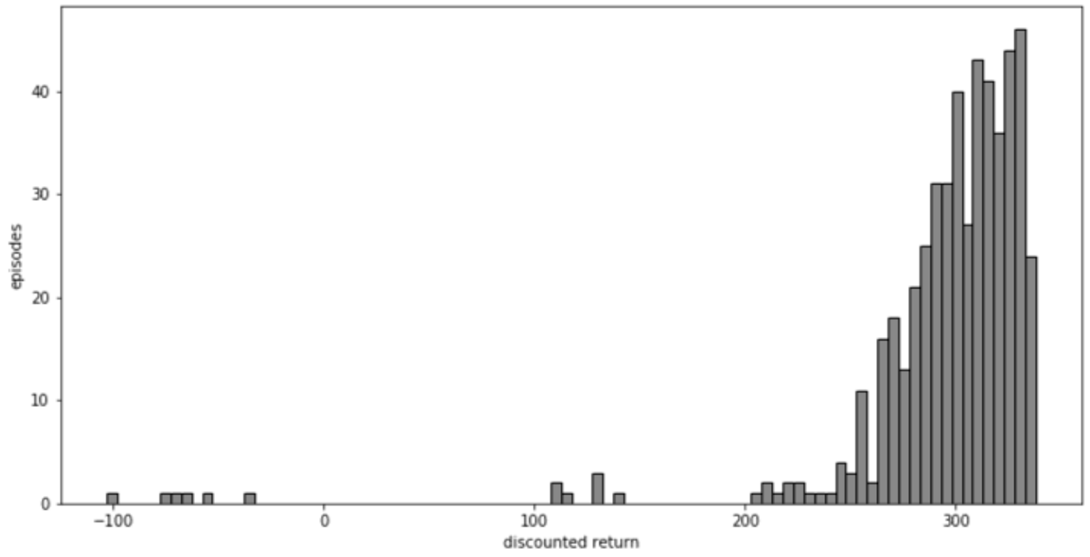
Table 8.9: Lower confidence bounds for the estimated expected returns in the first sensitivity scenario.

Policy \ Bound type	MPeB	AM	BCa	Maximum
π_e	103.84	94.00	296.92	296.92
π_b	-2.22	7.49	208.83	208.83

the accuracy of the off-policy evaluation, we generate 450 episodes of the pipeline life when the new policy is in place and compute each episode's return. The return empirical mean of these 450 on-policy episodes is 295.13, which was estimated to be 304 by the WPDIS algorithm. Therefore, we conclude that the methodology provides a more accurate estimate for the expected return when the policy is near-optimal. Figures 8.7a and 8.7b show the histograms of the returns under both policies for 450 episodes. In figure 8.7a we observe the similar bimodal distribution as in figure 8.6a; however, the evaluation policy almost eliminates the lower return episodes, figure 8.7b.



(a)



(b)

Figure 8.7: Histogram of the returns for on-policy evaluation of: 8.7a behavior policy and 8.7b evaluation policy in the first sensitivity scenario.

Table 8.10: The stochastic behavior policy used for data generation in the second sensitivity scenario.

CR	**NCD\NCL**	**0-0.2**	**0.2-0.4**	**0.4-0.6**	**0.6-1**
0	**0-1**	[1,0,0,0,0]	[1,0,0,0,0]	[1,0,0,0,0]	[1,0,0,0,0]
>0	**0-0.33**	[0.84, 0.05, 0.005, 0.1, 0.005]	[0.84, 0.05, 0.005, 0.1, 0.005]	[0.05 , 0.1 , 0.84, 0.005 , 0.005]	[0.05, 0.1, 0.84, 0.005 , 0.005]
>0	**0.33-0.66**	[0.84, 0.05, 0.005, 0.1, 0.005]	[0.84, 0.05, 0.005, 0.1, 0.005]	[0.05 , 0.1 , 0.84, 0.005 , 0.005]	[0.05, 0.1, 0.84, 0.005 , 0.005]
>0	**0.66-1+**	[0.84, 0.05, 0.005, 0.1, 0.005]	[0.84, 0.05, 0.005, 0.1, 0.005]	[0.05 , 0.1 , 0.84, 0.005 , 0.005]	[0.05, 0.1, 0.84, 0.005 , 0.005]

Table 8.11: The estimated expected returns by the Trustable Deep RL algorithm for the behavior and evaluation policies in the second sensitivity scenario.

Estimated expected return	$\hat{J}^b$	$\hat{J}^e$
Value	-93	175

Table 8.12: Lower confidence bounds for the estimated expected returns in the second sensitivity scenario.

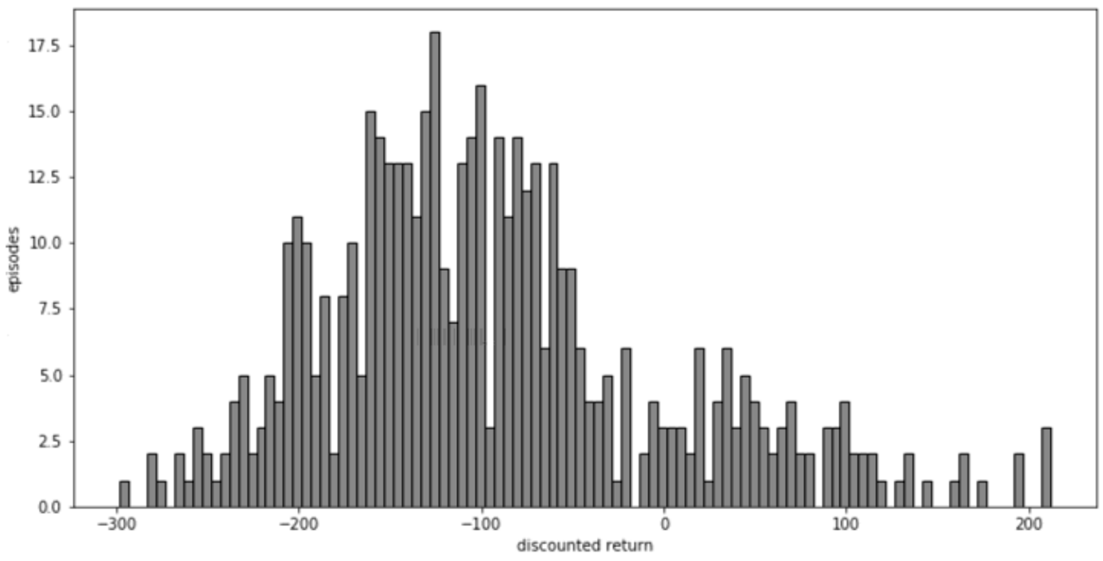
Policy \ Bound type	MPeB	AM	BCa	Maximum
π_e	-44.17	-22.84	150.02	150.02
π_b	-299.43	-238.96	-90.52	-90.52

The second sensitivity scenario is to place a less optimal policy than table 8.1 for data generation, and then investigate how the algorithm's output policy would improve upon that, and how accurate the off-policy evaluation would be. We designed the second sensitivity scenario's behavior policy to perform maintenance actions much less often than table 8.1. Moreover, the probability of applying inhibitor in moderate and severe corrosion is larger than pigging. The second scenario's behavior policy is shown in table 8.10

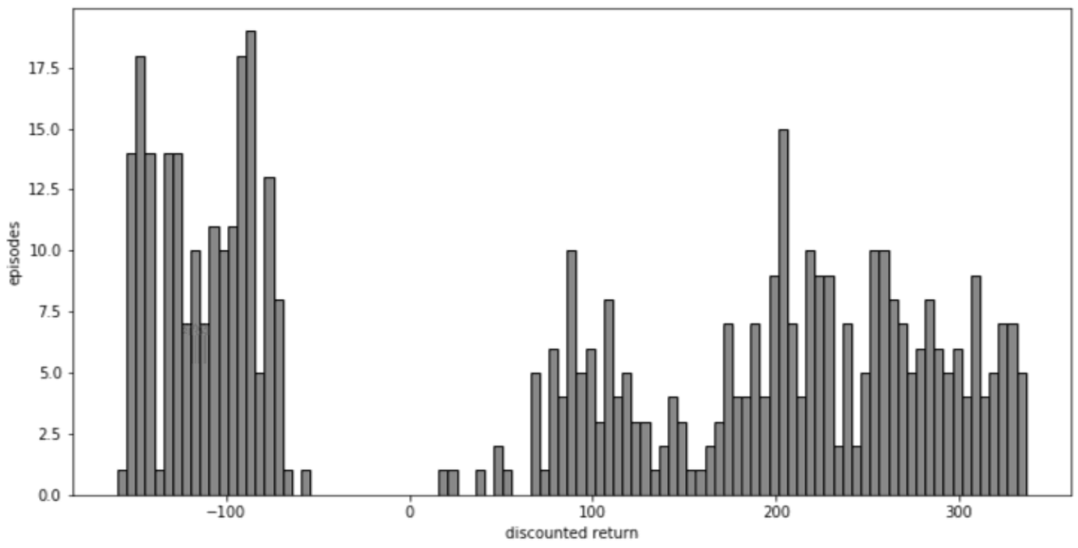
In the next step, we generate 1050 episodes with maximum length of 30 years as the training set and another 450 episodes as the test set while the second sensitivity scenario's behavior policy is in place. The generated data set is processed similar to section 8.1. Afterward, we follow algorithm 10 for training and evaluation of the Trustable Deep RL algorithm. All the algorithm's requirements, such as the hazard zones, lower and upper ranges of the returns, and the WPDIS batch size are same as the previous sections, except for the behavior policy. The evaluation results of the new policy suggested by the algorithm are brought in tables 8.11 and 8.12.

The above tables indicate that the new policy is estimated to have the expected return of 175 with 90% lower confidence bound of approximately 150.02. Whereas the behavior policy is estimated to have the expected return of -93 with 90% lower confidence bound of approximately -90.52. Hence, the Trustable Deep RL algorithm successfully found a strongly better policy than the input policy which is trustable for implantation in the pipeline test bench. In the next step, to examine the accuracy of the off-policy evaluation steps, we generate 450 episodes of the pipeline life when the new policy is in place and compute each episode's return. The empirical mean of these 450 episodes' returns is 82.75, which was estimated to be 175 by the WPDIS algorithm. The large difference between the WPDIS-estimated return and the empirical mean return infers that the WPDIS's accuracy for the second sensitivity scenario is less than the other two studied scenarios.

The histograms of the returns are brought in figures 8.8a and 8.8b to better compare the performances of the behavior and evaluation policies in the second scenario. We observe that the behavior policy's return histogram is a single-mode, right-skewed distribution with mean around -100, figure 8.8a. Whereas the evaluation policy's return histogram is a bimodal distribution with a sharp but narrow distribution around -110 and a wide and flat distribution around 200 8.8b. The emergence of the right distribution in the evaluation policy's histogram shows that the Trustable Deep RL algorithm successfully found a better performing policy.



(a)



(b)

Figure 8.8: Histogram of the returns for on policy evaluation of: 8.8a behavior policy and 8.8b evaluation policy in the second sensitivity scenario.

Table 8.13: Summary of the Trustable Deep RL performance in the sensitivity analysis and base-line scenarios.

	Near-optimal π_b, table 8.7	Base-line π_b, table 8.1	Poor π_b table, 8.10
Behavior policy empirical mean	210	44	-93
Evaluation policy estimated expected return	304	226	175
Evaluation policy empirical mean	295	190	82

In summary, the sensitivity analysis approach we employed tested the Trustable Deep RL algorithm's performance in different scenarios of the behavior policy optimality. Table 8.13 very concisely summarizes the sensitivity analysis results. We observe that as we enhance the behavior policy, the Trustable Deep RL algorithm also improves its suggested policy. However, the accuracy of its estimate on the new policy's expected return decreases. One reason might be when we use a poor policy for generating data, the algorithm finds a policy that is very different from the data generation policy, causing the IS weights of the WPDIS estimator to explode or vanish, which leads to estimation inaccuracy. On the other hand, when the data generation policy is near-optimal, the algorithm's suggested policy would vary less from the data generation policy; hence the WPDIS estimator provides more accurate estimates.

CHAPTER 9

Conclusion and Future Work

Deep RL is one of the most promising avenues of AI research that has gained significant breakthroughs in several applications. Despite all the recent advances made in the field, there is still a long way to deploy deep RL in real and highly risk-sensitive systems, e.g., clinical treatment policy and industrial installation's maintenance management policy. RL has several limitations that hinder its application to everyday life problems, including state definition and state visibility, credit assignment, and restricted online interaction. In this dissertation, we tackle the two main deep RL limitations that we believe they need more fundamental research, namely trustability and data efficiency. In the following, we summarize each chapter and highlight the novelties. This dissertation's contributions are chapters 5 and 6, where the Trustable Deep RL algorithm and the pipeline simulated test bench are proposed.

The dissertation began with a formal description of RL and mathematical representation of its elements in chapter 2. We introduced two classes of RL solutions, namely value-based methods and policy-based methods, and presented some of the vanilla off-policy RL algorithms that set the ground for more advanced algorithms presented or developed in the following chapters. We stated that RL algorithms' objective function is to maximize the expected return and an agent or a decision-making mechanism has a better policy if its expected return is larger. The expected return plays the central role in the agent's trustability and performance evaluation discussed in later chapters.

In chapter 3, we introduced the trustability taxonomy in RL algorithms. An RL agent is regarded as trustable if it:

1. acts safely, i.e., it does not perform risky or undesirable behavior,
2. behaves at least as good as the previously deployed policy.

This definition is one of the fundamental contributions of the dissertation. The two trustability aspects have been addressed separately in the RL community. We unify them and tie them together by introducing the RL agent's trustability concept. The issue of avoiding undesirable or risky actions has been studied under the term safe RL in the literature, and the background work on safe RL is brought in section 3.1. Providing performance guarantees for a new recommended policy prior to deployment is tackled under the term off-policy evaluation. The background work on off-policy evaluation is brought in section 3.2.

Chapter 4 focused on the off-policy deep RL algorithms' data efficiency. Leveraging chapter 2 discussion on various types of RL algorithms, this chapter presented the state of the art data efficiency techniques in deep Q-networks and deep actor-critic networks. Moreover, the architecture of the network used in the Trustable Deep RL methodology is presented in chapter 5.

This dissertation's most significant contribution is the Trustable Deep RL algorithm with efficient data utilization presented in chapter 6. This chapter began by explicitly listing characteristics of the target environments. It continued by mentioning all the implemented techniques to increase the deep RL agent's data efficiency, among which the weight initialization of the actor and the critic networks is the novel technique proposed in this dissertation. Another novelty of this methodology is to apply two sets of policy search constraints as risk barriers to ensure the safety of the deep network's policy. One constraint is advised by the expert, and the other one is derived from data.

The methodology is required to take one more step, beyond safety, to gain the users' trust, which is to estimate the new policy's performance prior to deployment. The algorithm estimates the new policy's expected return by implementing the WPDIS method introduced in chapter 3. The algorithm further provides confidence bounds on the estimated expected return values computed from a K-sized batch of episodes. This points to another contribution of the dissertation, which is utilizing expected return batch estimators with concentration inequalities. Although the proposed approach cannot bound single episodes returns, it well serves the purpose of comparing the previous policy with the new policy.

The training and evaluation steps of the Trustable Deep RL algorithm are presented in single pseudocode, algorithm 10. First, the algorithm efficiently trains an actor-critic network, then the network's output policy passes through the safety constraints. Lastly, the resulting policy would be accepted for implementation if its estimated expected return and lower confidence bound are larger than the previous policy.

Finally, At the end of chapter 6, we brought the Trustable Deep RL deployment algorithm, algorithm 11. This algorithm encompasses another contribution of this dissertation that is making advice for future data gathering, i.e., in unprecedented regions of the state space, the algorithm could provide suggestions on exploring the most promising actions.

Chapter 7 presents a test bench for optimizing the corrosion-related maintenance management in the dry gas pipeline. The test bench is another contribution of the dissertation. It is an effort to facilitate the development of RL methodologies in pipeline integrity management without risking the real system. The test bench is a physics-based corrosion degradation model that could interact with a decision-maker agent and adjust itself to the decision-maker's maintenance orders. In this dissertation, it offers a test environment for the Trustable Deep RL methodology.

Lastly, chapter 8 demonstrated and discussed the results of applying the data-efficient Trustable Deep RL methodology to the pipeline's corrosion maintenance management problem. This chapter assessed the data efficiency, safety, and accuracy of the off-policy evaluation. Besides, it presented the methodology's sensitivity to the behavior (data generation) policy's optimality. In summary, when the behavior policy is a relatively good (around optimal) policy, the Trustable Deep RL algorithm finds a superior policy that is not very different from the behavior policy; also the accuracy of the estimated performance and confidence bounds are satisfactory. In case of a bad behavior policy, the Trustable Deep RL algorithm still could find a superior policy; however, due to the large difference between the behavior and the new policy, the off-policy evaluation's accuracy is unsatisfactory.

9.1 Future Work

The methodology and test environment presented in this dissertation have highlighted several topics on which further research would be beneficial:

1. The test environment presented here is a simulated environment. Although we used that in an offline fashion similar to real systems, we cannot ignore that it is an emulator, intrinsically different from real-world problems. One direction for future work is to apply the presented methodology on real data sets, e.g., medical treatment policy. Specifically, we intend to implement our methodology on sepsis treatment policy because it is emerging as a benchmark for deep RL medical applications. We aim to leverage upon the MIMIC III database [JPS+16], which is a publicly available database and contains hospital admissions from approximately 38,600 adults.

2. In the proposed methodology, the actor-critic network is trained by batches of uniform random samples from the episode replay memory. Uniform sampling strategy underrates the informative and rich experiences that happen rarely. Hence, it would be beneficial to prioritize episodes and replay critical but rare episodes more frequently. To the best of the author's knowledge, there is no research on prioritizing episodes, unlike experience tuples that are well studied in the literature. Considering the increasing interest in off-policy, episodic RL, this would be a promising avenue of future research. Our suggestion is to prioritize episodes that have a higher number of rare transitions.

3. One of the assumptions in the Trustable Deep RL methodology is the agent's action space is discreet and finite. However, many applications, including robot control and medication dosage administration, have continuous action space. One direction for future work is to extend the methodology to problems with continuous action space. Since the proposed methodology classifies as a policy gradient approach, it is easily extendable to continuous action space.

4. Another restrictive assumption in the proposed Trustable Deep RL methodology is the environment must be fully observable because the underlying techniques are based on the Markov decision process assumption. The methodology's extension to partially observable environments would significantly enhance the proposed methodology's practicality to real environments. We believe the bottleneck would be to adjust the actor-critic network to partially observable environments. Our suggestion is to follow the recent work of Srinivasan et al. [SLZ$^+$18] and Nian et al. [NIR20] who aimed at modifying the actor-critic deep RL network to handle sequential decision making in complex partially observable problems.

5. The stationary assumption is also restrictive since many real sequential decision-making problems are in dynamic environments where the environment responds to the same set of decisions drift over time. One approach to extend the methodology to non-stationary environments is to frequently update the training set and retraining the networks with the most recent observations. The challenge in this approach is how to determine when is the best time to perform retraining. In other words, how we determine when the nature of the environment's responses are shifting.

Bibliography

[ACN10] Pieter Abbeel, Adam Coates, and Andrew Y Ng, *Autonomous helicopter aerobatics through apprenticeship learning*, The International Journal of Robotics Research **29** (2010), no. 13, 1608–1639.

[ALL$^+$09] John Asmuth, Lihong Li, Michael L Littman, Ali Nouri, and David Wingate, *A bayesian sampling approach to exploration in reinforcement learning*, Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence, 2009, pp. 19–26.

[Alt99] Eitan Altman, *Constrained markov decision processes*, Vol. 7, CRC Press, 1999.

[AMP$^+$10] Naoki Abe, Prem Melville, Cezar Pendus, Chandan K Reddy, David L Jensen, Vince P Thomas, James J Bennett, Gary F Anderson, Brent R Cooley, Melissa Kowalczyk, et al., *Optimizing debt collections using constrained reinforcement learning*, Proceedings of the 16th acm sigkdd international conference on knowledge discovery and data mining, 2010, pp. 75–84.

[AN04] Pieter Abbeel and Andrew Y Ng, *Apprenticeship learning via inverse reinforcement learning*, Proceedings of the twenty-first international conference on machine learning, 2004, pp. 1.

[AN09] ______, *Exploration and apprenticeship learning in reinforcement learning*, Proceedings of the 22nd international conference on machine learning, 2009, pp. 1–8.

[And69] Theodore Wilbur Anderson, *Confidence limits for the expected value of an arbitrary bounded random variable with a continuous distribution function*, STANFORD UNIV CA DEPT OF STATISTICS, 1969.

[BBB08] Arnab Basu, Tirthankar Bhattacharyya, and Vivek S Borkar, *A learning algorithm for risk-sensitive cost*, Mathematics of operations research **33** (2008), no. 4, 880–898.

[BCJS$^+$15] Reinaldo AC Bianchi, Luiz A Celiberto Jr, Paulo E Santos, Jackson P Matsuura, and Ramon Lopez de Mantaras, *Transferring knowledge as heuristics in reinforcement learning: A case-based approach*, Artificial Intelligence **226** (2015), 102–121.

[BI93] Leemon C Baird III, *Advantage updating*, WRIGHT LAB WRIGHT-PATTERSON AFB OH, 1993.

[BNT$^+$19] Ali Baheri, Subramanya Nageshrao, H Eric Tseng, Ilya Kolmanovsky, Anouck Girard, and Dimitar Filev, *Deep reinforcement learning with enhanced safety for autonomous highway driving*, arXiv preprint arXiv:1910.12905 (2019).

[BNVB13] 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** (2013), 253–279.

[Bor01] Vivek S Borkar, *A sensitivity formula for risk-sensitive cost and the actor–critic algorithm*, Systems & Control Letters **44** (2001), no. 5, 339–346.

[BSDS$^+$18] Reinaldo AC Bianchi, Paulo E Santos, Isaac J Da Silva, Luiz A Celiberto, and Ramon Lopez de Mantaras, *Heuristically accelerated reinforcement learning by means of case-based reasoning and transfer learning*, Journal of Intelligent & Robotic Systems **91** (2018), no. 2, 301–312.

[CB00] James Carpenter and John Bithell, *Bootstrap confidence intervals: when, which, what? a practical guide for medical statisticians*, Statistics in medicine **19** (2000), no. 9, 1141–1164.

[CH03] Suman Chakravorty and David Hyland, *Minimax reinforcement learning*, Aiaa guidance, navigation, and control conference and exhibit, 2003, pp. 5718.

[CJMDMB10] Luiz A Celiberto Jr, Jackson P Matsuura, Ramón López De Màntaras, and Reinaldo AC Bianchi, *Using transfer learning to speed-up reinforcement learning: a cased-based approach*, 2010 latin american robotics symposium and intelligent robotics meeting, 2010, pp. 55–60.

[Clo97] Jeffery A Clouse, *On integrating apprentice learning and reinforcement learning.* (1997).

[CNDGG18] Yinlam Chow, Ofir Nachum, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh, *A lyapunov-based approach to safe reinforcement learning*, Advances in neural information processing systems, 2018, pp. 8092–8101.

[CS87] Kun-Jen Chung and Matthew J Sobel, *Discounted mdp's: Distribution functions and exponential utility maximization*, SIAM journal on control and optimization **25** (1987), no. 1, 49–62.

[CW18] Kamil Ciosek and Shimon Whiteson, *Expected policy gradients*, Thirty-second aaai conference on artificial intelligence, 2018.

[DD04] Kurt Driessens and Sašo Džeroski, *Integrating guidance into relational reinforcement learning*, Machine Learning **57** (2004), no. 3, 271–304.

[DG04] Laurent Doyen and Olivier Gaudoin, *Classes of imperfect repair models based on reduction of failure intensity or virtual age*, Reliability Engineering & System Safety **84** (2004), no. 1, 45–56.

[DH97] Anthony Christopher Davison and David Victor Hinkley, *Bootstrap methods and their application*, Vol. 1, Cambridge university press, 1997.

[dL$^+$09] Javier de Lope et al., *Learning autonomous helicopter flight with evolutionary reinforcement learning*, International conference on computer aided systems theory, 2009, pp. 75–82.

[DRBM18] Will Dabney, Mark Rowland, Marc G Bellemare, and Rémi Munos, *Distributional reinforcement learning with quantile regression*, Thirty-second aaai conference on artificial intelligence, 2018.

[DWS12] Thomas Degris, Martha White, and Richard S Sutton, *Off-policy actor-critic*, arXiv preprint arXiv:1205.4839 (2012).

[DXWC19] Canhuang Dai, Liang Xiao, Xiaoyue Wan, and Ye Chen, *Reinforcement learning with safe exploration for network security*, Icassp 2019-2019 ieee international conference on acoustics, speech and signal processing (icassp), 2019, pp. 3057–3061.

[ESM$^+$18] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al., *Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures*, arXiv preprint arXiv:1802.01561 (2018).

[ET94] Bradley Efron and Robert J Tibshirani, *An introduction to the bootstrap*, CRC press, 1994.

[FF19] Christopher Frye and Ilya Feige, *Parenting: Safe reinforcement learning from human input*, arXiv preprint arXiv:1902.06766 (2019).

[GAF13] Javier Garcıa, Daniel Acera, and Fernando Fernández, *Safe reinforcement learning through probabilistic policy reuse*, RLDM 2013 (2013), 14.

[Gar94] Linda Garverick, *Corrosion in the petrochemical industry*, ASM international, 1994.

[Gas03] Chris Gaskett, *Reinforcement learning under circumstances beyond its control* (2003).

[GB10] Xavier Glorot and Yoshua Bengio, *Understanding the difficulty of training deep feedforward neural networks*, Proceedings of the thirteenth international conference on artificial intelligence and statistics, 2010, pp. 249–256.

[GBB04] Evan Greensmith, Peter L Bartlett, and Jonathan Baxter, *Variance reduction techniques for gradient estimates in reinforcement learning*, Journal of Machine Learning Research **5** (2004), no. Nov, 1471–1530.

[Ger12] Alborz Geramifard, *Practical reinforcement learning using representation learning and safe exploration for large scale markov decision processes*, Ph.D. Thesis, 2012.

[GF11] Javier Garcia and Fernando Fernandezez, *Safe reinforcement learning in high-risk tasks through policy improvement*, 2011 ieee symposium on adaptive dynamic programming and reinforcement learning (adprl), 2011, pp. 76–83.

[GF12] Javier Garcia and Fernando Fernández, *Safe exploration of state and action spaces in reinforcement learning*, Journal of Artificial Intelligence Research **45** (2012), 515–564.

[GF15] Javier Garcıa and Fernando Fernández, *A comprehensive survey on safe reinforcement learning*, Journal of Machine Learning Research **16** (2015), no. 1, 1437–1480.

[Gos09] Abhijit Gosavi, *Reinforcement learning for model building and variance-penalized control*, Proceedings of the 2009 winter simulation conference (wsc), 2009, pp. 373–379.

[GP13] Clement Gehring and Doina Precup, *Smart exploration in reinforcement learning using absolute temporal difference errors*, Proceedings of the 2013 international conference on autonomous agents and multi-agent systems, 2013, pp. 1037–1044.

[GRH13] Alborz Geramifard, Joshua Redding, and Jonathan P How, *Intelligent cooperative control architecture: a framework for performance improvement using safe learning*, Journal of Intelligent & Robotic Systems **72** (2013), no. 1, 83–103.

[GRRH11] Alborz Geramifard, Joshua Redding, Nicholas Roy, and Jonathan P How, *Uav cooperative control with stochastic risk models*, Proceedings of the 2011 american control conference, 2011, pp. 3393–3398.

[G$^+$19] Sahab Singh Gurjar et al., *In line inspection ili interval for cross country pipelines*, Spe oil and gas india conference and exhibition, 2019.

[GW05] Peter Geibel and Fritz Wysotzki, *Risk-sensitive reinforcement learning applied to control under constraints*, Journal of Artificial Intelligence Research **24** (2005), 81–108.

[HBIK95] Mance E Harmon, Leemon C Baird III, and A Harry Klopf, *Advantage updating applied to a differential game*, Advances in neural information processing systems, 1995, pp. 353–360.

[HBW19] Wenhui Huang, Francesco Braghin, and Zhuo Wang, *Learning to drive via apprenticeship learning and deep reinforcement learning*, 2019 ieee 31st international conference on tools with artificial intelligence (ictai), 2019, pp. 1536–1540.

[Heg94] Matthias Heger, *Consideration of risk in reinforcement learning*, Machine learning proceedings 1994, 1994, pp. 105–111.

[HM72] Ronald A Howard and James E Matheson, *Risk-sensitive markov decision processes*, Management science **18** (1972), no. 7, 356–369.

[HMVH$^+$18] 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*, Thirty-second aaai conference on artificial intelligence, 2018.

[HSN17] Josiah P Hanna, Peter Stone, and Scott Niekum, *Bootstrapping with models: Confidence intervals for off-policy evaluation*, Thirty-first aaai conference on artificial intelligence, 2017.

[HSSU08] Alexander Hans, Daniel Schneegaß, Anton Maximilian Schäfer, and Steffen Udluft, *Safe exploration for reinforcement learning.*, Esann, 2008, pp. 143–148.

[IGW18] Ehsan Imani, Eric Graves, and Martha White, *An off-policy policy gradient theorem using emphatic weightings*, Advances in neural information processing systems, 2018, pp. 96–106.

[Ins15] Future Life Institute, *Annual report*, 2015.

[Jia10] Jiming Jiang, *Large sample techniques for statistics*, Springer Science & Business Media, 2010.

[JL15] Nan Jiang and Lihong Li, *Doubly robust off-policy value evaluation for reinforcement learning*, arXiv preprint arXiv:1511.03722 (2015).

[JPS$^+$16] Alistair EW Johnson, Tom J Pollard, Lu Shen, H Lehman Li-wei, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark, *Mimic-iii, a freely accessible critical care database*, Scientific data **3** (2016), 160035.

[JWC98] Guofei Jiang, Cang-Pu Wu, and George Cybenko, *Minimax-based reinforcement learning with state aggregation*, Proceedings of the 37th ieee conference on decision and control (cat. no. 98ch36171), 1998, pp. 1236–1241.

[KS14] Alok Kulkarni and Sampada Sathe, *Healthcare applications of the internet of things: A review*, International Journal of Computer Science and Information Technologies **5** (2014), no. 5, 6229–6232.

[KW09] Rogier Koppejan and Shimon Whiteson, *Neuroevolutionary reinforcement learning for generalized helicopter control*, Proceedings of the 11th annual conference on genetic and evolutionary computation, 2009, pp. 145–152.

[KW11] ———, *Neuroevolutionary reinforcement learning for generalized control of simulated helicopters*, Evolutionary intelligence **4** (2011), no. 4, 219–241.

[Lar20] Kathy Riggs Larsen, *Protecting a pipeline when its coating has aged*, 2020.

[LCPR05] Edith LM Law, Melanie Coggan, Doina Precup, and Bohdana Ratitch, *Risk-directed exploration in reinforcement learning*, Planning and Learning in A Priori Unknown or Dynamic Domains **97** (2005).

[LHP+15] 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).

[Lin93] Long-Ji Lin, *Reinforcement learning for robots using neural networks*, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.

[LLG+15] Bo Liu, Ji Liu, Mohammad Ghavamzadeh, Sridhar Mahadevan, and Marek Petrik, *Finite-sample analysis of proximal gradient td algorithms.*, Uai, 2015, pp. 504–513.

[LLTZ18] Qiang Liu, Lihong Li, Ziyang Tang, and Dengyong Zhou, *Breaking the curse of horizon: Infinite-horizon off-policy estimation*, Advances in neural information processing systems, 2018, pp. 5356–5366.

[LWH19] Hepeng Li, Zhiqiang Wan, and Haibo He, *Constrained ev charging scheduling based on safe deep reinforcement learning*, IEEE Transactions on Smart Grid (2019).

[Mar52] Harry Markowitz, *Portfolio selection*, The journal of finance **7** (1952), no. 1, 77–91.

[Mas07] Pascal Massart, *Concentration inequalities and model selection*, Vol. 6, Springer, 2007.

[Mas90] ______, *The tight constant in the dvoretzky-kiefer-wolfowitz inequality*, The annals of Probability (1990), 1269–1283.

[MB05] Frederic Maire and Vadim Bulitko, *Apprenticeship learning for initial value functions in reinforcement learning*, Planning and Learning in A Priori Unknown or Dynamic Domains (2005), 23.

[MBM+16] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu, *Asynchronous methods for deep reinforcement learning*, International conference on machine learning, 2016, pp. 1928–1937.

[Mit13] Callie Mitchell, *Wooo – pig – sooie!" - the business of pipeline integrity ii*, 2013.

[MJL+18] Reza Mahjourian, Navdeep Jaitly, Nevena Lazic, Sergey Levine, and Risto Miikkulainen, *Hierarchical policy design for sample-efficient learning of robot table tennis through self-play*, arXiv preprint arXiv:1811.12927 (2018).

[MKS$^+$13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller, *Playing atari with deep reinforcement learning*, arXiv preprint arXiv:1312.5602 (2013).

[MKS$^+$15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al., *Human-level control through deep reinforcement learning*, Nature **518** (2015), no. 7540, 529.

[MN02] Oliver Mihatsch and Ralph Neuneier, *Risk-sensitive reinforcement learning*, Machine learning **49** (2002), no. 2-3, 267–290.

[MP09] Andreas Maurer and Massimiliano Pontil, *Empirical bernstein bounds and sample variance penalization*, arXiv preprint arXiv:0907.3740 (2009).

[MPMY$^+$14] Bahman Modiri, Mohammad Pourgol Mohammad, Mojtaba Yazdani, Farzad Nasirpouri, and Farzin Salehpour, *Piping anti–corrosion coating life assessment*, Asme international mechanical engineering congress and exposition, 2014, pp. V014T08A013.

[MSHB16] Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare, *Safe and efficient off-policy reinforcement learning*, Advances in neural information processing systems, 2016, pp. 1054–1062.

[MST$^+$05] Richard Maclin, Jude Shavlik, Lisa Torrey, Trevor Walker, and Edward Wild, *Giving advice about preferred actions to reinforcement learners via knowledge-based kernel regression*, Aaai, 2005, pp. 819–824.

[MSWT05] Richard Maclin, Jude Shavlik, Trevor Walker, and Lisa Torrey, *Knowledge-based support-vector regression for reinforcement learning*, Reasoning, Representation, and Learning in Computer Games (2005), 61.

[MT11] Shie Mannor and John Tsitsiklis, *Mean-variance optimization in markov decision processes*, arXiv preprint arXiv:1104.5601 (2011).

[MWM] Zahra Mahmoodzadeh, Keoyuan Wu, and Ali Mosleh, *Smart condition-based maintenance with reinforcement learning for dry gas pipeline subject to internal corrosion*, Under review for Sensors Journal.

[NBS10] Scott Niekum, Andrew G Barto, and Lee Spector, *Genetic programming for reward function search*, IEEE Transactions on Autonomous Mental Development **2** (2010), no. 2, 83–90.

[NCDL19] Ofir Nachum, Yinlam Chow, Bo Dai, and Lihong Li, *Dualdice: Behavior-agnostic estimation of discounted stationary distribution corrections*, Advances in neural information processing systems, 2019, pp. 2315–2325.

[NDK$^+$19] Ofir Nachum, Bo Dai, Ilya Kostrikov, Yinlam Chow, Lihong Li, and Dale Schuurmans, *Algaedice: Policy gradient from arbitrary experience*, arXiv preprint arXiv:1912.02074 (2019).

[New17] BBC News, *Google ai defeats human go champion*, 2017.

[New18] ———, *Ai finds novel way to beat classic q*bert atari video game*, 2018.

[NFK06] Yuriy Nevmyvaka, Yi Feng, and Michael Kearns, *Reinforcement learning for optimized trade execution*, Proceedings of the 23rd international conference on machine learning, 2006, pp. 673–680.

[NIR20] Xiaodong Nian, Athirai A Irissappane, and Diederik Roijers, *Dcrac: Deep conditioned recurrent actor-critic for multi-objective partially observable environments*, Proceedings of the 19th international conference on autonomous agents and multiagent systems, 2020, pp. 931–938.

[NR$^+$00] Andrew Y Ng, Stuart J Russell, et al., *Algorithms for inverse reinforcement learning.*, Icml, 2000, pp. 2.

[OBLB18] Ahmed Oussous, Fatima-Zahra Benjelloun, Ayoub Ait Lahcen, and Samir Belfkih, *Big data technologies: A survey*, Journal of King Saud University-Computer and Information Sciences **30** (2018), no. 4, 431–448.

[oMECCfPP91] American Society of Mechanical Engineers. Code Committee for Pressure Piping, *Manual for determining the remaining strength of corroded pipelines: a supplement to asme b31 code for pressure piping*, Ill., 1991.

[oPP18] The Canadian Association of Petroleum Producers, *Mitigation of internal corrosion in carbon steel gas pipeline systems*, 2018.

[PDR10] Sankara Papavinasam, Alex Doiron, and R Winston Revie, *Model to predict internal pitting corrosion of oil and gas pipelines*, Corrosion **66** (2010), no. 3, 035006–035006.

[PJP00] Roland Palmer-Jones and Dominic Paisley, *Repairing internal corrosion defects in pipelines-a case study*, International pipelines rehabilitation and maintenance conference, 4$^{\text{o}}$, 2000.

[Pre00] Doina Precup, *Eligibility traces for off-policy policy evaluation*, Computer Science Department Faculty Publication Series (2000), 80.

[PSB$^+$18] Hélène Plisnier, Denis Steckelmacher, Tim Brys, Diederik M Roijers, and Ann Nowé, *Directed policy gradient for safe reinforcement learning with human advice*, arXiv preprint arXiv:1808.04096 (2018).

[QVIRRGVR13] Pablo Quintía Vidal, Roberto Iglesias Rodríguez, Miguel Ángel Rodríguez González, and Carlos Vázquez Regueiro, *Learning on real robots from experience and simple user feedback* (2013).

[RKD$^+$18] Ramya Ramakrishnan, Ece Kamar, Debadeepta Dey, Julie Shah, and Eric Horvitz, *Discovering blind spots in reinforcement learning*, Proceedings of the 17th international conference on autonomous agents and multiagent systems, 2018, pp. 1017–1025.

[SB18] Richard S Sutton and Andrew G Barto, *Reinforcement learning: An introduction*, MIT press, 2018.

[SC11] Halit Bener Suay and Sonia Chernova, *Effect of human guidance and state space size on interactive reinforcement learning*, 2011 ro-man, 2011, pp. 1–6.

[SCMHL19] Jonathan Serrano-Cuevas, Eduardo F Morales, and Pablo Hernández-Leal, *Safe reinforcement learning using risk mapping by similarity*, Adaptive Behavior (2019), 1059712319859650.

[SHM$^+$16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al., *Mastering the game of go with deep neural networks and tree search*, nature **529** (2016), no. 7587, 484.

[Sin17] Amrinder Singh, *Internal coating-a must in gas pipelines*, ResearchGate preprint DOI: 10.13140/RG.2.2.13576.01283 (2017).

[SK00] William D Smart and Leslie Pack Kaelbling, *Practical reinforcement learning in continuous spaces*, Icml, 2000, pp. 903–910.

[SLA$^+$15] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz, *Trust region policy optimization*, International conference on machine learning, 2015, pp. 1889–1897.

[SLH$^+$14] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller, *Deterministic policy gradient algorithms*, 2014.

[SLZ$^+$18] Sriram Srinivasan, Marc Lanctot, Vinicius Zambaldi, Julien Pérolat, Karl Tuyls, Rémi Munos, and Michael Bowling, *Actor-critic policy optimization in partially observable multiagent environments*, Advances in neural information processing systems **31** (2018), 3422–3435.

[SML$^+$15] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel, *High-dimensional continuous control using generalized advantage estimation*, arXiv preprint arXiv:1506.02438 (2015).

[SMSM00] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour, *Policy gradient methods for reinforcement learning with function approximation*, Advances in neural information processing systems, 2000, pp. 1057–1063.

[SQAS15] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver, *Prioritized experience replay*, arXiv preprint arXiv:1511.05952 (2015).

[SS94] Pranab K Sen and Julio M Singer, *Large sample methods in statistics: an introduction with applications*, Vol. 25, CRC press, 1994.

[STK18] Elad Sarafian, Aviv Tamar, and Sarit Kraus, *Constrained policy improvement for safe and efficient reinforcement learning*, arXiv preprint arXiv:1805.07805 (2018).

[SYH$^+$18] Bradly C Stadie, Ge Yang, Rein Houthooft, Xi Chen, Yan Duan, Yuhuai Wu, Pieter Abbeel, and Ilya Sutskever, *Some considerations on learning to explore via meta-reinforcement learning*, arXiv preprint arXiv:1803.01118 (2018).

[Tah13] Hamdy A Taha, *Operations research: an introduction*, Pearson Education India, 2013.

[TB08] Andrea L Thomaz and Cynthia Breazeal, *Teachable robots: Understanding human teaching behavior to build more effective robot learners*, Artificial Intelligence **172** (2008), no. 6-7, 716–737.

[TB$^+$06] Andrea Lockerd Thomaz, Cynthia Breazeal, et al., *Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance*, Aaai, 2006, pp. 1000–1005.

[Tho15] Philip S Thomas, *Safe reinforcement learning*, Ph.D. Thesis, 2015.

[Tok10] Michel Tokic, *Adaptive ε-greedy exploration in reinforcement learning based on value differences*, Annual conference on artificial intelligence, 2010, pp. 203–210.

[TS09] Matthew E Taylor and Peter Stone, *Transfer learning for reinforcement learning domains: A survey*, Journal of Machine Learning Research **10** (2009), no. Jul, 1633–1685.

[TSL07] Matthew E Taylor, Peter Stone, and Yaxin Liu, *Transfer learning via inter-task mappings for temporal difference learning*, Journal of Machine Learning Research **8** (2007), no. Sep, 2125–2167.

[TTG15] Philip S Thomas, Georgios Theocharous, and Mohammad Ghavamzadeh, *High-confidence off-policy evaluation*, Twenty-ninth aaai conference on artificial intelligence, 2015.

[TWSM05] Lisa Torrey, Trevor Walker, Jude Shavlik, and Richard Maclin, *Using advice to transfer knowledge acquired in one reinforcement learning task to another*, European conference on machine learning, 2005, pp. 412–424.

[UJ19] Masatoshi Uehara and Nan Jiang, *Minimax weight and q-function learning for off-policy evaluation*, arXiv preprint arXiv:1910.12809 (2019).

[VHG11] Koen Hermans Jessica Vleugel, Michelle Hoogwout, and Imre Gelens, *Reinforcement learning with avoidance of unsafe regions*, BSc Project (2011).

[VHGS16] Hado Van Hasselt, Arthur Guez, and David Silver, *Deep reinforcement learning with double q-learning*, Thirtieth aaai conference on artificial intelligence, 2016.

[WAD17] Yu-Xiang Wang, Alekh Agarwal, and Miroslav Dudik, *Optimal and adaptive off-policy evaluation in contextual bandits*, Proceedings of the 34th international conference on machine learning-volume 70, 2017, pp. 3589–3597.

[Wat89] Christopher John Cornish Hellaby Watkins, *Learning from delayed rewards* (1989).

[Waw09] Paweł Wawrzyński, *Real-time reinforcement learning by sequential actor–critics and experience replay*, Neural Networks **22** (2009), no. 10, 1484–1497.

[WBH$^+$16] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas, *Sample efficient actor-critic with experience replay*, arXiv preprint arXiv:1611.01224 (2016).

[WC20] Shaomin Wu and Inma T Castro, *Maintenance policy for a system with a weighted linear combination of degradation processes*, European Journal of Operational Research **280** (2020), no. 1, 124–133.

[WHM11] Thomas J Walsh, Daniel K Hewlett, and Clayton T Morrison, *Blending autonomous exploration and apprenticeship learning*, Advances in neural information processing systems, 2011, pp. 2258–2266.

[Wie64] Norbert Wiener, *God and golem, inc: a comment on certain points where cybernetics impinges on religion*, Vol. 42, MIT press, 1964.

[Wil92] Ronald J Williams, *Simple statistical gradient-following algorithms for connectionist reinforcement learning*, Machine learning **8** (1992), no. 3-4, 229–256.

[WM19] Keo-Yuan Wu and Ali Mosleh, *Effect of temporal variability of operating parameters in corrosion modelling for natural gas pipelines subject to uniform corrosion*, Journal of Natural Gas Science and Engineering **69** (2019), 102930.

[WMMY11] Ronald E Walpole, Raymond H Myers, Sharon L Myers, and Keying Ye, *Probability & statistics for engineers & scientists*, Pearson Prentice Hall, 2011.

[WSH$^+$16] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas, *Dueling network architectures for deep reinforcement learning*, arXiv preprint arXiv:1511.06581 (2016).

[WT01] Lex Weaver and Nigel Tao, *The optimal reward baseline for gradient-based reinforcement learning*, Proceedings of the seventeenth conference on uncertainty in artificial intelligence, 2001, pp. 538–545.

[YHL$^+$17] Chaowei Yang, Qunying Huang, Zhenlong Li, Kai Liu, and Fei Hu, *Big data and cloud computing: innovation opportunities and challenges*, International Journal of Digital Earth **10** (2017), no. 1, 13–53.

[YKTA17] Yongsheng Yang, Faisal Khan, Premkumar Thodi, and Rouzbeh Abbassi, *Corrosion induced failure analysis of subsea pipelines*, Reliability Engineering & System Safety **159** (2017), 214–222.

[YSKS15] M Yadav, RR Sinha, Sumit Kumar, and TK Sarkar, *Corrosion inhibition effect of spiropyrimidinethiones on mild steel in 15% hcl solution: insight from electrochemical and quantum studies*, RSC advances **5** (2015), no. 87, 70832–70848.

[ZBL19] Yinan Zhang, Devin Balkcom, and Haoxiang Li, *Towards physically safe reinforcement learning under supervision*, arXiv preprint arXiv:1901.06576 (2019).

[ZBW19] Shangtong Zhang, Wendelin Boehmer, and Shimon Whiteson, *Generalized off-policy actor-critic*, Advances in neural information processing systems, 2019, pp. 1999–2009.

[Zho10] Wenxing Zhou, *System reliability of corroding pipelines*, International Journal of Pressure Vessels and Piping **87** (2010), no. 10, 587–595.

[ZKZ09] Yufan Zhao, Michael R Kosorok, and Donglin Zeng, *Reinforcement learning design for cancer clinical trials*, Statistics in medicine **28** (2009), no. 26, 3294–3315.

[ZZ13] Shenwei Zhang and Wenxing Zhou, *System reliability of corroding pipelines considering stochastic process-based models for defect growth and internal pressure*, International Journal of Pressure Vessels and Piping **111** (2013), 120–130.

[ZZZ$^+$18] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li, *Drn: A deep reinforcement learning framework for news recommendation*, Proceedings of the 2018 world wide web conference, 2018, pp. 167–176.