Master's Thesis

Reinforcement Learning: Theoretical Perspectives and Applications in Continual Learning

Tomasz Arczewski and Tadeusz Dziarmaga

Supervised by dr. hab. Marcin Mazur

Faculty of Mathematics and Computer Science
Jagiellonian University

Kraków, 2024

Introduction

Reinforcement learning (RL) is a powerful framework for modeling and solving decision-making problems where an agent interacts with a dynamic environment to maximize a cumulative reward over time. At the core of RL is the idea of learning the optimal strategy, which the agent learns by trial and error – this means starting from random behavior and interatively improving decisions using feedback from the environment. This procedure is typically formalized through the use of Markov Decision Processes (MDPs), which provide a mathematical framework for modeling the agent-environment interaction and the sequential nature of decision-making tasks.

The objective of this thesis is to present an extensive theoretical analysis of reinforcement learning across various contexts. The main sources on which we have based the theoretical part of our work are due to Sutton and Barto (2018), Lattimore and Szepesvári (2017), Szepesvári (2020), and Szepesvári (2022). This is complemented by empirical studies focused on continual reinforcement learning (CRL), which is a domain where the agent encounters a sequence of tasks rather than a single, isolated one. All experiments conducted in CRL represent our original work and contribution of this thesis. The source code is available at https://github.com/Teddy298/continualworld-ppo.

In Section 1, we begin by establishing the foundational concepts of RL, starting with a definition of infinite-horizon Markov Decision Processes. In this common MDP setting, learning continues indefinitely, and immediate rewards are generally prioritized over future ones. Following this, we explore the core tools and methods for solving infinite-horizon MDPs, with an emphasis on fundamental algorithms such as Value Iteration and General Policy Iteration. We provide formal proofs of convergence for these algorithms, demonstrating their role in finding optimal policies. We also explain the use of direct methods and Monte Carlo methods for evaluating policies. Additionally, we examine the differences between on-policy and off-policy algorithms.

In Section 2, we analyze how function approximation can be used for large and complex environments, for which fundamental approaches can be intractable. We introduce neural networks and gradient descent, which are key components for function approximation in reinforcement learning. These techniques form the foundation of deep reinforcement learning, where deep neural networks are used to largely improve the capacity of RL algorithms for high-dimensional tasks.

Section 3 delves into a theoretical analysis of the Q-learning algorithm, a widely used approach in reinforcement learning for solving infinite-horizon MDPs. First, we focus on the tabular version of Q-learning, exploring its convergence properties and the conditions under which it achieves optimality. Additionally, we examine Deep Q-Networks (DQN), which extend Q-learning by incorporating function approximation in the form of deep neural network to handle more complex environments.

Section 4 shifts our attention to policy gradient methods, which represent a different approach to learning optimal policies. Unlike previous methods that focus on estimating the value of actions or states to derive an optimal policy indirectly, policy gradient methods aim to directly optimize the policy itself, and in this thesis, we investigate fundamental components such as the Policy Gradient Theorem, REINFORCE algorithm, and actor-critic methods.

In subsequent Section 5, we revisit Value Iteration and General Policy Iteration in scenarios where taking the limit to infinity is computationally infeasible. Here, we analyze the performance of these algorithms after a finite number of iterations, providing insights into their convergence rates, error bounds, and computational complexities.

In Section 6, we explore multi-armed bandits, a special case of MDPs with a single state, allowing for a more granular analysis of exploration-exploitation trade-offs. In this section, we analyze classical algorithms such as Explore-Then-Commit (ETC) and Upper Confidence Bound (UCB), focusing on regret minimization, a key performance metric in bandit problems.

Section 7 extends the analysis to finite-horizon MDPs, where decisions are made over a limited number of time steps or stages, known as the horizon. In this context, we focus on analyzing online learning algorithms, where the algorithm operates in an unknown environment and learns about it by actively collecting data through interaction. We define this setting rigorously and provide performance bounds for specific algorithms. For the main result in this section (Theorem 7.8), based on Szepesvári (2020), our contribution includes correcting some of the constants in the concentration inequalities and clarifying several steps that were not explained in enough detail.

Finally, in Section 8, we present the original experimental contribution of this thesis in the domain of continual reinforcement learning. These experiments extend the Continual World benchmark (Wołczyk et al., 2021) with Proximal Policy Optimization (PPO) algorithm carefully combined with continual learning methods, which involved a high level of commitment. While the experiments in Wołczyk et al. (2021) were conducted with an algorithm called Soft Actor-Critic (SAC), our goal was to investigate whether the CRL methods can be successfully adapted to the different family of methods to which PPO belongs. The code for the experiments was implemented by us in Python, using Tensorflow 2 framework.

Contents

Contribution

The contributions to the sections of this thesis are as follows:

0. **Introduction** – T. Arczewski, T. Dziarmaga (equal contribution),
1. **Introduction to Reinforcement Learning** – T. Arczewski,
2. **Function Approximation** – T. Arczewski,
3. **Q-learning and Deep Q-network** – T. Dziarmaga,
4. **Policy Gradient** – T. Dziarmaga,
5. **Computational Complexity of RL Algorithms** – T. Dziarmaga,
6. **Multi-Armed Bandits** – T. Arczewski,
7. **Finite-Horizon MDPs** – T. Dziarmaga,
8. **Continual Reinforcement Learning** – T. Arczewski, T. Dziarmaga (equal contribution).

Acknowledgements

We thank dr. Maciej Wołczyk for his guidance and support in the Continual Reinforcement Learning project. We gratefully acknowledge Polish high-performance computing infrastructure PLGrid (HPC Center: ACK Cyfronet AGH) for providing computer facilities and support within computational grant no. PLG/2024/017145.

1 Introduction to Reinforcement Learning

Reinforcement learning (RL) operates on the principle of trial and error, where an agent learns to make optimal decisions by interacting with an environment (Busoniu et al., 2017; Sutton and Barto, 2018; Szepesvári, 2022). A Markov Decision Process (MDP) is a suitable mathematical model for reinforcement learning, effectively representing sequential decision-making problems where an agent interacts with the environment over discrete time steps. In the MDP, the environment is represented as a set of states, and the agent can take actions that transition it from one state to another, receiving rewards as feedback for its actions. The key aspect of MDPs is the Markov property, which states that the future state of the system depends only on the current state and the action taken, not on the history of states and actions that led to the current state.

In this section we consider an infinite-horizon MDP, in which the decision-making process continues indefinitely. From now on, we will refer to this as the MDP for simplicity. However, in Section 7, we will shift our focus to the finite-horizon MDP, where the decision-making process has a finite number of time steps. The content in this section builds on the foundational principles outlined by Sutton and Barto (2018), Agarwal et al. (2019), Szepesvári (2020), and Szepesvári (2022).

More formally we define Markov Decision Process as in (Agarwal et al., 2019).

Definition 1.1 (Markov Decision Process)**.** *A tuple $M = (\mathcal{S}, \mathcal{A}, P, R, \gamma, \mu)$ is a MDP if the following conditions hold.*

- *$\mathcal{S}$ is a state space. We will assume it is finite unless stated otherwise.*
- *$\mathcal{A}$ is an action space. Similarly to the state space, we will assume it is finite unless stated otherwise.*
- *$P : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ is the transition function, where by $\mathcal{P}(\mathcal{S})$ we denote the space of probability distributions over S. $P(s'|s, a)$ is the probability of transition into state s' when taking action a in state s.*
- *For every state-action pair (s, a), $r(s, a)$ is a random variable which indicates the immediate reward obtained when taking action a in state s. For every state-action pair (s, a) we will assume that this random variable has finite expected value $\mathbb{E}[r(s, a)] = R(s, a) < \infty$.*
- *$\gamma \in [0, 1)$ is a discount factor.*
- *$\mu \in \mathcal{P}(\mathcal{S})$ is the initial state distribution. It determines the distribution from which the initial state is generated.*

We typically use the term "finite MDP" for Markov Decision Processes with finite state and action spaces, which is the assumption we usually rely on. In a given MDP the agent interacts with the environment starting at some initial state s_0 drawn from μ. Then, at each time step $t = 0, 1, 2, \ldots$, the agent takes an action $a_t \in A$, obtains the immediate reward $r(s_t, a_t)$ and observes the next state $s_{t+1} \sim P(\cdot|s_t, a_t)$, where by $\sim$ we denote sampling from a distribution.

1.1 Episodic and Continuing Tasks Scenarios

In some of the practical algorithms we will consider two kinds of tasks scenarios, episodic and continuing. In episodic case the agent-environment interaction divides into subsequences called episodes, such as a single play of a game. At the conclusion of every episode, there may be one or more states designated as "terminal states". Upon reaching any of these terminal states, the system resets to some initial state to begin a new episode. In continuing case there are no terminal states, and the agent interaction with the environment continues without limit.

1.2 Basic Reinforcement Learning Concepts

In the context of reinforcement learning, the agent requires a strategy to determine its actions within the environment. This strategy is formally defined as a policy.

A policy is represented as a function, denoted by π, which takes the current state of the environment, denoted by s, and returns a distribution over possible actions that the agent can take in that state. Formally, this can be expressed as $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$, where $\mathcal{P}(\mathcal{A})$ denotes the set of probability distributions over actions. We will denote the set of all possible policies as Π.

There are different types of policies based on their behavior in selecting actions. One type are deterministic policies, which for every state s, directly select a single action with probability 1. This simplifies the representation of the policy as it can be treated as a direct mapping from set of all states to set set of all actions.

Another type are stochastic policies, which for each state s return a distribution over actions. The agent's decision-making process involves sampling from this distribution, introducing randomness into its actions. Stochastic policies are particularly useful in situations where exploration of different actions is necessary to learn about the environment or to handle uncertainty.

Another key concept in reinforcement learning is a value function. It serves as a fundamental tool for estimating the expected cumulative reward when an agent is in a particular state and follows a certain policy. It essentially quantifies the long-term desirability of states within an environment. This is formalized in the following definition.

Definition 1.2 (Value function). *We define value function $V^\pi : \mathcal{S} \to \mathbb{R}$ as*

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \middle| s_0 = s, \forall i \in \mathbb{N} : a_i \sim \pi(s_i)\right].$$

Optimal value function is a function $V^* : \mathcal{S} \to \mathbb{R}$, such that:

$$V^*(s) = \sup\left\{V^\pi(s) : \pi \in \Pi\right\}.$$

In reinforcement learning, we are interested in finding optimal policy, which is any policy, whose value function is equal to the optimal value function.

Definition 1.3 (Optimal policy). *Policy π is optimal if its value function is equal to the optimal value function, that is:*

$$\forall s \in \mathcal{S} : V^\pi(s) = V^*(s)$$

In the remaining part of this section, we show that at least one such policy exists.

The action-value function, often denoted as $Q^\pi(s, a)$, extends the concept of the value function by incorporating the notion of taking a specific action a initially. It estimates the expected cumulative reward starting from a particular state s, taking action a, and then following a certain policy π.

Definition 1.4 (Action-value function). *For given policy π, we define action-value function of that policy as:*

$$Q^\pi(s,a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t,a_t) \middle| s_0 = s, a_0 = a, \forall i \in \mathbb{N}_+ : a_i \sim \pi(s_i)\right].$$

Remark 1.1. *We have:*

$$\begin{aligned}
Q^\pi(s,a) &= \mathbb{E}\left[R(s,a) + \sum_{t=1}^{\infty} \gamma^t r(s_t,a_t) \middle| \begin{array}{l} s_0 = s, a_0 = a, \\ \forall i \in \mathbb{N}_+ : a_i \sim \pi(s_i) \end{array}\right] \\
&= \mathbb{E}\left[R(s,a) + V^\pi(s_1) \middle| \begin{array}{l} s_0 = s, a_0 = a, \\ \forall i \in \mathbb{Z}_+ : a_i \sim \pi(s_i) \end{array}\right] \\
&= R(s,a) + \sum_{s' \in \mathcal{S}} P(s' \mid s,a) V^\pi(s')
\end{aligned}$$

Remark 1.2. *We have:*

$$\begin{aligned}
V^\pi(s) &= \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t,a_t) \middle| \begin{array}{l} s_0 = s, \\ \forall i \in \mathbb{N} : a_i \sim \pi(s_i) \end{array}\right] \\
&= \sum_{a \in \mathcal{A}} \pi(a \mid s) \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t,a_t) \middle| \begin{array}{l} s_0 = s, a_0 = a, \\ \forall i \in \mathbb{N}_+ : a_i \sim \pi(s_i) \end{array}\right] \\
&= \sum_{a \in \mathcal{A}} \pi(a \mid s_0) Q^\pi(s,a)
\end{aligned}$$

Next, we define the Bellman operators, which serve as essential tools in reinforcement learning. These operators play a crucial role in dynamic programming algorithms, enabling agents to iteratively update their value estimates and make optimal decisions over time.

Definition 1.5 (Expected Bellman operator). *Let $V : \mathcal{S} \to \mathbb{R}$ be a function and $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ be a policy. We define expected Bellman operator $\mathcal{T}_\pi$ as follows:*

$$(\mathcal{T}_\pi V)(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) V(s')\right).$$

Definition 1.6 (Bellman operator). *Let $V : \mathcal{S} \to \mathbb{R}$ be a function. We define Bellman operator $\mathcal{T}$ as follows:*

$$\begin{aligned}
(\mathcal{T}V)(s) &= \max_{a \in \mathcal{A}} \left\{R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) V(s')\right\} \\
&= \max_{a \in \mathcal{A}} \left\{R(s,a) + \gamma \mathbb{E}\left[V(s') \mid s' \sim P(\cdot \mid s,a)\right]\right\}
\end{aligned}$$

Remark 1.3. *We have:*

$$\sup_{\pi \in \Pi} \mathcal{T}_\pi = \mathcal{T}$$

Proof. Obviously for any policy π, any function $V : \mathcal{S} \to \mathbb{R}$ and any state $s \in \mathcal{S}$, we have $(\mathcal{T}V)(s) \geq (\mathcal{T}_\pi V)(s)$. Hence we have:

$$\sup_{\pi \in \Pi} \mathcal{T}_\pi \leq \mathcal{T}.$$

Now if we take a policy π which for given function V and given state s with probability 1 takes action $a = \arg\max_{a\in\mathcal{A}} \left\{R(s,a) + \gamma \sum_{s'\in\mathcal{S}} P\left(s' \mid s,a\right) V\left(s'\right)\right\}$, we obtain that $\mathcal{T}_\pi = \mathcal{T}$, hence:

$$\sup_{\pi\in\Pi} \mathcal{T}_\pi \geq \mathcal{T},$$

and we reach the desired conclusion. □

We can prove that both expected Bellman operator and Bellman operator are contractions in $\|\cdot\|_\infty$, which by Banach fixed-point theorem (Banach, 1922) proves the existence of the fixed point both for $\mathcal{T}$, and for $\mathcal{T}_\pi$ for any policy π. Soon we will see that the fixed point of operator $\mathcal{T}$ is the optimal value function.

Lemma 1.4. *For any policy $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$, expected Bellman operator $\mathcal{T}_\pi$ is a γ-contraction in $\|\cdot\|_\infty$, i.e. for any $V_1 : \mathcal{S} \to \mathbb{R}, V_2 : \mathcal{S} \to \mathbb{R}$, it holds that $\|\mathcal{T}_\pi V_1 - \mathcal{T}_\pi V_2\|_\infty \leq \gamma\|V_1 - V_2\|_\infty$.*

Proof. Let $V_1 : \mathcal{S} \to \mathbb{R}, V_2 : \mathcal{S} \to \mathbb{R}$ be any functions. We start by analyzing the absolute difference between $\mathcal{T}_\pi V_1$ and $\mathcal{T}_\pi V_2$ at a particular state $s \in \mathcal{S}$.

$$\begin{aligned}
&|(\mathcal{T}_\pi V_1)(s) - (\mathcal{T}_\pi V_2)(s)| \\
&= \left|\sum_{a\in\mathcal{A}} \pi(a \mid s) \left[R(s,a) + \gamma \sum_{s'\in\mathcal{S}} P\left(s' \mid s,a\right) V_1(s') - \right.\right. \\
&\qquad\qquad \left.\left. -R(s,a) - \gamma \sum_{s'\in\mathcal{S}} P\left(s' \mid s,a\right) V_2(s')\right]\right| \\
&= \gamma \sum_{a\in\mathcal{A}} \pi(a \mid s) \sum_{s'\in\mathcal{S}} P\left(s' \mid s,a\right) |V_1(s') - V_2(s')| \\
&\leq \gamma \sum_{a\in\mathcal{A}} \pi(a \mid s) \sum_{s'\in\mathcal{S}} P\left(s' \mid s,a\right) \|V_1 - V_2\|_\infty \\
&= \gamma \left\|V_1 - V_2\right\|_\infty,
\end{aligned}$$

Now, using the above inequality, we can bound $\|\mathcal{T}_\pi V_1 - \mathcal{T}_\pi V_2\|_\infty$:

$$\begin{aligned}
\|\mathcal{T}_\pi V_1 - \mathcal{T}_\pi V_2\|_\infty &= \max_{s\in\mathcal{S}} \left\{|(\mathcal{T}_\pi V_1)(s) - (\mathcal{T}_\pi V_2)(s)|\right\} \\
&\leq \gamma \left\|V_1 - V_2\right\|_\infty,
\end{aligned}$$

which concludes the proof. □

Lemma 1.5. *Bellman operator $\mathcal{T}$ is a γ-contraction in $\|\cdot\|_\infty$, i.e. for any $V_1 : \mathcal{S} \to \mathbb{R}, V_2 : \mathcal{S} \to \mathbb{R}$, it holds that $\|\mathcal{T}V_1 - \mathcal{T}V_2\|_\infty \leq \gamma\|V_1 - V_2\|_\infty$.*

Proof. Let $V_1 : \mathcal{S} \to \mathbb{R}$ and $V_2 : \mathcal{S} \to \mathbb{R}$ be any two functions. We begin by analyzing the absolute difference between $\mathcal{T}V_1$ and $\mathcal{T}V_2$ at an arbitrary state $s \in \mathcal{S}$. Specifically, we consider the difference between the maximum values that define the Bellman operator for each function.

$$\begin{aligned}
&|(\mathcal{T}V_1)(s) - (\mathcal{T}V_2)(s)| = \\
&= \left| \max_{a\in\mathcal{A}} \left\{ R(s,a) + \gamma \sum_{s'\in\mathcal{S}} P(s' \mid s,a) V_1(s') \right\} - \right. \\
&\qquad\qquad \left. - \max_{a'\in\mathcal{A}} \left\{ R(s,a') + \gamma \sum_{s'\in\mathcal{S}} P(s' \mid s,a') V_2(s') \right\} \right| \\
&\le \max_{a\in\mathcal{A}} \left| \left\{ \left(R(s,a) + \gamma \sum_{s'\in\mathcal{S}} P(s' \mid s,a) V_1(s') \right) - \right. \right. \\
&\qquad\qquad \left. \left. - \left(R(s,a) + \gamma \sum_{s'\in\mathcal{S}} P(s' \mid s,a) V_2(s') \right) \right\} \right| \\
&= \gamma \max_{a\in\mathcal{A}} \left| \sum_{s'\in\mathcal{S}} P(s' \mid s,a) \left(V_1(s') - V_2(s') \right) \right| \\
&\le \gamma \max_{a\in\mathcal{A}} \left| \sum_{s'\in\mathcal{S}} P(s' \mid s,a) \left\| V_1 - V_2 \right\|_\infty \right| \\
&= \gamma \max_{a\in\mathcal{A}} \left| \left\| V_1 - V_2 \right\|_\infty \right| \\
&= \gamma \left\| V_1 - V_2 \right\|_\infty .
\end{aligned}$$

Now, using the above inequality, we can bound $\|\mathcal{T}V_1 - \mathcal{T}V_2\|_\infty$:

$$\begin{aligned}
&\left\| \mathcal{T}V_1 - \mathcal{T}V_2 \right\|_\infty = \\
&= \max_{s\in\mathcal{S}} \left\{ |(\mathcal{T}V_1)(s) - (\mathcal{T}V_2)(s)| \right\} \\
&\le \gamma \left\| V_1 - V_2 \right\|_\infty ,
\end{aligned}$$

which concludes the proof. □

Theorem 1.6. *Let π be any policy. Then:*

1. $\lim_{n\to\infty} \|\mathcal{T}_\pi^n V - V^\pi\|_\infty = 0$ *for any $V : \mathcal{S} \to \mathbb{R}$, where $V^\pi : \mathcal{S} \to \mathbb{R}$ is a unique function, such that: $\mathcal{T}_\pi V^\pi = V^\pi$.*
2. $\lim_{n\to\infty} \|\mathcal{T}^n V - V^*\|_\infty = 0$ *for any $V : \mathcal{S} \to \mathbb{R}$, where $V^* : \mathcal{S} \to \mathbb{R}$ is a unique function, such that: $\mathcal{T}V^* = V^*$.*

Proof. The proof follows immediately from Lemma 1.4, Lemma 1.5 and Banach fixed-point theorem. □

We can now prove that that the stationary point of the operator $\mathcal{T}^\pi$ is equal to the value function V^π of the policy π.

Proposition 1.7. *Let π be any policy and V be a stationary point of the expected Bellman operator $\mathcal{T}^\pi$. Then we have $V = V^\pi$.*

Proof. Let $s \in S$ be any state. Then we have:

$$\begin{aligned}
V^\pi(s) &= \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \middle| s_0 = s, \forall i \in \mathbb{N} : a_i \sim \pi\left(s_t\right)\right] \\
&= \mathbb{E}\left[r(s_0, a_0) | s_0 = s, a_0 \sim \pi\left(s_0\right)\right] + \\
&\quad + \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^t r(s_t, a_t) \middle| s_0 = s, \forall i \in \mathbb{N} : a_i \sim \pi\left(s_t\right)\right] \\
&= \sum_{a \in \mathcal{A}} \pi(a \mid s) \Bigg(R(s,a) + \\
&\quad + \gamma \sum_{s_1 \in \mathcal{S}} P\left(s_1 \mid s, a\right) \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_{t+1}, a_{t+1}) \middle| \forall i \in \mathbb{N}_+ : a_i \sim \pi\left(s_t\right)\right]\Bigg) \\
&= \sum_{a \in \mathcal{A}} \pi(a \mid s) \left(R(s,a) + \gamma \sum_{s_1 \in \mathcal{S}} P\left(s_1 \mid s, a\right) V_\pi(s_1) \right) \\
&= (T_\pi V^\pi)(s)
\end{aligned}$$

Thus V^π is a stationary point of $\mathcal{T}_\pi$ which ends the proof. □

We can show that both expected Bellman operator and Bellman operator are monotonic.

Proposition 1.8 (Monotonicity of Bellman operators). *For any policy π and any $U, V : \mathcal{S} \to \mathbb{R}$, such that $U \leq V$, we have: $\mathcal{T}_\pi U \leq \mathcal{T}_\pi V$ and $\mathcal{T} U \leq \mathcal{T} V$.*

Proof. It follows that:

$$\mathcal{T}_\pi V - \mathcal{T}_\pi U = \sum_{a \in \mathcal{A}} \pi(a \mid s) \gamma \left(\sum_{s' \in \mathcal{S}} P\left(s' \mid s, a\right) \left(V\left(s'\right) - U\left(s'\right)\right) \right) \geq 0,$$

which ends the proof of the fist part of the proposition. Now let s be any state and a^* be an action for which:

$$\mathcal{T} U(s) = R(s, a^*) + \gamma \sum_{s' \in \mathcal{S}} P\left(s' \mid s, a^*\right) V\left(s'\right)$$

Such action exists, since $\mathcal{A}$ is finite. Then we have for any state $s \in \mathcal{S}$:

$$(\mathcal{T} U)(s) = R(s, a^*) + \gamma \sum_{s' \in \mathcal{S}} P\left(s' \mid s, a^*\right) U\left(s'\right) \leq R(s, a^*) + \gamma \sum_{s' \in \mathcal{S}} P\left(s' \mid s, a^*\right) V\left(s'\right)$$

$$\leq \max_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P\left(s' \mid s, a\right) V\left(s'\right) \right\} = (\mathcal{T} V)(s),$$

which ends the proof of the second part of the proposition. □

Definition 1.7 (Greedy policy). *A policy π is greedy with respect to a function $V : \mathcal{S} \to \mathbb{R}$, if for every states $s \in \mathcal{S}$, π chooses some action a that maximizes:*

$$R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a) V(s')$$

with probability one.

Lemma 1.9. *Let policy π be greedy with respect to $V : \mathcal{S} \to \mathbb{R}$. Then:*

$$\mathcal{T}_\pi V = \mathcal{T} V$$

Proof. Let $V : \mathcal{S} \to \mathbb{R}$ be a function and π be a policy that is greedy with respect do V. Then we have:

$$\begin{aligned}
\mathcal{T}_\pi V &= \sum_{a \in \mathcal{A}} \pi(a \mid s) \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P\left(s' \mid s, a\right) V\left(s'\right) \right) \\
&= \max_{a \in \mathcal{A}} \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P\left(s' \mid s, a\right) V\left(s'\right) \right) \\
&= \mathcal{T} V.
\end{aligned}$$

□

Lemma 1.10. *Any policy π that is greedy with respect to V^* is optimal. This means that for all $s \in \mathcal{S}$:*

$$V^\pi(s) = V^*(s).$$

Proof. Let $s \in \mathcal{S}$. We know, by the definition of V^*, that $V^\pi(s) \le V^*(s)$. To prove inequality in opposite direction we will first show that $V^*(s) \le \left(\mathcal{T} V^*\right)(s)$. Applying operator $\mathcal{T}_\pi$ to both sides of $V^\pi(s) \le V^*(s)$, using the monotonicity of expected Bellman operator from Proposition 1.8 and using $\mathcal{T}_\pi V^\pi = V^\pi$ from Proposition 1.7, we get:

$$V^\pi(s) = (\mathcal{T}_\pi V^\pi)(s) \le (\mathcal{T}_\pi V^*)(s).$$

Now, by taking supremum of both sides over $\pi \in \Pi$, we obtain:

$$V^*(s) = \sup_{\pi \in \Pi} V^\pi(s) \le \sup_{\pi \in \Pi} (\mathcal{T}_\pi V^*)(s) = \left(\mathcal{T} V^*\right)(s),$$

where the last equality follows from Remark 1.3. Above inequality with Lemma 1.9 yields:

$$V^*(s) \le \left(\mathcal{T} V^*\right)(s) = \left(\mathcal{T}_\pi V^*\right)(s) \tag{1}$$

We can apply $\mathcal{T}^\pi$ and Proposition 1.8 to Equation (1) to get:

$$V^*(s) \le (\mathcal{T}_\pi V^*)(s) \le \left(\mathcal{T}_\pi \left(\mathcal{T}_\pi V^*\right)\right)(s) = \left(\mathcal{T}_\pi^2 V^*\right)(s).$$

Repeating this procedure iteratively we obtain the following inequality for any $n \in \mathbb{N}$:

$$V^*(s) \le \left(\mathcal{T}_\pi^n V^*\right).$$

We know from Theorem 1.6 that $\mathcal{T}_\pi^n V$ converges to V^π in $\|\cdot\|_\infty$ for any function $V : \mathcal{S} \to \mathbb{R}$, so by taking limits on both sides in above inequality we get:

$$V^*(s) \le V^\pi(s).$$

Thus $V^\pi(s) = V^*(s)$. □

The simple conclusion from Lemma 1.10 is the existence of deterministic optimal policy, as one is the greedy policy with respect to V^*.

Corollary 1.11. *In any MDP there exists a deterministic optimal policy.*

Theorem 1.12. *It holds that $\mathcal{T} V^* = V^*$.*

Proof. Let π be any policy that is greedy with respect to V^*. Then by Lemma 1.9 and Lemma 1.10:

$$\mathcal{T} V^* = \mathcal{T}_\pi V^* = \mathcal{T}_\pi V^\pi = V^\pi = V^*.$$

□

1.3 Value Iteration

We showed that fixed point of an operator $\mathcal{T}$ is in fact the optimal value function. Using the results from Theorem 1.6 and Lemma 1.10 we can construct a simple meta-algorithm which finds optimal policy. The algorithm calculates the limit of $\mathcal{T}^n V$ for any function $V : \mathcal{S} \to \mathbb{R}$. Then we take a greedy policy with respect to the result. From Theorem 1.6 and Theorem 1.12 we know that such a limit exists and is the optimal value function. From Lemma 1.10 we know that the greedy policy with respect to this result must be the optimal policy. The above procedure is summarized in Algorithm 1.

Algorithm 1 Value Iteration (VI)

```
Require: $V : \mathcal{S} \to \mathbb{R}$
  $V^* \leftarrow \lim_{n\to\infty} \mathcal{T}^n V$
  $\pi^* \leftarrow greedy(V^*)$
  return $\pi*$
```

In practice, when we are not able to directly calculate limit in Algorithm 1, we can iterate over $\mathcal{T}$ finitely many times. From Banach Fixed-Point Theorem, we know that $\mathcal{T}^n V$ approaches V^* exponentially fast. In Section 5 we will analyse such approach theoretically in depth.

1.4 Policy Iteration

Given any policy π, Theorem 1.6 guarantees existence of a fixed point $V^\pi : \mathcal{T}_\pi V^\pi = V^\pi$, as well as convergence to V^π by applying operator $\mathcal{T}_\pi$ to any value function $V_0 : \mathcal{S} \to \mathbb{R}$, i.e.

$$\lim_{n\to\infty} \|\mathcal{T}_\pi^n V_0 - V^\pi\|_\infty = 0.$$

Using this property we can develop a meta-algorithm that, starting from any initial policy π_0, iteratively alternates between two steps. First, it computes the value function V_{π_k} of the current policy π_k, using the expected Bellman operator. Then, it generates a new policy π_{k+1}, based on the current policy π_k and the value function V^{π_k}, such that $V^{\pi_{k+1}} \geq V^{\pi_k}$. We call such meta-algorithm General Policy Iteration (GPI) and present it in Algorithm 2.

Algorithm 2 General Policy Iteration (GPI)

```
Require: $\pi_0$
  for $n = 0, 1, 2, ...$ do
      calculate $V^{\pi_n}$
      find $\pi_{n+1}$, such that $V_{\pi_{n+1}} \geq V_{\pi_n}$
  end for
```

Within the General Policy Iteration algorithm, two components demand particular attention: the methodology for computing the value function and the strategy for finding superior policies. Various approaches, ranging from dynamic programming to Monte Carlo simulation, offer distinct trade-offs in computational complexity and accuracy. Similarly, there are many ways for calculating policy improvement step. However, the most popular one is taking a greedy policy with respect to the value function of previous policy. This is a subject of the following lemma.

Lemma 1.13 ((Szepesvári, 2020)). *Let π and π' be two policies, such that π' is greedy with respect to V^π. Then:*

$$V^\pi \leq \mathcal{T} V^\pi \leq V^{\pi'}.$$

Proof. We have:

$$V^\pi = \mathcal{T}_\pi V^\pi \leq \mathcal{T} V^\pi = \mathcal{T}_{\pi'} V^\pi,$$

where the inequality holds from Definition 1.6 of the Bellman operator $\mathcal{T}$ and the last equality follows from Lemma 1.9 , because π' is greedy with respect to V^π. We will prove by induction on $n \geq 1$ that:

$$V^\pi \leq \mathcal{T} V^\pi \leq \mathcal{T}_{\pi'}^n V^\pi. \tag{2}$$

We have already proved the case for $n = 1$. Assume now that the inequality holds for some $n \geq 1$. We will show that it also holds for $n + 1$. Applying $\mathcal{T}_{\pi'}$ to both sides of Equation (2), we obtain from Proposition 1.8 that:

$$\mathcal{T}_{\pi'} V^\pi \leq \mathcal{T}_{\pi'}^{n+1} V^\pi.$$

Thus, we have:

$$V^\pi \leq \mathcal{T} V^\pi = \mathcal{T}_{\pi'} V^\pi \leq \mathcal{T}_{\pi'}^{n+1} V^\pi,$$

which finishes the inductive step. Now if we take $n \to \infty$ in Equation (2) we get from Theorem 1.6:

$$V^\pi \leq \mathcal{T} V^\pi \leq \lim_{n\to\infty} \mathcal{T}_{\pi'}^n V^\pi = V^{\pi'},$$

which ends the proof of the lemma. □

We will now prove a theorem, from which the convergence of GPI with greedy step follows.

Theorem 1.14. *Let $\{\pi_k\}_{k\geq 0}$ be the sequence of policies produced by GPI with greedy step procedure. Then, for any $k \geq 0$, we have:*

$$\|V^{\pi_k} - V^*\|_\infty \leq \gamma^k \|V^{\pi_0} - V^*\|_\infty.$$

Proof. For a given $k \geq 0$ applying Lemma 1.13 k times, starting with π_0, we obtain:

$$\mathcal{T}^k V^{\pi_0} \leq V^{\pi_k} \leq V^*.$$

Thus,

$$V^* - V^{\pi_k} \leq V^* - \mathcal{T}^k V^{\pi_0}.$$

Hence:

$$\|V^* - V^{\pi_k}\|_\infty \leq \|V^* - \mathcal{T}^k V^{\pi_0}\|_\infty = \|\mathcal{T}^k V^* - \mathcal{T}^k V^{\pi_0}\|_\infty \leq \gamma^k \|V^* - V^{\pi_0}\|_\infty,$$

where in the equality above, we used the fact that V^* is a stationary point of $\mathcal{T}$ from Theorem 1.12, and in the last inequality, we used the fact from Lemma 1.5 that $\mathcal{T}$ is a γ-contraction in $\|\cdot\|_\infty$. □

From Theorem 1.14 we derive the following corollary.

Corollary 1.15. *The GPI algorithm with greedy step converges to the optimal policy.*

Proof. From Theorem 1.14 we know that that:

$$\lim_{k\to\infty} V^{\pi_k} = V^*,$$

so the statement follows immediately from the definition of the optimal policy. $\square$

When it comes to calculating value functions in reinforcement learning, a variety of methods are available, each of them with distinct advantages, limitations, and applicability to different types of problems. These methods include various computational strategies designed to estimate the expected future rewards for specific states or state-action pairs. They range from traditional dynamic programming techniques to modern methods such as temporal difference learning and Monte Carlo simulations. In this discussion, we will explore some of the most widely-used methodologies.

1.5 Direct Methods

Assuming finite set of states and access to the transition function and expected values of the rewards in given MDP, we can directly compute the value function of given policy π. From Proposition 1.7 we know that value function of policy π is a fixed point of expected Bellman operator $\mathcal{T}_\pi$. For given policy π let us define matrix $P_\pi \in \mathbb{R}^{\mathcal{S}\times\mathcal{S}}$, such that for any $s \in \mathcal{S}$ and any $s' \in \mathcal{S}$:

$$P_\pi(s, s') = \sum_{a\in\mathcal{A}} \pi(a \mid s) \sum_{s'\in\mathcal{S}} P(s' \mid s, a). \tag{3}$$

Intuitively, matrix P_π can be viewed as transition matrix of policy π. Considering matrix P_π, we will also need a simple lemma:

Lemma 1.16. *Let $x \in \mathbb{R}^{\mathcal{S}}$ and $\pi \in \Pi$. We have:*

$$\|x\|_\infty \geq \|P_\pi x\|_\infty.$$

Proof. Note that P_π is a stochastic matrix, i.e., all entries of P_π are non-negative, and each of its rows sums to 1. Thus, each element of $P_\pi x$ is a weighted average of elements of x, with weights summing to 1, from which the statement follows. $\square$

Similarly, we define a vector $R_\pi \in \mathbb{R}^{\mathcal{S}}$, such that for any $s \in \mathcal{S}$:

$$R_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a \mid s) R(s, a).$$

We can now think of the value function of policy π as a vector in $\mathbb{R}^{\mathcal{S}}$, and from Proposition 1.7 we obtain that:

$$V^\pi = R_\pi + \gamma P_\pi V^\pi.$$

Solving for V^π with assumption of invertibility of $(I - \gamma P_\pi)$ gives us:

$$V^\pi = (I - \gamma P_\pi)^{-1} R_\pi. \tag{4}$$

The value function V^π can be determined through Equation (4) provided that the matrix $(I - \gamma P_\pi)$ is invertible. Indeed, this condition holds true universally, which we prove in the following lemma.

Lemma 1.17. *Matrix $(I - \gamma P_\pi)$ is invertible.*

Proof. Let $x \in \mathbb{R}^{\mathcal{S}}$ be a non-zero vector. Then we have:

$$\begin{aligned}
\|(I - \gamma P_\pi)x\|_\infty &= \|x - \gamma P_\pi x\|_\infty \\
&\geq \|x\|_\infty - \gamma \|P_\pi x\|_\infty \\
&\geq \|x\|_\infty - \gamma \|x\|_\infty \\
&= (1 - \gamma) \|x\|_\infty \\
&> 0,
\end{aligned}$$

where the first inequality follows from the triangle inequality, and the second inequality follows from Lemma 1.16. Thus, it follows that $I - \gamma P^\pi$ has a full rank, which implies its invertibility. □

1.6 Monte Carlo Methods

In the case where the environment is episodic (see Section 1.1) and we do not have access to the MDP dynamics, we can estimate the true expectation by a Monte Carlo method. Specifically, we can sample returns to approximate the expected values:

$$\begin{aligned}
G_1 &= r(s_0^1, a_0^1) + \gamma r(s_1^1, a_1^1) + ... + \gamma^{N_1} r(s_{N_1}^1, a_{N_1}^1) = \sum_{n=0}^{N_1} \gamma^n r(s_n^1, a_n^1), \\
G_2 &= r(s_0^2, a_0^2) + \gamma r(s_1^2, a_1^2) + ... + \gamma^{N_2} r(s_{N_2}^2, a_{N_2}^2) = \sum_{n=0}^{N_2} \gamma^n r(s_n^2, a_n^2), \\
&\vdots \\
G_k &= r(s_0^k, a_0^k) + \gamma r(s_1^k, a_1^k) + ... + \gamma^{N_k} r(s_{N_k}^k, a_{N_k}^k) = \sum_{n=0}^{N_k} \gamma^n r(s_n^k, a_n^k).
\end{aligned}$$

By following a specific policy within the environment, we generate trajectories or episodes of experience. Each trajectory provides us with a sample return, which is the sum of rewards received from the initial state to the terminal state. These returns are then averaged over multiple episodes to obtain an estimate of the expected value function, i.e.

$$G = \frac{1}{k} \sum_{n=1}^{k} G_n.$$

One of the key advantages of Monte Carlo estimation is its model-free nature. It does not rely on knowledge of the transition function or the reward distribution, making it applicable to a wide range of scenarios, including those where the environment is complex or poorly understood.

1.7 On-Policy and Off-Policy Algorithms

In the framework of Markov Decision Processes within reinforcement learning, we usually distinguish between two types of algorithms: on-policy algorithms and off-policy algorithms. On-policy algorithms directly learn and improve the policy that is used to make decisions. This means that they explore and update their policy while following the same policy during training. In contrast, off-policy algorithms decouple the behavior policy used for exploration from the target policy being optimized. This means they can learn from data generated by any policy, not just the one they are currently following.

1.8 Bellman Operators for Action-Value Functions

We can define Bellman operators for action-value functions from Definition 1.4 in a very similar way we defined it for value functions.

Definition 1.8 (Expected Bellman operator for action-value functions). *Let $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ be function and $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ be a policy. We define expected Bellman operator for action-value functions $\mathcal{T}_\pi$ as follows:*

$$(\mathcal{T}_\pi Q)(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) \left(\sum_{a' \in \mathcal{A}} \pi(a' \mid s') Q(s',a') \right).$$

Definition 1.9 (Bellman operator for action-value functions). *Let $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ be function and $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ be a policy. We define Bellman operator $\mathcal{T}$ for action-value functions as follows:*

$$\begin{aligned}(\mathcal{T} Q)(s,a) &= R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) \left(\max_{a' \in \mathcal{A}} Q(s',a') \right) \\ &= R(s,a) + \gamma \mathbb{E} \left[\max_{a' \in \mathcal{A}} Q(s',a') \mid s' \sim P(\cdot \mid s,a) \right].\end{aligned}$$

Similarly to Bellman operators for value functions, we can show that Bellman operators for action-value functions are contractions in $\|\cdot\|_\infty$.

Lemma 1.18. *Let π be any policy. Then:*

1. *Expected Bellman operator for action-value functions $\mathcal{T}_\pi$ is contraction in $\|\cdot\|_\infty$, i.e. for any $Q_1, Q_2 : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, it holds that $\|\mathcal{T}_\pi Q_1 - \mathcal{T}_\pi Q_2\|_\infty \leq \gamma \|Q_1 - Q_2\|_\infty$*
2. *Bellman operator for action-value functions $\mathcal{T}$ for is contraction in $\|\cdot\|_\infty$, i.e. for any $Q_1, Q_2 : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, it holds that $\|\mathcal{T} Q_1 - \mathcal{T} Q_2\|_\infty \leq \gamma \|Q_1 - Q_2\|_\infty$*

Proof. Proofs are analogous to Lemma 1.4 and Lemma 1.5. □

We define optimal action-value function as function $Q^* : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, given by the following formula:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) V^*(s').$$

In the following lemma, we prove that Q^π and Q^* are fixed points of operators $\mathcal{T}^\pi$ and $\mathcal{T}$, respectively.

Lemma 1.19. *Let π be any policy. Then:*

$$\mathcal{T}_\pi Q^\pi = Q^\pi \text{ and } \mathcal{T} Q^* = Q^*.$$

Proof. We have:

$$\begin{aligned}(\mathcal{T}_\pi Q^\pi)(s,a) &= R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) \left(\sum_{a' \in \mathcal{A}} \pi(a' \mid s') Q^\pi(s',a') \right) \\ &= R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) V^\pi(s) \\ &= Q^\pi(s,a),\end{aligned}$$

where the second equality follows from Remark 1.2 and third equality follows from Remark 1.1. We also have:

$$\begin{aligned}(\mathcal{T}Q^*)(s,a) &= R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) \left(\max_{a' \in \mathcal{A}} Q^*(s',a') \right) \\ &= R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) V^*(s') \\ &= Q^*(s,a).\end{aligned}$$

□

The contraction property of $\mathcal{T}_\pi$ and $\mathcal{T}$ for action-value functions ensures the existence of unique fixed points for these operators. Lemma 1.19 demonstrates that these unique points correspond to the action-value function of policy π and the optimal action-value function, respectively. Additionally, the Banach fixed-point theorem ensures that by iteratively applying these operators to any initial function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, the resulting sequences will converge to the fixed points. Consequently, we can formulate a theorem analogous to Theorem 1.6.

Theorem 1.20. *Let π be any policy. Then:*

1. $\mathcal{T}_\pi Q^\pi = Q^\pi$ *and* $\lim_{n\to\infty} \|\mathcal{T}_\pi^n Q - Q^\pi\|_\infty = 0$ *for any* $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
2. $\mathcal{T} Q^* = Q^*$ *and* $\lim_{n\to\infty} \|\mathcal{T}^n Q - Q^*\|_\infty = 0$ *for any* $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$.

Proof. The proof is a direct consequence of Lemma 1.18, Lemma 1.19 and the Banach fixed-point theorem. □

2 Function Approximation

We often encounter environments that are characterized by vast and complex state spaces. This poses a significant challenge when using conventional approaches to represent value functions. The problem arises when the state space of the environment is extremely large. In such cases, traditional methods that map each possible state or state-action pair to a value become impractical. Moreover, in such an environment, an agent is likely to encounter states that it has never seen before, making it impossible to have a predefined value for every possible situation in a lookup table.

To address this issue, we turn to function approximation – a powerful concept that allows us to generalize beyond the states we have observed and estimate value functions. By introducing function approximation, we can create more scalable and flexible RL algorithms capable of operating in complex environments. An approximator can be any function that maps input data to a real-valued output. Specifically, we consider a function $f : \mathbb{R}^n \times \mathbb{R}^m \ni (x, \theta) \to f(x, \theta) \in \mathbb{R}$, where x represents the input data and θ are the parameters of the function. The key requirement for this approximator is that it must be differentiable with respect to θ.

In the context of Markov Decision Processes in reinforcement learning, we often work with states and state-action pairs. To use an approximator effectively, we need a numerical representation for each state or state-action pair. Let us denote this representation function as $\Phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^n$, where $\mathcal{S}$ represents the state space and $\mathcal{A}$ represents the action space. Essentially, $\Phi(s, a)$ provides a compact representation of the state-action pair (s, a). Now, the purpose of using such an approximator is to find optimal parameters θ_0 such that $f(\Phi(s, a), \theta_0)$ approximates a certain function $R(s, a)$. As defined in the context of MDPs, the function $R(s, a)$ typically represents the expected return associated with taking action a in state s.

In practice, various function approximators can be used, such as linear models, neural networks, or kernel methods. These approximators allow us to generalize from observed data and learn useful representations for RL tasks. Our thesis focuses solely on neural networks, which will be discussed in the next subsection. For convenience, we use notation $f_\theta(s, a) := f(\Phi(s, a), \theta)$.

2.1 Neural Network

Neural networks, also known as artificial neural networks, are the most frequently used function approximators in reinforcement learning (Goodfellow, 2016). They consist of multiple layers of interconnected nodes (neurons), where each layer applies a linear transformation followed by a nonlinear activation function. This layered structure allows neural networks to capture complex, nonlinear relationships between the input features and the output values.

Formally, a neural network function approximator is defined as:

$$f_\theta(s, a) = (f_{n-1} \circ f_{n-2} \circ ... \circ f_0)(\Phi(s, a)),$$

where each layer transformation f_k is given by:

$$f_k(x) = g_k(W_k^\top x + b_k),$$

with $\theta = (W_{n-1}, b_{n-1}, W_{n-2}, b_{n-2}, ..., W_0, b_0)$, and $W_k \in \mathbb{R}^{h_{k-1} \times h_k}, b_k \in \mathbb{R}^{h_k}$. The function g_k is a nonlinear activation function applied element-wise to the linear transformation of the input.

The most common activation functions used in neural networks include:

- sigmoid function, $\sigma(x) = \frac{1}{1+\exp(-x)}$,
- hyperbolic tangent function, $\tanh(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$,
- rectifier function, $ReLu(x) = \max(0, x)$.

The depth (number of layers) and width (number of neurons per layer) of the neural network can be adjusted to capture the complexity of the function being approximated. Deeper networks with more layers can model more intricate patterns, while wider networks with more neurons per layer can capture more detail in the data.

2.2 Gradient Descent

To train a neural network, its parameters must be optimized so that the network can accurately approximate the desired function. This optimization is commonly achieved through an iterative process known as Gradient Descent (Nocedal and Wright (1999), Boyd and Vandenberghe (2004), Bubeck et al. (2015)). Gradient Descent is a fundamental optimization technique which, in the case of neural networks, is used to minimize the loss function by iteratively updating the parameters of the network in the direction that reduces the value of the loss function the most.

Gradient Descent is a fundamental optimization technique used to minimize functions. The basic idea is to iteratively update parameters in the direction that reduces the function value the most. We present Gradient Descent algorithm in Algorithm 3.

Algorithm 3 Gradient Descent

```
Require: $\theta_0$, $\alpha$
  for $t = 1, 2, ...$ do
      Compute the gradient $\nabla f(\theta_t)$
      update $\theta_{t+1} = \theta_t - \alpha \nabla f(\theta_t)$
  end for
```

In Gradient Descent, the entire dataset is used to compute the gradient in each iteration. While this approach can be effective, it may become computationally expensive, especially for large datasets. To address this issue, Stochastic Gradient Descent (SGD) offers an alternative approach by using a subset of the data in each update step (Bubeck et al. (2015); Birge and Louveaux (2011)). Stochastic Gradient Descent improves efficiency by approximating the gradient using a small batch of data, rather than the full dataset. The SGD algorithm is presented in Algorithm 4.

Algorithm 4 Stochastic Gradient Descent

```
Require: $\theta_0$, learning rate $\alpha$, and batch size $b$
  for $t = 1, 2, ...$ do
      for each batch $B$ of size $b$ do
          Sample a mini-batch $B$ from the dataset
          Compute the gradient $\nabla f(\theta_t; B)$ based on the mini-batch $B$
          Update $\theta_{t+1} = \theta_t - \alpha \nabla f(\theta_t; B)$
      end for
  end for
```

3 Q-learning and Deep Q-network

In this chapter, we describe two reinforcement learning algorithms designed to approximate the optimal action-value function for a given MDP. We begin with Q-learning, an off-policy algorithm that estimates this function using a tabular representation of state-action pairs, and then explore the Deep Q-Network (DQN) algorithm, which extends Q-learning algorithm by leveraging neural network function approximation to manage environments with high-dimensional state spaces, where tabular representation becomes impractical.

3.1 Q-learning

We will now discuss an off-policy algorithm called Q-learning, which is designed to learn the optimal action-value function. The algorithm maintains a table, where each entry corresponds to a state-action pair and stores a quantity known as the Q-value, representing the agent's current estimate of the optimal action-value function for that pair. We can initialize the Q-values arbitrarily, for example all to be equal to zero.

The algorithm updates these values based on the principles of the Bellman operator for action-value functions from Definition 1.9. This update rule establishes a recursive relationship in which the Q-value of a state-action pair is adjusted based on the reward received after taking the action and the maximum Q-value achievable in the next state by selecting the best possible action. The whole procedure is described in Algorithm 5.

Algorithm 5 Q-learning algorithm

```
Require: $\pi_0$
  for $t = 0, 1, 2, ...$ do
      1) Observe current state $s_t$
      2) Perform action $a_t$ in state $s_t$ and receive immediate reward $r_t$
      3) Observe next state $s_{t+1}$
      4) Perform an update of the estimated Q-values:
            $Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t[r_t + \gamma \max_{a \in A} Q_t(s_{t+1}, a) - Q_t(s_t, a_t)]$
      where $0 \leq \alpha_t < 1$ is the learning rate at time step $t$.
  end for
```

With some assumptions on the learning rates and boundedness of the rewards, it can be proved, that Q-learning converges to the optimal action-value function (and therefore the target policy, which is greedy with respect to estimated action-value function is the optimal policy).

For every state-action pair (s, a) we define $n^i(s, a)$ to be the time step in which action a was tried in state s for the i^{th} time.

Theorem 3.1 (Q-learning convergence (Watkins and Dayan, 1992))**.** *Given a finite MDP with bounded rewards, the Q-learning algorithm converges with probability* 1 *to the optimal action-value function Q^*, as long as the following conditions hold:*
(1) $\sum_{i=1}^{\infty} \alpha_{n^i(s,a)} = \infty$ for every state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$,
(2) $\sum_{i=1}^{\infty} \alpha^2_{n^i(s,a)} < \infty$ for every state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$.

Note that, in particular, the condition (1) implies that every state-action pair must be visited infinitely often.

The Q-learning algorithm can be used in practice in both episodic and continuing tasks scenarios. A common policy responsible for taking actions in the environment is a policy which with some probability ϵ chooses random action to explore the environment and with probability $1 - \epsilon$ chooses the action that maximizes the estimated Q-values. We will call such a policy an ϵ-greedy policy. At the beginning of the learning process we can explore more and make ϵ bigger, as the Q-values, which aim to estimate the optimal action-value function can be inaccurate. As the learning proceeds and the estimates are getting more accurate, we can decrease ϵ and focus more on exploiting our approximation of optimal action-value function. In episodic tasks for all terminal states s_f the Q-values $Q(s_f, \cdot)$ are initialized to zero and are never updated, since the reward obtained from these states is always zero. The target policy in Q-learning algorithm is the greedy policy with respect to the Q-values, thus Q-learning can be seen as an off-policy algorithm, because the policy which takes actions in the environment is different from the target policy (it incorporates the "ϵ-exploration" part).

The overall scheme of the algorithm is presented in Algorithm 6 based on (Sutton and Barto, 2018).

Algorithm 6 Q-learning algorithm for episodic tasks

```
Initialize $Q(s, a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$ arbitrarily except that for terminal states
$s_f \in \mathcal{S}$: $Q(s_f, \cdot) = 0$
for episode = $1, ..., M$ do
    Observe the initial state $s_1$
    for each step $t = 1, ...$ of the episode do
        Sample action $a_t$ from $\epsilon$-greedy policy $\pi$
        Execute action $a_t$ in state $s_t$, observe reward $r_t$ and next state $s_{t+1}$
        Update the Q-value according to:
            $Q(s_t, a_t) = Q(s_t, a_t) + \alpha \Big( r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \Big)$
        if state $s_{t+1}$ is terminal then
            break
        end if
    end for
end for
```

3.2 Deep Q-Network

Assuming access to a neural network function approximation, we can use the intuition from Q-learning algorithm to train a powerful deep reinforcement learning agent, namely one that leverages deep neural networks. This concept encapsulates the essence of the Deep Q-Network (DQN) algorithm (Mnih et al., 2013). DQN has the ability to handle high-dimensional state spaces and, consequently, complex environments.

As the Q-values table of Q-learning can be very large for high-dimensional state spaces, DQN uses a neural network with weights θ to approximate the action-value function. The neural network takes the state as input and outputs Q-values for all possible actions, effectively generalizing across similar states. We denote by θ_i the parameters of the neural network at time step i. Let $\mathcal{E}$ be the emulator of the environment, which, given state-action pair (s, a), outputs the next state s' of the environment. The loss function $L_i(\theta_i)$ that DQN aimes to minimize is different in every time step i of the algorithm.

Precisely, it is given by the following formula:

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}[(y_i - Q(s,a;\theta_i))^2], \tag{5}$$

where $y_i = \mathbb{E}_{s'\sim\mathcal{E}}[r(s,a) + \gamma \max_{a'} Q(s',a';\theta_{i-1})|s,a]$ is known as the target for iteration i, and $\rho(s,a)$ is some distribution over state-action pairs which we will specify later. Note that the gradient of the loss function $L_i(\theta_i)$ with respect to parameters θ_i(parameters θ_{i-1} are held fixed) in iteration i can be expressed as follows:

$$\begin{aligned}\nabla_{\theta_i} L_i(\theta_i) = 2 \cdot \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{E}}\Bigg[\Big(r(s,a) + \gamma \max_{a'} Q(s',a';\theta_{i-1})\\ -Q(s,a;\theta_i)\Big)\,\nabla_{\theta_i} Q(s,a;\theta_i)\Bigg].\end{aligned} \tag{6}$$

Based on the loss function defined in Equation (5), the weights of Q-network may be updated by the gradient descent algorithm, using the computation from Equation (6). However, the full gradient descent is computationally expensive, thus DQN performs stochastic gradient descent, by sampling the state-action pairs (s,a) from the distribution $\rho(s,a)$ and corresponding next states s' from the emulator of the environment $\mathcal{E}$.

Similarly as in Q-learning, DQN's behavior policy is an ϵ-greedy policy, thus it is also an off-policy algorithm. In DQN, the distribution $\rho(s,a)$ is acquired from a critical component known as the replay buffer. This buffer stores transitions gathered during the agent's interactions with the environment. In each iteration t, the behavior policy selects an action a_t given the current state s_t, subsequently observing the reward r_t and the next state s_{t+1}. The complete transition tuple (s_t, a_t, r_t, s_{t+1}) is then stored in the replay buffer D. The replay buffer has some finite capacity N, so only the last N transitions are stored. This helps managing the memory usage and ensures that the agent focuses mostly on the most recent and relevant experiences while still retaining some diversity. For the optimization of the loss function, state-action pairs are sampled from D in a uniform manner.

The full scheme of the DQN procedure is presented in Algorithm 7, based on the procedure described in (Mnih et al., 2013). Note that the algorithm described in the original paper was tailored specifically for Atari games, necessitating preprocessing of game images to extract meaningful states. We omit the preprocessing step in our algorithm for simplicity, and assume direct access to the states of the environment.

Algorithm 7 DQN algorithm

```
Initialize replay memory $D$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, ..., M$ do
    Initialise the states sequence with $s_1$
    for $t = 1, ..., T$ do
        With probability $\epsilon$ select a random action $a_t$
        otherwise select $a_t = \arg\max_a Q(s_t, a; \theta)$
        Execute action $a_t$, observe reward $r_t$ and next state $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $D$
        Sample random minibatch of transitions $s_j, a_j, r_j, s_{j+1}$ from $D$
        Set $y_j = \begin{cases} r_j & \text{for terminal } s_{j+1} \\ r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta), & \text{for non-terminal } s_{j+1} \end{cases}$
        Perform a gradient step on $(y_j - Q(s_j, a_j; \theta))^2$
    end for
end for
```

4 Policy Gradient

We will now consider different kind of methods for reinforcement learning problems within the framework of infinite-horizon Markov Decision Processes (MDPs). So far we have focused on methods, where the policy is taking actions based on the estimated value function. Instead, we can learn a parameterized policy $\pi_\theta(a|s)$, where $\theta \in \mathbb{R}^d$ is a vector of the policy parameters. These parameters can be learned for example by a neural network.

For this to work properly we need some sensible objective function $J(\theta)$, which will depend on the parameters θ. We can then optimize this objective, e.g., by stochastic gradient descent from Algorithm 4:

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t).$$

The objective function is usually the expected cumulative discounted return obtained from the initial distribution μ, i.e., $J(\theta) = \sum_{s \in \mathcal{S}} \mu(s) V_{\pi_\theta}(s)$.

In theory, we can use any parameterization for the policy as long as $\pi_\theta(a|s)$ is differentiable with respect to θ. In practice, a common choice for discrete action spaces is a softmax policy, i.e.,

$$\pi_\theta(a|s) = \frac{\exp f(s, a, \theta)}{\sum_{b \in A} \exp f(s, b, \theta)},$$

where f can be any differentiable function, for example, linear function or neural network. With a softmax parameterization we can learn an adaptive exploration strategy, as we assign higher probabilities to actions that have yielded higher rewards in the past. Therefore, we can focus the exploration mostly on promising actions, which is not the case for uniform exploration such as in the Q-learning algorithm from Section 3. This can be beneficial, for example, in environments with huge action spaces, if only a few actions lead to positive rewards, since uniform exploration may cause us to spend a lot of time investigating completely unpromising actions.

A common choice of parameterization for continous action spaces is a Gaussian parameterization. In this case, a policy takes actions for each state according to a Normal distribution, whose mean and standard deviation are parameterized by the parameters θ.

As such parameterization is non-differentiable, usually some kind of reparameterization trick is used to make it functional.

4.1 Policy Gradient Theorem

We can express the gradient of the objective function $J(\theta) = \sum_{s\in\mathcal{S}} \mu(s)V_{\pi_\theta}(s)$ in terms of a discounted weighting of states encountered by policy π, starting from the initial distribution μ, which is given by the following formula:

$$d^\pi(s) = \sum_{t=0}^{\infty} \gamma^t p(s_t = s|\mu, \pi),$$

where $p(s_t = s|\mu, \pi)$ is a probability of reaching state s in t steps, starting from the initial distribution μ, and following policy π. This is a subject of the Policy Gradient Theorem.

Theorem 4.1 (Policy Gradient Theorem, Sutton et al. (1999))**.** *For any MDP we have:*

$$\nabla_\theta J(\theta) = \sum_{s\in S} d^{\pi_\theta}(s) \sum_{a\in A} \frac{\partial \pi_\theta(a|s)}{\partial\theta} Q^\pi(s,a). \tag{7}$$

Proof. For any $s \in \mathcal{S}$ we have:

$$\begin{aligned}
\frac{\partial V^{\pi_\theta}(s)}{\partial\theta} &= \frac{\partial}{\partial\theta} \sum_{a\in A} \pi_\theta(a|s) Q^{\pi_\theta}(s,a) = \\
&= \sum_{a\in A} \left[\frac{\partial \pi_\theta(a|s)}{\partial\theta} Q^{\pi_\theta}(s,a) + \pi_\theta(a|s) \frac{\partial}{\partial\theta} Q^{\pi_\theta}(s,a) \right] = \\
&= \sum_{a\in A} \left[\frac{\partial \pi_\theta(a|s)}{\partial\theta} Q^{\pi_\theta}(s,a) + \pi_\theta(a|s) \frac{\partial}{\partial\theta} \left[r(s,a) + \sum_{s'\in S} \gamma P(s'|s,a) V^{\pi_\theta}(s') \right] \right] = \\
&= \sum_{a\in A} \left[\frac{\partial \pi_\theta(a|s)}{\partial\theta} Q^{\pi_\theta}(s,a) + \pi_\theta(a|s) \sum_{s'\in S} \gamma P(s'|s,a) \frac{\partial}{\partial\theta} V^{\pi_\theta}(s') \right].
\end{aligned}$$

Now, we can use the obtained equality for any state s' and get:

$$\frac{\partial V^{\pi_\theta}(s')}{\partial\theta} = \sum_{a'\in A} \left[\frac{\partial \pi_\theta(a'|s')}{\partial\theta} Q^{\pi_\theta}(s',a') + \pi_\theta(a'|s') \sum_{s''\in S} \gamma P(s''|s',a') \frac{\partial}{\partial\theta} V^{\pi_\theta}(s'') \right].$$

Substituting it back into previous equality we get:

$$\begin{aligned}
\frac{\partial V^{\pi_\theta}(s)}{\partial\theta} = \sum_{a\in A} \Bigg[& \frac{\partial \pi_\theta(a|s)}{\partial\theta} Q^{\pi_\theta}(s,a) \\
& + \pi_\theta(a|s) \sum_{s'\in S} \gamma P(s'|s,a) \sum_{a'\in A} \Bigg[\frac{\partial \pi_\theta(a'|s')}{\partial\theta} Q^{\pi_\theta}(s',a') \\
& + \pi_\theta(a'|s') \sum_{s''\in S} \gamma P(s''|s',a') \frac{\partial}{\partial\theta} V^{\pi_\theta}(s'') \Bigg] \Bigg].
\end{aligned}$$

After repeatedly unrolling this equality, we get:

$$\frac{\partial V^{\pi_\theta}(s)}{\partial\theta} = \sum_{x\in\mathcal{S}} \sum_{k=0}^{\infty} \gamma^k p(s_k = x|s_0 = s, \pi) \sum_{a\in\mathcal{A}} \frac{\partial \pi_\theta(a|x)}{\partial\theta} Q^{\pi_\theta}(x,a),$$

where $p(s_k = x|s_0 = s, \pi)$ is probability of reaching state s in k steps, starting from the initial state s_0, and following policy π. Note that, in particular, we have for any natural number k:

$$p(s_k = x|\mu, \pi) = \sum_{s \in \mathcal{S}} \mu(s) p(s_k = x|s_0 = s, \pi).$$

Thus, we finally obtain:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \sum_{s \in \mathcal{S}} \mu(s) \frac{\partial V^{\pi_\theta}(s)}{\partial \theta} \\
&= \sum_{s \in \mathcal{S}} \mu(s) \sum_{x \in \mathcal{S}} \sum_{k=0}^{\infty} \gamma^k p(s_k = x \mid s_0 = s, \pi) \sum_{a \in \mathcal{A}} \frac{\partial \pi_\theta(a \mid x)}{\partial \theta} Q^{\pi_\theta}(x, a) \\
&= \sum_{x \in \mathcal{S}} \sum_{k=0}^{\infty} \gamma^k p(s_k = x \mid \mu, \pi) \sum_{a \in \mathcal{A}} \frac{\partial \pi_\theta(a \mid x)}{\partial \theta} Q^{\pi_\theta}(x, a) \\
&= \sum_{x \in \mathcal{S}} d^{\pi_\theta}(x) \sum_{a \in \mathcal{A}} \frac{\partial \pi_\theta(a \mid x)}{\partial \theta} Q^{\pi_\theta}(x, a),
\end{aligned}$$

which completes the proof. □

4.2 REINFORCE

Based on the intuition from Theorem 4.1, we can now derive a practical algorithm called REINFORCE (Sutton and Barto, 2018). From Theorem 4.1 we have:

$$\begin{aligned}
&\nabla J(\theta) = \sum_{s \in S} d^{\pi_\theta}(s) \sum_{a \in A} \frac{\partial \pi_\theta(a|s)}{\partial \theta} Q^{\pi}(s, a) = \\
&= \sum_{s \in S} \sum_{k=0}^{\infty} \gamma^k p(s_k = s|\mu, \pi_\theta) \sum_{a \in A} \frac{\partial \pi_\theta(a|s)}{\partial \theta} Q^{\pi_\theta}(s, a) = \\
&= \sum_{s \in S} \sum_{k=0}^{\infty} p(s_k = s|\mu, \pi_\theta) \sum_{a \in A} \pi_\theta(a|s) \frac{\partial \pi_\theta(a|s)}{\partial \theta} \frac{1}{\pi_\theta(a|s)} \gamma^k Q^{\pi_\theta}(s, a) = \\
&= \sum_{s \in S} \sum_{k=0}^{\infty} p(s_k = s|\mu, \pi_\theta) \sum_{a \in A} \pi_\theta(a|s) [\frac{\partial \ln \pi_\theta(a|s)}{\partial \theta} \gamma^k Q^{\pi_\theta}(s, a)].
\end{aligned}$$

Thus, to perform the stochastic gradient descent from Algorithm 4, we can follow the policy π_θ, and in each step t perform the following update:

$$\theta_{t+1} = \theta_t + \alpha \gamma^t \tilde{Q}^{\pi_\theta}(s, a) \frac{\partial \ln \pi_\theta(a|s)}{\partial \theta},$$

where $\tilde{Q}^{\pi_\theta}(s, a)$ is some estimator of the true action-value function $Q^{\pi_\theta}(s, a)$. We can, for example, sample an episode of interaction with the environment by following the current policy from state-action pair (s, a) and calculate, for each step t, the discounted sample return from that point onward. The pseudocode for the REINFORCE algorithm is presented in Algorithm 8.

In practice, the version of the REINFORCE algorithm presented in Algorithm 8 may be high variance, due to noisy estimates of the action-value functions in form of the sample returns. To address this issue, we can note that Theorem 4.1 can be generalized to a more convenient form.

Algorithm 8 REINFORCE algorithm

```
Initialize parameters $\theta$ arbitrarily
for each episode do
    Generate an episode $s_0, a_0, r_1, ..., s_{T-1}, a_{T-1}, r_T$, following policy $\pi_\theta$
    for each step of the episode $t = 0, 1, ..., T-1$ do
        $G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$
        $\theta = \theta + \alpha\gamma^t G_t \nabla_\theta \ln \pi_\theta(a_t|s_t)$
    end for
end for
```

Proposition 4.2. *Equation* (7) *can be generalized to the form:*

$$\nabla_\theta J(\theta) = \sum_{s \in S} d^{\pi_\theta}(s) \sum_{a \in A} \frac{\partial \pi_\theta(a|s)}{\partial \theta} [Q^\pi(s,a) - b(s)],$$

where $b(s)$ is a "baseline" function which does not vary with a.

Proof. For any $s \in \mathcal{S}$ we have:

$$\sum_{a \in A} b(s) \frac{\partial \pi_\theta(a|s)}{\partial \theta} = b(s) \sum_{a \in A} \frac{\partial \pi_\theta(a|s)}{\partial \theta} = b(s) \frac{\partial}{\partial \theta} \sum_{a \in A} \pi_\theta(a|s) = b(s) \frac{\partial}{\partial \theta} 1 = 0.$$

□

Using the above result, we can include the baseline function into the REINFORCE algorithm as follows:

$$\theta_{t+1} = \theta_t + \alpha\gamma^t [G_t - b(s_t)] \frac{\partial \ln \pi_\theta(a_t|s_t)}{\partial \theta}.$$

A common choice for the baseline function is an estimate of the value function of the input state s_t.

4.3 Actor-Critic Algorithms

Instead of using a sample of trajectory for approximating the action-value function of policy π as in REINFORCE algorithm, it can be approximated using some kind of model such as neural network. The policy can still be updated using the Policy Gradient Theorem. Algorithms that integrate these approaches are known as actor-critic algorithms. Intuitively, we can think of these methods as follows: the actor directly approximates the policy, while the critic evaluates the action-value function of the policy suggested by the actor, providing feedback that guides the actor toward better decisions over time. The critic can for example update the parameters w of the estimated action-value function $Q_w(s,a)$. A typical loss function to be optimized by the critic is given by the following formula:

$$L(w) = \mathbb{E}_{(s,a)\sim\rho(\cdot);s'\sim P(\cdot|s,a)}[Q_w(s,a) - y]^2,$$

where $\rho(\cdot)$ is some distribution of state-action pairs and

$$y = r(s,a) + \gamma Q_w(s', a')$$

is a target for the action-value function, and the gradient does not propagate through y. This loss uses the intuition from the expected Bellman operator for action-value functions from Definition 1.8, that the action-value function of a policy should be equal to the sum of immediate reward and the discounted action-value of the next state. We can then update the parameters w by stochastic gradient descent from Algorithm 4, sampling from the currently estimated policy and the environment, i.e.,

$$w = w + \alpha_w(r + \gamma Q_w(s', a') - Q_w(s, a))\nabla_w Q_w(s, a),$$

where r is a sample from the distribution of random variable $r(s, a)$.

A typical Actor-Critic algorithm scheme is presented in Algorithm 9.

Algorithm 9 Actor-Critic algorithm (Weng, 2018)

```
Initialize policy parameters $\theta$ and state-value weights $w$ arbitrarily
for episode $= 1, ..., M$ do
    Observe the initial state $s_1$
    for $t = 1, ..., T$ do
        Sample action $a_t \sim \pi(\cdot|s_t)$
        Execute action $a_t$, observe reward $r_t$ and next state $s_{t+1}$
        Sample next action $a_{t+1} \sim \pi(\cdot|s_{t+1})$
        Update the policy parameters
            $\theta = \theta + \alpha_\theta Q_w(s_t, a_t)\nabla_\theta \ln \pi_\theta(a_t|s_t)$
        Compute
            $\delta_t = r_t + \gamma Q_w(s_{t+1}, a_{t+1}) - Q_w(s_t, a_t),$
        and use it to update the parameters of action-value function estimator:
            $w = w + \alpha_w \delta_t \nabla_w Q_w(s_t, a_t)$
    end for
end for
```

5 Computational Complexity of RL Algorithms

In this section we will take a closer look on Value Iteration from Algorithm 1 and General Policy Iteration from Algorithm 2. Specifically, we will study how close these algorithms get to the optimal solution as a function of the number of iterations. This approach allows us to provide insights into their convergence properties and analyze their computational complexity. The results discussed in this chapter are based on (Szepesvári, 2020).

In this section we will consider value functions as vectors $V \in \mathbb{R}^{\mathcal{S}}$, while in previous sections they were considered as functions $V : \mathcal{S} \to \mathbb{R}$. Thus, if we write for example $V \geq 0$ it means that $\forall s \in \mathcal{S}$: $V(s) \geq 0$. In this chapter we will assume all the rewards lie in the [0, 1] interval for simplicity. Similarly as before we will assume that the state space $\mathcal{S}$ and the action space $\mathcal{A}$ are finite, and denote by S and A the corresponding cardinalities of these spaces.

5.1 Finite Value Iteration

We have already shown that iterative computation, starting from some initial value function $V_0 \in \mathbb{R}^S$ and obtaining subsequent value functions using the Bellman operator, leads to the sequence of value functions $\{V_k\}_{k\geq 0}$ which approaches the optimal value function V^*. In practice, taking the limit in Algorithm 1 can be intractable, especially for large state and action spaces. A more pragmatic approach involves performing the update with Bellman operator for a finite number of iterations. This finite horizon should be chosen sufficiently large so that the resulting value function is close enough to the optimal value function V^*. The algorithm then returns the policy which is greedy with respect to the obtained value function. We call such algorithm Finite Value Iteration and present it in Algorithm 10.

Algorithm 10 Finite Value Iteration (FVI)

```
Require: $V_0 \in \mathbb{R}^{\mathcal{S}}$, finite horizon $k \in \mathbb{N}_+$
  for $i = 1, 2, ..., k$ do:
      $V_i \leftarrow \mathcal{T} V_{i-1}$
  end for
  $\pi \leftarrow greedy(V_k)$
  return $\pi$
```

The idea of Finite Value Iteration algorithm is to obtain a value function, which is sufficiently close to the optimal value function. In the next theorem, we will prove a lower bound (dependent on the desired accuracy ϵ) on the number of iterations needed to reach a value function, which lies in the ϵ-neighbourhood of the optimal value function.

Theorem 5.1 ((Szepesvári, 2020)). *For any $\epsilon > 0$, if we set the initial vector $V_0 = 0$ and perform FVI procedure, then for $k \geq \ln \frac{1}{\epsilon(1-\gamma)} / \ln \frac{1}{\gamma}$ we have:*

$$\|V_k - V^*\|_\infty \leq \epsilon.$$

Proof. By assumption that rewards lie in the $[0, 1]$ interval, for any policy π we have:

$$0 \leq V^\pi \leq \sum_{i=0}^{\infty} \gamma^i = \frac{1}{1-\gamma},$$

and therefore:

$$\|V^*\|_\infty \leq \frac{1}{(1-\gamma)}.$$

As we know from Lemma 1.5 the Bellman Operator is a γ-contraction and from Theorem 1.12 that V^* is a stationary point of $\mathcal{T}$:

$$\|V^* - V_k\|_\infty = \|\mathcal{T}^k V^* - \mathcal{T}^k V_0\|_\infty \leq \gamma^k \|V^* - V_0\|_\infty = \gamma^k \|V^* - \mathbf{0}\|_\infty = \gamma^k \|V^*\|_\infty \leq \frac{\gamma^k}{1-\gamma}.$$

Clearly, the smallest k which satisfies the inequality $\frac{\gamma^k}{1-\gamma} \leq \epsilon$ is $k = \ln(\frac{1}{(\epsilon(1-\gamma))})/\ln(\frac{1}{\gamma})$, which concludes the proof. □

An important question is whether a near-optimal value function produced by the Finite Value Iteration will result in a policy, whose value function is also near-optimal. We will prove a lemma which guarantees that when V_k approaches V^*, the value function V^{π_k} of the policy π_k, which is greedy with respect to V_k, also approaches V^*. We will prove a lemma, which guarantees that, when V_k approaches V^* then also the value function V^{π_k} of the policy π_k, which is greedy with respect to V_k, approaches V^*.

Lemma 5.2 ((Szepesvári, 2020))**.** *For arbitrary vector $V \in \mathbb{R}^{\mathcal{S}}$ and policy π, which is greedy with respect to V, we have:*

$$V^\pi \geq V^* - \frac{2\gamma\|V_* - V\|_\infty}{1-\gamma}.$$

Proof. First, we will need two equalities, which are straightforward to see after expanding Bellman operators, i.e.:
For any $V \in \mathbb{R}^S$ and $C \in \mathbb{R}$:

$$\mathcal{T}_\pi(V + C) = \mathcal{T}_\pi V + C\gamma, \tag{8}$$

$$\mathcal{T}(V + C) = \mathcal{T}V + C\gamma. \tag{9}$$

Let $\epsilon = \|V^* - V\|_\infty$, which implies that:

$$-\epsilon \leq V^* - V \leq \epsilon \tag{10}$$

and

$$-\epsilon \leq V - V^* \leq \epsilon. \tag{11}$$

For a vector $x \in \mathbb{R}^{\mathcal{S}}$ we will use notation $|x|$ for a componentwise absolute value of that vector, i.e., $|x|_i = |x_i|$ for $i = 1, 2, ..., S$. From Lemma 1.16 we know that for any policy π:

$$\|P_\pi x\|_\infty \leq \|x\|_\infty,$$

where P_π is the transition matrix of policy π defined in Equation (3).

Now, let $\delta := V_* - V$. Our goal is to establish an upper bound for δ. We will achieve this through a series of steps. We have:

$$V^* - V^\pi = \mathcal{T}V^* - \mathcal{T}_\pi V^\pi \leq \mathcal{T}(V + \epsilon) - \mathcal{T}_\pi V^\pi,$$

which follows from Equation (10) and Proposition 1.8. Now, using Equation (9) we obtain:

$$\mathcal{T}(V + \epsilon) - \mathcal{T}_\pi V^\pi = \mathcal{T}V + \gamma\epsilon - \mathcal{T}_\pi V^\pi.$$

Policy π is greedy with respect to V, therefore by Lemma 1.9 we have:

$$\mathcal{T}V + \gamma\epsilon - \mathcal{T}_\pi V^\pi = \mathcal{T}_\pi V + \gamma\epsilon - \mathcal{T}_\pi V^\pi.$$

Continuing, we have:

$$\mathcal{T}_\pi V + \gamma\epsilon - \mathcal{T}_\pi V^\pi \leq \mathcal{T}_\pi(V^* + \epsilon) + \gamma\epsilon - \mathcal{T}_\pi V^\pi,$$

which follows from Equation (11) and Proposition 1.8. Now using Equation (8) we get:

$$\mathcal{T}_\pi(V^* + \epsilon) + \gamma\epsilon - \mathcal{T}_\pi V^\pi = \mathcal{T}_\pi V^* + 2\gamma\epsilon - \mathcal{T}_\pi V^\pi.$$

By expanding $\mathcal{T}_\pi$:

$$\mathcal{T}_\pi V^* + 2\gamma\epsilon - \mathcal{T}_\pi V^\pi = \gamma P^\pi(V^* - V^\pi) + 2\gamma\epsilon = \gamma P^\pi \delta + 2\gamma\epsilon.$$

From the above derivations we finally get:

$$\delta \leq \gamma P^\pi \delta + 2\gamma\epsilon.$$

Now if we take componentwise absolute value of both sides, use the triangle inequality and Lemma 1.16, we obtain:

$$|\delta| \leq |\gamma P_\pi \delta + 2\gamma\epsilon| \leq \gamma|P_\pi\delta| + 2\gamma\epsilon \leq \gamma\|\delta\|_\infty + 2\gamma\epsilon.$$

Now taking maximum over components of $|\delta|$ we get:

$$\|\delta\|_\infty \leq \gamma\|\delta\|_\infty + 2\gamma\epsilon,$$

and hence:

$$\|\delta\|_\infty \leq \frac{2\gamma\epsilon}{1-\gamma},$$

which concludes the proof (see definitions of δ and ϵ). □

We will now focus on the computational complexity of the FVI algorithm.

Theorem 5.3 (Computational complexity of FVI algorithm (Szepesvári, 2020))**.** *For any fixed target accuracy δ, after $O\left(S^2 A \ln(\frac{2\gamma}{\delta(1-\gamma)^2})/\ln(\frac{1}{\gamma})\right)$ arithmetic and logic operations, starting from the initial value function $V_0 = 0$, FVI produces a policy π such that:*

$$V^\pi \geq V^* - \delta.$$

Proof. From Theorem 5.1 we know, that, after $k \geq \ln\frac{1}{\epsilon(1-\gamma)}/\ln\frac{1}{\gamma}$ iterations of the algorithm, we have $\|V_k - V^*\|_\infty \leq \epsilon$. Thus, from Lemma 5.2, for a policy π_k that is greedy with respect to V_k, we get:

$$V^{\pi_k} \geq V^* - \frac{2\gamma\epsilon}{1-\gamma}.$$

Now, if we set $\epsilon = \frac{\delta(1-\gamma)}{2\gamma}$, we obtain:

$$V^{\pi_k} \geq V^* - \delta.$$

One can see that computing $V_{k+1} = \mathcal{T}V_k$ takes $O(S^2A)$ arithmetic and logic operations. To compute the value of the new value function at a particular state, we need to compute $\max_{a\in\mathcal{A}}\left\{R(s,a) + \gamma\sum_{s'\in\mathcal{S}} P\left(s' \mid s,a\right) V^{\pi_k}\left(s'\right)\right\}$. Since the cardinality of the state space $\mathcal{S}$ is S, the cost of evaluating the argument of the maximum is $O(S)$. Thus the full maximization problem's cost is $O(SA)$, and as we need to compute that for every state, the overall complexity is indeed $O(S^2A)$. We obtain our result by multiplying complexity of each iteration with the sufficient value of $k = \ln(\frac{1}{(\epsilon(1-\gamma))})/\ln(\frac{1}{\gamma})$. □

5.2 General Policy Iteraion

We will now take a closer look at the General Policy Iteration algorithm from Algorithm 2. Specifically, we will consider this algorithm with greedy policy improvement step. From now on, when referring to policy iteration, we will be referring specifically to General Policy Iteration with a greedy policy improvement step.

Similarly as with FVI algorithm from Algorithm 10, we will delve deeper into the runtime and computational complexity of policy iteration algorithm. We will first prove a theorem, which shows that policy iteration eliminates one sub-optimal action choice at some state after respective number of iterations, which depends only on the discount horizon γ.

Theorem 5.4 ((Szepesvári, 2020)). *For any sub-optimal policy π_0, if we perform policy iteration, then there exists a state $s \in \mathcal{S}$, such that, for any $k > \ln \frac{1}{(1-\gamma)} / \ln \frac{1}{\gamma}$*

$$\pi_k(s) \neq \pi_0(s).$$

This result will be proved later, as we first need some additional definitions. For policies π and π', we define function $A : \Pi \times \Pi \to \mathbb{R}^S$ by the following formula:

$$A(\pi', \pi) = \mathcal{T}_{\pi'} V^{\pi} - V^{\pi}.$$

Function $A(\cdot, \cdot)$ have some useful property.

Proposition 5.5 ((Szepesvári, 2020)). *Let π^* be any optimal policy. Then for any policy π we have:*

$$A(\pi, \pi^*) \leq 0.$$

Proof. Let π be any policy. Then:

$$A(\pi, \pi^*) = \mathcal{T}_{\pi} V^{\pi^*} - V^{\pi^*} = \mathcal{T}_{\pi} V^* - V^* = \mathcal{T}_{\pi} V^* - \mathcal{T} V^* \leq 0,$$

where the inequality follows from Remark 1.3. □

We will also need the following two propositions.

Proposition 5.6 ((Szepesvári, 2020)). *For any two policies π and π' we have:*

$$V^{\pi'} - V^{\pi} = (I - \gamma P^{\pi'})^{-1} A(\pi', \pi).$$

Proof. As we know from Lemma 1.17 that the matrix $(I - \gamma P^{\pi'})$ is invertible, we have:

$$V^{\pi'} = (I - \gamma P^{\pi'})^{-1} R_{\pi'},$$

thus we have:

$$V^{\pi'} - V^{\pi} = (I - \gamma P^{\pi'})^{-1}(R_{\pi'} - (I - \gamma P^{\pi'}) V^{\pi}) = (I - \gamma P^{\pi'})^{-1} A(\pi', \pi).$$

□

Proposition 5.7 ((Szepesvári, 2020)). *For any policy π and any $x \in \mathbb{R}^{\mathcal{S}}$, we have:*

$$\|(I - \gamma P_{\pi})^{-1} x\|_{\infty} \leq \frac{1}{(1-\gamma)} \|x\|_{\infty}.$$

Proof. From Lemma 1.16, we know that for any policy π and any vector $x \in \mathbb{R}^{\mathcal{S}}$:

$$\|P_\pi x\|_\infty \leq \|x\|_\infty.$$

Thus, in particular we have:

$$\|P_\pi (P_\pi x)\|_\infty \leq \|P_\pi x\|_\infty \leq \|x\|_\infty,$$

and analogously we get for any $i \in \mathbb{N}_+$:

$$\|P_\pi^i x\|_\infty \leq \|x\|_\infty. \tag{12}$$

From Lemma 1.17 we know that $(I - \gamma P_\pi)^{-1} = \sum_{i=0}^{\infty} \gamma^i P_\pi^i$, thus, from the triangle inequality and Lemma 1.16, we get:

$$\|(I - \gamma P_\pi)^{-1} x\|_\infty \leq \sum_{i=0}^{\infty} \gamma^i \|P_\pi^i x\|_\infty \leq \frac{1}{(1-\gamma)} \|x\|_\infty.$$

□

We are now ready to prove Theorem 5.4.

Proof. of Theorem 5.4
We have:

$$\begin{aligned} 0 \leq -A(\pi_k, \pi^*) &= (I - \gamma P_{\pi_k})(V^* - V^{\pi_k}) \\ &= (V^* - V^{\pi_k}) - \gamma P_{\pi_k}(V^* - V^{\pi_k}) \\ &\leq V^* - V^{\pi_k}, \end{aligned}$$

where in the first inequality we used Proposition 5.5, the first equality follows from Proposition 5.6, and the last inequality holds because $V^* \geq V^{\pi_k}$ and matrix P_{π_k} is stochastic and thus have only non-negative entries. Let π^* be an arbitrary optimal policy. We have:

$$\begin{aligned} \|A(\pi_k, \pi^*)\|_\infty &\leq \|V^* - V^{\pi_k}\|_\infty \\ &\leq \gamma^k \|V^* - V^{\pi_0}\|_\infty \\ &= \gamma^k \|V^{\pi^*} - V^{\pi_0}\|_\infty \\ &= \gamma^k \|(I - \gamma P_{\pi_0})^{-1}(-A(\pi_0, \pi^*))\|_\infty \\ &\leq \frac{\gamma^k}{1-\gamma} \|A(\pi_0, \pi^*)\|_\infty, \end{aligned} \tag{13}$$

where in the second inequality we used Theorem 1.14, in the last equality we used Proposition 5.6 and for the last inequality we used Proposition 5.7.
Now let s be the state that satisfies $-A(\pi_0, \pi^*)(s) = \|A(\pi_0, \pi^*)\|_\infty$. Note that such state exists because the state space $\mathcal{S}$ is finite. We have:

$$\begin{aligned} 0 \leq -A(\pi_k, \pi^*)(s) &\leq \|A(\pi_k, \pi^*)\|_\infty \\ &\leq \frac{\gamma^k}{1-\gamma} \|A(\pi_0, \pi^*)\|_\infty \\ &= \frac{\gamma^k}{1-\gamma} \left(-A(\pi_0, \pi^*)(s)\right), \end{aligned}$$

where for the first inequality we used Proposition 5.5 and for the third inequality we used Equation (13).

Now, take $k > \ln \frac{1}{(1-\gamma)} / \ln \frac{1}{\gamma}$, and note that then we have $\frac{\gamma^k}{1-\gamma} < 1$. Note that, since π_0 is a sub-optimal policy, we have:

$$A(\pi_0, \pi^*) = \mathcal{T}_{\pi_0} V^* - V^* \neq 0,$$

since otherwise $\mathcal{T}_{\pi_0} V^* = \mathcal{T} V^*$, and thus π_0 would be an optimal policy (as greedy with respect to V^*). Thus, we have $0 < \|A(\pi_0, \pi^*)\|_\infty = -A(\pi_0, \pi^*)(s)$ and hence:

$$-A(\pi_k, \pi^*)(s) \leq \frac{\gamma^k}{1-\gamma}(-A(\pi_0, \pi^*)(s)) < -A(\pi_0, \pi^*)(s).$$

From the definition of $A(\cdot, \cdot)$, we get:

$$\begin{aligned} 0 &< A(\pi_k, \pi^*)(s) - A(\pi_0, \pi^*)(s) \\ &= \mathcal{T}_{\pi_k} V^*(s) - \mathcal{T}_{\pi_0} V^*(s) \\ &= [R_{\pi_k}(s) + \gamma P_{\pi_k}(s) V^*] \\ &\quad - [R_{\pi_0}(s) + \gamma P_{\pi_0}(s) V^*] . \end{aligned}$$

Hence we have:

$$R_{\pi_k}(s) + \gamma P_{\pi_k}(s) V^* > R_{\pi_0}(s) + \gamma P_{\pi_0}(s) V^*.$$

Thus, $\pi_k(s) \neq \pi_0(s)$, which ends the proof. $\square$

We can now prove lower bounds for the runtime and computational complexity of General Policy Iteration with greedy step.

Theorem 5.8 (Runtime and computational bounds for GPI with greedy step (Szepesvári, 2020))**.** *Let $k' = \lceil \ln \frac{1}{(1-\gamma)} / \ln \frac{1}{\gamma} \rceil$. Then performing General Policy Iteration with greedy step, starting from an arbitrary initial policy π_0, after at most*

$$k = k'(SA - S) = O\left((SA - S) \ln \tfrac{1}{(1-\gamma)} / \ln \tfrac{1}{\gamma}\right)$$

iterations, the policy π_k is optimal. In particular, policy iteration computes an optimal policy with at most $O\left((S^4 A + S^3 A^2) \ln \frac{1}{(1-\gamma)} / \ln \frac{1}{\gamma}\right)$ arithmetic and logic operations.

Proof. From Theorem 5.4, we know that after every $O\left(\ln \frac{1}{(1-\gamma)} / \ln \frac{1}{\gamma}\right)$ iterations, policy iteration continues to eliminate suboptimal actions until none remain. However, for any state there are at most $A - 1$ suboptimal actions in that state, and as there are S states, this procedure can be done at most $S(A-1)$ times. Thus, after at most $k'S(A-1)$ iterations, the policy iteration algorithm produces the optimal policy.

Note that for given policy π_k its value can be computed by solving the following equation:

$$V^{\pi_k} = (I - \gamma P_{\pi_k})^{-1} R_{\pi_k}.$$

Given policy π_k, the cost of computing R_{π_k} is $O(S)$, since π_k is deterministic. Similarly, given access to transition function P, the cost of computing matrix P_{π_k} is $O(S^2)$. Now, the complexity of inverting a $S \times S$ matrix using eg. Gauss–Jordan elimination is $O(S^3)$, and the complexity of multiplying $S \times S$ matrix with S-dimensional vector is $O(S^2)$. Thus, the cost of computing V_{π_k}, given π_k is $O(S^3)$. To make the greedy step and compute the policy π_{k+1}, greedy with respect to value function V^{π_k}, we need to compute for every state $s \in \mathcal{S}$:

$$\underset{a \in \mathcal{A}}{\arg\max} \left\{ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P\left(s' \mid s, a\right) V^{\pi_k}\left(s'\right) \right\}.$$

The cost of evaluating $\left\{R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P\left(s' \mid s, a\right) V^{\pi_k}\left(s'\right)\right\}$ for given action a is $O(S)$, and since there are A actions and we need to find the argmax for every state $s \in \mathcal{S}$, the cost of computing policy π_{k+1} given V^{π_k} is $O(S^2 A)$. By summing the costs of the two steps involved in policy iteration and then multiplying this sum by $k'(SA - S)$, we arrive at the final result. $\square$

6 Multi-Armed Bandits

The problem of Multi-armed Bandits represents a fundamental and well-studied area of reinforcement learning, in which the system operates within a fixed state space, $S = \{s\}$. In this scenario, the objective is to maximize the cumulative reward over a finite time horizon N. Unlike in classic reinforcement learning objective, there is no discount factor (see Definition 1.1 of MDP). The term "Multi-armed Bandit" is derived from the classic "one-armed bandit", a type of gambling machine commonly found in casinos. In a one-armed bandit, the player pulls a lever to receive a random payout, which follows a particular probability distribution.

The Multi-armed Bandit problem generalises this concept by introducing n different levers, or "arms", each associated with its own probability distribution of rewards. The challenge lies in the fact that these reward distributions are generally unknown to the player, necessitating a strategy to estimate them through exploration and exploitation. The act of pulling an arm, or selecting an action a, is an experimental step designed to gather information about the associated reward distribution.

The essence of the problem is to balance exploration, which involves trying different arms to understand their reward distributions, with exploitation, which means selecting the arm that currently seems to offer the highest reward based on the information gathered so far. This balance is crucial for achieving the highest possible cumulative reward within the given time horizon N.

6.1 Formal Definition of Multi-Armed Bandits Problem

Definition 6.1 ((Lattimore and Szepesvári, 2017)). *Multi-armed Bandit is a collection of distributions $\nu = \{P_a : a \in \mathcal{A}\}$, where $\mathcal{A}$ is a set of available actions.*

At each time step $t \in \{1, 2, ..., N\}$, an agent chooses action $a_t \in \mathcal{A}$ and receives reward R_t which is sampled from distribution P_{a_t}. The agent's goal is to maximise cumulative reward:

$$S_N = \sum_{t=1}^{N} R_t.$$

The policy of an agent in a Multi-armed Bandit problem is defined similarly as in general reinforcement learning problem. However, due to the fact that there is only one state, the policy can be simplified. Instead of being a function $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$, mapping states to probability distributions over actions, the policy in the Multi-armed Bandit context is treated as a probability distribution directly over the actions: $\pi \in \mathcal{P}(\mathcal{A})$. At each time step, the policy changes, thus throughout learning we obtain sequence of policies: $\pi_0, \pi_1, ..., \pi_N$.

Often it is assumed that Multi-armed Bandit's collection of distributions ν comes from one family, such as: Bernoulli, uniform, Gaussian with known variance, Gaussian with unknown variance, etc. One particularly important family of distributions that is frequently assumed is the subgaussian family.

Definition 6.2 (Subgaussian random variable). *A random variable X is σ-subgaussian, if for all $\lambda \in \mathbb{R}$*

$$\mathbb{E}\left[\exp\left(\lambda X\right)\right] \leq \exp\left(\frac{\lambda^2\sigma^2}{2}\right).$$

A subgaussian random variable is a type of random variable that exhibits tail behavior similar to that of a Gaussian distribution but is more general. This means that the probability of large deviations from the mean is bounded in a similar way to a Gaussian distribution, making them useful for analysis in contexts where we desire controlled and predictable tail behavior. Subgaussian random variables provide a powerful tool for theoretical analysis because they allow for the use of concentration inequalities, which are crucial in deriving performance guarantees for various algorithms in the bandit setting.

Lemma 6.1 ((Lattimore and Szepesvári, 2017)). *Let X and Y be independent, σ_X and σ_Y-subgaussian random variables, respectively. Then:*

1. $\mathbb{E}[X] = 0$ *and* $\mathbb{V}[X] \leq \sigma_X^2$,
2. cX *is* $\|c\|\sigma_X$*-subgaussian for all* $c \in \mathbb{R}$,
3. $X + Y$ *is* $\sqrt{\sigma_X^2 + \sigma_Y^2}$ *-subgaussian.*

6.2 Regret

Let ν be Multi-armed Bandit, and $P_a \in \nu$. We define:

$$\mu_a(\nu) = \int_{-\infty}^{\infty} x dP_a(x),$$

This integral represents the expected value of the reward obtained from arm a. Now, we define μ^* as the highest mean among all arms:

$$\mu^*(\nu) = \max_{a \in \mathcal{A}} \mu_a.$$

Definition 6.3 (Regret (Lattimore and Szepesvári, 2017)). *Let ν be a Multi-armed Bandit and π be a policy. The regret of π on ν is*

$$R_n(\pi, \nu) = n\mu^*(\nu) - \mathbb{E}\left[\sum_{t=1}^{n} R_t\right].$$

Regret measures the difference between the reward we could have obtained by always selecting the best possible arm and the expected reward actually obtained by following a given policy π. Minimizing regret is essential in bandit problems because it implies that the policy π is effectively learning to identify and exploit the best arm over time.

Regret can be understood as the cost of exploration. If the policy always chose the optimal arm, the regret would be zero. However, since the policy must explore to learn which arm is optimal, it incurs regret when suboptimal arms are chosen.

Additionally, note that minimizing regret is equivalent to maximizing the cumulative reward S_n. Therefore, designing a policy to minimize regret indirectly ensures that the cumulative reward is maximized, which is the ultimate goal in Multi-armed Bandit problems.

We introduce the definition of suboptimality gap of arm a, which will be very useful for bounding regret of certain algorithms:

Definition 6.4 (Suboptimality gap). *Let $\nu = \{P_i : i \in \mathcal{A}\}$ be Multi-armed Bandit and $a \in \mathcal{A}$. Suboptimality gap of ν at a is defined as:*

$$\Delta_a(\nu) := \mu^*(\nu) - \mu_a(\nu).$$

When ν is clear from the context, we will usually omit writing ν and use Δ_a, μ^*, μ_a instead.

In the following subsections, we will treat A_t as a random variable, representing the action chosen by the agent in step t.

6.3 Explore-Then-Commit

Explore-Then-Commit (ETC) is one of the simplest and most straightforward algorithms for solving Multi-Armed Bandits. The basic idea is to divide the learning process into two distinct phases: exploration and exploitation.

1. **Exploration phase**. During this phase, the agent pulls each arm a predetermined number of times m, where $1 \leq m \leq \frac{N}{k}$, with N being the total number of time steps and k being the number of arms. The parameter m determines how many times each arm is explored. It is crucial to choose m appropriately to balance the time spent exploring with the time available for exploiting the best arm.
2. **Exploitation phase**. After the exploration phase, the agent calculates the mean reward for each arm based on the samples collected. In each subsequent round, the agent pulls the arm with the highest mean reward, exploiting the knowledge gained during the exploration phase.

Let $\hat{\mu}_i(s)$ be the average of rewards, that an agent have received by pulling an arm i up to time step s, precisely:

$$\hat{\mu}_i(s) := \frac{\sum_{t=1}^{s} \mathbb{I}(A_t = i) R_t}{\sum_{t=1}^{s} \mathbb{I}(A_t = i)}.$$

The algorithm is given in Algorithm 11.

Algorithm 11 Explore-Then-Commit (ETC))

```
Require: m
  for $t = 1, 2, ..., N$ do
      $A_t = \begin{cases} (t \mod k) + 1, \text{if } t \leq mk. \\ \arg\max_{i \in \{1,2,...,k\}} \hat{\mu}_i(mk), \text{otherwise.} \end{cases}$
  end for
```

Theorem 6.2 ((Lattimore and Szepesvári, 2017)). *Let $\nu = \{P_a : a \in \{1, 2, ..., k\}\}$ be Multi-Armed Bandit, such that P_a is 1-subgaussian, for every $a \in \{1, 2, ..., k\}$. When following ETC with $1 \leq m \leq \frac{N}{k}$:*

$$R_N \leq m \sum_{i=1}^{k} \Delta_i + (N - mk) \sum_{i=1}^{k} \Delta_i \exp\left(-\frac{m\Delta_i^2}{4}\right).$$

Let us examine the upper bound on regret in Theorem 6.2 more closely. The first term represents the regret accumulated during the exploration phase. Since the agent might be pulling suboptimal arms during exploration, this term accumulates the suboptimality gap Δ_i for each arm i scaled by m.

The second term captures the regret during the exploitation phase, in which the agent commits to pulling the arm with the highest observed mean reward. However, there is a probability of error in identifying the optimal arm, especially if the exploration phase is short.

6.4 Upper Confidence Bound (UCB)

The Upper Confidence Bound (UCB) algorithm is a more sophisticated and adaptive approach for solving Multi-armed Bandit problems compared to the Explore-Then-Commit strategy. One of the primary advantages of UCB over ETC is its dynamic balancing of exploration and exploitation. While ETC divides the process into distinct phases, UCB continuously adapts its strategy based on the rewards observed so far, making it more efficient in many scenarios.

The UCB algorithm works by maintaining an upper confidence bound for the estimated reward of each arm. At each time step, the arm with the highest upper confidence bound is selected. The algorithm can be formally described as given in Algorithm 12.

Algorithm 12 Upper Confidence Bound (Lai and Robbins, 1985)

```
Require: $\delta$
  for each arm $i = 1, 2, ..., k$ do
      pull arm $i$ once to initialize estimates
  end for
  for $t = k+1, k+2, ..., N$ do
      for each arm $i = 1, 2, ..., k$ do
          compute $ucb\_score_i(t) = \hat{\mu}_i(t) + \sqrt{\frac{2\log\frac{1}{\delta}}{\sum_{s=1}^{t}\mathbb{I}(A_s=i)}}$
      end for
      $A_t = \arg\max_i ucb\_score_i(t)$
  end for
```

Intuition. The intuition behind UCB is to ensure that each arm is explored sufficiently while favoring arms that have shown higher rewards in the past. The term $\sqrt{\frac{2\log 1/\delta}{\sum_{s=1}^{t}\mathbb{I}(A_s=i)}}$ represents the exploration bonus, which decreases as the number of pulls of arm i increases. This means that arms with fewer pulls will have a higher confidence bound, encouraging exploration. Conversely, arms with higher average rewards $\mu_i(t)$ will be favored during exploitation.

Formalization. The following lemma is crucial for deriving intuition for UCB.

Lemma 6.3. *Let $(X_i - \mu)$ be sequence of σ-subgaussian random variables, where $\mu \in \mathbb{R}$, $\hat{\mu} := \frac{\sum_{t=1}^{n} X_t}{n}$ and $\delta \in (0,1)$. Then with probability at least $1-\delta$:*

$$\mu \geq \hat{\mu} + \sqrt{\frac{2\sigma^2 \log\frac{1}{\delta}}{n}}.$$

Let us consider interacting with a 1-subgaussian Multi-armed Bandit. UCB is derived by creating an auxiliary metric for each arm which is:

$$ucb\_score_i(t) := \hat{\mu}_i(t) + \sqrt{\frac{2\log\frac{1}{\delta}}{\sum_{t=1}^{s}\mathbb{I}(A_t = i)}}.$$

If the arm with highest score, say arm i, was pulled many times, then ucb_score_i will be approximately μ_i, as the second term goes to 0 relatively quickly. For every other arm $j \neq i$, we have:

$$\mu_i(t) \approx \hat{\mu}_i(t) + \sqrt{\frac{2\log\frac{1}{\delta}}{\sum_{s=1}^{t}\mathbb{I}(A_s = i)}} \geq \hat{\mu}_j(t) + \sqrt{\frac{2\log\frac{1}{\delta}}{\sum_{s=1}^{t}\mathbb{I}(A_s = j)}} \leq \hat{\mu}_j(t) \approx \mu_j(t).$$

Theorem 6.4 (Regret of UCB (Lattimore and Szepesvári, 2017))**.**
Let $\nu = \{P_a : a \in \{1, 2, ..., k\}\}$ be Multi-Armed Bandit, such that P_a is 1-subgaussian, for every $a \in \{1, 2, ..., k\}$. When following UCB with $\delta = \frac{1}{N^2}$, the regret is bounded by:

$$R_N \leq 8\sqrt{Nk\log N} + 3\sum_{i=1}^{k}\Delta_i.$$

6.5 UCB Without Defined Horizon

In practical scenarios, we often encounter situations where we do not have access to prior knowledge about the horizon N, or the horizon is effectively infinite. Regret bound for the classic UCB, presented in the previous subsection, relies on a specific confidence level that depends on N (see Theorem 6.4), which can be a limitation. However, it is possible to adapt UCB to work without this information while still maintaining a regret bound of the same order. Algorithm is presented in Algorithm 13. The only difference from the original UCB is the use of a dynamic confidence level, $\delta_t = \frac{1}{1+t\log^2 t}$, which allows the algorithm to function without knowing N. The main idea behind UCB without a defined horizon is to dynamically adjust the confidence intervals as more data is collected. The exploration bonus term $\sqrt{\frac{2\log(1+t\log^2 t)}{\sum_{s=1}^{t}\mathbb{I}(A_s=i)}}$ ensures that even as t grows, the algorithm continues to explore each arm sufficiently. This dynamic adjustment allows the algorithm to adapt to an indefinite or unknown horizon, maintaining a balance between exploration and exploitation.

Algorithm 13 Upper Confidence Bound without horizon

```
Require: $\delta$
  for each arm $i = 1, 2, ..., k$ do
      pull arm $i$ once to initialize estimates
  end for
  for $t = k + 1, k + 2, ..., N$ do
      for each arm $i = 1, 2, ..., k$ do
          compute $ucb\_score_i(t) = \hat{\mu}_i(t) + \sqrt{\frac{2\log(1+t\log^2 t)}{\sum_{s=1}^{t}\mathbb{I}(A_s=i)}}$
      end for
      $A_t = \arg\max_i ucb\_score_i(t)$
  end for
```

Theorem 6.5 (Regret of UCB without horizon (Lattimore & Szepesvári, 2020))**.** *Let $\nu = \{P_a : a \in \{1, 2, ..., k\}\}$ be Multi-Armed Bandit, such that P_a is 1-subgaussian, for every $a \in \{1, 2, ..., k\}$. There exist a universal constant $C \in \mathbb{R}$, such that when following UCB without horizon, the regret in step n is bounded by:*

$$R_n \leq C\sum_{i=1}^{k}\Delta_i + 2\sqrt{Cnk\log n}.$$

We see that this algorithm's regret is of order $\mathcal{O}(\sqrt{nk\log n})$, similarly to the original UCB, without the need for knowledge of N.

6.6 Minimax Optimal Strategy in the Stochastic Case (MOSS)

The Minimax Optimal Strategy in the Stochastic case (MOSS) algorithm, introduced in Audibert and Bubeck (2009), aims to improve the worst-case regret of the Upper Confidence Bound (UCB) algorithm by setting an optimal confidence level. MOSS achieves this by removing the $\log(N)$ factor present in the classic UCB regret bound, leading to a more favorable performance in certain scenarios. However, one limitation of the MOSS algorithm is that it depends on the horizon N. The algorithm is presented in Algorithm 14.

Algorithm 14 MOSS (Minimax Optimal Strategy in the Stochastic case)

```
Require: $\delta$
  for each arm $i = 1, 2, ..., k$ do
      pull arm $i$ once to initialize estimates
  end for
  for $t = k + 1, k + 2, ..., N$ do
      for each arm $i = 1, 2, ..., k$ do
          compute $moss\_score_i(t) = \hat{\mu}_i(t) + \sqrt{\frac{\max\left(0, \log\left(\frac{N}{kT_i(t-1)}\right)\right)}{\sum_{s=1}^{t-1} \mathbb{I}(A_s = i)}}$
      end for
      $A_t = \arg\max_i moss\_score_i(t)$
  end for
```

Theorem 6.6 (Regret of MOSS (Lattimore and Szepesvári, 2017))**.** *Let $\nu = \{P_a : a \in \{1, 2, ..., k\}\}$ be Multi-Armed Bandit, such that P_a is 1-subgaussian, for every $a \in \{1, 2, ..., k\}$. When following MOSS, the regret is bounded by:*

$$R_N \leq 49\sqrt{kN} + \sum_{i=1}^{k} \Delta_i.$$

7 Finite-Horizon MDPs

In many real-world scenarios, decisions have to be made within a set time period. For example, when planning a trip, managing a project, or playing a game, there is a clear time limit for making choices. In this section, we will switch to finite-horizon Markov Decision Processes which formalize this setting. This framework is similar, but slightly different than the episodic task scenarios discussed in Section 1.1. In episodic tasks the episode ends, when one of terminal states is encountered. In finite-horizon there are no terminal states, and the episode ends after some fixed number of steps H. Moreover, in finite-horizon MDPs, rewards are not discounted because the goal is to maximize the total expected reward over a fixed number of time steps. As a result, there is no need to balance the importance of immediate and future rewards.

Most of the results discussed in this chapter are based on (Szepesvári, 2020). Specifically, the proof of Theorem 7.8 relies heavily on this work, but certain steps were not fully detailed and some constants in the concentration inequalities were incorrectly specified. We have addressed these gaps by clarifying the steps and providing corrected constants.

To define a finite-horizon MDP, we can leverage the Definition 1.1 of the MDP $M = (\mathcal{S}, \mathcal{A}, P, r, \gamma, \mu)$, with $\gamma = 1$ (no discounting), and additionaly specify the finite-horizon $H \geq 1$. Thus, finite-horizon MDP can be seen as the tuple $M = (\mathcal{S}, \mathcal{A}, P, r, H, \mu)$. We will assume that the state space $\mathcal{S}$ and the action space $\mathcal{A}$ are finite, and the reward function r is deterministic and bounded in the $[0, 1]$ interval.

In finite-horizon MDP the learner interacts with the environment over episodes $k = 1, 2, ..., K$ of length $H \geq 1$ each. For any $h = 1, 2, ..., H$, we will denote by $S_h^{(k)}$ the state visited by the agent in episode k and time step h. Similarly, we define $A_h^{(k)}$ to be the action taken by the agent in episode k and time step h. Thus, the data collected by the learner in episode k is as follows:

$$S_0^{(k)},\, A_0^{(k)},\, r(S_0^{(k)}, A_0^{(k)}),\, S_1^{(k)}, ..., S_{H-1}^{(k)},\, A_{H-1}^{(k)},\, r(S_{H-1}^{(k)}, A_{H-1}^{(k)}),\, S_H^{(k)}.$$

As we transition to a finite horizon MDP, it is essential to revisit and adjust some of the core concepts discussed previously to align with the new setting. Most of the intuitions are preserved, but some changes in the definitions are necessary. Below, we will redefine key concepts, introduce some new ones, and establish some notions for convenience in proofs to ensure they are appropriate for a finite horizon MDP.

Similarly as before, policy $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ is a mapping from states to distributions over actions. For any policy π and any state s, we will define $P_\pi(s) \in \mathbb{R}^{\mathcal{S}}$ to be the vector such that:

$$(P_\pi(s))_{s'} = \sum_{a \in \mathcal{A}} \pi(a \mid s) P(s' \mid s, a).$$

We denote by $r_\pi : \mathcal{S} \to \mathbb{R}$ a function such that, for given state s, we have:

$$r_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) r_a(s).$$

For finite-horizon MDPs we compute the value function of a given policy π recursively as follows:

$$V_{h,P}^\pi \in \mathbb{R}^{\mathcal{S}} : V_{h,P}^\pi(s) = r_\pi(s) + \langle P_\pi(s), V_{h+1,P}^\pi \rangle,\ 0 \leq h \leq H-1$$
$$V_{H,P}^\pi \in \mathbb{R}^{\mathcal{S}} : V_{H,P}^\pi(s) = 0.$$

Thus $V^{\pi}_{h,P}$ can be seen as the expected sum of rewards obtained from stage h onwards, when following policy π in finite-horizon MDP with transitions P.

We will denote by $V_k = \sum_{h=0}^{H-1} r(S_h^{(k)}, A_h^{(k)})$ the reward collected by the agent in episode k.

The goal of the agent is to minimize its cumulative regret R_K defined as follows:

$$R_K := \sum_{k=1}^{K} \left(V^*_{0,P}(S_0^{(k)}) - V_k \right),$$

where $V^*_{0,P}$ is the optimal value function, i.e., $V^*_{0,P}(s) = \sup_\pi \left\{ V^\pi_{0,P}(s) \right\}$. Intuitively, cumulative regret measures the difference between the reward actually received by the learning agent and the reward that could have been received by following the optimal policy from the begining. Many theoretical algorithms aim to achieve sublinear cumulative regret, meaning that it grows slower than linearly with respect to the number of episodes.

Similarly as in Definition 1.3 for infinite-horizon MDPs, we can define the optimal policy for finite-horizon MDPs as follows:

Definition 7.1 (Optimal Policy for Finite-Horizon MDP). *In a finite-horizon MDP, a policy π is optimal if its value function is equal to the optimal value function; that is, for any $s \in \mathcal{S}$:*

$$V^\pi_{0,P}(s) = V^*_{0,P}(s).$$

It can be proved that for any finite-horizon Markov Decision Process, there always exists a deterministic optimal policy. This is formalized in Theorem 7.1, for the proof see Proposition 4.4.3 in (Puterman, 2014).

Theorem 7.1. *For any finite-horizon MDP with finite state space $\mathcal{S}$ and finite action space $\mathcal{A}$ there exists a deterministic optimal policy.*

7.1 Upper Confidence Reinforcement Learning (UCRL)

We will now analyse the Upper Confidence Reinforcement Learning (UCRL) algorithm (Auer and Ortner, 2006), which is designed for online learning. In the context of reinforcement learning, this means the algorithm operates in an unknown environment and learns about it by actively collecting data through interaction. The UCRL algorithm aims to minimize cumulative regret without prior knowledge of the true transition dynamics of a finite-horizon Markov Decision Process. To achieve this, it estimates these dynamics based on the data collected through exploration. However, seeking favorable transitions alone is insufficient for achieving low regret; the algorithm must also ensure it collects valuable rewards during the exploration phase. This necessitates balancing the exploration-exploitation trade-off, similarly as in multi-armed bandits problems. The intuition behind the UCRL algorithm is to use the collected data to form confidence set around the transition probabilities, ensuring with high probability that the true transition probabilities fall within these sets. For data collection, the algorithm employs the strategy known as the "optimism principle" to explore the environment effectively while collecting high-valued rewards. It does so by computing the optimal policy based on the optimistic estimate of the true transition dynamics, derived from the estimated confidence set. Note that for large state and action spaces, this computation can be infeasible in practice, making UCRL algorithm more suitable for problems with smaller state-action spaces. The complete algorithm is detailed in Algorithm 15. It is necessary for the proofs to assume that the policy π_k can be computed deterministically given previous data in each episode k. To handle ties in the argmax, some deterministic tie-breaking mechanism should be implemented, ensuring that repeated applications of the algorithm on the same data will yield identical policies.

Algorithm 15 UCRL algorithm

```
for episode $k = 1, ..., K$ do
    Compute confidence set $C_{k,\delta}$
    Sample the initial state $S_0^{(k)}$
    Compute the optimistic model $\tilde{P}^{(k)} = \arg\max_{P \in C_{k,\delta}} V^*_{0,P}(S_0^{(k)})$
    Compute deterministic policy $\pi_k = \arg\max_\pi V^\pi_{0,\tilde{P}^{(k)}}(S_0^{(k)})$ in some deterministic way
    Use policy $\pi_k$ to interact with the MDP and observe data
        $S_0^{(k)}, A_0^{(k)}, r(S_0^{(k)}, A_0^{(k)}), S_1^{(k)}, ..., S_{H-1}^{(k)}, A_{H-1}^{(k)}, r(S_{H-1}^{(k)}, A_{H-1}^{(k)}), S_H^{(k)}$
end for
```

To estimate the transition probabilities of the MDP, we should first establish certain definitions. For any state-action pair (s, a), let $P_a^{(k)}(s) \in \mathbb{R}^{\mathcal{S}}$ be a vector such that:

$$(P_a^{(k)}(s))_{s'} = \frac{N_k(s, a, s')}{1 \vee N_k(s, a)}$$

for any state s', where $a \vee b = \max(a, b)$ and

$$N_k(s, a) = \sum_{k' < k} \sum_{h < H} \mathbb{I}(S_h^{(k')} = s, A_h^{(k')} = a)$$

denotes number of visitations of state-action pair (s, a) prior to episode k, as well as

$$N_k(s, a, s') = \sum_{k' < k} \sum_{h < H} \mathbb{I}(S_h^{(k')} = s, A_h^{(k')} = a, S_{h+1}^{(k')} = s')$$

denotes number of visitations of state-action-state triple (s, a, s') prior to episode k. Thus, $P_a^{(k)}(s)$ can be seen as an estimate of the transition probabilities from state-action pair (s, a), calculated in episode k, based on the observed transitions from previous episodes.

We also define the confidence sets:

$$C_{k,\delta} = \left\{ P : \forall s, a \; \|P_a^{(k)}(s) - P_a(s)\|_1 \leq \beta_\delta(N_k(s, a)) \right\},$$

which are dependent on a given function $\beta_\delta : \mathbb{N} \to (0, \infty)$.

We denote the true transition function of the MDP by P^* to distinguish it from other estimated transition functions.

Now we want to choose function β_δ such that $P^* \in C_{k,\delta}$ with at least $1 - \delta$ probability, where δ is some small positive real number, which specifies the desired probability. However, we seek for a confidence set $C_{k,\delta}$ that is not trivial, i.e., not excessively large.

We will now choose a specific β_δ. This is a subject of the following lemma.

Lemma 7.2 ((Szepesvári, 2020)). *Let $\beta_\delta(0) = 1$ and $\beta_\delta(n) = \sqrt{8\frac{S\ln(2)+ln(n(n+1)SA/\delta)}{n}}$ for $n \in \mathbb{Z}_+$. Define for every $k = 1, ..., K$ the confidence set $C_{k,\delta}$ as follows:*

$$C_{k,\delta} = \left\{P : \forall_{s,a} \|P_a^{(k)}(s) - P_a(s)\|_1 \leq \beta_\delta(N_k(s,a))\right\}.$$

Then, with probability at least $1 - \delta$, we have $P^* \in C_{k,\delta}$ for every $k = 1, ..., K$.

For the proof we will also need the following lemma, which provides the so-called Hoeffding's inequality.

Lemma 7.3 (Hoeffding's inequality, (Lattimore and Szepesvári, 2017)). *Let $X_1, ..., X_n$ be independent real-valued random variables, such that for each $t = 1, ..., n$ there exist some $a_t \leq b_t$, such that $X_t \in [a_t, b_t]$ almost surely. Then for every $\epsilon > 0$:*

$$\mathbb{P}\left[\sum_{t=1}^n (X_t - \mathbb{E}X_t) \geq \epsilon\right] \leq \exp\left(\frac{-2\epsilon^2}{\sum_{t=1}^n (b_t - a_t)^2}\right).$$

We will also need a following simple proposition.

Proposition 7.4. *Let $p \in \mathbb{R}^{\mathcal{S}}$ be an arbitrary vector. We have:*

$$\|p\|_1 = \max_{x\in\{\pm1\}^{\mathcal{S}}} \langle p, x\rangle$$

Proof. Note that $\langle p, x\rangle$ is maximal if for non-negative p_i we take $x_i = 1$ and for negative p_i we take $x_i = -1$, but this corresponds exactly to $\|p\|_1$. □

We can now prove Lemma 7.2.

Proof. Fix a state-action pair (s, a). We will denote by $S_u \in \mathcal{S}$ the state following the state-action pair (s, a) after u^{th} time the state action pair (s, a) was visited during the whole procedure of Algorithm 15. Let n be the number of times (s, a) was visited.
For $n \geq 0$ we define a vector $P_{n,a}(s) \in \mathbb{R}^{\mathcal{S}}$, such that for any state s':

$$(P_{n,a}(s))_{s'} = \begin{cases} \frac{1}{n}\sum_{u=1}^n \mathbb{I}(S_u = s') & \text{for } n \geq 1 \\ 0 & \text{for } n = 0. \end{cases}$$

Note that in particular we have $P_{N_k(s,a),a}(s) = P_a^{(k)}(s)$, which follows from the definition of $P_a^{(k)}(s)$. As the random variable $N_k(s, a)$ takes values in the set of natural numbers, it is sufficient to prove that for all $n \geq 0$ with probability at least $1 - \delta$ we have:

$$\forall_{s,a} \|P_{n,a}(s) - P_a^*(s)\|_1 \leq \beta_\delta(n).$$

For $n = 0$ this inequality holds trivially, since for any state-action pair (s, a) we have:

$$\|P_a^*(s)\|_1 \leq 1 = \beta_\delta(0).$$

From now on we will consider the case $n \geq 1$. From Proposition 7.4 we have:

$$\|P_{n,a}(s) - P_a^*(s)\|_1 = \max_{x\in\{\pm1\}^{\mathcal{S}}} \langle P_{n,a}(s) - P_a^*(s), x\rangle. \tag{14}$$

For any fixed $x \in \{\pm1\}^{\mathcal{S}}$, we have:

$$\langle P_{n,a}(s) - P_a^*(s), x\rangle = \frac{1}{n}\sum_{u=1}^n \sum_{s'\in S} x_{s'} \left(\mathbb{I}(S_u = s') - (P_a^*(s))_{s'}\right) = \frac{1}{n}\sum_{u=1}^n \Delta_{u,x}$$

where $\Delta_{u,x} = \sum_{s' \in S} x_{s'} \left(\mathbb{I}(S_u = s') - (P_a^*(s))_{s'}\right)$. It is straightforward to see that $(\Delta_{u,x})_{u=1}^n$ is a sequence of independent random variables, since the identity of the state following the state-action pair (s, a) does not depend on the time at which this state-action pair was observed.
We also have $\mathbb{E}[\Delta_u] = 0$ for any $u = 1, ..., n$, since $\mathbb{E}[\mathbb{I}(S_u = s')] = (P_a^*(s))_{s'}$, and $|\Delta_{u,x}| \leq 2$, since from the triangle inequality:

$$\begin{aligned}
|\Delta_{u,x}| &\leq \sum_{s' \in S} |x_{s'}| \mathbb{I}(S_u = s') + \sum_{s' \in S} |x_{s'}| \left(P_a^*(s)\right)_{s'} \\
&= \sum_{s' \in S} \mathbb{I}(S_u = s') + \sum_{s' \in S} \left(P_a^*(s)\right)_{s'} \\
&= 2.
\end{aligned}$$

Thus, we can use Lemma 7.3 with $\epsilon = \sqrt{8n \ln \frac{1}{\delta}}$ for sequence of independent random variables $\Delta_{u,x}$ which implies that:

$$\mathbb{P}\left(\sum_{u=1}^n \Delta_{u,x} \geq \sqrt{8n \ln \frac{1}{\delta}}\right) = \mathbb{P}\left(\frac{1}{n}\sum_{u=1}^n \Delta_{u,x} \geq \sqrt{\frac{8 \ln \frac{1}{\delta}}{n}}\right) \leq \delta. \tag{15}$$

We will now use a well-known principle called the union bound, which states that for any finite or countable set of events, the probability that at least one occurs is less or equal to the sum of their individual probabilities. Using obtained Equation (14) and noting that $|\{\pm 1\}^{\mathcal{S}}| = 2^{\mathcal{S}}$, we can can substitute δ with $\frac{\delta}{2^{\mathcal{S}}}$ in Equation (15), and take the union bound for events $A_x = \left\{\frac{1}{n}\sum_{u=1}^n \Delta_{u,x} \geq \sqrt{\frac{8 \ln \frac{2^{\mathcal{S}}}{\delta}}{n}}\right\}$ for all $x \in \{\pm 1\}^{\mathcal{S}}$, obtaining that with probability at least $1 - \delta$:

$$\|P_{n,a}(s) - P_a^*(s)\|_1 \leq \sqrt{8\frac{\ln \frac{2^{\mathcal{S}}}{\delta}}{n}} = \sqrt{8\frac{S\ln(2) + \ln(1/\delta)}{n}}.$$

To conclude the proof we need to take the union bound over $s \in \mathcal{S}$, $a \in \mathcal{A}$ and $n \geq 1$. Noting that $\sum_{n=1}^\infty \frac{\delta}{n(n+1)} = \sum_{n=1}^\infty (\frac{\delta}{n} - \frac{\delta}{n+1}) = \delta$ we can take the union bound over infinite set $\{n \in \mathbb{N}_+\}$. Hence, similarly as before, we obtain that with probability at least $1 - \delta$ for all $n \geq 1$, $s \in \mathcal{S}$, $a \in \mathcal{A}$ we have:

$$\|P_{n,a}(s) - P_a^*(s)\|_1 \leq \sqrt{8\frac{S\ln(2) + \ln(n(n+1)SA/\delta)}{n}} = \beta_\delta(n),$$

which ends the proof. □

We will now state the Azuma-Hoeffding inequality, which will be useful later. For this we need to define a sequence of random variables adapted to a filtration.

Definition 7.2 (Sequence of random variables adapted to a filtration)**.** *A sequence of random variables $\{X_t\}_{t \geq 0}$ is said to be adapted to a filtration $\mathbb{F} = (\mathcal{F}_t)$, if for each time t, the random variable X_t is measurable with respect to the σ-algebra $\mathcal{F}_t$.*

Lemma 7.5 (Azuma-Hoeffding inequality, (Lattimore and Szepesvári, 2017))**.** *Let $X_1, ..., X_n$ be a sequence of random variables adapted to a filtration $\mathbb{F} = (\mathcal{F}_t)_{t=0}^{t=n}$. Suppose that for $t = 1, ..., n$ there exist some $a_t \leq b_t$ such that $X_t \in [a_t, b_t]$ almost surely. Then for any $\epsilon > 0$*

$$\mathbb{P}\left(\sum_{t=1}^n X_t - \mathbb{E}\left[X_t | \mathcal{F}_{t-1}\right] \geq \epsilon\right) \leq \exp\left(\frac{-2\epsilon^2}{\sum_{t=1}^n (b_t - a_t)^2}\right).$$

We will also need a following well-known inequality:

Lemma 7.6. *Suppose that $x_1, x_2, ..., x_n$ are positive real numbers. Then:*

$$\frac{x_1 + ... + x_n}{n} \leq \sqrt{\frac{x_1^2 + ... + x_n^2}{n}}.$$

We will need a following lemma:

Lemma 7.7 ((Szepesvári, 2020))**.** *For any $y, z, h \in \mathbb{R}$ such that $z > h, y + z > h, h > 0$ we have:*

$$\sqrt{y + z - h} \leq \sqrt{z - h} + \frac{y}{2\sqrt{z - h}}.$$

Proof. Consider a function $f(x) = \sqrt{x - h}$. This function is concave on (h, ∞). Hence, for all y, z such that $z > h$ and $y + z > h$: $f(y + z) \leq f(z) + yf'(z)$ from which the thesis follows. □

We are ready to prove the main theorem of this section.

Theorem 7.8 (UCRL Regret, (Szepesvári, 2020))**.** *The cumulative regret produced by the UCRL algorithm satisfies with probability at least $1 - 3\delta$:*

$$R_K \leq 4c_\delta H\sqrt{SAHK} + 4c_\delta H^2 SA + H\sqrt{8HK \ln \frac{1}{\delta}} + H\sqrt{2K \ln \frac{1}{\delta}},$$

where $c_\delta = \sqrt{8S\ln(2) + 8\ln(HK(HK + 1)SA/\delta)}$.

Proof. Let π_k be the policy used by the UCRL algorithm in episode k. We will define an event $\varepsilon = \left\{P^* \in \bigcap_{k=1,...,K} C_{k,\delta}\right\}$. By Lemma 7.2 we know that $\mathbb{P}(\varepsilon) \geq 1 - \delta$. In the following we will assume that event ε has occured. We decompose the cumulative regret as follows:

$$\begin{aligned} R_K = \sum_{k=1}^{K} \left(V^*_{0,P^*}(S_0^{(k)}) - V_k\right) = & \underbrace{\sum_{k=1}^{K} \left(V^*_{0,P^*}(S_0^{(k)}) - V^*_{0,\tilde{P}_k}(S_0^{(k)})\right)}_{(I)} \\ & + \underbrace{\sum_{k=1}^{K} \left(V^{\pi_k}_{0,\tilde{P}_k}(S_0^{(k)}) - V^{\pi_k}_{0,P^*}(S_0^{(k)})\right)}_{(II)} \\ & + \underbrace{\sum_{k=1}^{K} \left(V^{\pi_k}_{0,P^*}(S_0^{(k)}) - V_k\right)}_{(III)}, \end{aligned} \tag{16}$$

where we have used the equality $V^*_{0,\tilde{P}_k}(S_0^{(k)}) = V^{\pi_k}_{0,\tilde{P}_k}(S_0^{(k)})$, which holds for every $k = 1, ..., K$, because we defined π_k as the optimal policy for the MDP with optimistic transition model $\tilde{P}_k$. Note that for every $k = 1, ..., K$ we have:

$$V^*_{0,P^*}(S_0^{(k)}) \leq V^*_{0,\tilde{P}_k}(S_0^{(k)}),$$

which follows from the choice of the optimistic model $\tilde{P}_k$ and the fact that $P^* \in C_{k,\delta}$. Thus, we can bound the first term in Equation (16) i.e.

$$(I) \leq 0. \tag{17}$$

This eliminates the dependence on the unknown optimal policy from the regret bound, justifying our use of the optimistic model.
Now we proceed to bounding the third term. Denote $X_k = V^{\pi_k}_{0,P^*}(S_0^{(k)}) - V_k$ and note that $\{X_k\}$ is not an i.i.d. sequence, because clearly the policy π_k is dependent on the data obtained up to episode k, and therefore we cannot use the Hoeffding's inequality to bound $\sum_{k=1}^K X_k$. However, if we define a filtration $\mathbb{F} = (\mathcal{F}_k)_{k=1}^{k=K}$ such that:

$$\mathcal{F}_k = \left\{S_0^{(l)}, A_0^{(l)}, r(S_0^{(l)}, A_0^{(l)}), S_1^{(l)}, ..., S_{H-1}^{(l)}, A_{H-1}^{(l)}, r(S_{H-1}^{(l)}, A_{H-1}^{(l)}), S_H^{(l)}\right\}_{l=1}^{k-1} \cup \left\{S_0^{(k)}\right\},$$

then we get $\mathbb{E}[V_k|\mathcal{F}_k] = V^{\pi_k}_{0,P^*}(S_0^{(k)})$ for each $k = 1, ..., K$. This holds because given previous data, the expected sum of reward collected in episode k is equal to the value function of policy π_k, which can be calculated deterministically given previous data. We also have $-H \leq X_k \leq H$, since, as we assume that rewards lie in the $[0,1]$ interval, both $V^{\pi_k}_{0,P^*}(S_0^{(k)})$ and V_k are non-negative and bounded by H. Hence, we can use the Azuma-Hoeffding inequality from Lemma 7.5 for sequence of random variables $X_1, ..., X_K$ and $\epsilon = H\sqrt{2K \ln \frac{1}{\delta}}$, to conclude that with probability at least $1 - \delta$

$$(III) = \sum_{k=1}^K X_k < H\sqrt{2K \ln \frac{1}{\delta}}. \tag{18}$$

Now it remains to bound the second term of the cumulative regret decomposition. For every $h = 0, ..., H$ and $k = 1, ..., K$ put:

$$\delta_h^{(k)} := V^{\pi_k}_{h,\tilde{P}^{(k)}}(S_h^{(k)}) - V^{\pi_k}_{h,P^*}(S_h^{(k)}).$$

Let $\mathbb{F} = (\mathcal{F}_{h,k})_{k=1,h=0}^{k=K,h=H-1}$ be a filtration, such that $\mathcal{F}_{h,k}$ contains all the data up to episode k and time step h, i.e.:

$$\mathcal{F}_{h,k} = \left\{S_0^{(1)}, A_0^{(1)}, r(S_0^{(1)}, A_0^{(1)}), ..., S_{h-1}^{(k)}, A_{h-1}^{(k)}, r(S_{h-1}^{(k)}, A_{h-1}^{(k)}), S_h^{(k)}\right\}.$$

From the definition of value function for finite-horizon MDP we get:

$$\begin{aligned}
\delta_h^{(k)} &= r_{\pi_k}(S_h^{(k)}) + \langle \tilde{P}^{(k)}_{\pi_k}(S_h^{(k)}), V^{\pi_k}_{h+1,\tilde{P}^{(k)}}\rangle - r_{\pi_k}(S_h^{(k)}) - \langle P^*_{\pi_k}(S_h^{(k)}), V^{\pi_k}_{h+1,P^*}\rangle \\
&= \langle \tilde{P}^{(k)}_{\pi_k}(S_h^{(k)}), V^{\pi_k}_{h+1,\tilde{P}^{(k)}}\rangle - \langle P^*_{\pi_k}(S_h^{(k)}), V^{\pi_k}_{h+1,P^*}\rangle \\
&= \langle \tilde{P}^{(k)}_{\pi_k}(S_h^{(k)}), V^{\pi_k}_{h+1,\tilde{P}^{(k)}}\rangle - \langle P^*_{\pi_k}(S_h^{(k)}), V^{\pi_k}_{h+1,P^*}\rangle \\
&+ \langle P^*_{\pi_k}(S_h^{(k)}), V^{\pi_k}_{h+1,\tilde{P}^{(k)}}\rangle - \langle P^*_{\pi_k}(S_h^{(k)}), V^{\pi_k}_{h+1,\tilde{P}^{(k)}}\rangle \\
&= \left(\langle \tilde{P}^{(k)}_{\pi_k}(S_h^{(k)}) - P^*_{\pi_k}(S_h^{(k)}), V^{\pi_k}_{h+1,\tilde{P}^{(k)}}\rangle\right) + \left(\langle P^*_{\pi_k}(S_h^{(k)}), V^{\pi_k}_{h+1,\tilde{P}^{(k)}} - V^{\pi_k}_{h+1,P^*}\rangle\right).
\end{aligned}$$

Since rewards are in $[0,1]$ we also have $\|V^{\pi_k}_{h+1,\tilde{P}^{(k)}}\|_\infty \leq H$. Therefore, we obtain the following estimate:

$$\begin{aligned}
\langle \tilde{P}^{(k)}_{\pi_k}(S_h^{(k)}) - P^*_{\pi_k}(S_h^{(k)}), V^{\pi_k}_{h+1,\tilde{P}^{(k)}}\rangle &= \langle \tilde{P}^{(k)}_{A_h^{(k)}}(S_h^{(k)}) - P^*_{A_h^{(k)}}(S_h^{(k)}), V^{\pi_k}_{h+1,\tilde{P}^{(k)}}\rangle \\
&\leq \|P^*_{A_h^{(k)}}(S_h^{(k)}) - \tilde{P}^{(k)}_{A_h^{(k)}}(S_h^{(k)})\|_1 H,
\end{aligned}$$

where equality is a consequence of the fact that π_k is a deterministic policy and thus: $\pi_k(A_h^{(k)}|S_h^{(k)}) = 1$. Then from the triangle inequality, the definition of the confidence set $C_{k,\delta}$, and since both P^* and $\tilde{P}^(k)$ belongs to $C_{k,\delta}$ we get:

$$\begin{aligned}\|P^*_{A_h^{(k)}}(S_h^{(k)}) - \tilde{P}^{(k)}_{A_h^{(k)}}(S_h^{(k)})\|_1 &\le \|P^*_{A_h^{(k)}}(S_h^{(k)}) - P^{(k)}_{A_h^{(k)}}(S_h^{(k)})\|_1 + \|P^{(k)}_{A_h^{(k)}}(S_h^{(k)}) - \tilde{P}^{(k)}_{A_h^{(k)}}(S_h^{(k)})\|_1 \\ &\le 2\beta_\delta(N_k(S_h^{(k)}, A_h^{(k)})).\end{aligned}$$

Note that we also have:

$$\langle P^*_{\pi_k}(S_h^{(k)}), V^{\pi_k}_{h+1,\tilde{P}^{(k)}} - V^{\pi_k}_{h+1,P^*}\rangle = \mathbb{E}[\delta^{(k)}_{h+1}|\mathcal{F}_{h,k}].$$

For convenience we will denote $\eta^{(k)}_{h+1} = \mathbb{E}[\delta^{(k)}_{h+1}|\mathcal{F}_{h,k}] - \delta^{(k)}_{h+1}$. Combining the obtained results we get:

$$\delta_h^{(k)} \le 2H\beta_\delta(N_k(S_h^{(k)}, A_h^{(k)})) + \delta^{(k)}_{h+1} + \eta^{(k)}_{h+1}.$$

Telescoping the above bound for $h = 0, ..., H-1$ and noting that, from the definition of the value function, we have $\delta_H^{(k)} = 0$, we obtain:

$$\delta_0^{(k)} \le \sum_{h=1}^{H} \eta_h^{(k)} + 2H\sum_{h=0}^{H-1}\beta_\delta(N_k(S_h^{(k)}, A_h^{(k)})).$$

Note that for $k = 1, ..., K$ and $h = 1, ..., H$ we have

$$\mathbb{E}[\eta^{(k)}_{h+1}|\mathcal{F}_{h,k}] = \mathbb{E}\left[\left(\mathbb{E}[\delta^{(k)}_{h+1}|\mathcal{F}_{h,k}] - \delta^{(k)}_{h+1}\right)|\mathcal{F}_{h,k}\right] = 0.$$

Since $|\delta_h^{(k)}| \le H$, we have $-2H \le \eta_h^{(k)} \le 2H$. Thus, from Azuma-Hoeffding inequality from Lemma 7.5, with $\epsilon = H\sqrt{8HK\ln\frac{1}{\delta}}$, we conclude that with probability at least $1-\delta$ the following condition holds:

$$\sum_{k=1}^{K}\sum_{h=1}^{H}\eta_h^{(k)} \le H\sqrt{8HK\ln\frac{1}{\delta}}.$$

Thus, we obtain that with probability at least $1-\delta$:

$$\begin{aligned}(II) = \sum_{k=1}^{K}\delta_0^{(k)} &= \sum_{k=1}^{K}\sum_{h=1}^{H}\eta_h^{(k)} + 2H\sum_{k=1}^{K}\sum_{h=0}^{H-1}\beta_\delta(N_k(S_h^{(k)}, A_h^{(k)})) \\ &\le H\sqrt{8HK\ln\frac{1}{\delta}} + 2H\underbrace{\sum_{k=1}^{K}\sum_{h=0}^{H-1}\beta_\delta(N_k(S_h^{(k)}, A_h^{(k)}))}_{(IV)}.\end{aligned} \tag{19}$$

Now it remains to bound the term (IV). Let us denote:

$$\begin{aligned}c_\delta &= \sqrt{8S\ln 2 + 8\ln HK(HK+1)SA/\delta}, \\ M_k(s,a) &= \sum_{h=0}^{H-1}\mathbb{I}(S_h^{(k)} = s, A_h^{(k)} = a).\end{aligned}$$

Then we have $N_k(s,a) = \sum_{i=1}^{k-1} M_i(s,a)$. Note that for $0 \le u \le HK$ the following condition holds:

$$\beta_\delta(u) \le \frac{c_\delta}{\sqrt{1 \vee u}}. \tag{20}$$

For $u = 0$ the inequality holds trivially, because

$$\beta_\delta(0) = 1 \le \sqrt{8S \ln 2} \le c_\delta.$$

For $1 \le u \le HK$ we have

$$\beta_\delta(u) = \sqrt{\frac{8S\ln 2 + 8\ln\frac{u(u+1)SA}{\delta}}{u}} \le \sqrt{\frac{8S\ln 2 + 8\ln\frac{HK(HK+1)SA}{\delta}}{u}} = \frac{c_\delta}{\sqrt{u}} = \frac{c_\delta}{\sqrt{1 \vee u}},$$

so Equation (20) also holds. Consequently we obtain:

$$\begin{aligned}(IV) = \sum_{k=1}^{K}\sum_{h=0}^{H-1} \beta_\delta(N_k(S_h^{(k)}, A_h^{(k)})) &\le c_\delta \sum_{k=1}^{K}\sum_{h=0}^{H-1} \frac{1}{\sqrt{1 \vee N_k(S_h^{(k)}, A_h^{(k)})}} \\ &= c_\delta \sum_{(s,a)\in S\times A} \sum_{k=1}^{K}\sum_{h=0}^{H-1} \frac{\mathbb{I}(S_h^{(k)} = s, A_h^{(k)} = a)}{\sqrt{1 \vee N_k(s,a)}} \\ &= c_\delta \sum_{(s,a)\in S\times A} \sum_{k=1}^{K} \frac{M_k(s,a)}{\sqrt{1 \vee \left(\sum_{i=1}^{k-1} M_i(s,a)\right)}}.\end{aligned}$$

Note that in our case, we have $M_k(s,a) \le H$ for all $k = 1, ..., K$. Thus:

$$c_\delta \sum_{(s,a)\in S\times A} \sum_{k=1}^{K} \frac{M_k(s,a)}{\sqrt{1 \vee (\sum_{i=1}^{k-1} M_i(s,a))}} \le c_\delta \sum_{(s,a)\in S\times A} \sum_{k=1}^{K} \frac{M_k(s,a)}{\sqrt{1 \vee (\sum_{i=1}^{k} M_i(s,a) - H)}}.$$

For every state-action pair (s,a) let us denote $k(s,a)$ to be the smallest integer such that $\sum_{i=1}^{k(s,a)} M_i(s,a) > H$, and if no such integer exists, set $k(s,a) = 0$. Then, as all M_k are non-negative, we have $\sum_{i=1}^{l} M_i(s,a) > H$ for any integer $l \ge k(s,a)$. Consequently:

$$\sum_{k=1}^{K} \frac{M_k(s,a)}{\sqrt{1 \vee (\sum_{i=1}^{k} M_i(s,a) - H)}} \le \sum_{k=1}^{k(s,a)-1} M_k(s,a) + M_{k(s,a)}(s,a) + \sum_{k=k(s,a)+1}^{K} \frac{M_k(s,a)}{\sqrt{\sum_{i=1}^{k} M_i(s,a) - H}}. \tag{21}$$

From our definition of $k(s,a)$ we have:

$$\sum_{k=1}^{k(s,a)-1} M_k(s,a) \le H.$$

To bound the last term in Equation (21) we use Lemma 7.7. Note that for any $k \ge k(s,a)+1$, applying Lemma 7.7 for $h = H$, $z = \sum_{i=1}^{k} M_i(s,a)$, $y = -M_k(s,a)$, we obtain the following estimate:

$$\frac{M_k(s,a)}{2\sqrt{\sum_{i=1}^{k} M_i(s,a) - H}} \le \sqrt{\sum_{i=1}^{k} M_i(s,a) - H} - \sqrt{\sum_{i=1}^{k-1} M_i(s,a) - H}.$$

Therefore we have:

$$\frac{M_K(s,a)}{2\sqrt{\sum_{i=1}^{K} M_i(s,a) - H}} \leq \sqrt{\sum_{i=1}^{K} M_i(s,a) - H} - \sqrt{\sum_{i=1}^{K-1} M_i(s,a) - H}$$

$$\frac{M_{K-1}(s,a)}{2\sqrt{\sum_{i=1}^{K-1} M_i(s,a) - H}} \leq \sqrt{\sum_{i=1}^{K-1} M_i(s,a) - H} - \sqrt{\sum_{i=1}^{K-2} M_i(s,a) - H}$$

$$\dots$$

$$\frac{M_{k(s,a)+1}(s,a)}{2\sqrt{\sum_{i=1}^{k(s,a)+1} M_i(s,a) - H}} \leq \sqrt{\sum_{i=1}^{k(s,a)+1} M_i(s,a) - H} - \sqrt{\sum_{i=1}^{k(s,a)} M_i(s,a) - H}.$$

Consequently, after summarizing the above inequalities we obtain:

$$\begin{aligned}\sum_{k=k(s,a)+1}^{K} \frac{M_k(s,a)}{\sqrt{\sum_{i=1}^{k} M_i(s,a) - H}} &\leq 2\sqrt{\sum_{i=1}^{K} M_i(s,a) - H} - 2\sqrt{\sum_{i=1}^{k(s,a)} M_i(s,a) - H} \\ &\leq 2\sqrt{\sum_{i=1}^{K} M_i(s,a) - H} \\ &\leq 2\sqrt{\sum_{i=1}^{K-1} M_i(s,a)} \\ &= 2\sqrt{N_K(s,a)},\end{aligned}$$

where the last inequality holds because $M_K(s,a) \leq H$. Going back to our main derivations we get:

$$\sum_{k=1}^{k(s,a)-1} M_k(s,a) + M_{k(s,a)}(s,a) + \sum_{k=k(s,a)+1}^{K} \frac{M_k(s,a)}{\sqrt{\sum_{i=1}^{k} M_i(s,a) - H}} \leq H + H + 2\sqrt{N_K(s,a)}.$$

Therefore:

$$\begin{aligned}c_\delta \sum_{(s,a)\in S\times A} \sum_{k=1}^{K} \frac{M_k(s,a)}{\sqrt{1 \vee (\sum_{i=1}^{k} M_i(s,a) - H)}} &\leq 2c_\delta HSA + 2c_\delta \sum_{(s,a)\in S\times A} \sqrt{N_K(s,a)} \\ &\leq 2c_\delta SA\sqrt{\sum_{(s,a)\in S\times A} \frac{N_K(s,a)}{SA}} + 2c_\delta HSA \\ &= 2c_\delta\sqrt{SA(H-1)(K-1)} + 2c_\delta HSA \\ &\leq 2c_\delta\sqrt{SAHK} + 2c_\delta HSA,\end{aligned}$$

where the second inequality follows from Lemma 7.6 and the equality follows from a simple observation that $\sum_{(s,a)\in S\times A} N_K(s,a) = (H-1)(K-1)$. Hence, we obtain:

$$(IV) = \sum_{k=1}^{K} \sum_{h=0}^{H-1} \beta_\delta(N_k(S_h^{(k)}, A_h^{(k)})) \leq 2c_\delta\sqrt{SAHK} + 2c_\delta HSA, \tag{22}$$

and thus, combining Equation (22) with Equation (19), we obtain that with probability at least $1-\delta$:

$$(II) \leq H\sqrt{8HK\ln\frac{1}{\delta}} + 4c_\delta H\sqrt{SAHK} + 4c_\delta H^2SA. \tag{23}$$

Finally, we can combine Equations (17), (18), and (23), taking the union bound over the event ε and two applications of Azuma-Hoeffding inequality from Lemma 7.5. We conclude that with probability at least $1-3\delta$:

$$\begin{aligned} R_K =& (I) + (II) + (III) \\ \leq& \underbrace{0}_{(I)} + \underbrace{H\sqrt{8HK\ln\frac{1}{\delta}} + 4c_\delta H\sqrt{SAHK} + 4c_\delta H^2SA}_{(II)} + \underbrace{H\sqrt{2K\ln\frac{1}{\delta}}}_{(III)} \\ =& 4c_\delta H\sqrt{SAHK} + 4c_\delta H^2SA + H\sqrt{8HK\ln\frac{1}{\delta}} + H\sqrt{2K\ln\frac{1}{\delta}}. \end{aligned}$$

□

7.2 Upper Confidence Bound Value Iteration and Minimax Lower Bound

In this subsection, we state the minimax lower bound on cumulative regret for Finite-Horizon MDPs and note that there exists an algorithm which matches it. The minimax lower bound describes the best possible performance that any algorithm can achieve under the worst-case scenario.

Theorem 7.9 (Cumulative Regret Minimax Lower Bound, (Domingues et al., 2021))**.** *Assume that we have $S \geq 6$, $A \geq 2$ and $H \geq 3\log_A S + 6$. Then for any algorithm $\mathbf{a}$, there exists an MDP $\mathcal{M}_{\mathbf{a}}$, such that for $K \geq HSA$ the cumulative regret R_K of algorithm $\mathbf{a}$ on MDP $\mathcal{M}_{\mathbf{a}}$ satisfies:*

$$R_K \geq \frac{1}{48\sqrt{6}}\sqrt{H^2SAK}.$$

An algorithm achieving this minimax lower bound, known as Upper Confidence Bound Value Iteration Bernstein-Friedman type, (UCBVI-BF), was demonstrated in (Azar et al., 2017).

Theorem 7.10 (UCBVI-BF regret)**.** *Assume that $K \geq H^2S^3A$ and $SA \geq H$. Consider a parameter $\delta > 0$. Then, with probability $1-\delta$, the cumulative regret of UCBVI-BF algorithm is bounded as follows:*

$$R_K \leq O\left(\sqrt{H^2SAK} \cdot L\right),$$

where $L = \ln(\frac{5H^2SAK}{\delta})$.

An important distinction is that UCBVI-BF allows the use of stage-dependent policies, i.e., $\pi : \mathcal{S} \times H \to \mathcal{P}(\mathcal{A})$, which is not the case for UCRL.

8 Continual Reinforcement Learning

In the domain of machine learning, models are typically trained in isolated environments to solve specific tasks within a fixed context. However, real-world applications require more dynamic and flexible learning approaches (Gama et al., 2014). To address this need, the field of continual learning has emerged (Parisi et al., 2019), which is defined as the problem of sequentially learning a model for a large number of tasks without forgetting the knowledge gained from previous tasks and with the ability to transfer knowledge to future tasks. This setup presents two main challenges: catastrophic forgetting, where the agent forgets how to perform previous tasks after learning new ones (McCloskey and Cohen, 1989; Goodfellow et al., 2013; Kirkpatrick et al., 2017), and transfer learning (Pan and Yang, 2009), which involves applying knowledge from one task to improve learning in related new tasks. In the context of RL, such a scenario occurs if the agent is trained on a sequence of tasks without access to previous environments. This approach is referred to as continual reinforcement learning (CRL) (Khetarpal et al., 2022).

This chapter deviates from the theoretical discussions of the previous sections. We focus on the empirical evaluation of Proximal Policy Optimization (PPO) (Schulman et al., 2017) algorithm within the continual reinforcement learning setting. PPO is a deep reinforcement learning algorithm that integrates policy gradient and actor-critic methods, which are covered in Section 4. In this chapter, which consists of our original work, we specifically examine the integration of PPO with various continual learning methods through extensive experimentation on the Continual World benchmark (Wołczyk et al., 2021). While the experiments in (Wołczyk et al., 2021) were conducted with off-policy algorithm Soft Actor-Critic (SAC) (Haarnoja et al., 2019), our goal was to investigate if the CL methods can be successfully adapted to an on-policy algorithm like PPO. For our experiments, we did not rely on any existing integrations; instead, we carefully combined PPO with continual learning methods, which involved considerable effort and complexity.

8.1 Continual World

Continual World (Wołczyk et al., 2021) is a benchmark specifically designed to evaluate RL agents on challenges that resemble properties of continual learning. It is constructed around a sequences of realistic robotic manipulation tasks from Meta-World (Yu et al., 2020). In our experiments, we use a sequence of the five following tasks:

- `faucet-close-v2`: the agent learns to close a faucet,
- `handle-press-side-v2`: the agent must press a handle down sideways,
- `push-v2`: the agent is tasked with pushing a puck to the goal location,
- `window-close-v2`: the agent needs to push close a window,
- `peg-unplug-side-v2`: the agent must unplug a peg sideways.

Visualizations of all the above tasks are available at `https://meta-world.github.io/figures/ml45.gif`.

8.2 Continual Learning Methods

The continual learning techniques we used in experiments are as follows.

- **Fine-Tuning**

  In our experiments, fine-tuning served as the baseline approach. With fine-tuning, the PPO agent learns each task sequentially without resetting the neural network. After completing a task, the network continues training on the next task. No specific continual learning techniques are applied in this baseline approach, providing a simple framework for assessing how well the agent adapts to new tasks while retaining previously learned information.

- **Elastic Weight Consolidation (EWC)** (Kirkpatrick et al., 2017)

  Elastic Weight Consolidation (EWC) was implemented to address catastrophic forgetting and forward transfer in our experiments. EWC introduces a penalty to the training process that prevents significant changes to important weights associated with previous tasks. It uses the Fisher Information Matrix (Kirkpatrick et al., 2017) to identify which weights are crucial for earlier tasks and keeps these weights stable while training on new tasks. This approach helps the agent retain knowledge from past tasks while still adapting to new ones.

- **PackNet** (Mallya and Lazebnik, 2018)

  PackNet was another technique we tested to handle catastrophic forgetting and forward transfer. It works by pruning, or removing, parts of the neural network after each task, which makes the network more efficient for the current task. It then retrains the remaining parts for new tasks while keeping the old weights unchanged. This method allows the network to handle multiple tasks without overwriting what it has already learned.

8.3 Metrics

For training each task, we utilized the default reward functions of Meta-World environments (Yu et al., 2020). However, for evaluation, we adopted a binary success metric. This metric assesses success based on the distance of a task-relevant object to its target position. Specifically, the metric is defined as

$$\mathbb{I}_{\|o-g\|_2<\epsilon},$$

where o and g represent the 3D coordinates of the object and the goal, respectively, and ϵ is a small, task-specific threshold, specified for each of the tasks in Table 12 of (Yu et al., 2020).

To evaluate the effectiveness of our experiments, we used forward transfer and forgetting metrics, as defined in Wołczyk et al. (2021), for each task in a sequence. Intuitively, forward transfer of a task is defined as a normalized area between the training curve of a particular task (trained in a sequence) and the training curve of the reference (a single-task experiment), and forgetting is defined as the decrease in performance after the end of its training. We calculated these metrics in the following way.

- **Forward Transfer** Let $p_i^b(t) \in [0,1]$ be the *success rate* of the model on the reference of task i (a single-task experiment), and $p_i(t)$ be the *success rate* of the model on task i at time t. The forward transfer for the task i, denoted by FT_i, is

$$FT_i := \frac{\mathrm{AUC}_i - \mathrm{AUC}_i^b}{1 - \mathrm{AUC}_i^b},$$

where:

$$\mathrm{AUC}_i := \frac{1}{n}\sum_{k=1}^{n} p_i\left((i-1)\Delta + kt\right),$$

$$\mathrm{AUC}_i^b := \frac{1}{n}\sum_{k=1}^{n} p_i^b(kt).$$

In the above formulas, Δ is the number of training steps for each task in a sequence, t is the frequency of testing performance, that is we test the model every t steps, and $n := \Delta/t$. The average forward transfer for the entire sequence of tasks is:

$$FT = \frac{1}{N}\sum_{i=1}^{N} FT_i,$$

where N is the number of tasks in the sequence.

- **Forgetting** To quantify forgetting for task i, we measure the difference between the *success rate* on that task at the end of its training and the *success rate* on that task at the end of whole learning process, i.e.,

$$F_i = p_i(i \cdot \Delta) - p_i(N\Delta).$$

The average forgetting for the entire sequence of tasks is:

$$F = \frac{1}{N}\sum_{i=1}^{N} F_i.$$

8.4 Experiments

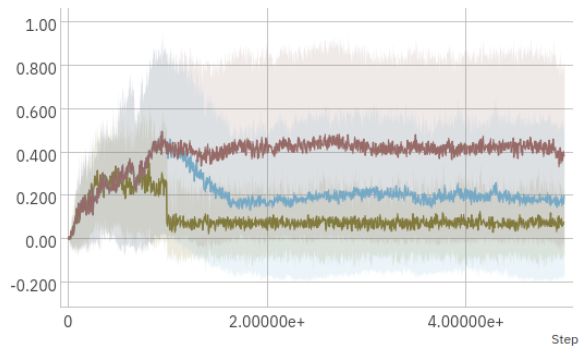
In this subsection, we present in detail our experimental approach to testing how well PPO performs with the integration of CL methods. All of the presented results are the original work of the authors.

As explained earlier we used sequence of $N = 5$ tasks. Each task was trained for $\Delta = 10^6$ steps of the agent in the environment. Every task i was evaluated every $t = 5 \cdot 10^3$ steps, during the entire learning process, but it was trained only during its specific interval $t \in [(i-1)\Delta, i\Delta]$. During each evaluation of task i, we ran 10 episodes, specifically for this task, using the current policy and measured the average *success rate* of those 10 episodes to approximate $p_i(t)$.

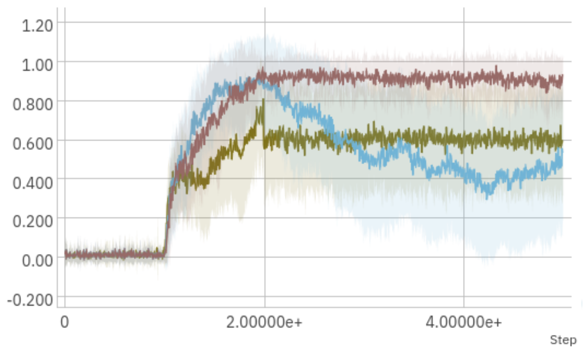
In Figure 1, we present the results of our experiments. The x-axis of these figures represents the current step of the learning process, while the y-axis presents the *success ratio* of the agent on a given task. Moreover, Table 1 presents the results of the discussed CL methods for Forgetting and Forward Transfer metrics.

Our results indicate that both Elastic Weight Consolidation (EWC) and PackNet techniques outperformed simple fine-tuning in terms of mitigating forgetting of previously learned tasks. This observation is evident from the figures and is further supported by Table 1. Comparing these results to Table 1 in Wołczyk et al. (2021), we observe a notably worse Forward Transfer for PackNet in our setting. We hypothesize that the observed decline in Forward Transfer performance for PackNet in our setting is due to its design being optimized for off-policy algorithms like SAC, which benefit from experience replay and efficient use of past data, while PPO, as an on-policy algorithm, does not take full advantage of PackNet's retraining mechanism.

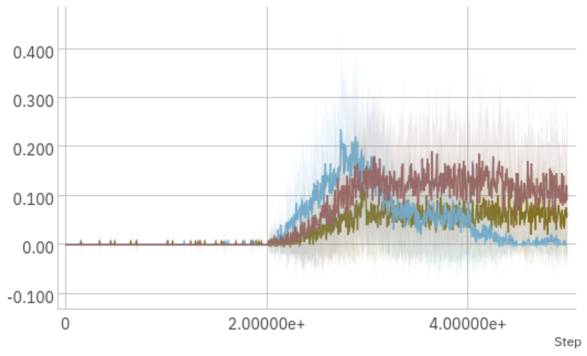
The code for the experiments is available at https://github.com/Teddy298/continualworld-ppo.



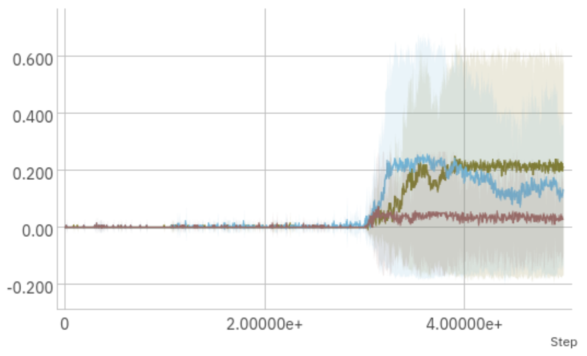
(a) faucet-close-v2



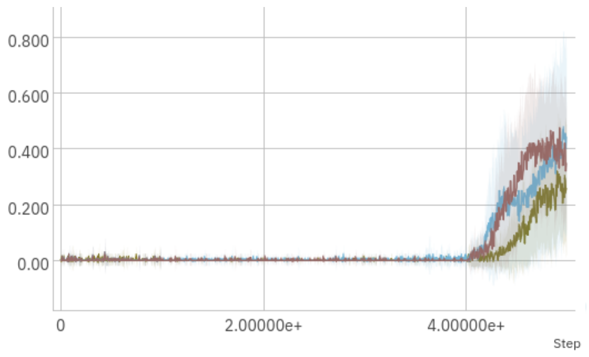
(b) handle-press-side-v2



(c) push-v2



(d) window-close-v2



(e) peg-unplug-side-v2

Figure 1: Comparison between different Continual Learning (CL) methods integrated with the Proximal Policy Optimization (PPO) algorithm. The tasks were trained during 10^6 time steps intervals, and their order is provided for training in alphabetical order, from (a) to (e). The x-axis represents the time step, while the y-axis shows the average *success rate*. Results are averaged over 20 random seeds. The three CL methods are represented by the following colors: (●) **Fine-tuning**, (●) **EWC**, (●) **PackNet**.

Method	Forgetting	Forward Transfer
Fine-Tuning	0.173	**0.089**
EWC	**0.023**	0.027
Packnet	0.038	-0.053

Table 1: Forgetting and Forward Transfer metrics for PPO combined with different CL methods. Results averaged over 20 random seeds. The observed behavior of PPO is similar to SAC, which was the algorithm evaluated in Wołczyk et al. (2021), across most metrics, except for the Forward Transfer metric for PackNet.

References

Agarwal, A., Jiang, N., and Kakade, S. M. (2019). Reinforcement learning: Theory and algorithms.

Audibert, J.-Y. and Bubeck, S. (2009). Minimax policies for adversarial and stochastic bandits. In *COLT*, pages 217–226.

Auer, P. and Ortner, R. (2006). Logarithmic online regret bounds for undiscounted reinforcement learning. In Schölkopf, B., Platt, J., and Hoffman, T., editors, *Advances in Neural Information Processing Systems*, volume 19. MIT Press.

Azar, M. G., Osband, I., and Munos, R. (2017). Minimax regret bounds for reinforcement learning.

Banach, S. (1922). Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fundamenta Mathematicae*, 3(1):133–181.

Birge, J. R. and Louveaux, F. (2011). *Introduction to stochastic programming.* Springer Science & Business Media.

Boyd, S. and Vandenberghe, L. (2004). *Convex optimization.* Cambridge university press.

Bubeck, S. et al. (2015). Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning*, 8(3-4):231–357.

Busoniu, L., Babuska, R., De Schutter, B., and Ernst, D. (2017). *Reinforcement learning and dynamic programming using function approximators.* CRC press.

Domingues, O. D., Ménard, P., Kaufmann, E., and Valko, M. (2021). Episodic reinforcement learning in finite mdps: Minimax lower bounds revisited. In Feldman, V., Ligett, K., and Sabato, S., editors, *Proceedings of the 32nd International Conference on Algorithmic Learning Theory*, volume 132 of *Proceedings of Machine Learning Research*, pages 578–598. PMLR.

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., and Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4):1–37.

Goodfellow, I. (2016). *Deep Learning.* MIT Press.

Goodfellow, I. J., Mirza, M., Xiao, D., Courville, A., and Bengio, Y. (2013). An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211.*

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S. (2019). Soft actor-critic algorithms and applications.

Khetarpal, K., Riemer, M., Rish, I., and Precup, D. (2022). Towards continual reinforcement learning: A review and perspectives. *Journal of Artificial Intelligence Research*, 75:1401–1476.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526.

Lai, T. L. and Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22.

Lattimore, T. and Szepesvári, C. (2017). Bandit algorithms.

Mallya, A. and Lazebnik, S. (2018). Packnet: Adding multiple tasks to a single network by iterative pruning.

McCloskey, M. and Cohen, N. J. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning.

Nocedal, J. and Wright, S. J. (1999). *Numerical optimization*. Springer.

Pan, S. J. and Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359.

Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., and Wermter, S. (2019). Continual lifelong learning with neural networks: A review. *Neural networks*, 113:54–71.

Puterman, M. L. (2014). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. In Solla, S., Leen, T., and Müller, K., editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press.

Szepesvári, C. (2020). Rl theory lecture notes. `https://rltheory.github.io/`. Accessed: 2024-05-01.

Szepesvári, C. (2022). *Algorithms for reinforcement learning*. Springer nature.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3):279–292.

Weng, L. (2018). Policy gradient algorithms. `https://lilianweng.github.io/posts/2018-04-08-policy-gradient/`. Accessed: 2024-05-01.

Wołczyk, M., Zając, M., Pascanu, R., Łukasz Kuciński, and Miłoś, P. (2021). Continual world: A robotic benchmark for continual reinforcement learning.

Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., and Levine, S. (2020). Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning*, pages 1094–1100. PMLR.