Chapter 1

Basics of Reinforcement Learning

In this chapter, we begin with motivating examples and then introduce the mathematical framework used to model decision-making problems in reinforcement learning. Let's start with a few real-world scenarios that illustrate the types of problems reinforcement learning aims to solve. These examples are described informally, without invoking RL-specific terminology. Our goal is to revisit them later and show how they can be modeled using the RL framework.

1.1 The Slot Machine Problem: Exploration vs Exploitation.

Imagine walking into a casino and finding 3 slot machines lined up in front of you. Each machine, when played, gives a random reward between 0 and 100. The machines differ in how they are programmed — some machines may tend to give higher rewards on average, while others may be worse.

Figure 1.1: An example of 3 slot machines. Some slot machines may give higher rewards on average.

You only have time to play a total of 1000 times, across all the machines combined, and your goal is to maximize the total reward you earn. This scenario raises an important question: how should you decide which machines to play?

- On the one hand, you would like to *explore* the different machines, by trying each of them enough times to estimate how good they are.
- On the other hand, you would also like to *exploit* your current knowledge by repeatedly playing the machines that seem to offer the best rewards so far.

Balancing this trade-off between *exploration* (gathering information) and *exploitation* (maximizing reward based on current knowledge) is one of the core challenges in reinforcement learning.

1.2 Learning to Play Chess: Credit Assignment

Let's consider another scenario. Suppose you are an AI agent learning how to play chess against a human opponent. At each turn, the agent observes the current *state* of the game — that is, the configuration of all pieces on the board — and decides which move to make.

Figure 1.2: An example chess board configuration. The agent observes the state of the board and decides on a move.

In principle, if the agent could determine the best move for every possible board configuration, it would be able to play optimally. However, learning what the best move is in each state is a highly challenging task for several reasons.

- First, the agent only receives feedback at the end of each game: it either wins, loses, or draws. The consequence of individual moves is not immediately known.
- Secondly, since chess involves rich strategic interactions between pieces, it can be hard to determine which specific actions, or sequences of actions, are responsible for this final outcome.

1.3 Reinforcement Learning Formulation

Both of the examples above, as well as many other real-world problems, can be cast in the reinforcement learning (RL) framework. In this framework, we model the interaction between two entities: **The agent**, which makes decisions by observing its current situation and choosing actions. **The environment**, which responds to the agent's actions by providing feedback in the form of rewards and updating the situation accordingly. The interaction proceeds as follows:

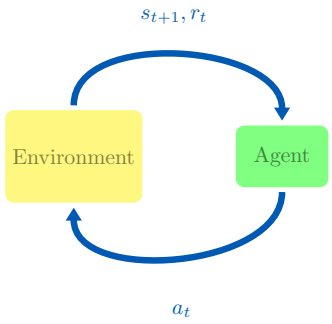

Figure 1.3: Interaction loop between the agent and environment in reinforcement learning. At each time step t, the agent observes the state s_t, takes an action a_t, and receives a new state s_{t+1} and reward r_t.

1. The agent begins in some *state* representing the current situation.
2. Based on the observed state, the agent chooses an *action* from the set of allowed actions.
3. The environment responds by:
   - Providing the agent with a *reward*, which reflects the immediate quality of the chosen action in the current state.
   - Updating the *state*, potentially in a stochastic (random) manner, based on the agent's action.
4. The agent observes the new state and repeats the process.

This sequence continues over a series of steps, often called an *episode*, which may end after a fixed number of steps or upon reaching a special terminal state. The goal of the agent is to learn a strategy that maximizes its expected cumulative reward over time. We now formalize this setup mathematically.

1.4 Markov Decision Process

The standard framework for modeling reinforcement learning problems is the *Markov Decision Process (MDP)*. This framework captures the sequential nature of decision-making under uncertainty.

Definition 1.1 (Markov Decision Process)**.** *A **finite-horizon Markov Decision Process (MDP)** is described by:*

1. State space S*: the set of all possible states.*
2. Action space A*: the set of all actions available to the agent.*
3. Transition function $T : S \times A \to \Delta(S)$[1]*: The transition function describes the next state given the current state and the action chosen by the agent. If the current state is s, and the agent takes action a, the next state s' is drawn from the distribution $T(s, a)$.*
4. Reward function $R : S \times A \to \Delta([0, 1])$*: The reward function describes the immediate reward given the current state and the action chosen by the agent. If the current state is s, and the agent takes action a, then the reward r is drawn from the distribution $R(s, a)$.*
5. Initial state $s_0 \in S$*: the starting state of each episode.*
6. Horizon $H \in \mathbb{N}$*: the number of time steps over which the agent interacts with the environment in an episode, before restarting from initial state s_0.*

Remark 1.2. *Instead of a fixed starting state s_0, one can assume an initial state distribution $\mu \in \Delta(S)$. This is a more general model, but assuming a fixed starting state simplifies analysis and is without loss of generality: a random initial state can be simulated by adding a dummy initial state and increasing the horizon by 1.*

We now formally define how an agent interacts with the environment in a given MDP. The interaction follows the following protocol:

1. The agent starts in state s_0.
2. For each time step $t = 0, 1, \ldots, H - 1$
   (a) The agent observes the state s_t
   (b) The agent selects an action $a_t \in A$ based on its policy.
   (c) The environment provides a reward $r_t \sim R(s_t, a_t)$.
   (d) The environment transitions to the next state $s_{t+1} \sim T(s_t, a_t)$.

[1] Given any finite set K, define $\Delta(K)$ as the set of all probability distributions over K, i.e.,

$$\Delta(K) := \{p : K \to [0, 1] : \sum_{k \in K} p(k) = 1\}.$$

The decisions made by the agent are specified by a *policy*. Formally, a policy is a function $\pi : S \to \Delta(A)$, where $\pi(s)$ is a distribution over actions given the current state s. That is, at each time t, the agent samples action $a_t \sim \pi(s_t)$.

Definition 1.3 (Goal)**.** *The goal of the agent is to find a policy that maximizes the expected cumulative reward over the horizon. In other words, the objective is to solve the optimization problem:*

$$\max_{\pi} \mathbb{E}_{T,R,\pi} \left[\sum_{t=0}^{H-1} r_t \right],$$

where the expectation is over the randomness in the transition function, reward function, and the policy.

In reality (similar to standard PAC learning settings), we only expect to find an approximate optimal policy with some constant probability.

1.5 Unrolling the MDP

In general, the transition function is stochastic, but to build intuition, it will also be useful to consider deterministic transition function. When the transitions are deterministic, we can visualize the unrolling of the MDP, into a tree (Figure 1.4).

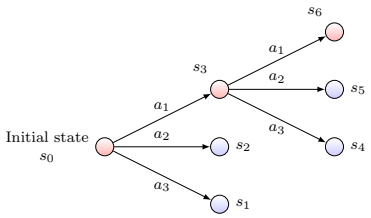

Figure 1.4: Unrolling of an MDP with states $S = \{s_0, s_1, s_2, \ldots, s_6\}$ and actions $A = \{a_1, a_2, a_3\}$. Each node corresponds to some state $s \in S$; edges correspond to some action $a \in A$, and transition to some state $s' \in S$ given by $T(s, a)$. The root of the tree is the initial state s_0.

1.6 Value Functions

To reason about how good a policy is, we define two central objects in reinforcement learning: the state value function and the state-action value function.

Definition 1.4 (Value Functions)**.** *Let $\pi : S \to \Delta(A)$ be a policy, and let H be the horizon. For each time step $h \in \{0, \ldots, H-1\}$, we define:*

- *The* state value function $V_h^\pi : S \to \mathbb{R}$ *as:*

$$V_h^\pi(s) := \mathbb{E}_{\pi,T,R}\left[\sum_{t=h}^{H-1} r_t \,\middle|\, s_h = s\right],$$

i.e., the expected total reward collected from time h to $H-1$, starting in state s and following policy π.

- *The* state-action value function $Q_h^\pi : S \times A \to \mathbb{R}$ *as:*

$$Q_h^\pi(s,a) := \mathbb{E}_{\pi,T,R}\left[\sum_{t=h}^{H-1} r_t \,\middle|\, s_h = s, a_h = a\right],$$

i.e., the expected total reward obtained by taking action a in state s at time h, and then following policy π until the end of the episode.

To simplify notation, we embed the time step h into the state (e.g., by replacing s with the pair (s,h)), and will often use $V^\pi(s)$ and $Q^\pi(s,a)$ to denote the value functions. We use V^* and Q^* to denote the value functions $V^*(s) = \max_\pi V^\pi(s)$ and $Q^*(s,a) = \max_\pi Q^\pi(s,a)$ and refer to the corresponding policy as the optimal policy π^*[2].

[2]even though optimal policy π^* is not unique, $V^*(s)$ and $Q^*(s,a)$ defined as $\max_\pi V^\pi(s)$ and $\max_\pi Q^\pi(s,a)$ are unique.