Chapter 2
Reinforcement Learning Background

This monograph focuses on transfer learning in reinforcement learning domains; some RL background is necessary. Our goal in this chapter is to briefly discuss RL concepts and notation used in this monograph so that the reader may understand later TL algorithms and experiments. Readers who desire a more comprehensive treatment of the reinforcement learning framework are referred to [Kaelbling et al. (1996)] and [Sutton and Barto (1998)].

2.1 Framing the Reinforcement Learning Problem

RL problems are typically framed in terms of Markov decision processes [Puterman (1994)] (MDPs). For the purposes of this monograph, “MDP” and “task” can be used interchangeably. An MDP is specified by the 4-tuple \( \langle S, A, T, R \rangle \). \( S \) is the set of states in the task. The available actions are enumerated in the set \( A \), although not every action may be possible in every state. The transition function, \( T : S \times A \mapsto S \), takes a state and an action and returns the state of the environment after the action is performed. Transitions may be non-deterministic, making the transition function a probability distribution function. The reward function, \( R : S \mapsto \mathbb{R} \), maps each state of the environment to a real-valued number which is the instantaneous reward achieved for reaching the state.

A learning agent senses its current state \( s \in S \). The agent’s observed state may be different from the true state if there is perceptual noise. If the task is episodic, the agent begins at a start state, \( s_{\text{initial}} \), and executes actions in the environment until it reaches a terminal state, \( s_{\text{final}} \), at which point it is returned to a start state. In some tasks where the agent is given no reward except when reaching a terminal state, it is convenient to think of the final states as goal states. An agent in an episodic task typically attempts to maximize the average reward per episode. In non-episodic tasks, the agent attempts to maximize the total reward, which may be discounted.\(^2\)

\(^1\) Some formulations also explicitly include a start state distribution, \( S_0 \), and a terminal state distribution, \( S_f \).

\(^2\) By utilizing a discount factor, \( \gamma \), the agent can weigh immediate rewards more heavily than future rewards, allowing it to maximize the expectation of an infinite sum of rewards.
Fig. 2.1 An agent interacts with an environment by sequentially selecting an action in an observed state, with the objective of maximizing an external reward signal.

Transfer learning methods are particularly relevant in MDPs that have a large or continuous state space, as these are the problems which are slow to learn *tabula rasa* and for which transfer may provide substantial benefits. Such tasks typically factor the state using *state variables*, so that $s = \langle x_1, x_2, \ldots, x_n \rangle$ (see Figure 2.1).

A policy, $\pi : S \mapsto A$, fully defines how a learner interacts with the environment by mapping perceived environmental states to actions. The success of an agent is determined by how well it maximizes the total reward it receives in the long run while acting under some policy $\pi$. An *optimal policy*, $\pi^*$, is a policy which does maximize the expectation of this value. Any reasonable learning algorithm attempts to modify $\pi$ over time so that the agent’s performance approaches that of $\pi^*$ in the limit.

Rather than computing a policy directly, many learning methods first estimate an *action-value function*, $Q : S \times A \mapsto \mathbb{R}$. $Q(s,a)$ is the expected return (or total reward) found when executing action $a$ from state $s$, and then greedily following the current policy thereafter. The current policy may be generated from $Q$ by simply selecting the action that has the highest value for the current state. Another possibility is to calculate the *value function*, $V : S \mapsto \mathbb{R}$, which maps states to the expected return. Value functions are typically learned when the agent has a model of the task: to generate a policy, the agent calculates $V(s')$ for all possible $s'$, which can only be done by knowing $T(s,a)$ for $a \in A$.

There are many possible approaches for learning a policy or action-value function, which will be discussed in Section 2.3. We first provide an overview of *function approximation* for RL, which will be crucial for learning in large tasks.