Finite Markov Decision Processes

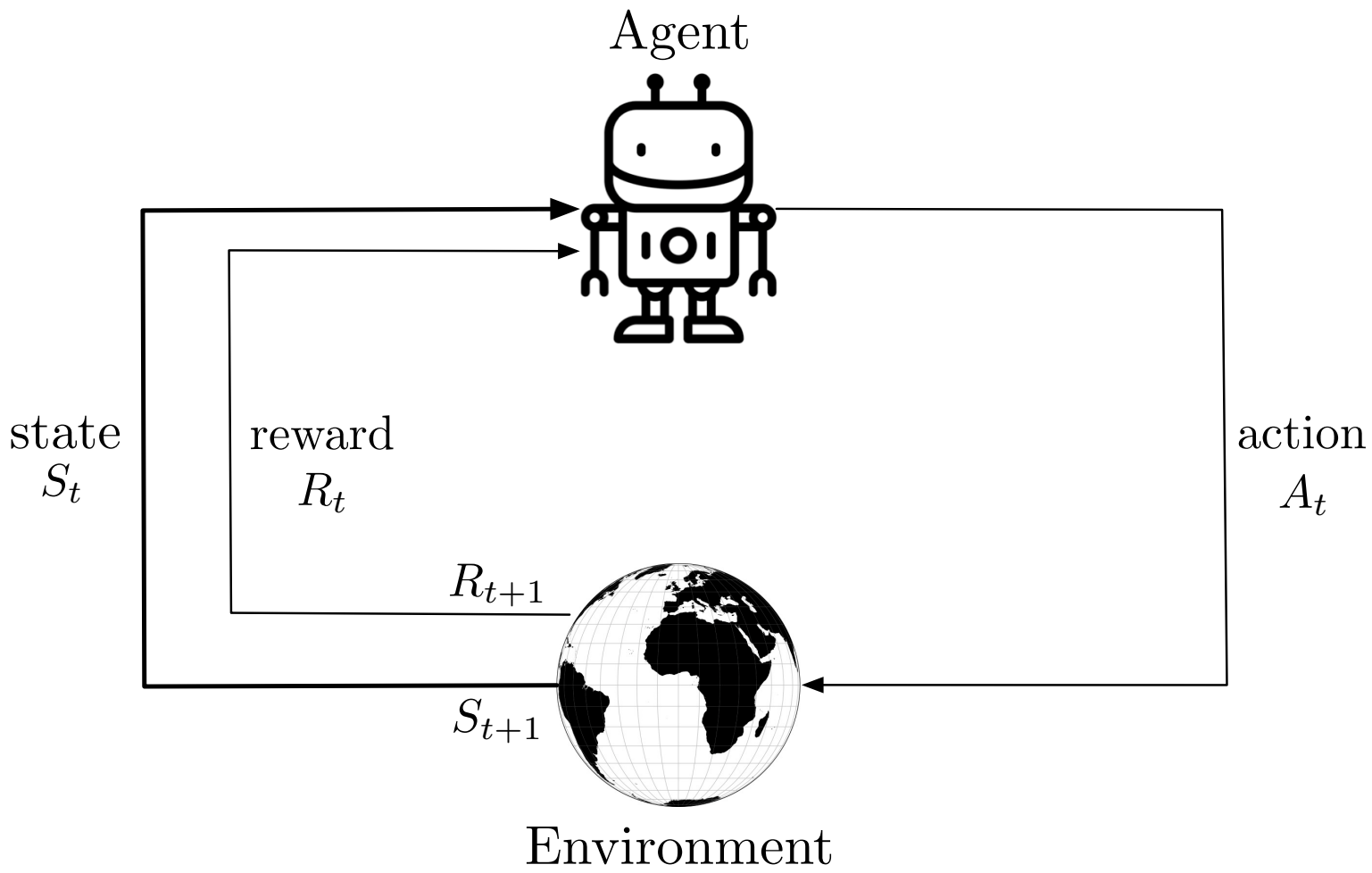
Reinforcement Learning

2019

Sepehr Maleki

University of Lincoln
School of Engineering

The Agent Environment Interface



(S. Maleki 2019)

Policy

- At each time step, the agent implements a mapping from states to probabilities of selecting each possible action.
- This mapping is called the agent's *policy* and is denoted by π_t.
- $\pi_t(a|s)$ is the probability that $A_t = a$ if $S_t = s$.
- $S_t \in \mathcal{S}$, where $\mathcal{S}$ is the set of possible states.
- $A_t \in \mathcal{A}(S_t)$, where $\mathcal{A}(S_t)$ is the set of actions available in state S_t
- Reinforcement learning methods specify how the agent changes its policy as a result of its experience.

(S. Maleki 2019)

Goals and Rewards

- At each time step, the reward is a simple number, $R_t \in \mathbb{R}$.
- Informally, the agent's goal is to maximise the total amount of reward it receives.
- This does not mean maximising the immediate reward, but cumulative reward in the long run.

(S. Maleki 2019)

Returns

- Denote the sequence of received rewards after time step t by $R_{t+1}, R_{t+2}, R_{t+3}, \ldots$.
- In general, we seek to maximise the expected return, G_t.
- The expected return is defined as some specific function of the reward sequence.
- In the simplest case the return is the sum of the rewards:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \ldots + R_T \ ,$$

where T is a final time step.

(S. Maleki 2019)

Discounting

When discounting is applied, the agent tries to select actions so that the sum of the discounted rewards it receives over the future is maximised:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \ ,$$

where γ is a parameter $0 \leq \gamma \leq 1$, called the discount rate.

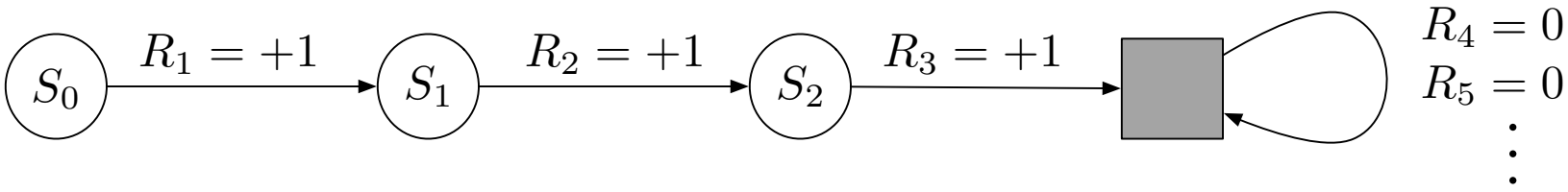
If $\gamma < 1$, the infinite sum has a finite value as long as the reward sequence R_k is bounded. If $\gamma = 0$, the agent is "myopic" in being concerned only with maximising immediate rewards.

(S. Maleki 2019)

Episodic and Continuing Tasks

Reinforcement learning tasks divide to those that:

1. the agent-environment interaction naturally breaks down into a sequence of separate episodes (episodic tasks).
2. the agent-environment interaction continues (continuing tasks).



(S. Maleki 2019)

The Markov Property (1)

- In the agent-environment interaction, we ideally like a state signal that summarises past sensations compactly.
- A state signal that succeeds in retaining all relevant information is said to be *Markov*, or to have *the Markov property*.

Assuming a finite number of states and reward values, consider how a general environment might respond at time $t+1$ to the action that the agent has taken at time t:

$$Pr\{R_{t+1} = r, S_{t+1} = s' | S_0, A_0, R_1, ..., S_{t-1}, A_{t-1}, R_t, S_t, A_t\} ,$$

for r, s', and all possible values of the past events:

$$S_0, A_0, R_1, ..., S_{t-1}, A_{t-1}, R_t, S_t, A_t$$

(S. Maleki 2019)

The Markov Property (2)

If the state signal has the Markov property, then the environment's response at $t+1$ depends only on the state and action representations at t:

$$p(s', r|s, a) = Pr\{R_{t+1} = r, S_{t+1} = s'|S_t = s, A_t = a\} \ ,$$

for r, s', S_t and A_t. In other words, a state is a Markov state iff:

$$Pr\{R_{t+1} = r, S_{t+1} = s'|S_t = s, A_t = a\} =$$
$$Pr\{R_{t+1} = r, S_{t+1} = s'|S_0, A_0, R_1, ..., S_{t-1}, A_{t-1}, R_t, S_t, A_t\} \ .$$

In this case, the environment and task as a whole are also said to have the Markov property.

Markov Decision Processes (1)

Definition: A reinforcement learning task that satisfies the Markov property is called a *Markov decision process*, or MDP.

If the state and action spaces are finite, then it is called a *finite Markov decision process* (finite MDP).

The dynamics of a finite MDP is fully specified by:

$$p(s', r|s, a) = Pr\{R_{t+1} = r, S_{t+1} = s'|S_t = s, A_t = a\} \ .$$

Therefore, one can compute anything else one might want to know about the environment.

(S. Maleki 2019)

Markov Decision Processes (2)

Expected rewards for state-action pairs:

$$r(s,a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a] = \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} p(s', r|s, a)$$

State-transition probabilities:

$$p(s'|s,a) = Pr\{S_{t+1} = s'|S_t = s, A_t = a\} = \sum_{r\in\mathcal{R}} p(s', r|s, a)$$

Expected rewards for state-action-next state triples:

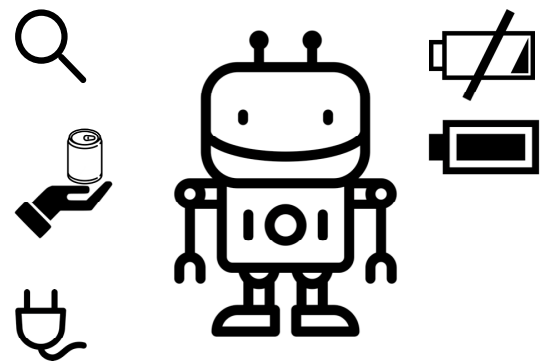
$$r(s,a,s') = \mathbb{E}[R_{t+1}|S_t = s, A_t = a, S_{t+1} = s'] = \frac{\sum_{r\in\mathcal{R}} rp(s', r|s, a)}{p(s'|s,a)}$$

(S. Maleki 2019)

Example: Recycling Robot (1)

The best way to find cans is to actively search for them, but this runs down the robot's battery, whereas waiting does not.

Whenever the robot is searching, the possibility exists that its battery will become depleted. In this case the robot must shut down and wait to be rescued (producing a low reward).



$$A(high) = \{\text{search, wait}\}$$
$$A(low) = \{\text{search, wait, recharge}\}$$

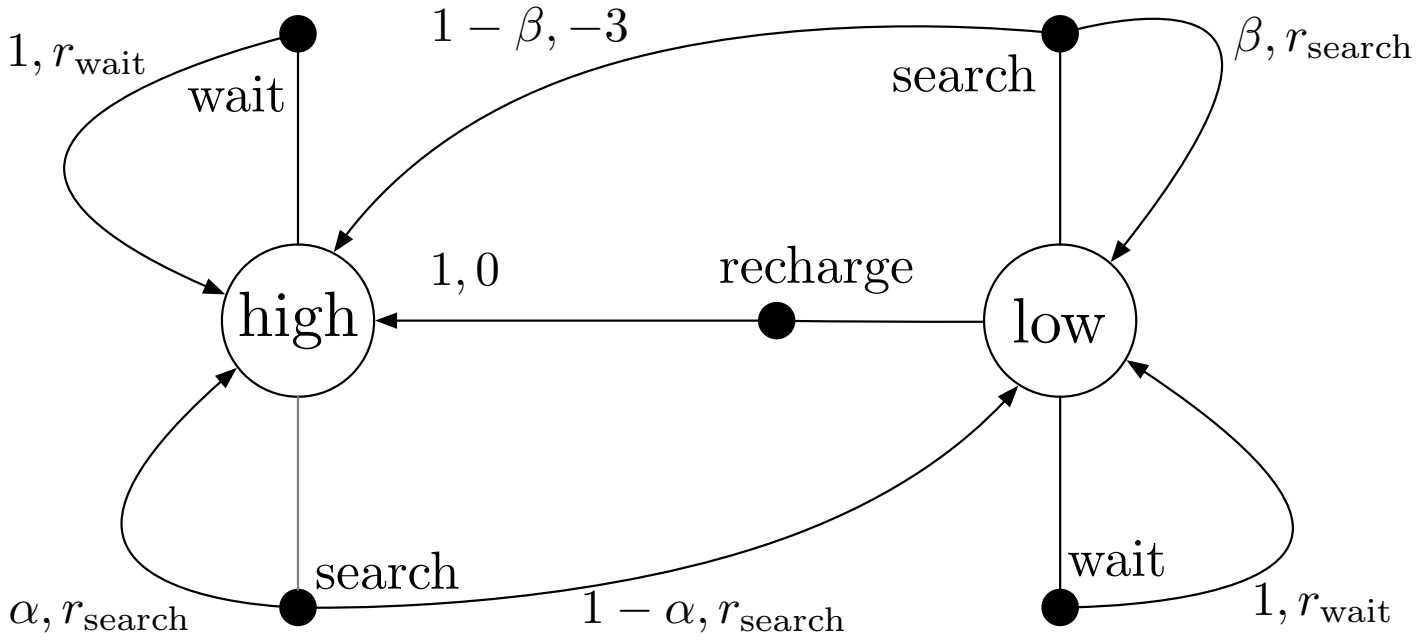
(S. Maleki 2019)

Example: Recycling Robot (2)

s	s'	a	$p(s'\|s,a)$	$r(s,a,s')$
high	high	search	α	r_{search}
high	low	search	$1-\alpha$	r_{search}
low	high	search	$1-\beta$	-3
low	low	search	β	r_{search}
high	high	wait	1	r_{wait}
high	low	wait	0	r_{wait}
low	high	wait	0	r_{wait}
low	low	wait	1	r_{wait}
low	high	recharge	1	0
low	low	recharge	0	0

(S. Maleki 2019)

Example: Recycling Robot (3)



(S. Maleki 2019)

Value Functions (1)

- Value functions estimate how good it is for the agent to be in (or perform a given action) in a given state.
- The notion of "how good" here is defined in terms of future rewards (expected return).
- Value functions are defined with respect to particular policies.

The state value function for policy π can be formally described with:

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right] .$$

Similarly, the action-value function for policy π is given by:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a\right] .$$

(S. Maleki 2019)

Bellman Equation for v_π (1)

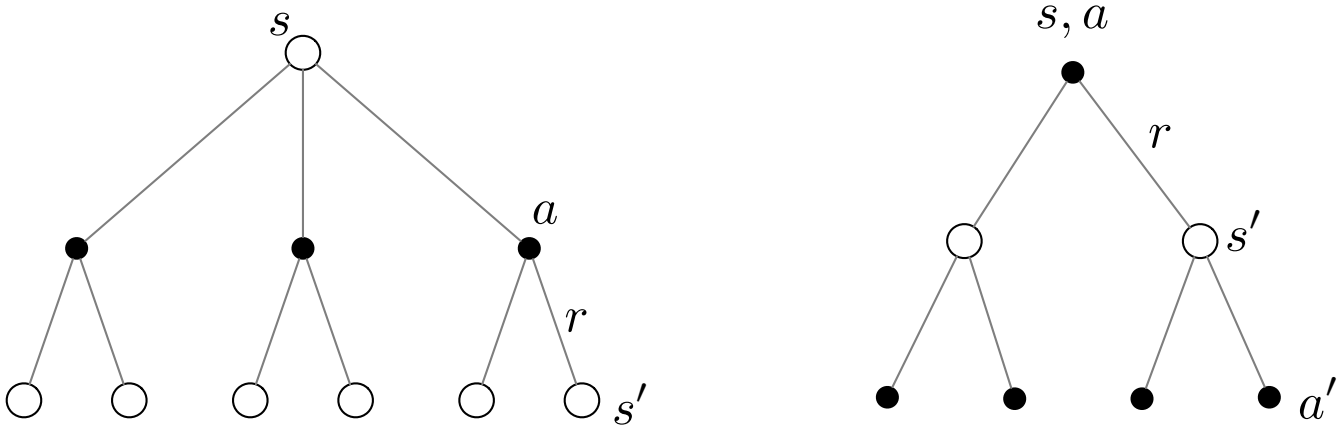
A fundamental property of value functions is that they satisfy particular recursive relationships:

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k R_{t+k+2}|S_t = s\right] \\
&= \sum_a \pi(a|s)\sum_{s'}\sum_r p(s', r|s, a)\left[r + \gamma\mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+2}|s_{t+1} = s'\right]\right] \\
&= \sum_a \pi(a|s)\sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]
\end{aligned}
$$

(S. Maleki 2019)

Bellman Equation for v_π (2)

- Bellman equation for v_π expresses a relationship between the value of a state and the values of its successor states.
- As if we are looking ahead from one state to its possible successor states, as suggested in the following backup diagrams:

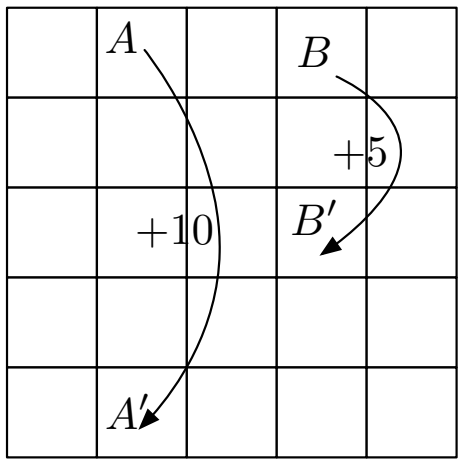


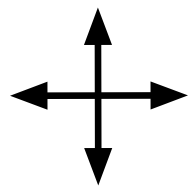
- Value of the start state must equal the (discounted) value of the expected next state, plus the reward expected along the way.

(S. Maleki 2019)

Example: Gridworld

- Actions that would take the agent off the grid leave its location unchanged, but also result in a reward of -1.
- Other actions result in a reward of 0, except those that move the agent out of the special states A and B.
- The agent selects all four actions with equal probability in all states.





Actions

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

$\gamma = 0.9$

(S. Maleki 2019)

Optimal Policy

- Solving a RL task means, roughly, finding a policy that achieves a lot of reward over the long run.
- A policy π is better than or equal to a policy π', if its expected return is greater than or equal to that of π' for all states:

$$\pi \geq \pi' \text{ iff } v_\pi(s) \geq v'_\pi(s) \ \ \forall s \in \mathcal{S} \ .$$

- There is always at least one policy (optimal policy) that is better than or equal to all other policies.
- Since there may be more than one optimal policy, we denote the family of optimal policies by π_*.

(S. Maleki 2019)

Optimal State-Value and Action-Value Functions

- Optimal policies π_* share the same state-value and action-value functions.
- The optimal state-value function is defined as:

$$v_*(s) = \max_\pi v_\pi(s) \quad \forall s \in \mathcal{S} \ .$$

- The optimal action-value function is defined as:

$$q_*(s,a) = \max_\pi q_\pi(s,a) \quad \forall s \in \mathcal{S}, \ \forall a \in \mathcal{A}(s) \ .$$

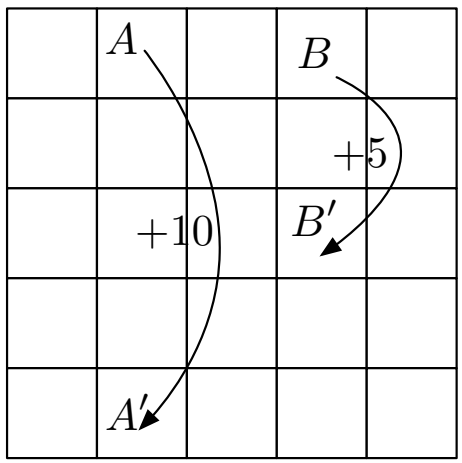
- The expected return for action a in state s for an optimal policy is therefore:

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \ .$$

(S. Maleki 2019)

Example: Gridworld

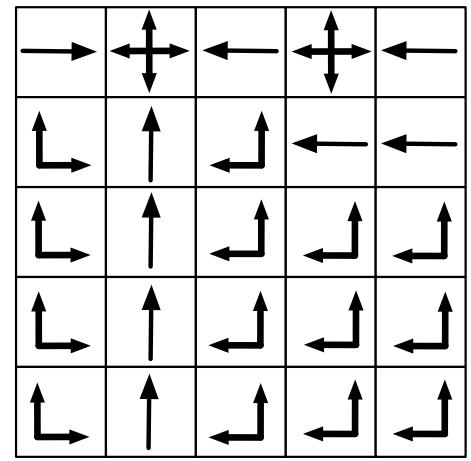
Solve the Bellman equation for v_* for the simple grid task introduced earlier:



Gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

v_*



π_*

(S. Maleki 2019)

Optimality and Approximation

- Explicitly solving the Bellman optimality equation provides one route to finding an optimal policy.
- However, these solutions are rarely directly useful as they rely on at least three assumptions that are rarely true in practice:
  1. We accurately know the dynamics of the environment ;
  2. We have enough computational resources ;
  3. The Markov property .
- In Many practical cases, the functions must be approximated, using some sort of more compact parameterised function representation.

(S. Maleki 2019)