3. Reinforcement Learning: Markov Decision Processes

Reinforcement Learning Notes

March 29, 2025

Contents

1 Introduction to Markov Decision Processes

1.1 Motivation and Context

Markov Decision Processes (MDPs) provide the formal mathematical framework for modeling reinforcement learning problems where an agent interacts with an environment through sequential decision-making. MDPs extend upon simpler paradigms like multi-armed bandits and contextual bandits by introducing:

- A fully specified notion of state
- State transitions influenced by agent actions
- Long-term consequences of decisions

In a multi-armed bandit problem, only the immediate reward is considered, with no concept of state. In contextual bandits, while rewards depend on states, the actions do not influence future states. MDPs provide the next evolutionary step, where an agent's actions both affect immediate rewards and influence the probability distribution of future states.

1.2 The Agent-Environment Interface

The standard agent-environment interface in reinforcement learning is formalized through MDPs:

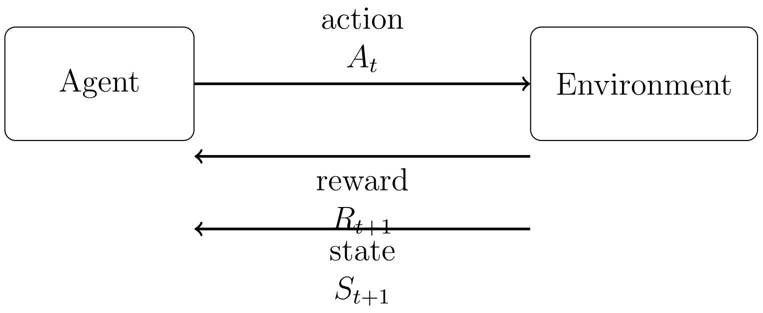


Figure 1: The agent-environment interaction in reinforcement learning (Sutton & Barto)

The interaction proceeds as follows:

- At each time step t, the agent observes the current state $S_t \in \mathcal{S}$

- Based on this state, the agent selects an action $A_t \in \mathcal{A}(S_t)$
- The environment responds with a reward $R_{t+1} \in \mathcal{R}$ and transitions to a new state S_{t+1}
- This creates a sequence of states, actions, and rewards: $S_0, A_0, R_1, S_1, A_1, R_2, S_2, \ldots$

The goal of the agent is to select actions that maximize the expected cumulative reward over time.

2 Components of MDPs

2.1 States, Actions, Rewards, and Transitions

A Markov Decision Process is defined by four essential components:

1. **States** ($\mathcal{S}$): The set of all possible states of the environment. A state $s \in \mathcal{S}$ provides all relevant information needed for decision-making. The state at time t is denoted S_t.
2. **Actions** ($\mathcal{A}$): The set of all possible actions the agent can take. For each state s, the agent can choose from a set of available actions $\mathcal{A}(s)$. The action at time t is denoted A_t.
3. **Transitions**: The state transition probability function $p(s'|s,a)$ defines the dynamics of the environment. It specifies the probability of transitioning to state s' given that the agent was in state s and took action a.
4. **Rewards** ($\mathcal{R}$): The reward function $r(s,a)$ defines the immediate reward received after taking action a in state s. The reward at time $t+1$ (after taking action A_t in state S_t) is denoted R_{t+1}.

2.2 The Markov Property

The defining characteristic of MDPs is the Markov property, which states that the future is independent of the past given the present. Formally, a state S_t is Markov if and only if:

$$P(S_{t+1} = s', R_{t+1} = r | S_t, A_t, R_t, S_{t-1}, A_{t-1}, \ldots, R_1, S_0, A_0) = P(S_{t+1} = s', R_{t+1} = r | S_t, A_t) \tag{1}$$

Notation Overview

- $P(S_{t+1} = s', R_{t+1} = r | S_t, A_t, \ldots)$ - The probability of transitioning to state s' and receiving reward r given the entire history
- $P(S_{t+1} = s', R_{t+1} = r | S_t, A_t)$ - The probability of transitioning to state s' and receiving reward r given only the current state and action
- S_t - The state at time step t
- A_t - The action taken at time step t
- R_t - The reward received at time step t

The Markov property implies that the current state captures all relevant information from the history of the process. This is a crucial assumption that simplifies the decision-making problem while still capturing the essential dynamics of many real-world sequential decision problems.

3 Formal MDP Definition

3.1 Mathematical Notation

A finite Markov Decision Process is formally defined as a 5-tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$ where:

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{R} \to [0, 1]$ is the state transition probability function
- $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function
- $\gamma \in [0, 1]$ is the discount factor

The dynamics of the environment are fully specified by the joint probability distribution:

$$p(s', r|s, a) = P(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a) \tag{2}$$

Notation Overview

- $p(s', r|s, a)$ - The probability of transitioning to state s' and receiving reward r, given that the agent was in state s and took action a
- $P(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a)$ - The probability that the state at time t is s' and the reward is r, given that the state at time $t-1$ was s and the action taken was a
- S_t - The state at time step t
- R_t - The reward received at time step t
- A_t - The action taken at time step t

From this joint probability function, we can derive other important quantities:

3.2 State Transition Probabilities

The state transition probabilities are derived from the joint probability by summing over all possible rewards:

$$p(s'|s, a) = \sum_{r\in\mathcal{R}} p(s', r|s, a) \tag{3}$$

Notation Overview

- $p(s'|s, a)$ - The probability of transitioning to state s' given state s and action a
- $p(s', r|s, a)$ - The joint probability of transitioning to state s' and receiving reward r, given state s and action a
- $\mathcal{R}$ - The set of all possible rewards

3.3 Expected Reward Functions

There are several ways to define the expected reward, each useful in different contexts:

$$r(s, a) = \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a] = \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} p(s', r|s, a) \tag{4}$$

Notation Overview

- $r(s, a)$ - The expected reward when taking action a in state s
- $\mathbb{E}[\cdot]$ - The expected value operator
- R_t - The reward at time t
- S_{t-1} - The state at time $t-1$
- A_{t-1} - The action taken at time $t-1$
- $p(s', r|s, a)$ - The joint probability of transitioning to state s' and receiving reward r, given state s and action a
- $\mathcal{S}$ - The set of all possible states
- $\mathcal{R}$ - The set of all possible rewards

3.4 Example of Expected Reward Calculation

Let's illustrate the calculation of the expected reward with a simple example:
Consider a tiny MDP with:

- 2 states: s_1 and s_2
- 1 action: a
- 2 possible rewards: 0 and 1

The transition probabilities are defined as:

- $p(s_1, 0|s_1, a) = 0.3$ (30% chance of staying in s_1 with reward 0)
- $p(s_2, 0|s_1, a) = 0.2$ (20% chance of moving to s_2 with reward 0)
- $p(s_1, 1|s_1, a) = 0.1$ (10% chance of staying in s_1 with reward 1)
- $p(s_2, 1|s_1, a) = 0.4$ (40% chance of moving to s_2 with reward 1)

To calculate the expected reward $r(s_1, a)$, we apply the formula:

$$\begin{aligned}
r(s_1, a) &= \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} p(s', r|s_1, a) \\
&= 0 \cdot \sum_{s'\in\mathcal{S}} p(s', 0|s_1, a) + 1 \cdot \sum_{s'\in\mathcal{S}} p(s', 1|s_1, a) \\
&= 0 \cdot (p(s_1, 0|s_1, a) + p(s_2, 0|s_1, a)) + 1 \cdot (p(s_1, 1|s_1, a) + p(s_2, 1|s_1, a)) \\
&= 0 \cdot (0.3 + 0.2) + 1 \cdot (0.1 + 0.4) \\
&= 0 \cdot 0.5 + 1 \cdot 0.5 \\
&= 0.5
\end{aligned}$$

This example shows how the inner sum (over s') collects probabilities for the same reward across different next states, and the outer sum (over r) adds up the weighted rewards.

We can also define the expected reward for state-action-next-state triples:

$$r(s, a, s') = \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r\in\mathcal{R}} r \frac{p(s', r|s, a)}{p(s'|s, a)} \tag{5}$$

Notation Overview

- $r(s, a, s')$ - The expected reward when transitioning from state s to state s' via action a
- $\mathbb{E}[\cdot]$ - The expected value operator
- R_t - The reward at time t
- S_{t-1} - The state at time $t-1$
- A_{t-1} - The action taken at time $t-1$
- S_t - The state at time t
- $p(s', r|s, a)$ - The joint probability of transitioning to state s' and receiving reward r, given state s and action a
- $p(s'|s, a)$ - The probability of transitioning to state s' given state s and action a
- $\mathcal{R}$ - The set of all possible rewards

3.5 MDP Constraints

For a valid MDP, the following constraints must be satisfied:

1. $\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
2. $p(s', r|s, a) \geq 0$ for all $s, s' \in \mathcal{S}$, $r \in \mathcal{R}$, and $a \in \mathcal{A}(s)$

These constraints ensure that $p(s', r|s, a)$ is a valid probability distribution.

4 Policy Definition

4.1 Formalization of Policies

A policy is a mapping from states to probabilities of selecting each possible action. Formally, a policy π is defined as:

$$\pi(a|s) = P(A_t = a|S_t = s) \tag{6}$$

Notation Overview

- $\pi(a|s)$ - The probability of selecting action a in state s under policy π
- $P(A_t = a|S_t = s)$ - The probability that the agent selects action $A_t = a$ when in state $S_t = s$
- A_t - The action taken at time step t
- S_t - The state at time step t

4.2 Types of Policies

Policies can be categorized based on their stochasticity:

1. **Deterministic Policies**: For each state, exactly one action is selected with probability 1, and all other actions have probability 0. A deterministic policy can be written as $\pi(s) = a$, indicating that action a is always selected in state s.
2. **Stochastic Policies**: Actions are selected according to a probability distribution. Stochastic policies are useful for exploration and for solving problems with partial observability or non-deterministic dynamics.

4.3 Stationary vs. Non-Stationary Policies

Another categorization of policies is based on their time-dependence:

1. **Stationary Policies**: The policy does not change over time. The mapping from states to action probabilities remains constant regardless of the time step.

2. **Non-Stationary Policies**: The policy changes over time. The mapping from states to action probabilities depends on the time step.

In most reinforcement learning contexts, we focus on finding optimal stationary policies, as they are simpler and often sufficient for solving MDPs.

4.4 Examples of Policies

Several common policy types used in reinforcement learning include:

- **Greedy Policy**: Always selects the action with the highest estimated value:
$$\pi(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_{a'} Q(s, a') \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

- **ϵ-Greedy Policy**: Selects the best action most of the time, but occasionally explores:
$$\pi(a|s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|} & \text{if } a = \arg\max_{a'} Q(s, a') \\ \frac{\epsilon}{|\mathcal{A}(s)|} & \text{otherwise} \end{cases} \tag{8}$$

- **Softmax Policy**: Selects actions with probabilities proportional to their estimated values:
$$\pi(a|s) = \frac{e^{Q(s,a)/\tau}}{\sum_{a'} e^{Q(s,a')/\tau}} \tag{9}$$
where τ is a temperature parameter controlling exploration.

Notation Overview

- $\pi(a|s)$ - The probability of selecting action a in state s under policy π
- $Q(s, a)$ - The estimated value of taking action a in state s
- $\arg\max_{a'}$ - The action that maximizes the given function
- ϵ - A small probability for random exploration
- $|\mathcal{A}(s)|$ - The number of possible actions in state s
- τ - Temperature parameter in softmax, controlling exploration (higher values increase randomness)

5 Episodic vs. Continuing Tasks

5.1 Episodic Tasks

Episodic tasks are those with a clear endpoint or terminal state:

- Each episode starts from an initial state and ends when a terminal state is reached
- Examples include board games (chess, Go), where games have a definite beginning and end
- The interaction breaks naturally into subsequences called "episodes"
- Each episode ends in a special terminal state, often with different rewards for different outcomes

In episodic tasks, we denote the set of all non-terminal states as $\mathcal{S}$ and the set of all states, including the terminal state, as $\mathcal{S}^+$.

5.2 Continuing Tasks

Continuing tasks are those that go on indefinitely without a natural endpoint:

- The agent-environment interaction does not break into episodes and continues without limit

- Examples include ongoing process control, perpetual robot operation, or stock portfolio management
- There is no terminal state, so the agent must learn to perform well over an indefinite horizon

5.3 Unified Notation

To unify the notation for both episodic and continuing tasks, we can use a single formulation by considering episodic tasks as continuing tasks with a special absorbing terminal state that transitions only to itself with a reward of zero.

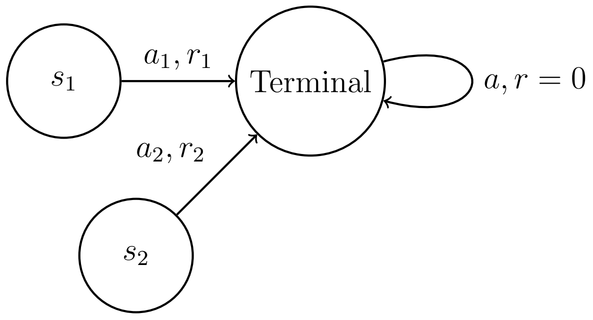


Figure 2: Representation of a terminal state in an episodic task

This unification allows us to treat both episodic and continuing tasks within the same mathematical framework.

6 Discounted Future Return

6.1 Definition of Return

In reinforcement learning, the agent aims to maximize not just immediate rewards but the cumulative reward over time. The return is the function of reward sequence that the agent seeks to maximize.

For a continuing task, the simplest formulation would be the sum of rewards:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \dots \tag{10}$$

Notation Overview

- G_t - The return at time step t
- R_{t+k} - The reward received k steps after time t

However, this sum could be infinite for continuing tasks, making it unsuitable as an optimization criterion.

6.2 Discounting

To address this issue, we introduce discounting, which reduces the importance of future rewards:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{11}$$

Notation Overview

- G_t - The discounted return at time step t
- R_{t+k+1} - The reward received $k+1$ steps after time t
- γ - The discount factor, $0 \leq \gamma \leq 1$
- $\sum_{k=0}^{\infty}$ - The sum over all future time steps

The discount factor γ determines how much the agent values future rewards:

- γ close to 0: Myopic evaluation, immediate rewards are strongly preferred
- γ close to 1: Far-sighted evaluation, future rewards are valued almost as much as immediate ones

6.3 Reasons for Discounting

There are several reasons to use discounting:

1. **Mathematical Convenience**: Ensures the sum is finite for continuing tasks, provided $\gamma < 1$
2. **Uncertainty about the Future**: Future rewards are more uncertain, which may justify valuing them less

3. **Alignment with Human Preferences**: People often prefer immediate rewards to delayed ones
4. **Avoidance of Infinite Returns**: Without discounting, optimal policies might not exist for continuing tasks

6.4 Episodic Returns

For episodic tasks with a terminal time step T, the return can be defined as:

$$G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1} \tag{12}$$

Notation Overview

- G_t - The return at time step t
- R_{t+k+1} - The reward received $k+1$ steps after time t
- γ - The discount factor, $0 \leq \gamma \leq 1$
- T - The final time step of the episode
- $T-t-1$ - The number of remaining steps in the episode after time t

6.5 Unified Return Formulation

To unify notation for both episodic and continuing tasks, we can use:

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \tag{13}$$

Notation Overview

- G_t - The return at time step t
- R_k - The reward received at time step k
- γ - The discount factor, $0 \leq \gamma \leq 1$
- T - The final time step (could be ∞ for continuing tasks)
- $k - t - 1$ - The number of time steps between receiving reward R_k and time t

This formulation handles both episodic tasks (where T is finite) and continuing tasks (where $T = \infty$). Note that for continuing tasks, $\gamma < 1$ is required to ensure the sum is finite.

6.6 Recursive Relationship for Return

The return has a useful recursive relationship:

$$G_t = R_{t+1} + \gamma G_{t+1} \tag{14}$$

Notation Overview

- G_t - The return at time step t
- R_{t+1} - The immediate reward received after time t
- γ - The discount factor, $0 \leq \gamma \leq 1$
- G_{t+1} - The return at time step $t + 1$

This recursive relationship is fundamental to many reinforcement learning methods, particularly those based on bootstrapping.

7 Value Functions

7.1 State-Value Function

The state-value function $v_\pi(s)$ represents how good it is for an agent to be in a particular state s when following policy π. Formally, it is defined as the expected return starting from state s and following policy π thereafter:

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] \tag{15}$$

Notation Overview

- $v_\pi(s)$ - The state-value function for policy π
- $\mathbb{E}_\pi[\cdot]$ - The expected value when following policy π
- G_t - The return at time step t
- S_t - The state at time step t

We can expand this definition using the recursive relationship for the return:

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] \tag{16}$$

Notation Overview

- $v_\pi(s)$ - The state-value function for policy π
- $\mathbb{E}_\pi[\cdot]$ - The expected value when following policy π
- R_{t+1} - The immediate reward
- γ - The discount factor, $0 \leq \gamma \leq 1$
- G_{t+1} - The return at time step $t+1$
- S_t - The state at time step t

7.2 Action-Value Function

The action-value function $q_\pi(s, a)$ represents how good it is to take a specific action a in state s when following policy π thereafter. Formally:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \tag{17}$$

Notation Overview

- $q_\pi(s, a)$ - The action-value function for policy π
- $\mathbb{E}_\pi[\cdot]$ - The expected value when following policy π
- G_t - The return at time step t
- S_t - The state at time step t
- A_t - The action taken at time step t

7.3 Relationship Between Value Functions

The state-value and action-value functions are related by:

$$v_\pi(s) = \sum_a \pi(a|s) q_\pi(s, a) \tag{18}$$

Notation Overview

- $v_\pi(s)$ - The state-value function for policy π
- $\pi(a|s)$ - The probability of taking action a in state s under policy π
- $q_\pi(s, a)$ - The action-value function for policy π
- $\sum_a$ - The sum over all possible actions in state s

This equation shows that the value of a state equals the expected value of the actions that might be taken in that state, weighted by their probability under policy π.

Conversely, the action-value function can be expressed in terms of the state-value function:

$$q_\pi(s, a) = \sum_{s', r} p(s', r|s, a)[r + \gamma v_\pi(s')] \tag{19}$$

Notation Overview

- $q_\pi(s, a)$ - The action-value function for policy π
- $p(s', r|s, a)$ - The probability of transitioning to state s' and receiving reward r, given state s and action a
- r - The immediate reward
- γ - The discount factor, $0 \leq \gamma \leq 1$
- $v_\pi(s')$ - The state-value function for policy π at the next state s'
- $\sum_{s',r}$ - The sum over all possible next states and rewards

This equation shows that the value of taking action a in state s equals the expected immediate reward plus the discounted value of the next state.

7.4 The Bellman Equation for State-Value Function

The Bellman equation expresses the recursive relationship in value functions. For the state-value function:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \tag{20}$$

Notation Overview

- $v_\pi(s)$ - The state-value function for policy π
- $\pi(a|s)$ - The probability of taking action a in state s under policy π
- $p(s', r|s, a)$ - The probability of transitioning to state s' and receiving reward r, given state s and action a
- r - The immediate reward
- γ - The discount factor, $0 \leq \gamma \leq 1$
- $v_\pi(s')$ - The state-value function for policy π at the next state s'
- $\sum_a$ - The sum over all possible actions in state s
- $\sum_{s',r}$ - The sum over all possible next states and rewards

The Bellman equation states that the value of a state equals the expected return for taking an action according to policy π, which consists of the immediate reward plus the discounted value of the next state.

7.5 The Bellman Equation for Action-Value Function

Similarly, the Bellman equation for the action-value function is:

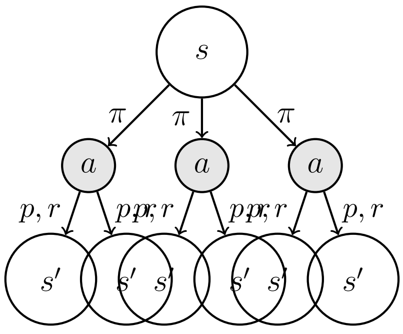
$$q_\pi(s,a) = \sum_{s',r} p(s',r|s,a) \left[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s',a') \right] \tag{21}$$

Notation Overview

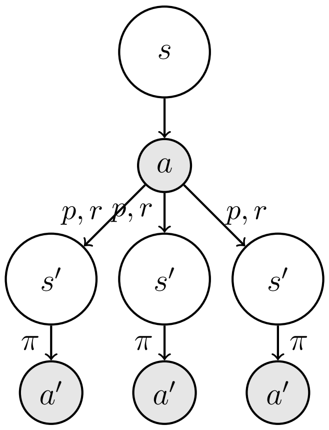
- $q_\pi(s,a)$ - The action-value function for policy π
- $p(s',r|s,a)$ - The probability of transitioning to state s' and receiving reward r, given state s and action a
- r - The immediate reward
- γ - The discount factor, $0 \leq \gamma \leq 1$
- $\pi(a'|s')$ - The probability of taking action a' in the next state s' under policy π
- $q_\pi(s',a')$ - The action-value function for policy π at the next state-action pair (s',a')
- $\sum_{s',r}$ - The sum over all possible next states and rewards
- $\sum_{a'}$ - The sum over all possible actions in the next state s'

7.6 Backup Diagrams for Bellman Equations

Backup diagrams provide a graphical representation of the Bellman equations, illustrating how values are propagated from future states back to the current state.



(a) State-value backup



(b) Action-value backup

Figure 3: Backup diagrams for (a) the state-value function v_π and (b) the action-value function q_π

In these diagrams:

- Solid circles represent states
- Open circles represent actions
- The root node at the top represents the state or state-action pair being evaluated
- The branches show possible trajectories
- Nodes are traversed from top to bottom when following trajectories
- Values are backed up from bottom to top when computing the value function

8 Optimal Policies and Value Functions

8.1 Definition of Optimal Policies

A policy π is defined as better than or equal to a policy π' if its expected return is greater than or equal to that of π' for all states:

$$\pi \geq \pi' \iff v_\pi(s) \geq v_{\pi'}(s) \text{ for all } s \in \mathcal{S} \tag{22}$$

Notation Overview

- $\pi \geq \pi'$ - Policy π is better than or equal to policy π'
- $v_\pi(s)$ - The state-value function for policy π
- $v_{\pi'}(s)$ - The state-value function for policy π'
- $\mathcal{S}$ - The set of all possible states

There is always at least one policy that is better than or equal to all other policies, called an optimal policy, denoted π_*. All optimal policies share the same optimal state-value function, v_*:

$$v_*(s) = \max_\pi v_\pi(s) \text{ for all } s \in \mathcal{S} \tag{23}$$

Notation Overview

- $v_*(s)$ - The optimal state-value function
- $\max_\pi$ - The maximum over all possible policies
- $v_\pi(s)$ - The state-value function for policy π
- $\mathcal{S}$ - The set of all possible states

They also share the same optimal action-value function, q_*:

$$q_*(s,a) = \max_\pi q_\pi(s,a) \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}(s) \tag{24}$$

Notation Overview

- $q_*(s,a)$ - The optimal action-value function
- $\max_\pi$ - The maximum over all possible policies
- $q_\pi(s,a)$ - The action-value function for policy π
- $\mathcal{S}$ - The set of all possible states
- $\mathcal{A}(s)$ - The set of actions available in state s

8.2 The Bellman Optimality Equations

The Bellman optimality equations characterize the optimal value functions:

$$v_*(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')] \tag{25}$$

Notation Overview

- $v_*(s)$ - The optimal state-value function
- $\max_a$ - The maximum over all actions $a \in \mathcal{A}(s)$
- $p(s', r|s, a)$ - The probability of transitioning to state s' and receiving reward r, given state s and action a
- r - The immediate reward
- γ - The discount factor, $0 \leq \gamma \leq 1$
- $v_*(s')$ - The optimal state-value function at the next state s'
- $\sum_{s',r}$ - The sum over all possible next states and rewards

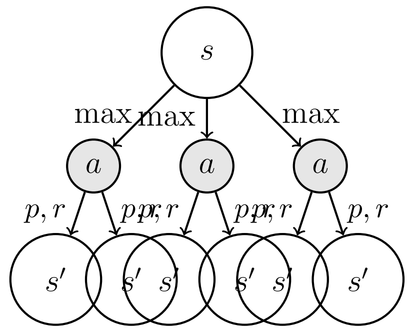
And for the action-value function:

$$q_*(s, a) = \sum_{s',r} p(s', r|s, a)[r + \gamma \max_{a'} q_*(s', a')] \tag{26}$$

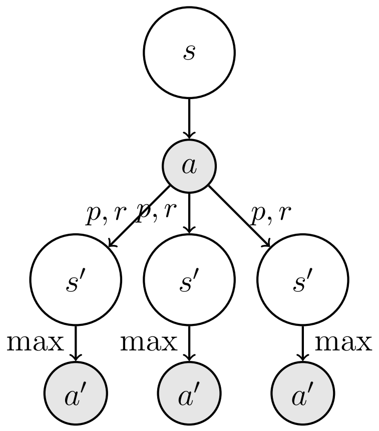
Notation Overview

- $q_*(s, a)$ - The optimal action-value function
- $p(s', r|s, a)$ - The probability of transitioning to state s' and receiving reward r, given state s and action a
- r - The immediate reward
- γ - The discount factor, $0 \leq \gamma \leq 1$
- $\max_{a'}$ - The maximum over all actions $a' \in \mathcal{A}(s')$
- $q_*(s', a')$ - The optimal action-value function at the next state-action pair (s', a')
- $\sum_{s',r}$ - The sum over all possible next states and rewards

8.3 Backup Diagrams for Bellman Optimality Equations



(a) Optimal state-value backup



(b) Optimal action-value backup

Figure 4: Backup diagrams for (a) the optimal state-value function v_* and (b) the optimal action-value function q_*

The key difference between these backup diagrams and those for v_π and q_π is the replacement of the policy distribution π with the max operator, indicating that the agent selects the action that maximizes value.

8.4 Deriving Optimal Policies

Given the optimal value functions, it is straightforward to determine an optimal policy:

$$\pi_*(s) = \arg\max_a q_*(s, a) \tag{27}$$

Notation Overview

- $\pi_*(s)$ - The optimal policy
- $\arg\max_a$ - The action that maximizes the given function
- $q_*(s, a)$ - The optimal action-value function

Alternatively, using the state-value function:

$$\pi_*(s) = \arg\max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')] \tag{28}$$

Notation Overview

- $\pi_*(s)$ - The optimal policy
- $\arg\max_a$ - The action that maximizes the given function
- $p(s', r|s, a)$ - The probability of transitioning to state s' and receiving reward r, given state s and action a
- r - The immediate reward
- γ - The discount factor, $0 \leq \gamma \leq 1$
- $v_*(s')$ - The optimal state-value function at the next state s'
- $\sum_{s',r}$ - The sum over all possible next states and rewards

Any policy that is greedy with respect to the optimal value function(s) is an optimal policy. There may be multiple optimal policies that share the same optimal value function.

9 Solving MDPs

9.1 Methods for Finding Optimal Policies

Several methods can be used to solve MDPs:

1. **Dynamic Programming (DP)**: When the MDP's dynamics (transition probabilities and reward function) are known, DP methods like policy iteration and value iteration can be used to compute optimal policies.

2. **Monte Carlo (MC) Methods**: When the dynamics are unknown, MC methods use sample episodes to estimate values and improve the policy.

3. **Temporal Difference (TD) Learning**: Combines ideas from DP and MC, updating value estimates based on other learned estimates without waiting for the end of an episode.

4. **Model-Free Methods**: Methods like Q-learning and SARSA that learn directly from interaction with the environment without explicitly modeling its dynamics.

5. **Model-Based Methods**: Methods that learn a model of the environment's dynamics and then use that model for planning.

9.2 The Generalized Policy Iteration Framework

Many reinforcement learning methods follow a generalized policy iteration (GPI) framework, which alternates between policy evaluation and policy improvement:

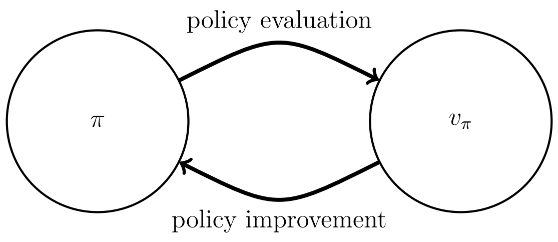


Figure 5: The generalized policy iteration framework

- **Policy Evaluation**: Compute the value function v_π for the current policy π
- **Policy Improvement**: Improve the policy by making it greedy with respect to the current value function
- This process continues until convergence to the optimal policy and value function

Different methods vary in how they implement these two steps and how they interleave them.

10 Example Applications of MDPs

10.1 Grid World Example

A classic example of an MDP is the grid world problem:

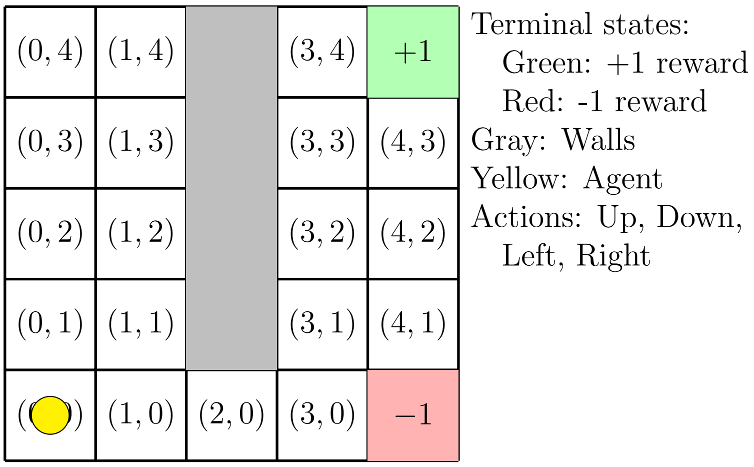


Figure 6: Example grid world MDP

In this environment:

- States are the grid positions (x, y)
- Actions are movements in four directions: up, down, left, right
- Transitions are deterministic (moving in a direction always results in the corresponding adjacent cell, unless blocked by a wall or grid boundary)
- Rewards are -1 for each step, except when reaching terminal states
- Terminal states have rewards of +1 (green) and -1 (red)
- The goal is to find the policy that maximizes the expected return (i.e., reach the green state with the minimum number of steps)

10.2 Robot Navigation

MDPs can model robot navigation problems:

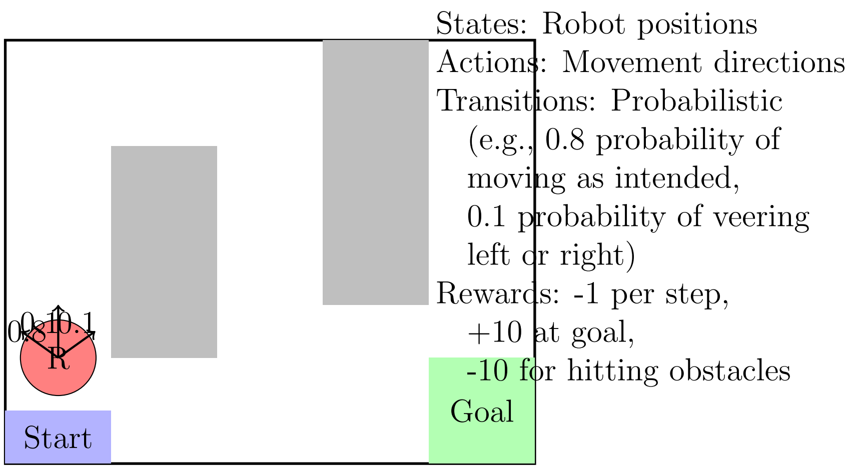

States: Robot positions
Actions: Movement directions
Transitions: Probabilistic (e.g., 0.8 probability of moving as intended, 0.1 probability of veering left or right)
Rewards: -1 per step, +10 at goal, -10 for hitting obstacles
0.8
0.1
0.1
R
Start
Goal

Figure 7: Robot navigation as an MDP

10.3 Resource Management

MDPs can also model resource allocation problems:

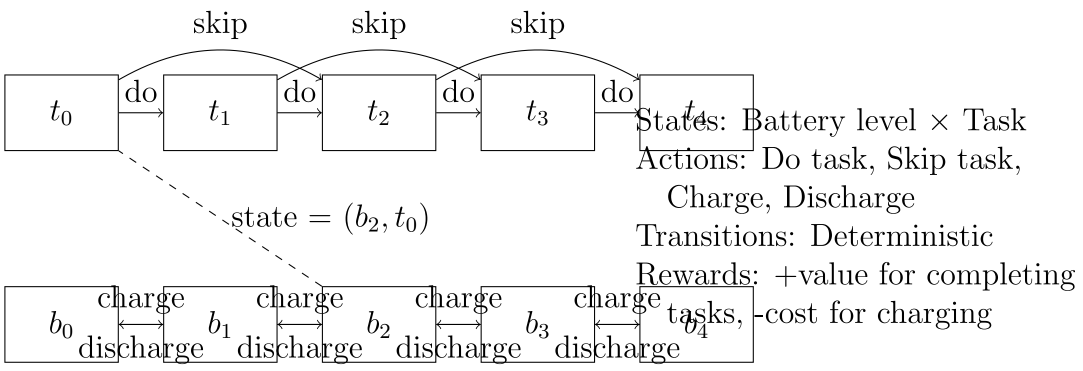

skip
skip
skip
t_0
do
t_1
do
t_2
do
t_3
do
t_4
state = (b_2, t_0)
b_0
charge
discharge
b_1
charge
discharge
b_2
charge
discharge
b_3
charge
discharge
b_4
States: Battery level × Task
Actions: Do task, Skip task, Charge, Discharge
Transitions: Deterministic
Rewards: +value for completing tasks, -cost for charging

Figure 8: Resource management as an MDP

11 Conclusion

Markov Decision Processes provide the formal mathematical framework for modeling sequential decision-making problems. They extend simpler frameworks like multi-armed bandits by introducing state transitions that are influenced by the agent's actions, allowing for the modeling of long-term consequences.

Key concepts covered in this chapter include:

- The four essential components of an MDP: states, actions, transitions, and rewards
- The formal MDP definition using mathematical notation and constraints
- Policies as mappings from states to action probabilities
- Episodic versus continuing tasks and the unification of their notation
- The discounting of future rewards and its mathematical and practical justifications
- Value functions and their recursive relationships expressed in the Bellman equations
- Optimal policies and value functions characterized by the Bellman optimality equations
- Various methods for solving MDPs to find optimal policies

Understanding MDPs is crucial for reinforcement learning, as they provide the theoretical foundation upon which algorithms are built. The concepts of value functions, policies, and the Bellman equations are fundamental to virtually all reinforcement learning methods, from classic techniques like dynamic programming to modern approaches based on deep neural networks.