



Reinforcement Learning

Finite Markov Decision Process

U Kang
Seoul National University



In This Lecture

- Markov Decision Process
 - Definition
 - Return, state, and action
 - Policy and value functions
 - Optimality and approximation



Overview

- Markov Decision Process (MDP)
 - A classical formalization of sequential decision making, where actions influence not just immediate rewards, but also subsequent situations, or states, and through those future rewards
 - Involve delayed reward and the need to tradeoff immediate and delayed reward
 - Estimate $q_*(s, a)$ for each action a in each state s , or the value $v_*(s)$ of each state given optimal action selections



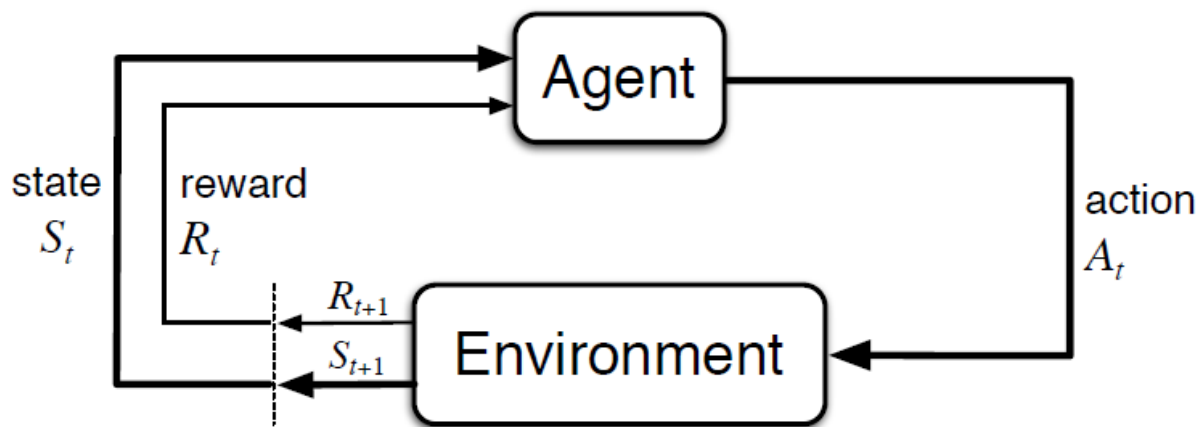
Outline

- ➔ **Agent-Environment Interface**
- Goals and Rewards
- Returns and Episodes
- Episodic and Continuing Tasks
- Policies and Value Functions
- Optimal Policies and Value Functions
- Optimality and Approximation
- Conclusion



Agent-Environment Interface

- Objective: learning from interaction to achieve a goal
- Agent: the learner and decision maker
- Environment: everything outside the agent
- Environment gives rewards, special numerical values that the agent seeks to maximize over time





Agent-Environment Interface

- Consider a sequence of time steps, $t = 0, 1, 2, \dots$
- At each time t , the agent receives state S_t and reward R_t (when $t \neq 0$), and on that basis selects an action A_t
- One step later, the agent receives state S_{t+1} and reward R_{t+1} , and on that basis selects an action A_{t+1}
- Sequence of interactions
 - $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, \dots$



MDP

■ Finite MDP

- States, actions, and rewards are finite
- $p(s', r | s, a) = P\{S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a\}$
- The probability distribution p defines the dynamics of MDP
 - $p: S \times R \times S \times A \rightarrow [0, 1]$
 - $\sum_{s'} \sum_r p(s', r | s, a) = 1$, for all s, a
- The probability $p(s', r | s, a)$ depends only on the preceding state and action, not earlier ones
 - Markov property



MDP

- From $p(s', r|s, a)$, we can compute anything we want to know about the environment
 - State-transition probabilities
 - $p(s'|s, a) = P\{S_t = s' | S_{t-1} = s, A_{t-1} = a\} = \sum_r p(s', r|s, a)$
 - Expected reward
 - $r(s, a) = E[R_t | S_{t-1} = s, A_{t-1} = a] = \sum_r r \sum_{s'} p(s', r|s, a)$
 - Expected reward for state-action-next state
 - $r(s, a, s') = E[R_t | S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_r r \frac{p(s', r|s, a)}{p(s'|s, a)}$



Flexibility of MDP

- The time steps need not refer to fixed intervals of real time; they can refer to arbitrary successive stages of decision making and acting
- The actions can be low-level controls, such as the voltages applied to the motors of a robot arm, or high-level decisions, such as whether or not to have lunch or to go to graduate school
- The states can take a wide variety of forms. E.g., low-level sensor readings, or high-level symbolic descriptions of objects in a room



Agent-Environment Boundary

- Different from physical boundary
 - The motors and mechanical linkages of a robot and its sensing hardware should usually be considered parts of the environment rather than parts of the agent
- Anything that cannot be changed arbitrarily by the agent is considered to be outside of it and thus part of its environment
- The agent–environment boundary represents the limit of the agent's *absolute control*, not of its knowledge



Abstraction by MDP

- MDP: a considerable abstraction of the problem of goal-directed learning from interaction
- Any problem of learning goal-directed behavior can be reduced to three signals between agent and environment
 - Actions: choices made by the agent
 - States: basis on which the choices are made
 - Rewards: the agent's goal



Example: Bioreactor

- Goal: determine moment-by-moment temperatures and stirring rates for a bioreactor (a large vat of nutrients and bacteria used to produce useful chemicals)
- Actions: target temperatures and target stirring rates
 - Represented as a vector
- States: thermocouple and other sensory readings, perhaps filtered and delayed, plus symbolic inputs representing the ingredients in the vat and the target chemical
 - Represented as a vector
- Rewards: moment-by-moment measures of the rate at which the useful chemical is produced by the bioreactor
 - Represented as a scalar



Example: Pick-and-Place Robot

- Goal: control the motion of a robot arm in a repetitive pick-and-place task, to learn movements that are fast and smooth
- Actions: voltages applied to each motor at each joint
- States: latest readings of joint angles and velocities
- Reward: +1 for each object successfully picked up and placed
 - To encourage smooth movements, on each time step a small, negative reward can be given as a function of the moment-to-moment “jerkiness” of the motion.



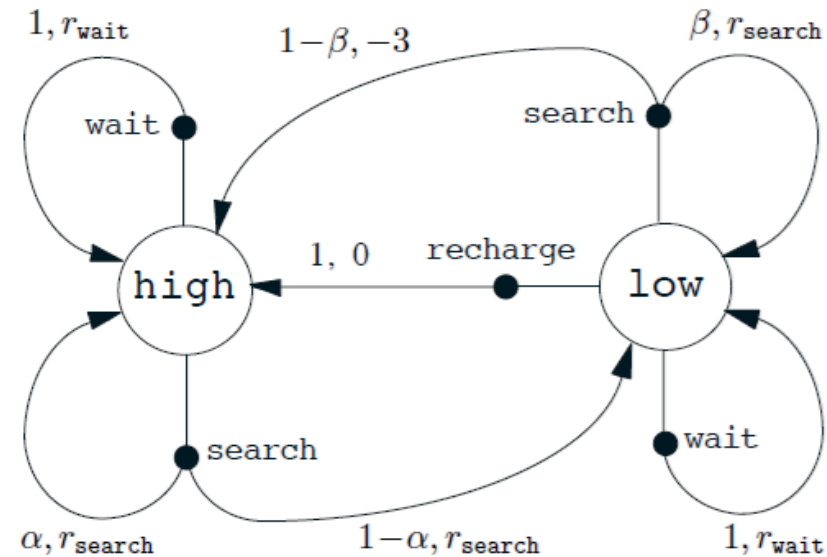
Example: Recycling Robot

- A mobile robot collecting empty soda cans, running on a rechargeable battery
- State: battery level (high, low)
- Actions: search (for a can), wait (for someone to bring a can), recharge (its battery)
- Actions sets
 - $A(\text{high}) = \{\text{search}, \text{wait}\}$
 - $A(\text{low}) = \{\text{search}, \text{wait}, \text{recharge}\}$



Example: Recycling Robot


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



Sutton and Barto,
Reinforcement
Learning, 2018



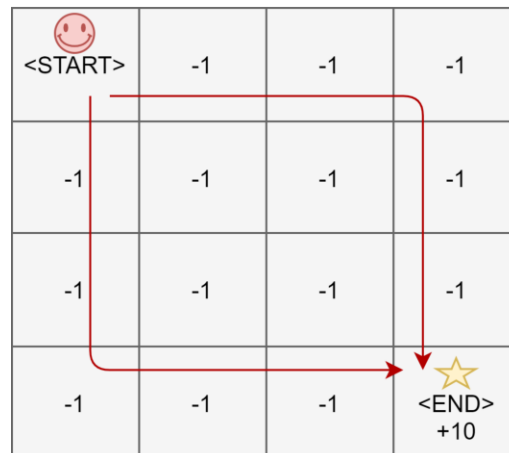
Outline

- Agent-Environment Interface
-  **Goals and Rewards**
- Returns and Episodes
- Episodic and Continuing Tasks
- Policies and Value Functions
- Optimal Policies and Value Functions
- Optimality and Approximation
- Conclusion



Goals and Rewards

- Reward hypothesis: the goal of an agent is to maximize the expected value of the cumulated reward
- This has proved to be flexible and widely applicable





Examples

- Making a robot learn to walk
 - Reward: proportional to the robot's forward motion, on each time step
- Making a robot learn how to escape from a maze
 - Reward: -1 for every time step that passes prior to escape; this encourages the agent to escape as quickly as possible
- Making a robot learn to find and collect empty soda cans for recycling
 - Reward: 0 most of the time, and $+1$ for each can collected. One might also want to give the robot negative rewards when it bumps into things or when somebody yells at it
- Learning to play chess
 - Reward: $+1$ for winning, -1 for losing, and 0 for drawing and for all nonterminal positions




Maximizing Rewards

- The agent always learns to maximize its long-term reward
- If we want it to do something for us, we must provide rewards to it in such a way that in maximizing them the agent will also achieve our goals.
- Reward signal is not the place to impart to the agent prior knowledge about *how* to achieve what we want it to do
 - E.g., a chess-playing agent should be rewarded only for actually winning, not for achieving subgoals such as taking its opponent's pieces or gaining control of the center of the board
 - Exploiting prior knowledge on *how*: initial policy or value functions
- Reward signal is your way of communicating to the robot *what* you want it to achieve, not how you want it achieved



Outline

- Agent-Environment Interface
- Goals and Rewards
-  **Returns and Episodes**
- Episodic and Continuing Tasks
- Policies and Value Functions
- Optimal Policies and Value Functions
- Optimality and Approximation
- Conclusion



Returns and Episodes

- Agent's goal: maximize the cumulative rewards in the long run
- Maximize expected return G_t
 - $G_t = R_{t+1} + R_{t+2} + R_{t+3} + \dots + R_T$
 - T : final time step
- What is the implication of T ?



Returns and Episodes

- Episode (trial): separated subsequence in agent-environment interaction
 - Plays of a game
 - Trips through a maze
 - Any sort of repeated interaction
- Each episode ends in a special terminal state, followed by a reset to a starting state
- Each episode may give different rewards



Returns and Episodes

- Episodic tasks: tasks with episodes
 - S : set of all nonterminal states
 - S^+ : set of all states plus the terminal state
- Continuing tasks
 - Agent-environment interaction does not break into episodes, but goes on continually without limit
 - E.g., a robot with a long life span
 - The return may be infinite; thus we need a concept of discounting



Returns and Episodes

- An agent selects actions to maximize the sum of discounted rewards
 - $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$
 - γ : discount rate in $[0, 1]$
- Returns at successive time steps are related to each other
 - $$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \dots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$



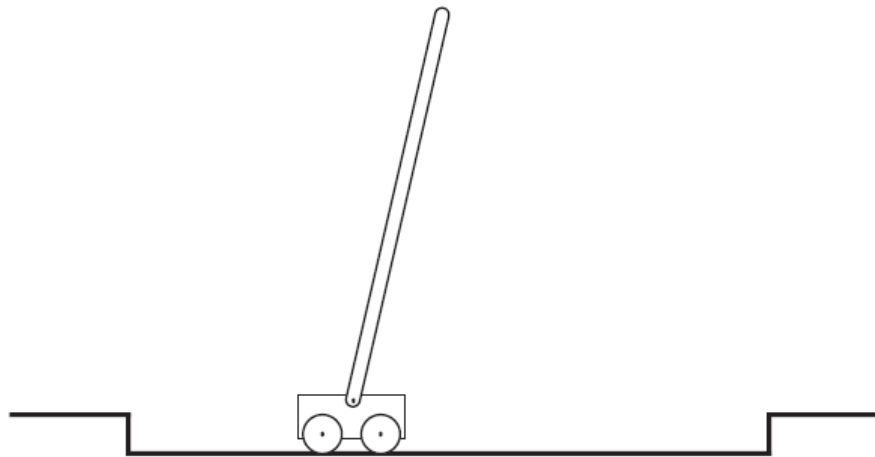
Returns and Episodes

- Discount rate determines the present value of future rewards
 - A reward received k time steps in the future is worth only γ^{k-1} times what it would be worth if it were received immediately
 - If $\gamma < 1$, G_t is finite as long as reward sequence $\{R_k\}$ is bounded
 - If the reward is constant $+1$, then $G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma}$
 - If $\gamma = 0$, the agent concerns only on the immediate rewards
 - As γ approaches 1, the agent becomes more far-sighted



Example: Pole-Balancing

- Objective: apply forces to a cart moving along a track to keep a pole hinged to the cart from falling over
- Failure: if the pole falls past a given angle from vertical or if the cart runs off the track






Example: Pole-Balancing

- Episodic task formulation
 - Reward +1 for every time step on which failure did not occur
- Continuing task formulation (with discounting)
 - Reward -1 on each failure and 0 at all other times
 - The return at each time is related to $-\gamma^K$, where K is the number of time steps before failure
- In either case, the return is maximized by keeping the pole balanced for as long as possible



Outline

- Agent-Environment Interface
- Goals and Rewards
- Returns and Episodes
-  **Episodic and Continuing Tasks**
- Policies and Value Functions
- Optimal Policies and Value Functions
- Optimality and Approximation
- Conclusion



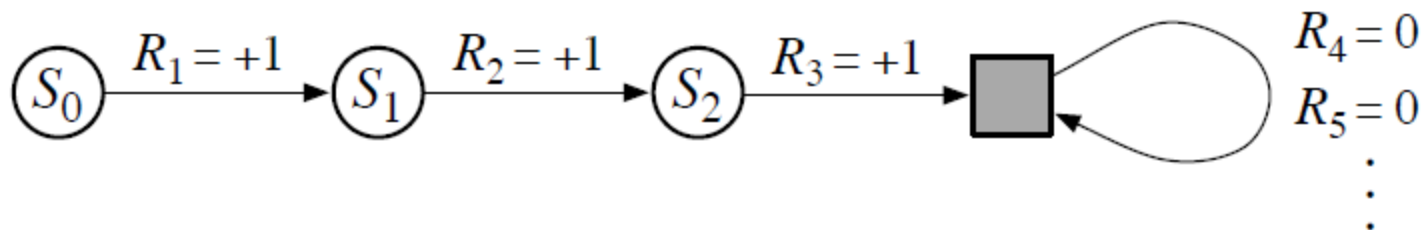
Episodic and Continuing Tasks

- Agent-environment interaction
 - Episodic tasks: interaction is broken into separate episodes
 - Continuing tasks
- It is useful to establish one notation to refer to both episodic and continuing tasks



Episodic and Continuing Tasks

- Absorbing state
 - Transitions only to itself and generates only rewards of 0
 - Useful to unify episodic and continuing tasks
 - Setting $T = 3$ or $T = \infty$ give the same result






Episodic and Continuing Tasks

- Unified notations for episodic and continuing tasks
 - $G_t = \sum_{k=t+1}^T \gamma^{k-t-1} R_k$
 - Episodic tasks: $\gamma = 1$
 - T can be finite, or $T = \infty$ with an absorbing state
 - Continuing tasks: $\gamma < 1, T = \infty$



Outline

- Agent-Environment Interface
- Goals and Rewards
- Returns and Episodes
- Episodic and Continuing Tasks
-  **Policies and Value Functions**
- Optimal Policies and Value Functions
- Optimality and Approximation
- Conclusion



Policies and Value Functions

- Almost all RL algorithms involve estimating value functions
 - Value function: map a state (or state-action pair) to a value representing how good it is in terms of expected return
- The rewards the agent can expect to receive in the future depend on what actions it will take; thus value functions are defined with respect to particular ways of acting, called policies



Policies and Value Functions

- Policy: a mapping from a state to probabilities of selecting each action
 - $\pi(a|s)$: the probability of selecting $A_t = a$, if $S_t = s$



Policies and Value Functions

■ Value function

- State-value function for policy π : expected return when starting in s and following π thereafter
 - $v_{\pi}(s) = E_{\pi}[G_t | S_t = s] = E_{\pi}[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s]$
- Action-value function for policy π : expected return when starting in s , taking action a , and following π thereafter
 - $q_{\pi}(s, a) = E_{\pi}[G_t | S_t = s, A_t = a] = E_{\pi}[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a]$
- The value of the final state is always 0



Policies and Value Functions

- Value functions v_π and q_π can be estimated from experience
 - Monte Carlo methods: average over many random samples of actual returns
 - $v_\pi(s)$: maintain average of the actual returns that have followed the state
 - $q_\pi(s, a)$: maintain average of the actual returns that have followed the state and the action
 - Function approximator
 - Monte Carlo method is limited when there are many states
 - Maintain v_π and q_π as parameterized functions (with fewer parameters than states)
 - Learn parameters to better match the observed returns



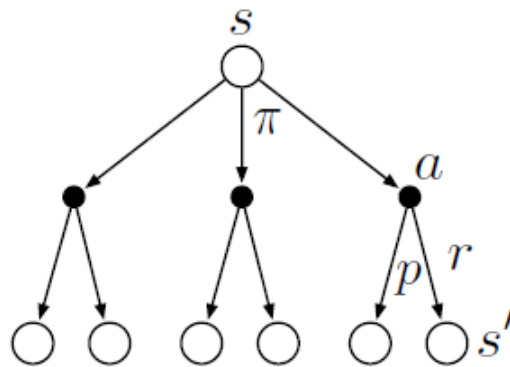
Value Function

- Recursive relationship of value functions v_π

- $$\begin{aligned} v_\pi(s) &= E_\pi[G_t | S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s] \\ &= E_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a) [r + \gamma E_\pi[G_{t+1} | S_{t+1} = s']] \\ &= \sum_a \pi(a|s) \sum_{s', r} p(s', r | s, a) [r + \gamma v_\pi(s')] \end{aligned}$$

Bellman equation for v_π

- The value function v_π is the unique solution to its Bellman equation



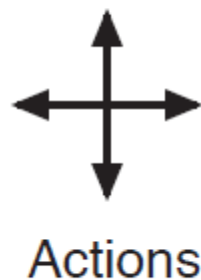
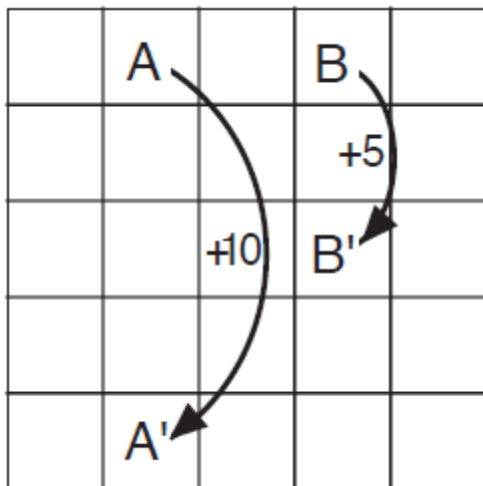
Backup diagram for v_π



Example: Gridworld

Sutton and Barto,
Reinforcement
Learning, 2018

- State: each cell
- Actions: north, south, east, and west
- Rewards: +10 (A \rightarrow A'), +5 (B \rightarrow B'), -1 for "off the grid", and 0 otherwise



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

$v_{\pi}(s)$ for equiprobable random policy,
when $\gamma = 0.9$

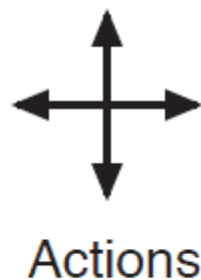
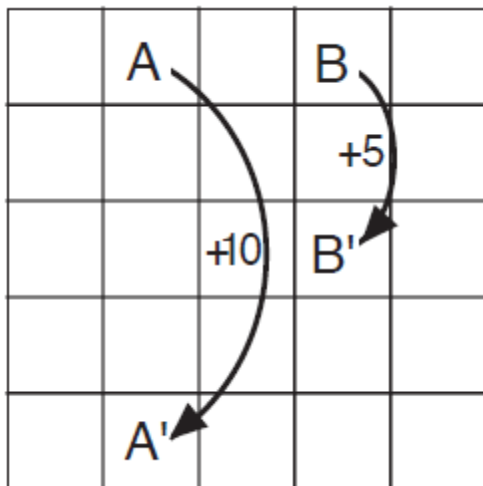


Example: Gridworld

Sutton and Barto,
Reinforcement
Learning, 2018

■ Observations

- Value of A is less than 10; Value of B is greater than 5
- Value of A' is negative
- Upper cells have higher values than bottom cells



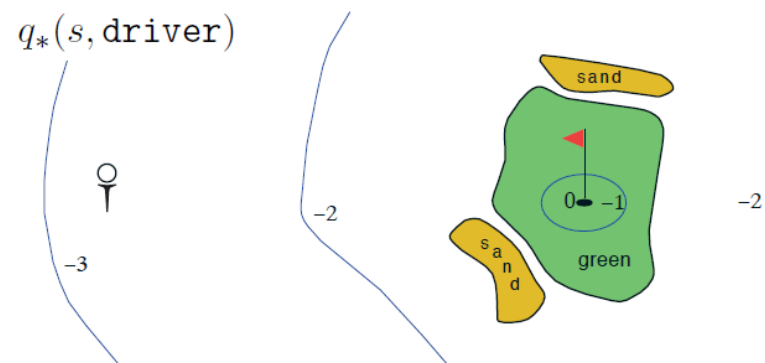
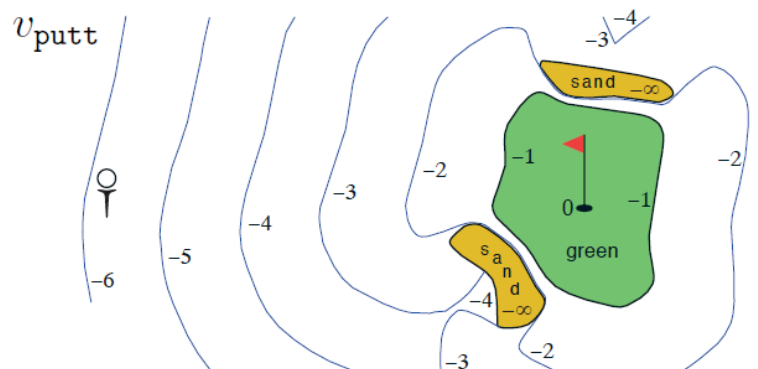
3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

$v_{\pi}(s)$ for equiprobable random policy,
when $\gamma = 0.9$




Example: Golf

- State: location of the ball
- Action: club (, and direction)
- Reward: -1 for each stroke until hitting the ball into the hole
- Observations
 - v_{putt}
 - Shorter gaps of values than in driver
 - $-\infty$ value for sand
 - $q_*(s, \text{driver})$
 - Best action-value for 'driver' action





Outline

- Agent-Environment Interface
- Goals and Rewards
- Returns and Episodes
- Episodic and Continuing Tasks
- Policies and Value Functions
-  **Optimal Policies and Value Functions**
- Optimality and Approximation
- Conclusion



Optimal Policies and Values

- The goal of RL is to find a policy that achieves the best reward in the long run
- A policy is better than the other if its expected return is greater than the other for all states
 - $\pi \geq \pi'$ iff $v_\pi(s) \geq v_{\pi'}(s)$, for all s
- Optimal policy π_* : better than or equal to all other policies
 - Share the optimal state-value function v_*
 - $v_*(s) = \max_\pi v_\pi(s)$ for all s
 - Share the optimal action-value function q_*
 - $q_*(s, a) = \max_\pi q_\pi(s, a)$ for all a and s
 - $q_*(s, a) = E[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]$



Optimal Policies and Values

■ Bellman optimality equation

- The value of a state under an optimal policy must equal the expected return for the best action from that state

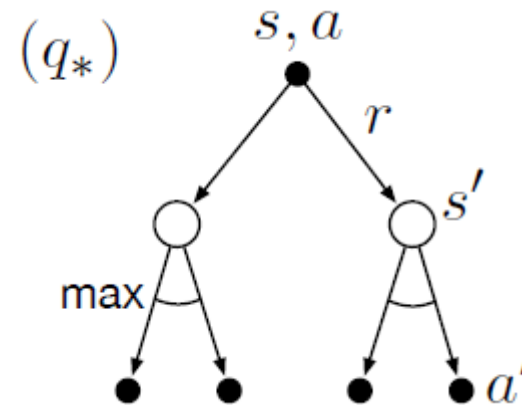
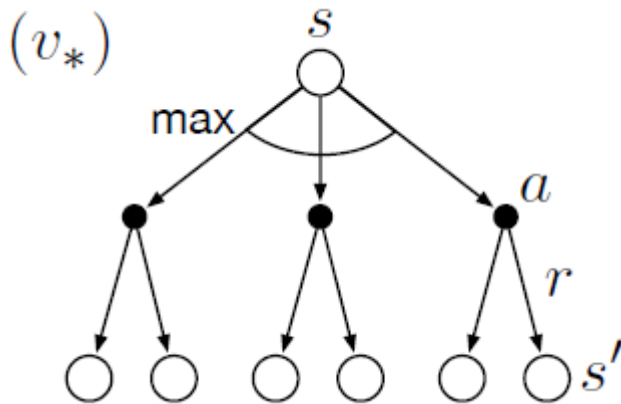
- $$\begin{aligned} v_*(s) &= \max_a q_{\pi^*}(s, a) \\ &= \max_a E_{\pi^*}[G_t | S_t = s, A_t = a] \\ &= \max_a E_{\pi^*}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\ &= \max_a E[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_*(s')] \end{aligned}$$

- $$\begin{aligned} q_*(s, a) &= E[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a] \\ &= \sum_{s', r} p(s', r | s, a) [r + \gamma \max_{a'} q_*(s', a')] \end{aligned}$$



Optimal Policies and Values

- Backup diagram for v_* and q_*





Optimal Policies and Values

- How to find the best policy, given v_* ?
 - $v_*(s) = \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma v_*(s')]$
 - For each state s , there will be one or more actions at which the maximum is obtained in the Bellman optimality equation; any policy that assigns nonzero probability only to these actions is optimal
 - Can be thought of as one-step search: the actions that appear best after a one-step search are optimal
 - Greedy search: any policy that is greedy with respect to the optimal evaluation function v_* is optimal
 - Considering the one-step consequences of actions is enough, since v_* already takes into account the reward consequences of all possible future behavior

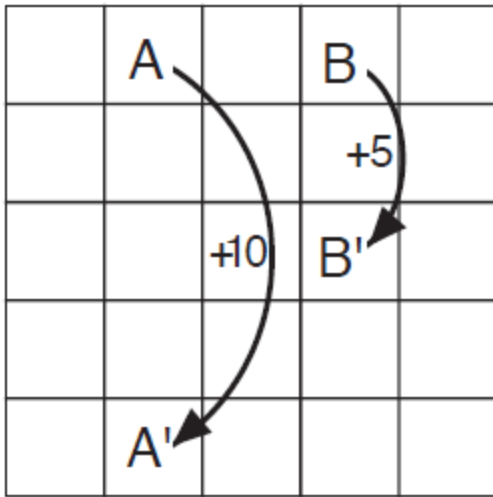


Optimal Policies and Values

- How to find the best policy, given q_* ?
 - $q_*(s, a) = \sum_{s', r} p(s', r | s, a) [r + \gamma \max_{a'} q_*(s', a')]$
 - The agent does not even have to do a one-step search!
 - For any state s , find any action that maximizes $q_*(s, a)$
 - The action-value function effectively caches the results of all one-step ahead searches



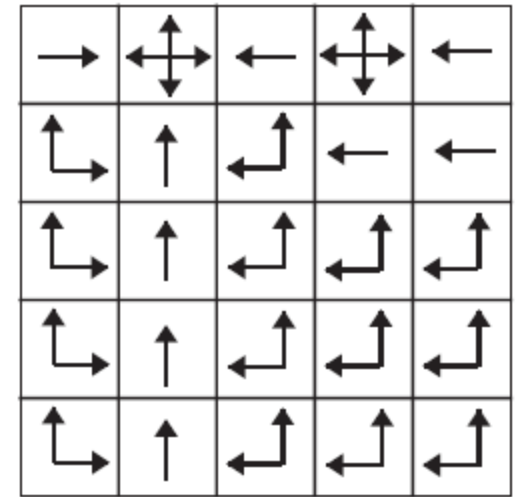
Example: Gridworld



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_*



Example: Recycling Robot

- h, l, s, w, re : high, low, search, wait, recharge
- For any choice of r_s, r_w, α, β , and γ , with $0 \leq \gamma < 1, 0 \leq \alpha, \beta \leq 1$, there is exactly one pair of numbers $v_*(h)$ and $v_*(l)$, that simultaneously satisfy the following equations

- $$v_*(h) = \max \left\{ \begin{array}{l} p(h|h,s)[r(h,s,h) + \gamma v_*(h)] + p(l|h,s)[r(h,s,l) + \gamma v_*(l)], \\ p(h|h,w)[r(h,w,h) + \gamma v_*(h)] + p(l|h,w)[r(h,w,l) + \gamma v_*(l)] \end{array} \right\}$$

$$= \max \left\{ \begin{array}{l} \alpha[r_s + \gamma v_*(h)] + (1 - \alpha)[r_s + \gamma v_*(l)], \\ 1[r_w + \gamma v_*(h)] + 0[r_w + \gamma v_*(l)] \end{array} \right\}$$

$$= \max \left\{ \begin{array}{l} r_s + \gamma[\alpha v_*(h) + (1 - \alpha)v_*(l)], \\ r_w + \gamma v_*(h) \end{array} \right\}.$$

- $$v_*(l) = \max \left\{ \begin{array}{l} \beta r_s - 3(1 - \beta) + \gamma[(1 - \beta)v_*(h) + \beta v_*(l)], \\ r_w + \gamma v_*(l), \\ \gamma v_*(h) \end{array} \right\}.$$



Bellman Optimality Equation

- Explicitly solving Bellman optimality equation leads to solving the RL problem; however, it is rarely useful in practice
 - 1) We do not know the exact dynamics of the environment
 - 2) We do not have enough computational resources to complete the computation
 - 3) Markov property may not be true
- E.g., backgammon
 - It has 10^{20} states, and thus take thousands of years to solve the Bellman equation
 - Needs approximate computation




Bellman Optimality Equation

- Many decision-making methods can be viewed as ways of approximately solving the Bellman optimality equation
 - $v_*(s) = \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma v_*(s')]$
- Heuristic search method
 - Expand the Bellman optimality equation several times (up to some depth), forming a “tree” of possibilities, and then using a heuristic evaluation function to approximate v_* at the “leaf” nodes
- DP (dynamic programming) is more closely related to the Bellman Optimality Equation
- Many RL methods approximately solve the Bellman optimality equation using actual experienced transitions in place of knowledge of the expected transitions



Outline

- Agent-Environment Interface
- Goals and Rewards
- Returns and Episodes
- Episodic and Continuing Tasks
- Policies and Value Functions
- Optimal Policies and Value Functions
-  **Optimality and Approximation**
- Conclusion



Optimality and Approximation

- An agent that learns an optimal policy has done very well, but in practice this rarely happens
- For the kinds of tasks in which we are interested, optimal policies can be generated only with extreme computational cost
- Optimality is an ideal that agents can only approximate to varying degrees, due to
 - Computational limitation
 - Memory limitation



Optimality and Approximation

- Computational limitation
 - Chess: it is difficult to exactly solve the Bellman optimality equation
- Memory limitation
 - For small and finite states, it is possible to make a table of storing values, policies, and models
 - When there are too many states, we need parameterized function approximation




Optimality and Approximation

- Approximation for RL presents challenges
 - In approximating optimal behavior, there may be many states that the agent faces with such a low probability that selecting suboptimal actions for them has little impact on the amount of reward the agent receives
 - E.g., Tesauro's backgammon player plays with exceptional skill even though it might make very bad decisions on board configurations that never occur in games against experts.
 - The online nature of RL makes it possible to approximate optimal policies in ways that put more effort into learning to make good decisions for frequently encountered states, at the expense of less effort for infrequently encountered states
 - This is one key property that distinguishes RL from other approaches to approximately solving MDPs



Outline

- Agent-Environment Interface
- Goals and Rewards
- Returns and Episodes
- Episodic and Continuing Tasks
- Policies and Value Functions
- Optimal Policies and Value Functions
- Optimality and Approximation
-  **Conclusion**



Conclusion

■ RL

- ❑ Learning from interaction how to behave in order to achieve a goal
- ❑ RL agent and its environment interact over a sequence of discrete time steps
- ❑ Actions: choices made by the agent
- ❑ States: basis for making the choices
- ❑ Rewards: the basis for evaluating the choices
- ❑ Environment: everything inside the agent is completely known and controllable by the agent; everything outside is incompletely controllable but may or may not be completely known
- ❑ Policy: a stochastic rule from state to action
- ❑ Agent's goal: maximize the amount of reward it receives over time



Conclusion

- Markov decision process (MDP)
 - State, action, and reward sets with well-defined transition probabilities
 - Return: function of future rewards that the agent seeks to maximize in expectation
 - Undiscounted return: appropriate for episodic tasks
 - Discounted return: appropriate for continuing tasks



Conclusion

- Markov decision process (MDP)
 - A policy's value functions map each state, or state–action pair to the expected return from that state, or state–action pair, given that the agent uses the policy
 - Optimal value functions: give the largest expected return achievable by any policy
 - Optimal policy: a policy whose value functions are optimal
 - The optimal value functions are unique for a given MDP, but there can be many optimal policies
 - Any policy that is greedy with respect to the optimal value functions must be an optimal policy
 - Bellman optimality equations should be satisfied by the optimal value functions; given the optimal value functions, an optimal policy can be determined easily



Conclusion

- Approximation
 - RL agent may have complete knowledge of the environment, or incomplete knowledge of it
 - Even if the agent has a complete knowledge of the environment, the agent cannot exactly compute it
 - Computational limitation
 - Memory limitation
 - Approximation is the key to RL



Exercise

- (Question 1)
- Suppose that we have a small gridworld and agent travels each state with the equiprobable random policy. There are four actions possible in each state, $A = \{up, down, right, left\}$, which deterministically cause the corresponding state transitions, except that actions that would take the agent off the grid in fact leave the state unchanged. The reward is -1 on all transitions until the terminal state (shaded) is reached.

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

0.0	-14.0	-18.0	-22.0
-14.0	-17.0	-21.0	-18.0
-18.0	-21.0	-17.0	-14.0
-22.0	-18.0	-14.0	0.0

$k = \infty$

	←	←	↙
↑	↖	↘	↓
↑	↗	↘	↓
↖	→	→	



Exercise

- (Question 1)
- Now, suppose a new state 15 is added to the gridworld just below state 13, and its actions, left, up, right, and down, take the agent to states 12, 13, 14, and 15, respectively. Assume that the transitions and the values of the original states are unchanged. What is $v_\pi(15)$ with the discount rate $\gamma = 0.9$ for the equiprobable random policy in this case?

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

0.0	-14.0	-18.0	-22.0
-14.0	-17.0	-21.0	-18.0
-18.0	-21.0	-17.0	-14.0
-22.0	-18.0	-14.0	0.0

$k = \infty$

	←	←	↙
↑	↖	↙	↓
↑	↗	↘	↓
↖	→	→	



Exercise

- (Answer)
- We need to calculate state-value function for state 15. Since we have converged state-value function for all states except the new state 15, we may calculate $v_\pi(15)$ with the neighbor state-value functions.
- From the state-value function equation $V_\pi(s) = \sum_{a \in A} \pi(s|a) * \sum_{s' \in S} P_{s,s'}^a * [r(s') + \gamma * V_\pi(s')]$, state-value function for the new state 15 can be formulized as,
- $$V_\pi(15) = \frac{1}{4} * [\{-1 + \gamma V_\pi(12)\} + \{-1 + \gamma V_\pi(13)\} + \{-1 + \gamma V_\pi(14)\} +$$



Questions?