ST449 Artificial Intelligence and Deep Learning

Lecture 6

Introduction to reinforcement learning

Milan Vojnovic

https://github.com/lse-st449/lectures

This lecture topics

- Multi-armed bandits
- Exploitation vs exploration tradeoff

- Reinforcement learning problem
- Markov Decision Process (MDP) formulation

Our scope

- Computational approaches to learning from interaction
  - Not about how people or animals learn
- Focus on
  - Idealized learning situations and
  - Efficiency of various learning methods
  - Perspective of an artificial intelligence researcher or engineer
- Reinforcement learning: learning what to do
  - Mapping situations to actions to maximize a reward
  - The learner must discover which actions yield the most reward by trying them
  - Two characteristics: trial-and-error search and delayed rewards

The RL learning problem

- Reinforcement learning is defined by a learning problem
  - Different from a learning method
  - Any method suited for solving an RL problem is considered a RL method

- Reinforcement learning is different from supervised learning

  - Supervised learning: examples provided by a knowledgeable external supervisor
    - Ex. statistical pattern recognition, image recognition, …

  - Reinforcement learning: an agent learns from its own experience

Tradeoff: exploitation vs exploration

- Exploitation: to maximize reward, the agent must prefer actions it has tried in the past and found to yield a reward
- Exploration: to discover which actions yield a high reward, the agent must try actions that it has not selected before
- Tradeoff between exploitation and exploration:
  - Neither exploitation nor exploration can be used exclusively
  - The agent must try various actions and progressively favor high-reward actions
- This trade-off does not arise in supervised learning

Multi-armed bandits

Multi-armed bandit problem

- k-armed bandit problem
- Each arm when pulled (experimented) yields an independent reward
- a_t = arm pulled at time step (experiment) t
- r_t = reward at time step t
- Expected reward at time step t: $Q_t^*(a) = \mathbf{E}[r_t \mid a_t = a]$ (unknown)

Action-value methods

- Action-value methods: estimating the value of actions and using these estimates to make action selection decisions

- Average reward received:

$$Q_t(a) = \frac{\sum_{i=1}^{t-1} r_i \mathbf{1}_{\{a_i=a\}}}{\sum_{i=1}^{t-1} \mathbf{1}_{\{a_i=a\}}}$$

- Greedy policy: greedy action selection

$$a_t = \arg\max_a Q_t(a)$$

- Greedy policy issues
  - Always exploits: may be suboptimal in the long run
  - Need to explore different arms to learn about their value

ϵ-greedy: simple action selection policy

- Initialization: choose a small value parameter $\epsilon \in (0,1]$

- At each step perform:
  - With probability $1 - \epsilon$: make a greedy choice
  - Otherwise: choose a randomly selected arm from the set of all arms

- This simple policy combines exploitation and exploration
  - At each time step, any given arm is selected with probability $\geq \epsilon/k$
  - Greedy choice made with probability $\geq 1 - \epsilon + \epsilon/k$

Incremental implementation

- $N_t(a)$ = number of time steps in which arm a is pulled by time step t
- $t_a(i)$ = time step of the i-th selection of arm a
- Average reward received from arm a by time step t:

$$Q_t(a) = \frac{\sum_{i=1}^{N_t(a)} r_{t_a(i)}}{N_t(a)}$$

- Suppose arm a is selected at time step t:

$$Q_{t+1}(a) = \frac{\sum_{i=1}^{N_t(a)} r_{t_a(i)} + r_t}{N_t(a)+1} = \frac{N_t(a)}{N_t(a)+1} Q_t(a) + \frac{1}{N_t(a)+1} r_t$$

$$= Q_t(a) + \frac{1}{N_t(a)+1}(r_t - Q_t(a))$$

ϵ-greedy bandit algorithm

- **Initialization**: $Q(a) = 0, N(a) = 0$, for $a = 1,2, \dots, k$

- **While** 1

$$a^* \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1 - \epsilon \\ \text{random arm} & \text{with probability } \epsilon \end{cases}$$

$$r \leftarrow \text{bandit\_reward}(a)$$

$$N(a) \leftarrow N(a) + 1$$

$$Q(a) \leftarrow Q(a) + \frac{1}{N(a)}(r - Q(a))$$

Example: four Bernoulli arms

Reward distributions

Bernoulli(0.1)

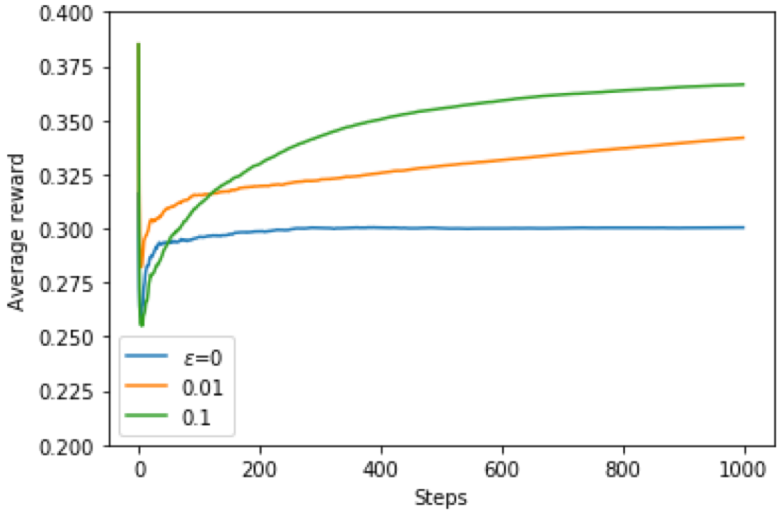
Bernoulli(0.4)

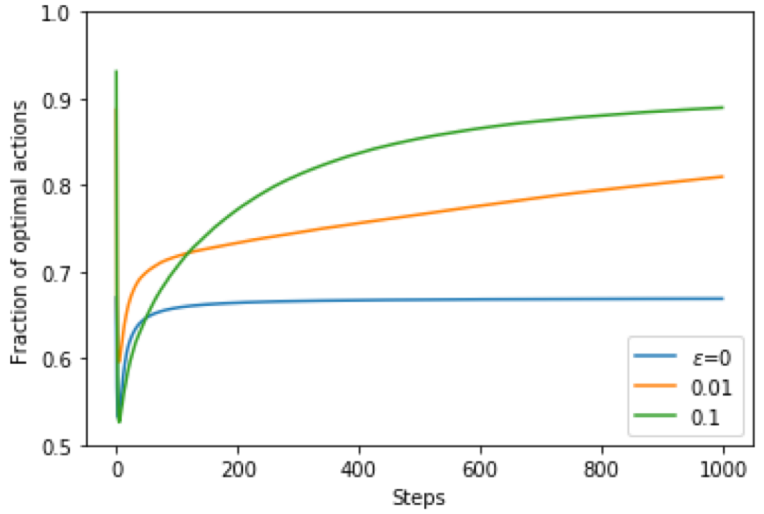
Bernoulli(0.1)

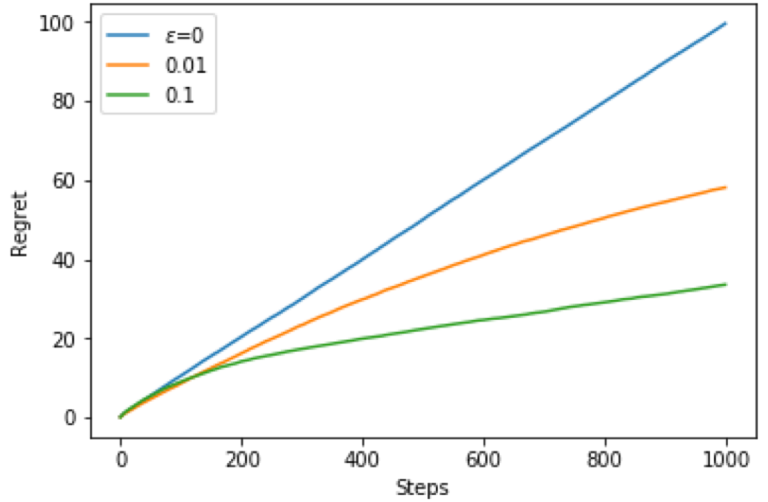
Bernoulli(0.1)

Best arm

Example: four Bernoulli arms (cont'd)







- Average results obtained by using 1000 independent runs
- $\text{Regret}(T) := \max_a Q^*(a)T - \sum_{t=1}^{T} Q^*(a_t)$

Tracking in nonstationary problems

- The incremental update for nonstationary problems:

  $Q_{t+1}(a) = Q_t(a) + \alpha(r_t - Q_t(a))$

  where $\alpha \in (0,1]$ is a step size parameter

- Exponential weighted moving average: for a such that $a_t = a$

  $Q_{t+1}(a) = Q_t(a) + \alpha(r_t - Q_t(a))$

  $= \alpha r_{t_a(N_t(a)+1)} + (1-\alpha) Q_{t_a(N_t(a))}(a)$

  $= \alpha\, r_{t_a(N_t(a)+1)} + (1-\alpha)\left[\alpha r_{t_a(N_t(a))} + (1-\alpha) Q_{t_a(N_t(a)-1)}\right]$

  …

  $= (1-\alpha)^{N_t(a)+1} Q_1(a) + \sum_{i=1}^{N_t(a)+1} \alpha(1-\alpha)^{N_t(a)+1-i} r_{t_a(i)}$

Upper-confidence-bound (UCB) algorithm

- Select action according to:

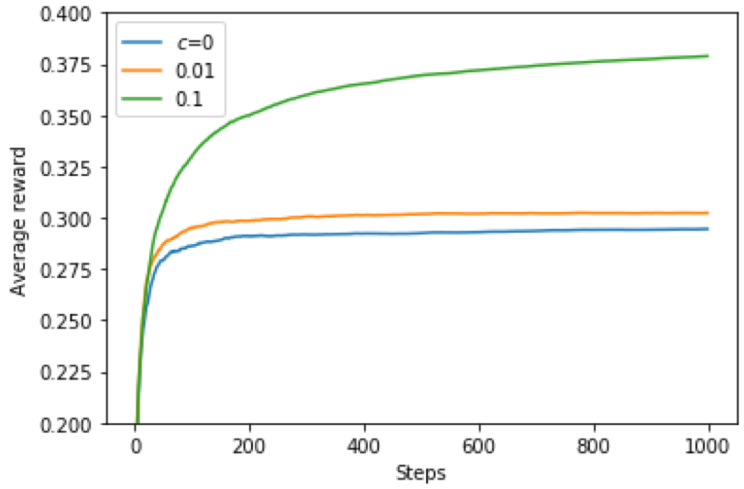
$$a_t = \arg\max_a \left(Q_t(a) + c\sqrt{\frac{\log(t)}{N_t(a)}} \right)$$

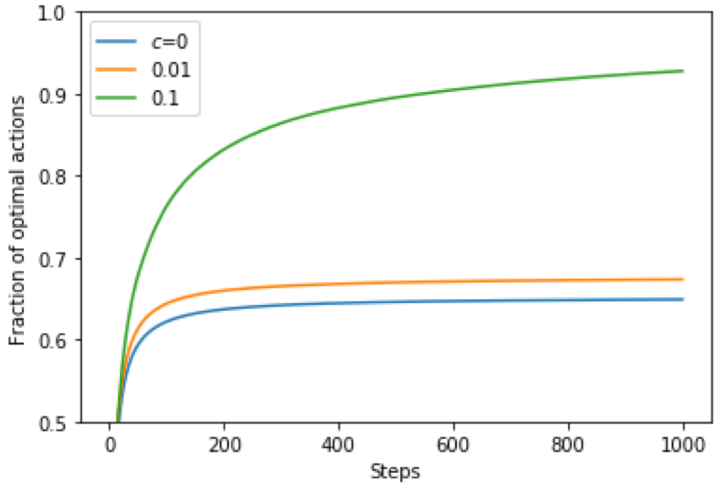
  where c is a positive constant

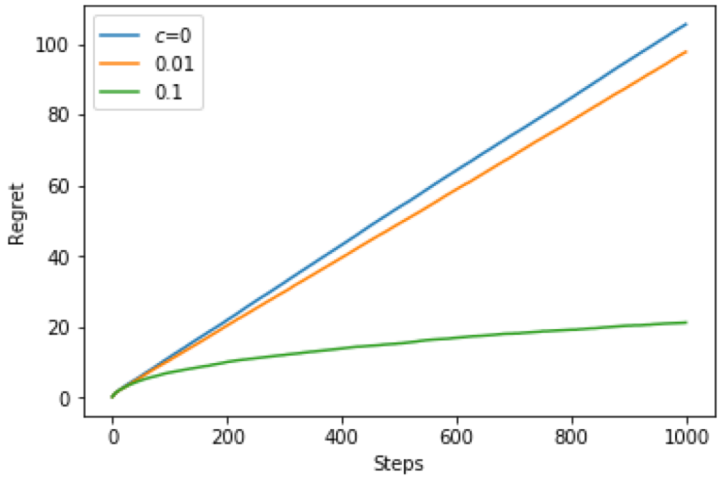
- The upper-confidence interval term $\sqrt{\log(t)/N_t(a)}$ accounts for uncertainty of the arm value estimate

- Exploration achieved by
  - Prioritizing arms that haven't been explored much (small $N_t(a)$)
  - For arms not being experimented for some number of steps, the upper confidence interval term increases proportionally to $\sqrt{\log(t)}$

Four Bernoulli arms revisited







Gradient bandit algorithms

- Each arm associated with a preference score $H_t(a)$

- Randomized policy:

$$\pi_t(a) \coloneqq \mathbf{Pr}[a_t = a] = \frac{e^{H_t(a)}}{\sum_{j=1}^{k} e^{H_t(j)}}, \text{ for } a = 1,2,\dots,k$$

- The preference scores updated as:

$$H_{t+1}(a) = H_t(a) + \eta(r_t - \bar{R}_t)\left(\mathbf{1}_{\{a_t=a\}} - \pi_t(a)\right), \text{ for } a = 1,2,\dots,k$$

where $\bar{R}_t$ is a baseline defined as average reward up to and including time step t

Gradient based algorithms (cont'd)

- The expected reward objective:

$$\mathbf{E}[r_t] = \sum_a \pi_t(a) Q^*(a)$$

- The gradient ascent algorithm:

$$H_{t+1}(a) = H_t(a) + \eta \frac{\partial}{\partial H_t(a)} \mathbf{E}[r_t] \text{ for } a = 1,2,\dots,k$$

- By a straightforward derivation:

$$\frac{\partial}{\partial H_t(a)} \pi_t(a') = (\mathbf{1}_{\{a'=a\}} - \pi_t(a))\pi_t(a')$$

Gradient based algorithms (cont'd)

- The a-th element of the gradient vector $\nabla_{H_t}\mathbf{E}[r_t]$ satisfies:

$$\frac{\partial}{\partial H_t(a)}\mathbf{E}[r_t] = \sum_{a'}(Q^*(a') - B_t)\frac{\partial}{\partial H_t(a)}\pi_t(a')$$

$$= \mathbf{E}\left[(Q^*(a_t) - B_t)\frac{\partial}{\partial H_t(a)}\pi_t(a_t)\frac{1}{\pi_t(a_t)}\right]$$

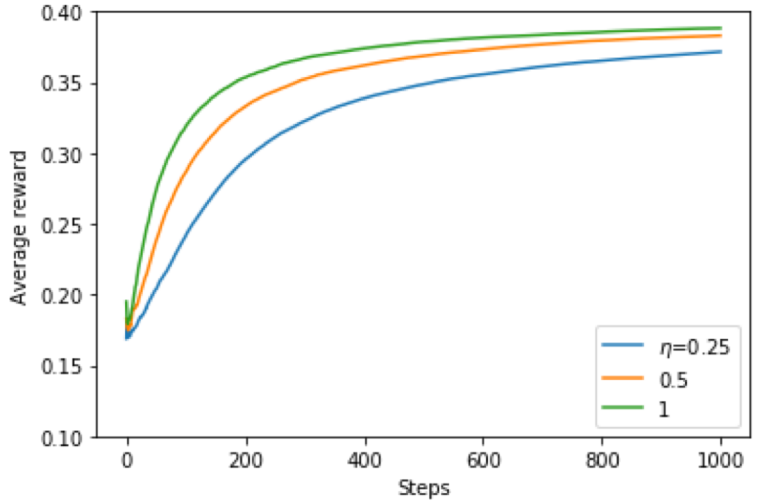
$$= \mathbf{E}\left[(r_t - B_t)\frac{\partial}{\partial H_t(a)}\pi_t(a_t)\frac{1}{\pi_t(a_t)}\right]$$

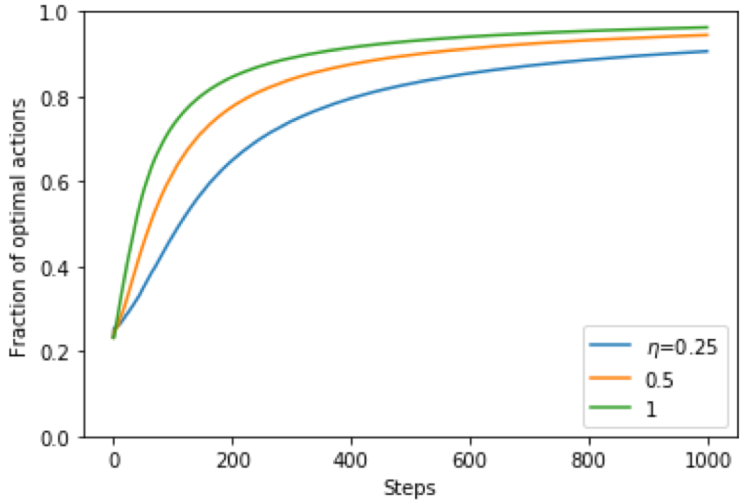
$$= \mathbf{E}\left[(r_t - B_t)(1_{\{a_t=a\}} - \pi_t(a))\right]$$

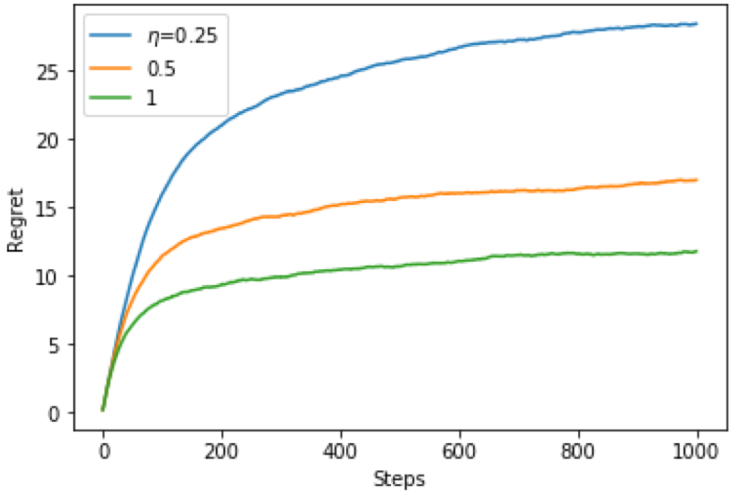
  where B_t is a baseline

- Interpretation: $(r_t - \bar{R}_t)\left(\mathbf{1}_{\{a_t=a\}} - \pi_t(a)\right)$ is the a-th element of a stochastic gradient vector

Four Bernoulli arms revisited







Reinforcement learning problem

Agent, environment, and tasks

- Agent: the learner (decision maker)
  - Makes actions aiming to maximize a long run reward
- Environment: the thing the agent interacts with
  - After each selection of an action, presents a new situation and gives an (instantaneous) reward to the agent
- Task: specification of an environment (one instance of a RL problem)

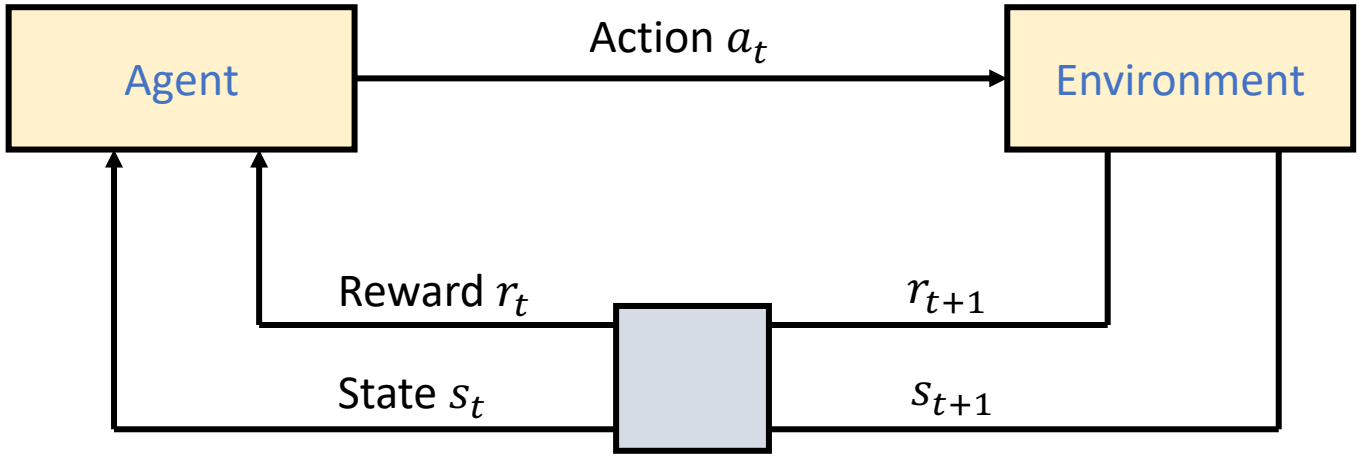
State and actions

- The agent and environment interact at discrete time steps $t = 0,1, \ldots$

- At each time step t, the agent:
  - Receives some representation of the **environment state** $s_t \in S$, where S is the set of possible states
  - Selects an **action** $a_t \in A(s_t)$, where $A(s_t)$ is the set of available actions in state s_t

- At next time step $t + 1$, the agent:
  - Receives a **numerical reward** r_{t+1} and
  - Finds itself in a **new state** s_{t+1}

Agent-environment interaction



Control engineering vocabulary

- agent = controller
- environment = controlled system (plant)
- action = control signal

The agent's policy

- At each time step, the agent implements a mapping from states to a probability distribution over actions

- This mapping is called the agent's policy:

$$\pi_t(s, a) = \mathbf{Pr}[a_t = a \mid s_t = s]$$

- Deterministic policy: π_t has all its mass on one action

Goals and rewards

- The agent's goal: maximize the cumulative reward defined as the total amount of reward received in long run

- In general, different from instantaneous rewards $r_t \in \mathbf{R}$ at time step t

- Examples:
  - Robot learning to walk
  - Robot learning to escape a maze
  - Robot learning to find and collect empty soda cans for recycling

Cumulative rewards for episodic tasks

- Episodic task: agent-environment interactions can be broken into subsequences or episodes
  - Each episode starts in an initial state and ends in a terminal state

- Examples of episodic tasks
  - Plays of a game
  - Trips through a maze
  - Other repeated interactions

- For episodic tasks, the cumulative reward can be defined as the sum of rewards:

$$R_t = r_{t+1} + r_{t+2} + \cdots + r_T$$

where T is the final time step of the task

Cumulative rewards for continuing tasks

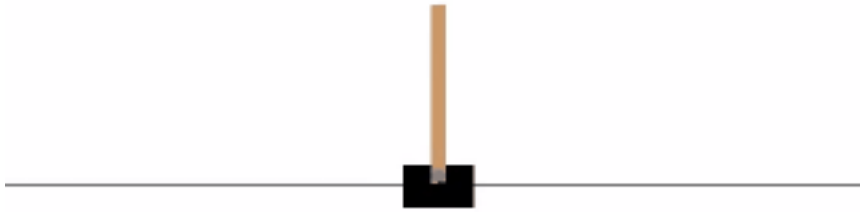
- Continuing task: an agent-environment interaction that goes on without a limit
  - Defining the cumulative reward as the sum of future rewards is problematic as it may be infinite

- Resolved by using the concept of discounted rewards:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

where γ is a parameter $0 \leq \gamma < 1$ (called the discount rate)

Example: cart-pole balancing



- Apply forces to a cart moving along a track to keep a pole hinged to the cart from falling over
- A failure occurs if the pole falls past a given angle from vertical or if the cart reaches an end of the track
- The pole is reset to vertical position after each failure
- Goal: minimize the failure rate
  - Reward of $+1$ for each time step in which a failure did not occur, otherwise 0
  - Reward of -1 for each time step in which a failure occurred, otherwise 0

Markov Decision Process (MDP)

- Key assumption: Markov property

$$\mathbf{Pr}[s_{t+1} = s', r_{t+1} = r \mid F_t] = \mathbf{Pr}[s_{t+1} = s', r_{t+1} = r \mid s_t, a_t]$$

  for every history $F_t = (s_t, a_t, r_t, \ldots, r_1, s_0, a_0)$

- Markov Decision Process (MDP): a RL task satisfying the Markov property

- Finite MDP: a MAP with finite action spaces, defined by
  - State and action sets, and
  - One-step dynamics of the environment

Markov Decision Process (cont'd)

- State transition probabilities: Given any state s and action a, the transition probability to the next state s' is given by

$$P_{s,s'}^{a} = \mathbf{Pr}[s_{t+1} = s' \,|\, s_t = s, a_t = a]$$

- Conditional expected rewards: the expected reward in the next time step, conditional on the current state s, action a, and next state s':

$$R_{s,s'}^{a} = \mathbf{E}[r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s']$$

Example: recycling robot MDP

- A mobile robot whose job is to collect empty soda cans in an office environment
- The robot has sensors for detecting cans and an arm and gripper that can pick them up and place them in an onboard bin
- The robot runs on a rechargeable battery
- High-level decisions how to search for cans are made by a reinforcement learning agent based on the current charge level of the battery

This agent has to decide whether the robot should

- Actively search for a can for a certain period of time, or
- Remain stationary and wait for someone to bring it a can, or
- Head back to its home base to recharge its battery

Example: recycling robot MDP (cont'd)

- Suppose the environment is such that
  - 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, there is a possibility that its battery depletes
  - If the battery depletes while the robot is searching, the robot must shut down and wait to be rescued incurring a reward penalty

- Suppose the environment (battery) has two states low and high:

$$\text{State space: } S = \{\text{low}, \text{high}\}$$

- The agent's action sets:

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

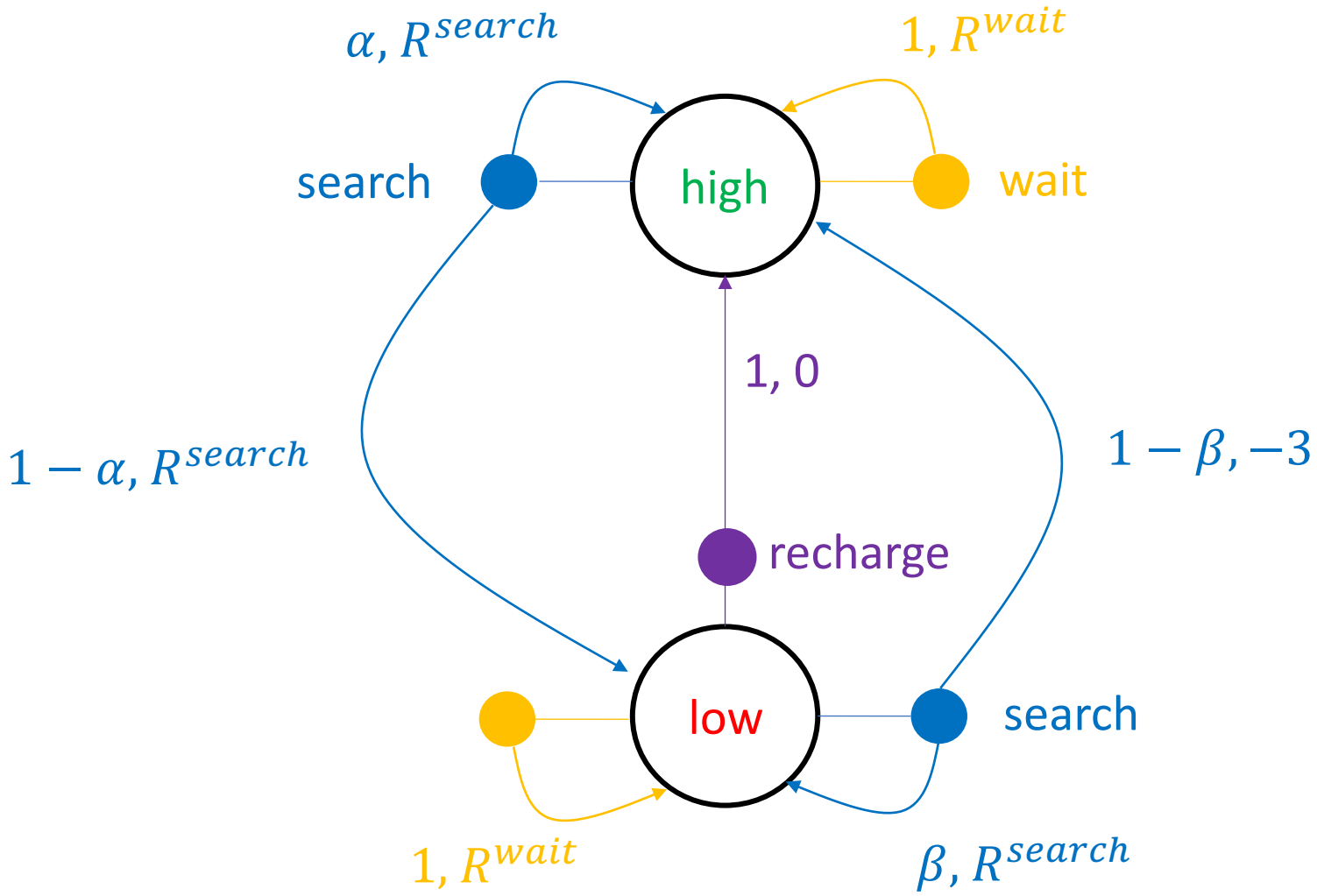
$$A(\text{low}) = \{\text{search}, \text{wait}, \text{recharge}\}$$

Example: recycling robot MDP (cont'd)

s	s'	a	$P^a_{s,s'}$	$R^a_{s,s'}$
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

Assumption "the best way to find cans is to actively search for them": $R^{search} > R^{wait}$

Example: recycling robot MDP (cont'd)



State value function

- State value function $V : S \rightarrow \mathbf{R}$: the expected future return from a given state

- The state-value function under a policy π:

$$V^{\pi}(s) = \mathbf{E}_{\pi}[R_t \mid s_t = s] = \mathbf{E}_{\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s]$$

Action value functions

- Action value function $Q : S \times A \rightarrow \mathbf{R}$: quantifies the expected future return by performing a given action at a given state

- The action value function under a policy π:

$$Q^{\pi}(s,a) = \mathbf{E}_{\pi}[R_t \mid s_t = s, a_t = a] = \mathbf{E}_{\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a]$$

Bellman equation

- The state value function satisfies the Bellman equation:

$$V^{\pi}(s) = \sum_{a \in A} \pi(s,a) \sum_{s' \in S} P^{a}_{s,s'} [R^{a}_{s,s'} + \gamma V^{\pi}(s')], \text{ for all } s \in S$$

- Exercise: show this

- The value function V^{π} is the unique solution of its Bellman equation

Exercise solution

- $V^{\pi}(s) = \mathbf{E}_{\pi}[R_t | s_t = s]$

$= \mathbf{E}_{\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s]$

$= \mathbf{E}_{\pi}[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_t = s]$

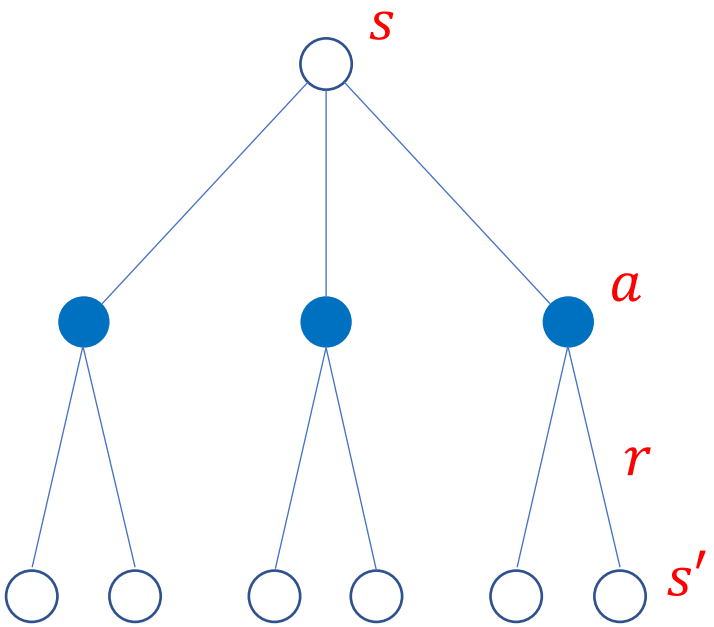
$= \sum_a \pi(s,a) \sum_{s'} P^a_{s,s'} \left[R^a_{s,s'} + \gamma \mathbf{E}_{\pi}[\sum_{k=0}^{\infty} \gamma^k \, r_{t+k+2} \mid s_{t+1} = s'] \right]$ (Markov property)

$= \sum_a \pi(s,a) \sum_{s'} P^a_{s,s'} [R^a_{s,s'} + \gamma V^{\pi}(s')]$

Backup diagrams

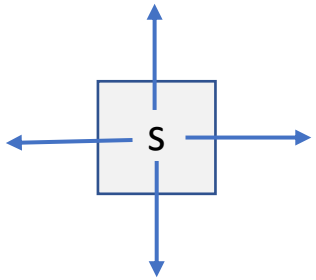
- Backup diagram: represents relationships that are basis of the update (backup) operators

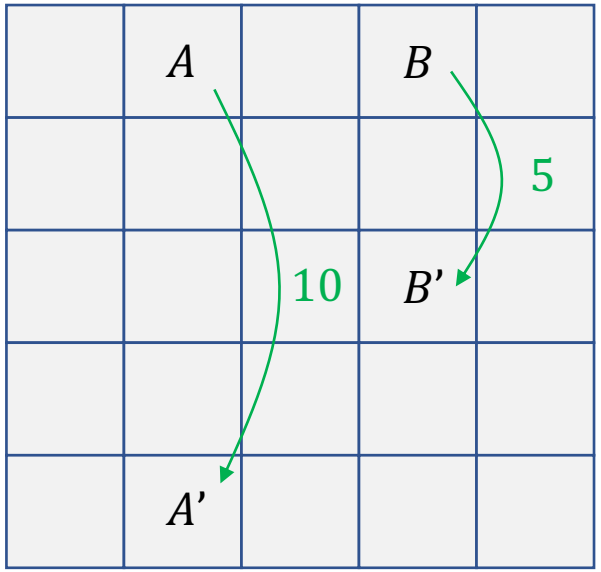
- Backup diagram for V^{π}:



Gridworld example

- Action set for state s:
  $A(s) = \{\text{north}, \text{south}, \text{east}, \text{west}\}$





- Actions that that would take the agent off the grid leave its location unchanged and yield a reward of -1

- Other actions result in a reward of 0, expect for

  state A: all actions yield a reward of 10 and take the agent to state A', and

  state B: all actions yield a reward of 5 and take the agent to state B'

Optimal value functions

- A policy π is said to be better than or equal to a policy π' if

$$V^{\pi}(s) \geq V^{\pi'}(s) \text{ for all } s \in S$$

- A policy π^* is said to be an optimal policy if

$$V^{\pi^*}(s) \geq \max_{\pi} V^{\pi}(s) \text{ for all } s \in S$$

- Optimal policies share the same optimal action value function:
  - $Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a)$ for all $s \in S, a \in A(s)$
  - Note: $Q^*(s,a) = \mathbf{E}[r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a]$

Bellman optimality equation

- The optimal value function V^* satisfies

$$V^*(s) = \max_a \sum_{s' \in S} P^a_{s,s'} [R^a_{s,s'} + \gamma V^*(s')], \text{ for } s \in S$$

- Exercise: show this

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

Bellman optimality equation cont'd

- $V^*(s) = \max_a Q^{\pi^*}(s,a)$

$= \max_a \mathbf{E}_{\pi^*}[R_t \mid s_t = s, a_t = a]$

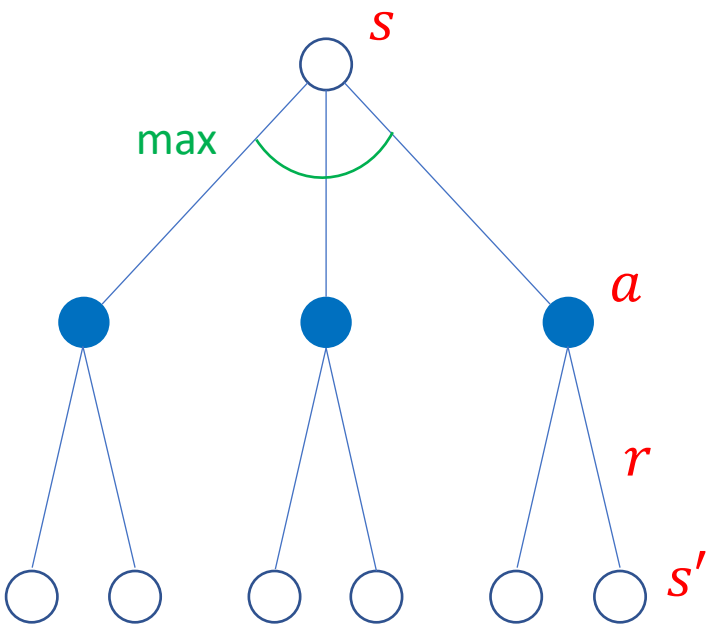
$= \max_a \mathbf{E}_{\pi^*}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\,, a_t = a]$

$= \max_a \mathbf{E}_{\pi^*}[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_t = s, a_t = a]$

$= \max_a \mathbf{E}_{\pi^*}[r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a]$

$= \max_a \sum_{s'} P^a_{s,s'} [R^a_{s,s'} + \gamma V^*(s')]$

Backup diagram

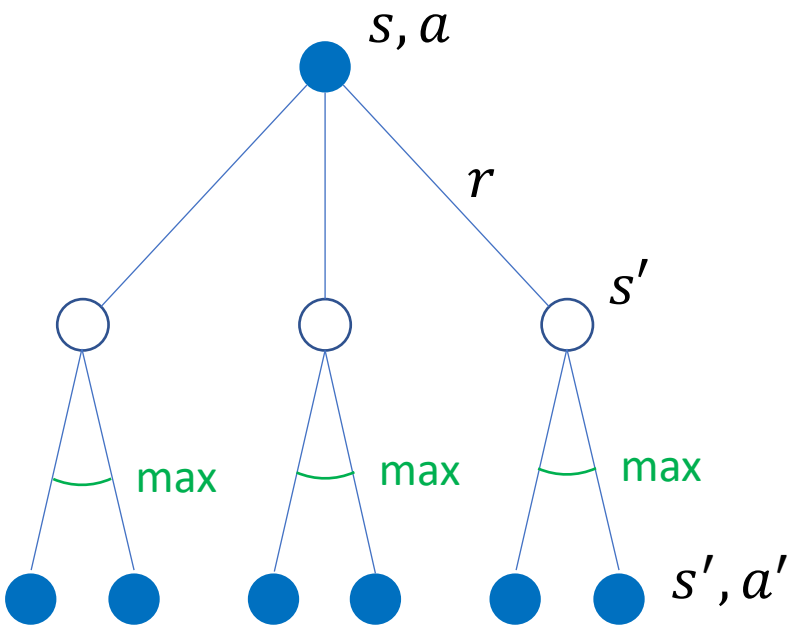


Bellman optimality equation cont'd

- For optimal action value function:

$$Q^*(s,a) = \sum_{s' \in S} P^a_{s,s'} [R^a_{s,s'} + \gamma \max_{a'} Q^*(s',a')], \text{ for } s \in S, a \in A(s)$$

- Backup diagram:



References

- R. S. Sutton and A. G. Barto, Reinforcement Learning, Chapters 1 and 3, 1998
- R. S. Sutton and A. G. Barto, Reinforcement Learning, Second edition, Chapter 2: Multi-armed bandits, 2018

Seminar exercises

- Getting started with OpenAI Gym
- Multi-armed bandits problem