

# ST449 Artificial Intelligence and Deep Learning

## Lecture 7

### Dynamic programming and Monte Carlo methods



Milan Vojnovic

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

# Topics of this lecture

- Elementary solution methods:
  - Dynamic programming
  - Monte Carlo methods
- Next lecture: elementary solution methods cont'd
  - Temporal-difference learning

# Dynamic programming (DP)

# Dynamic Programming

- **Dynamic Programming (DP)**: a collection of algorithms used to compute optimal policies **given a perfect knowledge of the environment as a MDP**
- DP algorithms are rarely used directly in practice
- However, they provide a foundation for other solution methods
  - Other methods can be seen as trying to achieve the same but with less computation and without assuming a perfect knowledge of the environment

# Our focus: finite MDPs

- Environment modeled by a **finite MDP**: finite state and action sets  $S, A(s), s \in S$
- Dynamics specified by the **transition probabilities**

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

and the **immediate expected rewards** for actions and state transitions:

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

- DP ideas can be applied also to problems with continuous state and action sets
  - A common approach is to use quantization

# Bellman optimality equations

- The optimal state value function  $V^*$  satisfies:

$$V^*(s) = \max_a \sum_{s' \in S} P_{s,s'}^a [R_{s,s'}^a + \gamma V^*(s')], \text{ for } s \in S$$

- The optimal action value function  $Q^*$  satisfies:

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

# Policy evaluation

- **Policy evaluation**: computation of the state value function  $V^\pi$  for a given policy  $\pi$
- Also referred to as **the prediction problem**
- For any given policy  $\pi$ , the existence and uniqueness of  $V^\pi$  are guaranteed whenever either
  - The discount rate satisfies  $\gamma < 1$ , or
  - Eventual termination is guaranteed from all states under policy  $\pi$
- $V^\pi$  is a solution of a system of  $|S|$  linear equations with  $|S|$  unknowns

# Iterative policy evaluation

- Iterative policy evaluation: an iterative solution method that outputs a sequence of approximate state-value functions  $V_0, V_1, \dots$ 
  - Initial value function  $V_0$  is chosen arbitrarily subject to the constraint that at any terminal state it has value equal to 0
  - Iterative update rule:

$$\begin{aligned} V_{k+1}(s) &= \mathbf{E}_{\pi}[r_{t+1} + \gamma V_k(s_{t+1}) \mid s_t = s] \\ &= \sum_a \pi(s, a) \sum_{s'} P_{s,s'}^a [R_{s,s'}^a + \gamma V_k(s')] \quad \text{for } s \in S \end{aligned}$$

- The limit point of  $V_k$  as  $k \rightarrow \infty$  is  $V^{\pi}$  under the same conditions that guarantee the existence of  $V^{\pi}$



# Iterative policy evaluation cont'd

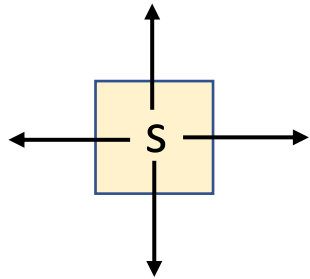
- The iterative policy evaluation in the previous slide is referred to as a **full backup iterative method**
  - In each time step, the state value function is updated based on the values of states evaluated in the previous time step
- An alternative backup method:
  - Updating the state value function at a state by using the most recent updates of the state value function at other states
  - Pseudo-code in the next slide

# Iterative policy evaluation: pseudo-code

- **Input:** a policy  $\pi$ ,  $\theta$  (a small positive number)
- Initialization:  $V(s) = 0$  for all  $s \in S^+$
- **Repeat:**
  - $\Delta \leftarrow 0$
  - For each**  $s \in S$ :
    - $v \leftarrow V(s)$
    - $V(s) \leftarrow \sum_a \pi(s, a) \sum_{s'} P_{s,s'}^a [R_{s,s'}^a + \gamma V(s')]$
    - $\Delta = \max\{\Delta, |v - V(s)|\}$
- until**  $\Delta < \theta$
- **Output:**  $V$  (approximation of  $V^\pi$ )

$S^+ := S \cup S^0$  where  $S^0$  is the set of terminal states

# Example: GridWorld



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



terminal state

- Undiscounted, episodic task
- Action set for state  $s$ :  $A(s) = \{\text{up, down, right, left}\}$
- Action that would take the agent off the grid leave the state unchanged
- Rewards: for each transition, the reward of value  $-1$

# GridWorld: values of equiprobable random policy

$V^\pi(s)$ :

0	-14 1	-20 2	-22 3
-14 4	-18 5	-20 6	-20 7
-20 8	-20 9	-18 10	-14 11
-20 12	-20 13	-14 14	0

Q1: What is the value of  $Q^\pi(11, \text{down})$ ?

Q2: What is the value of  $Q^\pi(7, \text{down})$  ?

# Value function evaluation

By symmetry:

0	$v_1$ 1	$v_2$ 2	$v_3$ 3
$v_1$ 4	$v_5$ 5	$v_6$ 6	$v_2$ 7
$v_2$ 8	$v_6$ 9	$v_5$ 10	$v_1$ 11
$v_3$ 12	$v_2$ 13	$v_1$ 14	0

Bellman optimality equation:

$$v_1 = \frac{1}{4}(-1 + v_2) + \frac{1}{4}(-1 + v_5) + \frac{1}{4}(-1 + 0) + \frac{1}{4}(-1 + v_1)$$

$$v_2 = \frac{1}{4}(-1 + v_3) + \frac{1}{4}(-1 + v_6) + \frac{1}{4}(-1 + v_1) + \frac{1}{4}(-1 + v_2)$$

$$v_3 = \frac{1}{4}(-1 + v_3) + \frac{1}{4}(-1 + v_2) + \frac{1}{4}(-1 + v_2) + \frac{1}{4}(-1 + v_3)$$

$$v_5 = \frac{1}{4}(-1 + v_6) + \frac{1}{4}(-1 + v_6) + \frac{1}{4}(-1 + v_1) + \frac{1}{4}(-1 + v_1)$$

$$v_6 = \frac{1}{4}(-1 + v_2) + \frac{1}{4}(-1 + v_5) + \frac{1}{4}(-1 + v_5) + \frac{1}{4}(-1 + v_2)$$

$$\Rightarrow (v_1, v_2, v_3, v_5, v_6) = (-14, -20, -22, -18, -20)$$

# GridWorld: optimal value and policy

0	-1	-2	-3
-1	-2	-3	-2
-2	-3	-2	-1
-3	-2	-1	0

	←	←	↙
↑	↖	↕	↓
↑	↕	↘	↓
↙	→	→	

# GridWorld: optimal value and policy (cont'd)

0	$v_1$ 1	$v_2$ 2	$v_3$ 3
$v_1$ 4	$v_5$ 5	$v_6$ 6	$v_2$ 7
$v_2$ 8	$v_6$ 9	$v_5$ 10	$v_1$ 11
$v_3$ 12	$v_2$ 13	$v_1$ 14	0

Bellman's optimality equations:

- 1  $v_1 = \max\{-1 + v_2, -1 + v_5, -1 + 0, -1 + v_1\}$
- 2  $v_2 = \max\{-1 + v_3, -1 + v_6, -1 + v_1, -1 + v_2\}$
- 3  $v_3 = \max\{-1 + v_3, -1 + v_2, -1 + v_2, -1 + v_3\}$
- 5  $v_5 = \max\{-1 + v_6, -1 + v_6, -1 + v_1, -1 + v_1\}$
- 6  $v_6 = \max\{-1 + v_2, -1 + v_5, -1 + v_5, -1 + v_2\}$

- 1  $\Rightarrow v_1 = -1$

&

- 2  $\Rightarrow v_2 = -2$

...

# Policy improvement

- The policy improvement theorem: suppose that  $\pi$  and  $\pi'$  are two deterministic policies such that

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

Then  $\pi'$  must be as good as, or better, than  $\pi$ , i.e.

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

Moreover, if the inequality in (C) is strict for at least one state, then the inequality in (R) must be strict for at least one state



# Proof of the policy improvement theorem

$$\begin{aligned} \bullet \quad & V^\pi(s) \leq Q^\pi(s, \pi'(s)) \\ &= \mathbf{E}_{\pi'}[r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s] \\ &\leq \mathbf{E}_{\pi'}[r_{t+1} + \gamma Q^\pi(s_{t+1}, \pi'(s_{t+1})) \mid s_t = s] \\ &= \mathbf{E}_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 V^\pi(s_{t+2}) \mid s_t = s] \\ &\vdots \\ &= \mathbf{E}_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \mid s_t = s] \\ &\leq V^{\pi'}(s) \end{aligned}$$

# Greedy policy improvement

- Greedy policy  $\pi'$  for any given policy  $\pi$  is given by

$$\pi'(s) = \operatorname{argmax}_a Q^\pi(s, a), \text{ for all } s \in S$$

- Greedy policy selects the best action in **one step lookahead**
- The greedy policy meets the conditions of the policy improvement theorem, thus it is as good as, or better than, the original policy

# Greedy policy improvement cont'd

- Note that:

$$\begin{aligned}\pi'(s) &= \operatorname{argmax}_a Q^\pi(s, a) \\ &= \operatorname{argmax}_a \mathbf{E}[r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s, a_t = a] \\ &= \operatorname{argmax}_a \sum_{s'} P_{s,s'}^a [R_{s,s'}^a + \gamma V^\pi(s')]\end{aligned}$$

- If  $\pi'$  is as good as but not better than  $\pi$ , then  $V^\pi = V^{\pi'} \Rightarrow V^{\pi'}$  satisfies the Bellman optimality equation  $\Rightarrow \pi$  and  $\pi'$  are optimal
- Policy improvement yields a strictly better policy, except when the original policy is already optimal

# Policy iteration

- **Policy iteration**: an iterative method that alternates between

(E) policy evaluation

(I) policy improvement

- $\pi_0 \rightarrow V^{\pi_0} \rightarrow \pi_1 \rightarrow V^{\pi_1} \rightarrow \dots \rightarrow \pi^* \rightarrow V^*$

- A finite MDP has a finite number of policies

$\Rightarrow$  the policy iteration method must converge in a finite number of steps

# Policy iteration: pseudo-code

**Initialization:**  $V(s) \in \mathbf{R}$ ,  $\pi(s) \in A(s)$  arbitrary for all  $s \in S$

**Repeat:**

$\Delta \leftarrow 0$

**For each**  $s \in S$ :

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_{s'} P_{s,s'}^{\pi(s)} [R_{s,s'}^{\pi(s)} + \gamma V(s')]$

$\Delta = \max\{\Delta, |v - V(s)|\}$

**until**  $\Delta < \theta$

$\text{polycystable} \leftarrow \text{True}$

**For each**  $s \in S$ :

$b \leftarrow \pi(s)$

$\pi(s) = \operatorname{argmax}_a \sum_{s'} P_{s,s'}^a [R_{s,s'}^a + \gamma V(s')]$

**If**  $b \neq \pi(s)$  **then**  $\text{polycystable} \leftarrow \text{False}$

**If**  $\text{polycystable}$ , **then** return, **else** go to policy evaluation

policy evaluation

policy improvement

# Value iteration

- Drawbacks of the policy iteration method:
  - Each iteration requires executing policy evaluation which requires multiple iterations (sweeps through the state space)
  - This can be computationally inefficient
- Value iteration idea: evaluate **only one step** in the policy evaluation

$$\begin{aligned} V_{k+1}(s) &= \max_a \mathbf{E}[r_{t+1} + \gamma V_k(s_{t+1}) | s_t = s, a_t = a] \\ &= \max_a \sum_{s'} P_{s,s'}^a [R_{s,s'}^a + \gamma V_k(s')] \end{aligned}$$

- The sequence  $V_0, V_1, \dots$  converges to  $V^*$  under the same conditions that guarantee the existence of  $V^*$

# Value iteration: pseudo-code

- Initialization:  $V(s) = 0$  for all  $s \in S^+$

- Repeat:

$\Delta \leftarrow 0$

For each  $s \in S$ :

$v \leftarrow V(s)$

$V(s) \leftarrow \max_a \sum_{s'} P_{s,s'}^a [R_{s,s'}^a + \gamma V(s')]$

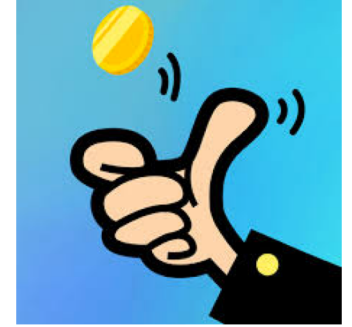
$\Delta = \max\{\Delta, |v - V(s)|\}$

until  $\Delta < \theta$

- **Output:** deterministic policy  $\pi$  given by

$$\pi(s) = \operatorname{argmax}_a \sum_{s'} P_{s,s'}^a [R_{s,s'}^a + \gamma V(s')] \text{ for all } s \in S$$

# Example: Gambler's problem



- A gambler makes bets on the outcomes of a sequence of coin flips
  - The gambler must decide for each coin flip what portion of his capital to stake
- **If outcome of the coin flip = heads then:**

The gambler **wins** as much money as he has staked on this flip
- **else**

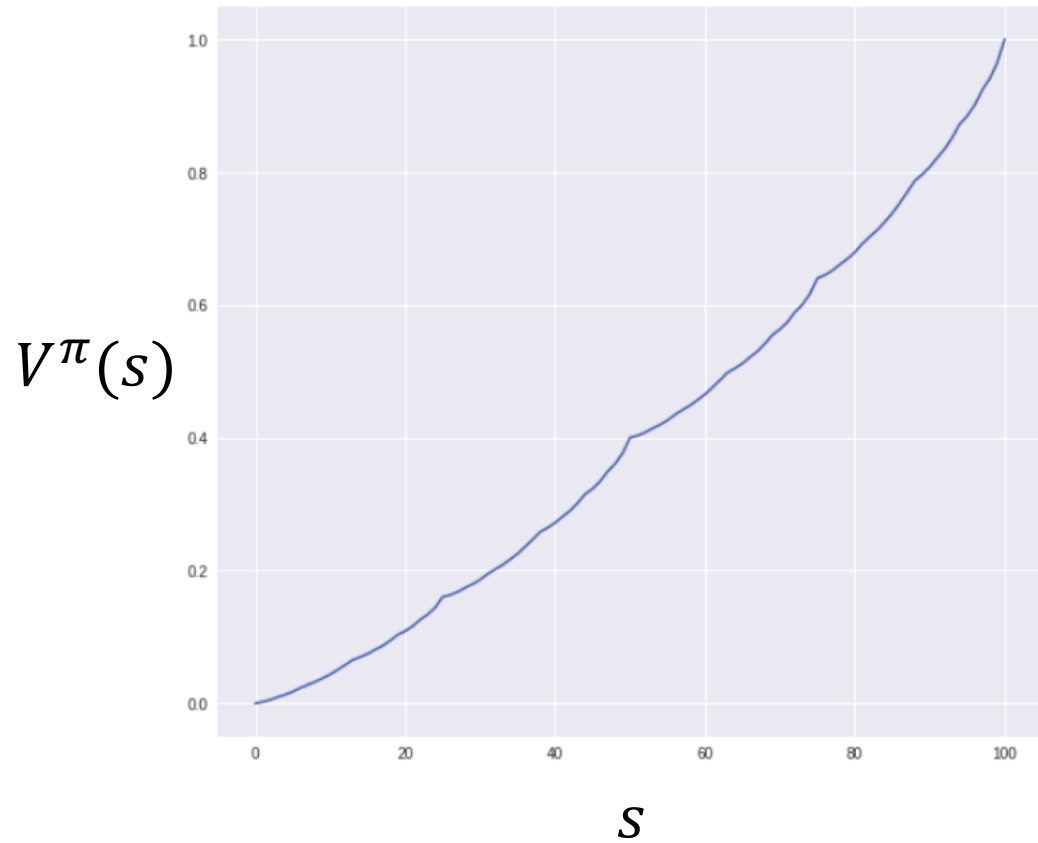
The gambler **loses** his stake
- The game ends when the gambler reaches his goal of **\$100** or **loses by running out of money**



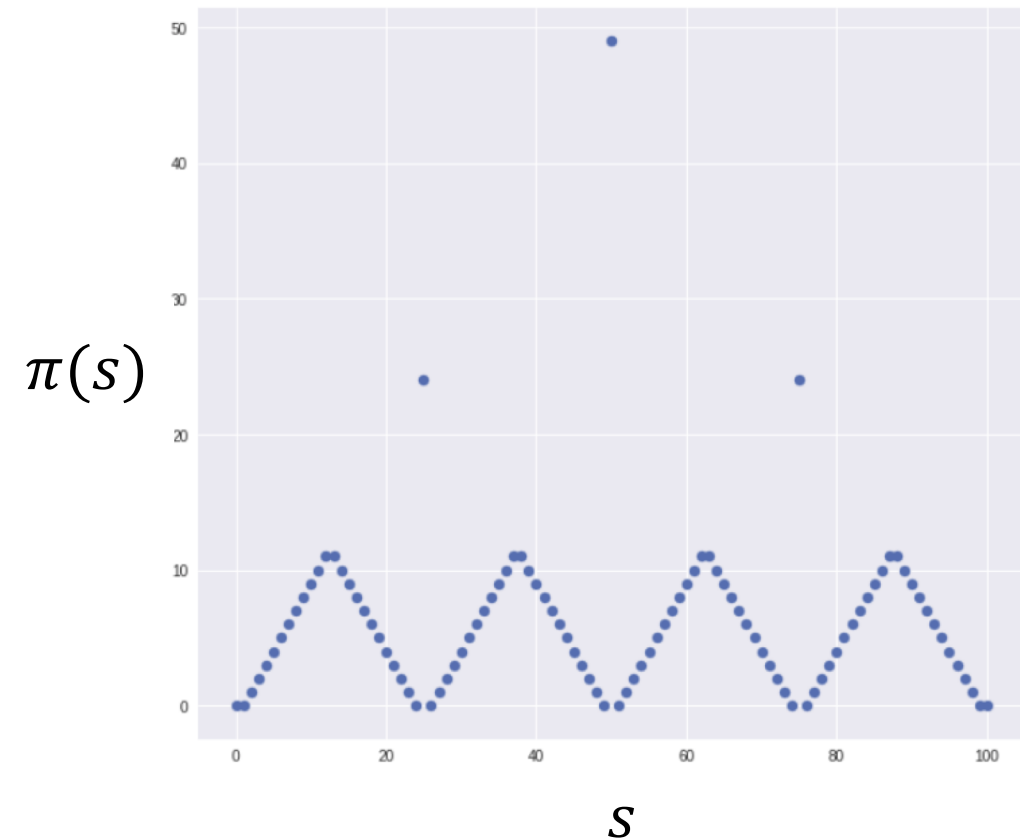
# Gambler's problem cont'd

- Formulated as an undiscounted, episodic, finite MDP problem
- State set:  $S = \{1, 2, \dots, 99\}$ ,  $S^+ = S \cup \{0, 100\}$
- Action sets:  $A(s) = \{1, 2, \dots, \min\{s, 100 - s\}\}$  for  $s \in S$
- $\Pr[\text{outcome of coin flip is heads}] = p$  (known parameter)
- Exercise (seminar session):
  - Show value function for different iterations
  - Show the optimal policy

# Gambler's problem cont'd



Optimal value function



Optimal policy

# Asynchronous DP

- Drawback of standard DP method:
  - Requires operations over the entire state set of the MDP
  - For large state sets this can be computationally expensive
- **Asynchronous DP**: backups up values in any order using whatever values of other states are available
  - Convergence guaranteed provided each state is visited with a positive rate
- Asynchronous DP algorithms referred to as **distributed DP algorithms**
  - Implementation in multiprocessor systems with communication delays between processors

# Monte Carlo methods

# Monte Carlo methods

- **Monte Carlo (MC) methods for reinforcement learning:** learning methods for solving the RL problem based on **averaging sample returns**
  - Estimating value functions and discovering optimal policies
  - Not assuming a perfect model of the environment
  - Based only on experience: sample sequences of states, actions and rewards from online or simulated interactions with an environment
- Defined for episodic tasks to ensure well-defined returns
- **Incremental methods in an episode-by-episode sense**
  - Estimates of value functions and policies are updated only upon the completion of an episode
  - Different from **step-by-step incremental methods** (next lecture)

# MC policy evaluation

- Suppose our goal is to estimate  $V^\pi(s)$ , the value of a state  $s$  for a given policy  $\pi$ , given a set of episodes obtained by following  $\pi$  and passing through state  $s$
- Types of visits to states:
  - A visit to  $s$ : each occurrence of state  $s$  in an episode
  - First visit to  $s$ : each first occurrence of  $s$  in an episode
- Types of MC methods:
  - The every-visit MC method:  $V^\pi(s)$  estimated by the average of the returns following **each visit** to  $s$  in a set of episodes
  - The first-visit MC method:  $V^\pi(s)$  estimated by the average of the returns following **each first visit** to  $s$  in an episode of a set of episodes
- The every-visit and first-visit MC methods are similar but have different theoretical properties
  - Q: Can you think of a fundamental difference between the two methods?

# The first-visit MC method: pseudo code

- **Initialization:**

$\pi \leftarrow$  policy to be evaluated

$V \leftarrow$  an arbitrary state-value function

$\text{Returns}(s) \leftarrow$  an empty list, for all  $s \in S$

- **Repeat:**

Generate an episode using policy  $\pi$

**For each** distinct  $s$  appearing in the episode:

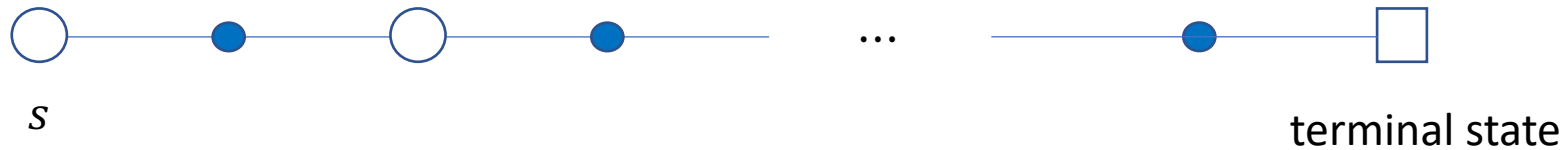
$R \leftarrow$  return following the first occurrence of  $s$

Append  $R$  to  $\text{Returns}(s)$

$V(s) \leftarrow \text{average}(\text{Returns}(s))$

# Backup diagram

- Backup diagram for MC estimation of  $V^\pi$  shows states **sampled** in one episode



- Unlike to the DP backup diagram that shows only **one-step transitions**



# Some pros and cons of MC methods

- Pros:

- Estimating the value of a single state is independent of the number of states
- One can generate many sample episodes starting from a given state to estimate the value of this state

- Cons:

- Incremental updates on an episode-by-episode basis which introduces delays
- Alternative methods allow for step-by-step incremental updates
  - Time-difference learning methods (discussed in next lecture)

# MC estimation of action values

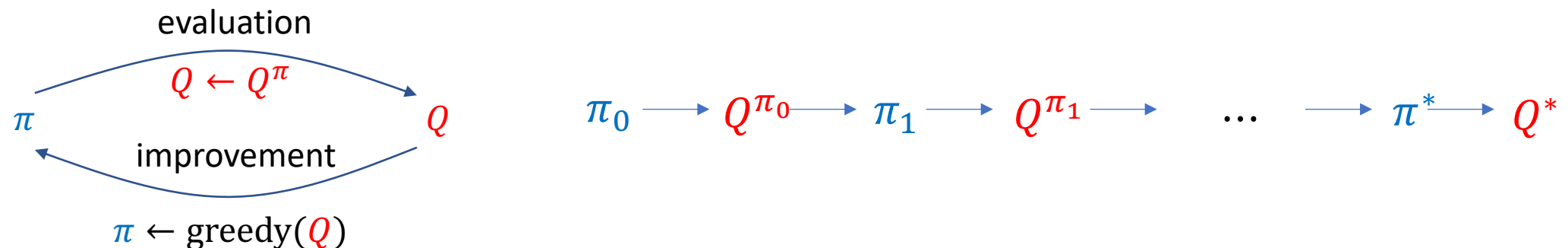
- Given a model, state values are sufficient to find a policy
  - Simply look one-step ahead and choose an action that leads to the best combination of reward and next state
- Without a model, state values are insufficient to find a policy
  - We need to explicitly estimate the value of each action to find a policy
- Primary goal of MC methods: estimate optimal action value function  $Q^*$
- The estimates can be defined analogously to estimating the state value function
  - E.g. the first-visit MC method

# MC estimation of action values (cont'd)

- **Issue**: many state-action pairs may never be visited under a policy
  - E.g. if  $\pi$  is a deterministic policy, only one action-state pair is observed for each distinct state
  - Need to maintain exploration !
- Two approaches to ensuring continual exploration:
  - **Exploring starts**: the first step of each episode starts at a state-action pair and every such pair has non-zero probability of being selected at the start
  - **Stochastic policies**: use policies that ensure a non-zero probability of selecting each action from the set of available actions in each given state

# MC control

- **MC control**: using MC estimation to approximate optimal policies
- Basic idea: use **generalized policy iteration**
  - Repeatedly alter the value function estimate to more closely approximate the value function of the current policy
  - Repeatedly improve the policy estimate wrt the current value function
- MC version of the standard policy iteration:



# MC control with exploring starts

- **Initialization:** for all  $s \in S, a \in A(s)$   
     $Q(s, a) \leftarrow \text{arbitrary}$   
     $\pi(s) \leftarrow \text{arbitrary}$   
     $\text{Returns}(s, a) \leftarrow \text{empty list}$
- **Repeat:**  
    Generate an episode using exploring starts and policy  $\pi$   
  
    **For each** pair  $(s, a)$  appearing in the episode:  
         $R \leftarrow \text{return following the first occurrence of } (s, a)$   
        Append  $R$  to  $\text{Returns}(s, a)$   
         $Q(s, a) \leftarrow \text{average}(\text{Returns}(s, a))$   
  
    **For each**  $s$  in the episode:  
         $\pi(s) \leftarrow \text{argmax}_a Q(s, a)$

# On-policy vs off-policy control methods

- **On-policy methods:** attempt to evaluate or improve the policy that is **used to make decisions**
- **Off-policy methods:** attempt to evaluate a policy by **observing episodes generated by using a different policy**

# Soft policies

- In on-policy control methods, policy  $\pi$  is usually a soft policy:

$$\pi(s, a) > 0 \text{ for all } s \in S \text{ and } a \in A(s)$$

- A policy  $\pi$  is said to be an  $\epsilon$ -soft policy if

$$\pi(s, a) \geq \frac{\epsilon}{|A(s)|} \text{ for all } s \in S \text{ and } a \in A(s)$$

- $\epsilon$ -greedy policy: chose an action with maximum estimated action value with probability  $1 - \epsilon$ , and otherwise select an action at random
  - Any non-greedy action is selected with probability  $\geq \epsilon/|A(s)|$
  - Greedy action is selected with probability  $1 - \epsilon + \epsilon/|A(s)|$

# An $\epsilon$ -soft on-policy MC control algorithm

- **Initialization:** for all  $s \in S, a \in A(s)$   
 $Q(s, a) \leftarrow$  arbitrary  
 $\pi(s) \leftarrow$  arbitrary  $\epsilon$ -soft policy  
 $\text{Returns}(s, a) \leftarrow$  empty list
- **Repeat:**  
Generate an episode using policy  $\pi$   
**For each** pair  $(s, a)$  appearing in the episode:  
     $R \leftarrow$  return following the first occurrence of  $(s, a)$   
    Append  $R$  to  $\text{Returns}(s, a)$   
     $Q(s, a) \leftarrow \text{average}(\text{Returns}(s, a))$   
  
**For each**  $s$  in the episode:  
     $a^* \leftarrow \operatorname{argmax}_a Q(s, a)$   
    **For each**  $a \in A(s)$ :  
        
$$\pi(s, a) \leftarrow \begin{cases} 1 - \epsilon + \epsilon \frac{1}{|A(s)|} & \text{if } a = a^* \\ \epsilon \frac{1}{|A(s)|} & \text{if } a \neq a^* \end{cases}$$



# Improvement guarantees

- Suppose  $\pi$  is any  $\epsilon$ -soft policy and  $\pi'$  is the  $\epsilon$ -greedy policy wrt  $Q^\pi$
- Fact:  $Q^\pi(s, \pi'(s)) \geq V^\pi(s)$  for all  $s \in S$
- By the policy improvement theorem:  $V^{\pi'}(s) \geq V^\pi(s)$  for all  $s \in S$

# Off-policy MC control

- The policy used to generate behavior (**behavior policy**) may be different to the policy used for evaluation and improvement (**estimation policy**)
- An advantage of off-policy MC methods: the estimation policy may be deterministic (e.g. greedy) while the behavior policy can continue to sample all auctions
- The key point: **evaluating one policy while following another**

# Evaluating one policy while following another

- Suppose episodes are generated by following policy  $\pi'$  while we want to estimate  $V^\pi$  or  $Q^\pi$  for a given policy  $\pi$  such that  $\pi \neq \pi'$
- Requirement:  $\pi(s, a) > 0 \Rightarrow \pi'(s, a) > 0$  for all  $s \in S, a \in A(s)$
- How can we construct an estimate of  $V^\pi(s)$  using returns, states, and actions observed under policy  $\pi'$ ?

# Estimating the value function

- $N$  := the number of episodes
- $T_i(s)$  := first time step at which state  $s$  is visited in episode  $i$ , otherwise infinity
- $T_i$  := the number of steps in episode  $i$
- $R_i(s)$  := reward from the first visit to state  $s$  in episode  $i$
- $N_s(N)$  := number of episodes with at least one visit of state  $s$

- **Estimator:**  $\hat{V}_N^{\pi'}(s) = \frac{1}{N_s(N)} \sum_{i=1}^N R_i(s) \mathbf{1}_{\{T_i(s) < T_i\}}$

- By the law of large numbers,

$$\lim_{N \rightarrow \infty} \hat{V}_N^{\pi'}(s) = \frac{\mathbf{E}[R_1(s) \mathbf{1}_{\{T_1(s) < T_1\}}]}{\mathbf{Pr}[T_1(s) < T_1]} = \mathbf{E}[R_1(s) \mid T_1(s) < T_1]$$

# Estimating the value function (cont'd)

- By the definition of the MDP environment:

$$\mathbf{E}[R_1(s) \mid T_1(s) < T_1] = \frac{\sum_{(s,a) \in \Pi_s} \left( \prod_{k=t_1(s)}^{t_1-1} \pi'(s_k, a_k) P_{s_k, s_{k+1}}^{a_k} \right) \left( \sum_{k=t_1(s)}^{t_1-1} R_{s_k, s_{k+1}}^{a_k} \right)}{\sum_{(s,a) \in \Pi_s} \left( \prod_{k=t_1(s)}^{t_1-1} \pi'(s_k, a_k) P_{s_k, s_{k+1}}^{a_k} \right)}$$

where  $\Pi_s$  contains  $\mathbf{s} = (s_{t_1(s)}, \dots, s_{t_1})$  and  $\mathbf{a} = (a_{t_1(s)}, \dots, a_{t_1-1})$  such that  $s_{t_1(s)} = s$

- To have an estimator of  $V^\pi(s)$  in the equation above we would like to have  $\pi(s_k, a_k)$  instead of  $\pi'(s_k, a_k)$

- Use the estimator:  $\hat{V}_N^\pi(s) = \frac{\sum_{i=1}^N \frac{p_i^\pi(s)}{p_i^{\pi'}(s)} R_i(s) \mathbf{1}_{\{T_i(s) < T_i\}}}{\sum_{i=1}^N \frac{p_i^\pi(s)}{p_i^{\pi'}(s)} \mathbf{1}_{\{T_i(s) < T_i\}}}$

where  $p_i^\varphi(s) := \prod_{k=T_i(s)}^{T_i-1} \varphi(s_k, a_k) P_{s_k, s_{k+1}}^{a_k}$  for a policy  $\varphi$

# Estimating the value function (cont'd)

- $$\lim_{N \rightarrow \infty} \hat{V}_N^\pi(s) = \frac{\mathbf{E} \left[ \frac{p_1^\pi(s)}{p_1^{\pi'}(s)} R_1(s) \mid T_1(s) < T_1 \right]}{\mathbf{E} \left[ \frac{p_1^\pi(s)}{p_1^{\pi'}(s)} \mid T_1(s) < T_1 \right]}$$

- Note that (as desired):

$$\begin{aligned} & \mathbf{E} \left[ \frac{p_1^\pi(s)}{p_1^{\pi'}(s)} R_1(s) \mid T_1(s) < T_1 \right] \\ &= \sum_{(s,a) \in \Pi_s} \left( \prod_{k=t_1(s)}^{t_1-1} \pi'(s_k, a_k) P_{s_k, s_{k+1}}^{a_k} \right) \frac{\left( \prod_{k=t_1(s)}^{t_1-1} \pi(s_k, a_k) P_{s_k, s_{k+1}}^{a_k} \right)}{\left( \prod_{k=t_1(s)}^{t_1-1} \pi'(s_k, a_k) P_{s_k, s_{k+1}}^{a_k} \right)} \left( \sum_{k=t_1(s)}^{t_1-1} R_{s_k, s_{k+1}}^{a_k} \right) \\ &= \sum_{(s,a) \in \Pi_s} \left( \prod_{k=t_1(s)}^{t_1-1} \pi(s_k, a_k) P_{s_k, s_{k+1}}^{a_k} \right) \left( \sum_{k=t_1(s)}^{t_1-1} R_{s_k, s_{k+1}}^{a_k} \right) \end{aligned}$$

# Estimating the value function (cont'd)

- Similarly, we have

$$\begin{aligned}
 & \mathbf{E} \left[ \frac{p_1^\pi(s)}{p_1^{\pi'}(s)} \mid T_1(s) < T_1 \right] \\
 &= \sum_{(s,a) \in \Pi_s} \left( \prod_{k=t_1(s)}^{t_1-1} \pi'(s_k, a_k) P_{s_k, s_{k+1}}^{a_k} \right) \frac{\left( \prod_{k=t_1(s)}^{t_1-1} \pi(s_k, a_k) P_{s_k, s_{k+1}}^{a_k} \right)}{\left( \prod_{k=t_1(s)}^{t_1-1} \pi'(s_k, a_k) P_{s_k, s_{k+1}}^{a_k} \right)} \\
 &= \sum_{(s,a) \in \Pi_s} \left( \prod_{k=t_1(s)}^{t_1-1} \pi(s_k, a_k) P_{s_k, s_{k+1}}^{a_k} \right)
 \end{aligned}$$

- Hence, we have (as desired):

$$\lim_{N \rightarrow \infty} \hat{V}_N^\pi(s) = \frac{\sum_{(s,a) \in \Pi_s} \left( \prod_{k=t_1(s)}^{t_1-1} \pi(s_k, a_k) P_{s_k, s_{k+1}}^{a_k} \right) \left( \sum_{k=t_1(s)}^{t_1-1} R_{s_k, s_{k+1}}^{a_k} \right)}{\sum_{(s,a) \in \Pi_s} \left( \prod_{k=t_1(s)}^{t_1-1} \pi(s_k, a_k) P_{s_k, s_{k+1}}^{a_k} \right)}$$

# No need to know the environment

- The important point for the estimator  $\hat{V}_N^\pi(s)$  is that it **does not require knowledge of the environment**
- The ratio of the weights depends only on the policies:

$$\frac{p_i^\pi(s)}{p_i^{\pi'}(s)} = \prod_{k=T_i(s)}^{T_i-1} \frac{\pi(s_k, a_k)}{\pi'(s_k, a_k)}$$

- The random reward  $R_i(s)$  is observable



# Alternative notation

- In the book, the estimator is defined equivalently but using different notation by considering only the episodes in which state  $s$  is visited (we only collect a sample of the reward in these episodes)
- The estimator  $\hat{V}_N^{\pi}(s)$  can be written as

$$V(s) = \frac{\sum_{j=1}^{n_s} \frac{p_j(s)}{p'_j(s)} R_j(s)}{\sum_{j=1}^{n_s} \frac{p_j(s)}{p'_j(s)}}$$

where  $n_s$  is the number of episodes in which state  $s$  is visited and  $p_j(s) = p_{i_s(j)}^{\pi}(s)$  and  $p'_j(s) = p_{i'_s(j)}^{\pi'}(s)$  where  $i_s(j)$  is the index of the episode in the sequence of all episodes for which state  $s$  occurred for the  $j$ -th time in an episode

# An off-policy MC control algorithm

- **Initialization:** for all  $s \in S, a \in A(s)$ :

$Q(s, a) \leftarrow$  arbitrary

$N(s, a) \leftarrow 0$  // numerator of  $Q(s, a)$

$D(s, a) \leftarrow 0$  // denominator of  $Q(s, a)$

$\pi \leftarrow$  an arbitrary deterministic policy

- **Repeat:**

Select a policy  $\pi'$  and use it to generate an episode:

$S_0, a_0, r_1, S_1, a_1, r_2, \dots, S_{T-1}, a_{T-1}, r_T, S_T$

$\tau \leftarrow$  latest time at which  $a_\tau \neq \pi(s_\tau)$

**For each** pair  $(s, a)$  appearing at the episode at  $\tau$  or later:

$t \leftarrow$  time of the first occurrence of  $(s, a)$  at  $t \geq \tau$

$w \leftarrow \prod_{k=t+1}^{T-1} \frac{1}{\pi'(s_k, a_k)}$

$N(s, a) \leftarrow N(s, a) + wR_t$

$D(s, a) \leftarrow D(s, a) + w$

$Q(s, a) \leftarrow \frac{N(s, a)}{D(s, a)}$

**For each**  $s \in S$ :

$\pi(s) = \operatorname{argmax}_a Q(s, a)$

# References

- R. S. Sutton and A. G. Barto, Reinforcement Learning, Chapters 4 and 5, 1998

# Seminar exercises

- Iterative policy evaluation: Gridworld problem
- Value iteration: Gambler's problem
- Monte Carlo prediction: Black Jack example
- Monte Carlo control: Black Jack example