

Reinforcement Learning

Lecture 2: Planning & Learning

Chengchun Shi

Lecture Outline

1. Preliminaries

2. Planning: Dynamic Programming

- 2.1 Policy Iteration
- 2.2 Value Iteration

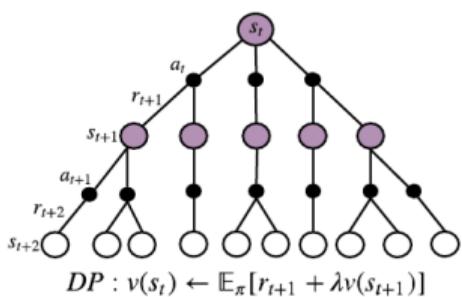
3. Learning: Monte Carlo Methods

- 3.1 MC Policy Evaluation (Prediction)
- 3.2 MC Policy Optimization (Control)

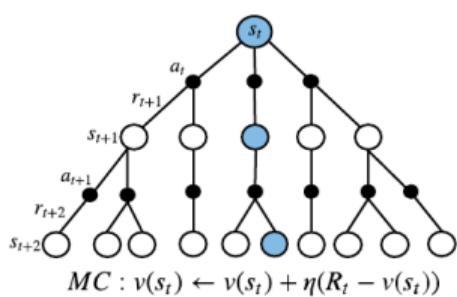
4. Learning: Temporal Difference Learning

- 4.1 TD Prediction
- 4.2 TD Control: SARSA

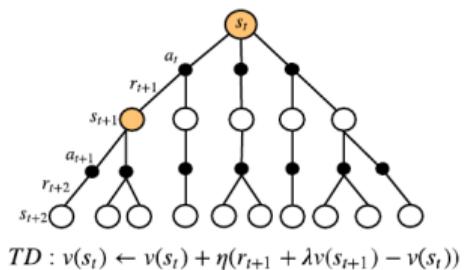
Lecture Outline (Cont'd)



Dynamic Programming (DP)



Monte Carlo (MC)



Temporal Difference (TD)

Lecture Outline

1. Preliminaries

2. Planning: Dynamic Programming

- 2.1 Policy Iteration
- 2.2 Value Iteration

3. Learning: Monte Carlo Methods

- 3.1 MC Policy Evaluation (Prediction)
- 3.2 MC Policy Optimization (Control)

4. Learning: Temporal Difference Learning

- 4.1 TD Prediction
- 4.2 TD Control: SARSA

Learning v.s. Planning

Two fundamental problems in sequential decision making

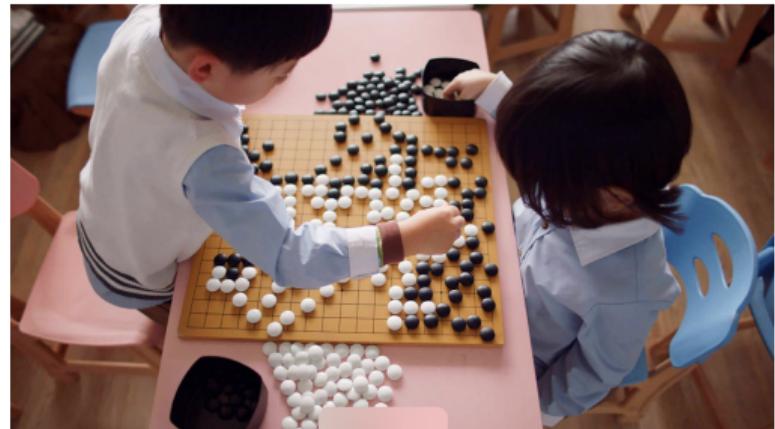
- **Planning**
 - A model of the environment (e.g., state transition, reward function) is **known**
 - The agent performs computations with its model, **without** any external interaction
 - Example: **Dynamic Programming**
- **Learning**
 - The environment is initially **unknown**
 - The agent **interacts** with the model
 - The agent **learns** the optimal policy from experience
 - Example: **Monte Carlo methods, temporal difference learning (Q-learning), policy-based learning, model-based learning**

Example: Go Game

- **Planning:** Rules of Go are known
- Exhaustive search of the optimal move
- No need to play Go with others



- **Learning:** No need to know the rules
- Learn the optimal move from experience
- Practice makes perfect



Models: Finite MDPs

- Environment modelled by a finite MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
- MDP model assumptions: **Markovianity & time-homogeneity**
- \mathcal{S} : state space (a **finite** set of states)
- \mathcal{A} : action space (a **finite** set of actions)
- \mathcal{P} : state transition probability matrix, $\mathcal{P}_{ss'}^a = \Pr(\mathcal{S}_{t+1} = s' | \mathcal{A}_t = a, \mathcal{S}_t = s)$
- \mathcal{R} : reward function, $\mathcal{R}_s^a = \mathbb{E}(R_t | \mathcal{A}_t = a, \mathcal{S}_t = s)$
- γ : discounted factor $\in [0, 1]$, allowed to be **1** if all sequences terminate (e.g., finite horizons)
- DP, MC and TD are **equally applicable** to settings with continuous state or action space

Bellman Equations

- Bellman equation for the (state) value function:

$$V^\pi(s) = \mathbb{E}^\pi[R_t + \gamma V^\pi(s_{t+1}) | S_t = s],$$

- or equivalently,

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left[R_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a V^\pi(s') \right].$$

- Bellman optimality equation for the **optimal** value function:

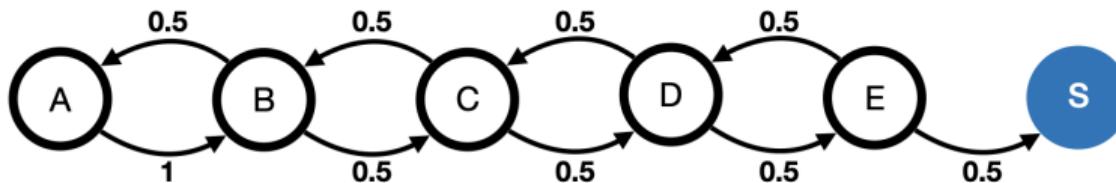
$$V^{\pi^{\text{opt}}}(s) = \max_a \mathbb{E}[R_t + \gamma V^{\pi^{\text{opt}}}(s_{t+1}) | A_t = a, S_t = s],$$

- or equivalently,

$$V^{\pi^{\text{opt}}}(s) = \max_{a \in \mathcal{A}} \left[R_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a V^{\pi^{\text{opt}}}(s') \right].$$

Bellman Equation: The Random Walk Example

- Consider a simple **random walk** on a path:



- Reward for transition to State S of value 1 , zero reward for other transitions
- Bellman equations:

$$V^\pi(A) = \mathbb{E}^\pi[R_t + \gamma V^\pi(S_{t+1}) | S_t = A] = \gamma V^\pi(B)$$

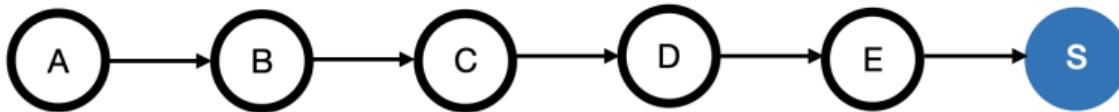
$$V^\pi(B) = \mathbb{E}^\pi[R_t + \gamma V^\pi(S_{t+1}) | S_t = B] = \frac{\gamma}{2} V^\pi(C) + \frac{\gamma}{2} V^\pi(A)$$

⋮

$$V^\pi(E) = \mathbb{E}^\pi[R_t + \gamma V^\pi(S_{t+1}) | S_t = E] = \frac{1}{2} + \frac{\gamma}{2} V^\pi(D)$$

Bellman Optimality Equation

- The **random walk** example:



- Reward for transition to State s of value 1, zero reward for other transitions
- Bellman optimality equations:

$$V^{\pi^{\text{opt}}}(A) = \max_a \mathbb{E}[R_t + \gamma V^{\pi^{\text{opt}}}(S_{t+1}) | A_t = a, S_t = A] = \gamma V^{\pi^{\text{opt}}}(B)$$

$$V^{\pi^{\text{opt}}}(B) = \max_a \mathbb{E}[R_t + \gamma V^{\pi^{\text{opt}}}(S_{t+1}) | A_t = a, S_t = B] = \gamma V^{\pi^{\text{opt}}}(C)$$

⋮

$$V^{\pi^{\text{opt}}}(E) = \max_a \mathbb{E}[R_t + \gamma V^{\pi^{\text{opt}}}(S_{t+1}) | A_t = a, S_t = E] = 1$$

State-Action Value Function

Definition

The state-action value function (better known as the **Q-function**) is expected return starting from s and a under π ,

$$Q^\pi(s, a) = \mathbb{E}^\pi(G_t | A_t = a, S_t = s) = \mathbb{E}^\pi \left(\sum_{i=0}^{+\infty} \gamma^i R_{i+t} | A_t = a, S_t = s \right).$$

- Q^π is **independent** of the time t in its definition, under **time-homogeneity**
- Q^π is the state value V^π under a Markov policy that implements a at the first time and follows π afterwards
- Reduces to action value function $\mathbb{E}^\pi(R_t | A_t = a)$ in Lecture 1 when $\gamma = 0$, $S = \emptyset$

State-Action Value Function (Cont'd)

Relationships between V^π and Q^π

- $Q^\pi \rightarrow V^\pi$:

$$V^\pi(s) = \mathbb{E}^\pi(G_t | S_t = s) = \sum_{a \in \mathcal{A}} \pi(a|s) \mathbb{E}^\pi(G_t | A_t = a, S_t = s) = \sum_{a \in \mathcal{A}} \pi(a|s) Q^\pi(s, a)$$

- $V^\pi \rightarrow Q^\pi$:

$$\begin{aligned} Q^\pi(s, a) &= \mathbb{E}(R_t | A_t = a, S_t = s) + \gamma \mathbb{E}(G_{t+1} | A_t = a, S_t = s) \\ &= \mathbb{E}(R_t | A_t = a, S_t = s) + \gamma \mathbb{E}[\mathbb{E}^\pi(G_{t+1} | S_{t+1}) | A_t = a, S_t = s] \\ &= \mathbb{E}[R_t + \gamma V^\pi(S_{t+1}) | A_t = a, S_t = s] \end{aligned}$$

1. Preliminaries

2. Planning: Dynamic Programming

- 2.1 Policy Iteration
- 2.2 Value Iteration

3. Learning: Monte Carlo Methods

- 3.1 MC Policy Evaluation (Prediction)
- 3.2 MC Policy Optimization (Control)

4. Learning: Temporal Difference Learning

- 4.1 TD Prediction
- 4.2 TD Control: SARSA

Dynamic Programming

Definition (Dynamic Programming)

A collection of algorithms used to compute optimal policies given **perfect** knowledge of the environment

- Dynamic programming (DP) is **rarely** used in practice (the environment is usually unknown)
- However, they provide a foundation for other solution methods
- “Dynamic programming” is used to solve many other statistical learning problems
 - Learning optimal **dynamic treatment regimes** (DTRs)
 - Multi-scale **change point detection**

Dynamic Programming Methods

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

- Policy Evaluation
- Policy Improvement

$$\pi_0 \longrightarrow V^{\pi_0} \longrightarrow \pi_1 \longrightarrow V^{\pi_1} \longrightarrow \dots \longrightarrow \pi^{opt} \longrightarrow V^{\pi^{opt}}$$

- **Value Iteration:** simultaneously combine **policy evaluation** and **policy improvement**

$$V^{\pi_0} \longrightarrow V^{\pi_1} \longrightarrow V^{\pi_2} \longrightarrow \dots \longrightarrow V^{\pi^{opt}} \longrightarrow \pi^{opt}$$

Lecture Outline

1. Preliminaries

2. Planning: Dynamic Programming

- 2.1 Policy Iteration
- 2.2 Value Iteration

3. Learning: Monte Carlo Methods

- 3.1 MC Policy Evaluation (Prediction)
- 3.2 MC Policy Optimization (Control)

4. Learning: Temporal Difference Learning

- 4.1 TD Prediction
- 4.2 TD Control: SARSA

Policy Iteration: Policy Evaluation

- Computation of the (state) value function \mathbf{V}^π for a given π
- According to the Bellman equation, for any s ,

$$\mathbf{V}^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left[\mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a \mathbf{V}^\pi(s') \right],$$

- written in matrix form, $\mathbf{V}^\pi = \mathcal{R} + \gamma \mathcal{P} \mathbf{V}^\pi$
- \mathbf{V}^π is a column vector with one entry per state

$$\begin{bmatrix} \mathbf{V}^\pi(1) \\ \vdots \\ \mathbf{V}^\pi(n) \end{bmatrix} = \sum_{a \in \mathcal{A}} \begin{bmatrix} \pi(a|1)\mathcal{R}_1^a \\ \vdots \\ \pi(a|n)\mathcal{R}_n^a \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11} & \cdots & \mathcal{P}_{1n} \\ \vdots & & \vdots \\ \mathcal{P}_{n1} & \cdots & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} \mathbf{V}^\pi(1) \\ \vdots \\ \mathbf{V}^\pi(n) \end{bmatrix},$$

where $\mathcal{P}_{ij} = \sum_{a \in \mathcal{A}} \pi(a|i) \mathcal{P}_{ij}^a$

Policy Evaluation (Cont'd)

- \mathbf{V}^π is a solution of a system of n linear equations with n unknowns
- It can be computed directly

$$\begin{aligned}\mathbf{V}^\pi &= \mathcal{R} + \gamma \mathcal{P} \mathbf{V}^\pi \\ (\mathbf{I} - \gamma \mathcal{P}) \mathbf{V}^\pi &= \mathcal{R} \\ \mathbf{V}^\pi &= (\mathbf{I} - \gamma \mathcal{P})^{-1} \mathcal{R}\end{aligned}$$

- $\mathbf{I} - \gamma \mathcal{P}$ is **invertible** when γ is strictly smaller than 1, since

$$\mathbf{x}^\top (\mathbf{I} - \gamma \mathcal{P}) \mathbf{x} = (1 - \gamma) \|\mathbf{x}\|_2^2 + \gamma \sum_{i,j} \mathcal{P}_{ij} (x_i - x_j)^2 > 0,$$

when $\mathbf{x} \neq \mathbf{0}$. The equality holds due to that each row of \mathcal{P} sums up to 1.

Policy Evaluation: Algorithm

- **Iterative Policy Evaluation:** an iterative method that outputs a sequence of value functions $V_0, V_1, V_2, \dots, V_k \rightarrow V^\pi$
- **Initial** value function V_0 is chosen arbitrarily subject to the **constraint** that at terminal state it has value **0**
- **Iterative** update rule (according to the Bellman equation):

$$V_{k+1} = \mathcal{R} + \gamma \mathcal{P} V_k$$

- **Convergence** is guaranteed when γ is strictly smaller than **1** (more in appendix), or eventual termination is guaranteed from all states under π

Policy Evaluation: Pseudocode

- **Input:** a policy π , a threshold parameter $\epsilon > 0$
- **Initialization:** $V(s) = 0$ for any $s \in \mathcal{S}$
- **Repeat:**

$$\Delta \leftarrow 0$$

For each $s \in \mathcal{S}$

$$\nu \leftarrow V(s)$$

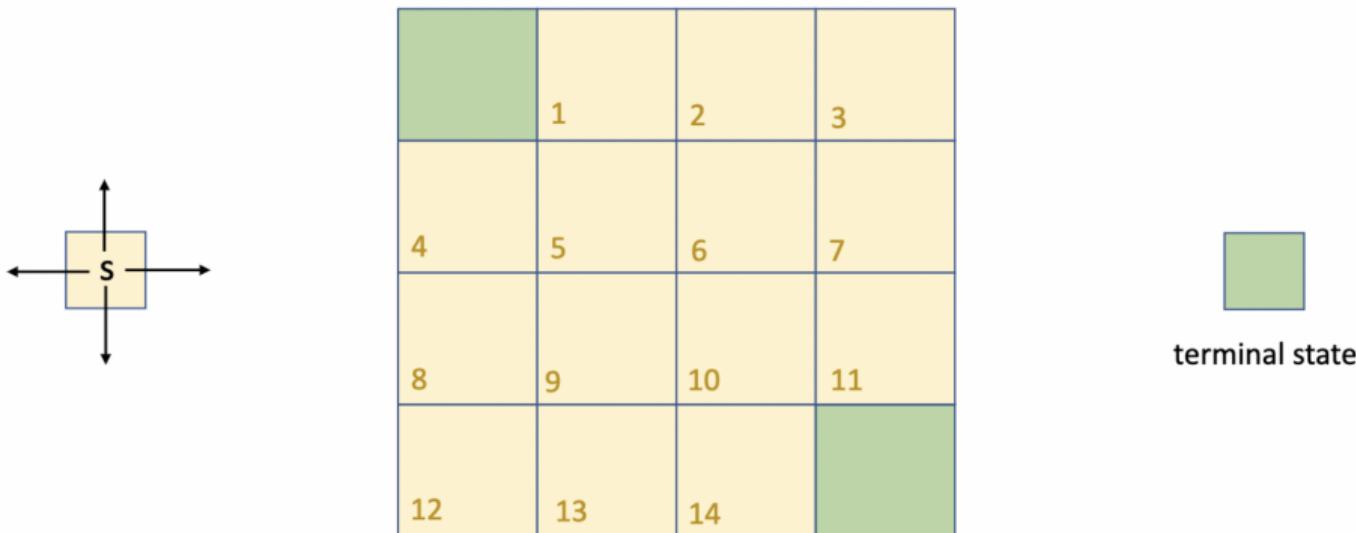
$$V(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s) \left[\mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a V(s') \right]$$

$$\Delta \leftarrow \max(\Delta, |\nu - V(s)|)$$

until $\Delta < \epsilon$

- **Output** V

GridWorld Example



- Undiscounted, episodic, finite MDP task
- $\mathcal{A} = \{\text{up, down, right, left}\}$. Actions leading out of the grid leave state unchanged
- Rewards: for each transition, the reward of value -1

GridWorld Example (Cont'd)

0	-14	-20	-22
-14	-18	-20	-20
-20	-20	-18	-14
-22	-20	-14	0

Figure: Values of uniform random policy

$$\pi(n|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = \pi(e|\cdot) = 0.25$$

GridWorld Example (Cont'd)

By symmetry and Bellman equation,

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

$$\begin{aligned}v_1 &= \frac{1}{4}(-1 + v_2) + \frac{1}{4}(-1 + v_5) + \frac{1}{4}(-1 + 0) + \frac{1}{4}(-1 + v_1) \\v_2 &= \frac{1}{4}(-1 + v_3) + \frac{1}{4}(-1 + v_6) + \frac{1}{4}(-1 + v_1) + \frac{1}{4}(-1 + v_2) \\v_3 &= \frac{1}{4}(-1 + v_3) + \frac{1}{4}(-1 + v_2) + \frac{1}{4}(-1 + v_2) + \frac{1}{4}(-1 + v_3) \\v_5 &= \frac{1}{4}(-1 + v_6) + \frac{1}{4}(-1 + v_6) + \frac{1}{4}(-1 + v_1) + \frac{1}{4}(-1 + v_1) \\v_6 &= \frac{1}{4}(-1 + v_2) + \frac{1}{4}(-1 + v_5) + \frac{1}{4}(-1 + v_5) + \frac{1}{4}(-1 + v_2)\end{aligned}$$

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

GridWorld Example (Cont'd)

$k = 0$	<table border="1"><tr><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr><tr><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr><tr><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr><tr><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr></table>	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	$k = 2$	<table border="1"><tr><td>0.0</td><td>-1.7</td><td>-2.0</td><td>-2.0</td></tr><tr><td>-1.7</td><td>-2.0</td><td>-2.0</td><td>-2.0</td></tr><tr><td>-2.0</td><td>-2.0</td><td>-2.0</td><td>-1.7</td></tr><tr><td>-2.0</td><td>-2.0</td><td>-1.7</td><td>0.0</td></tr></table>	0.0	-1.7	-2.0	-2.0	-1.7	-2.0	-2.0	-2.0	-2.0	-2.0	-2.0	-1.7	-2.0	-2.0	-1.7	0.0	$k = 10$	<table border="1"><tr><td>0.0</td><td>-6.1</td><td>-8.4</td><td>-9.0</td></tr><tr><td>-6.1</td><td>-7.7</td><td>-8.4</td><td>-8.4</td></tr><tr><td>-8.4</td><td>-8.4</td><td>-7.7</td><td>-6.1</td></tr><tr><td>-9.0</td><td>-8.4</td><td>-6.1</td><td>0.0</td></tr></table>	0.0	-6.1	-8.4	-9.0	-6.1	-7.7	-8.4	-8.4	-8.4	-8.4	-7.7	-6.1	-9.0	-8.4	-6.1	0.0
0.0	0.0	0.0	0.0																																																		
0.0	0.0	0.0	0.0																																																		
0.0	0.0	0.0	0.0																																																		
0.0	0.0	0.0	0.0																																																		
0.0	-1.7	-2.0	-2.0																																																		
-1.7	-2.0	-2.0	-2.0																																																		
-2.0	-2.0	-2.0	-1.7																																																		
-2.0	-2.0	-1.7	0.0																																																		
0.0	-6.1	-8.4	-9.0																																																		
-6.1	-7.7	-8.4	-8.4																																																		
-8.4	-8.4	-7.7	-6.1																																																		
-9.0	-8.4	-6.1	0.0																																																		
$k = 1$	<table border="1"><tr><td>0.0</td><td>-1.0</td><td>-1.0</td><td>-1.0</td></tr><tr><td>-1.0</td><td>-1.0</td><td>-1.0</td><td>-1.0</td></tr><tr><td>-1.0</td><td>-1.0</td><td>-1.0</td><td>-1.0</td></tr><tr><td>-1.0</td><td>-1.0</td><td>-1.0</td><td>0.0</td></tr></table>	0.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.0	$k = 3$	<table border="1"><tr><td>0.0</td><td>-2.4</td><td>-2.9</td><td>-3.0</td></tr><tr><td>-2.4</td><td>-2.9</td><td>-3.0</td><td>-2.9</td></tr><tr><td>-2.9</td><td>-3.0</td><td>-2.9</td><td>-2.4</td></tr><tr><td>-3.0</td><td>-2.9</td><td>-2.4</td><td>0.0</td></tr></table>	0.0	-2.4	-2.9	-3.0	-2.4	-2.9	-3.0	-2.9	-2.9	-3.0	-2.9	-2.4	-3.0	-2.9	-2.4	0.0	$k = \infty$	<table border="1"><tr><td>0.0</td><td>-14.</td><td>-20.</td><td>-22.</td></tr><tr><td>-14.</td><td>-18.</td><td>-20.</td><td>-20.</td></tr><tr><td>-20.</td><td>-20.</td><td>-18.</td><td>-14.</td></tr><tr><td>-22.</td><td>-20.</td><td>-14.</td><td>0.0</td></tr></table>	0.0	-14.	-20.	-22.	-14.	-18.	-20.	-20.	-20.	-20.	-18.	-14.	-22.	-20.	-14.	0.0
0.0	-1.0	-1.0	-1.0																																																		
-1.0	-1.0	-1.0	-1.0																																																		
-1.0	-1.0	-1.0	-1.0																																																		
-1.0	-1.0	-1.0	0.0																																																		
0.0	-2.4	-2.9	-3.0																																																		
-2.4	-2.9	-3.0	-2.9																																																		
-2.9	-3.0	-2.9	-2.4																																																		
-3.0	-2.9	-2.4	0.0																																																		
0.0	-14.	-20.	-22.																																																		
-14.	-18.	-20.	-20.																																																		
-20.	-20.	-18.	-14.																																																		
-22.	-20.	-14.	0.0																																																		

Figure: Value functions at each iteration

Policy Iteration: Policy Improvement

- Identify some π' that is no worse than π based on V^π
- For any s , consider a hybrid policy
 - implements a at the first time
 - follows π afterwards
- Its value is given by $Q^\pi(s, a)$ (can be computed based on V^π)
- Select π' among the class of hybrid policies that **maximizes** the value

$$\pi'(s) = \arg \max_a Q^\pi(s, a)$$

- Its value is given by $Q^\pi(s, \pi'(s)) \geq V^\pi(s)$, since the hybrid policy class contains π
- Surprisingly, according to **policy improvement theorem**, $V^{\pi'}(s) \geq V^\pi(s)$ for any s !

Policy Improvement (Cont'd)

Given a policy π , improve π by acting **greedily** with respect to V^π ,

$$\begin{aligned}\pi'(\mathbf{s}) &= \arg \max_{\mathbf{a}} Q^\pi(\mathbf{s}, \mathbf{a}) = \arg \max_{\mathbf{a}} \mathbb{E}[\mathcal{R}_t + \gamma V^\pi(\mathbf{S}_{t+1}) | A_t = \mathbf{a}, S_t = \mathbf{s}] \\ &= \arg \max_{\mathbf{a}} [\mathcal{R}_s^{\mathbf{a}} + \gamma \sum_{\mathbf{s}'} \mathcal{P}_{ss'}^{\mathbf{a}} V^\pi(\mathbf{s}')]\end{aligned}$$

Theorem

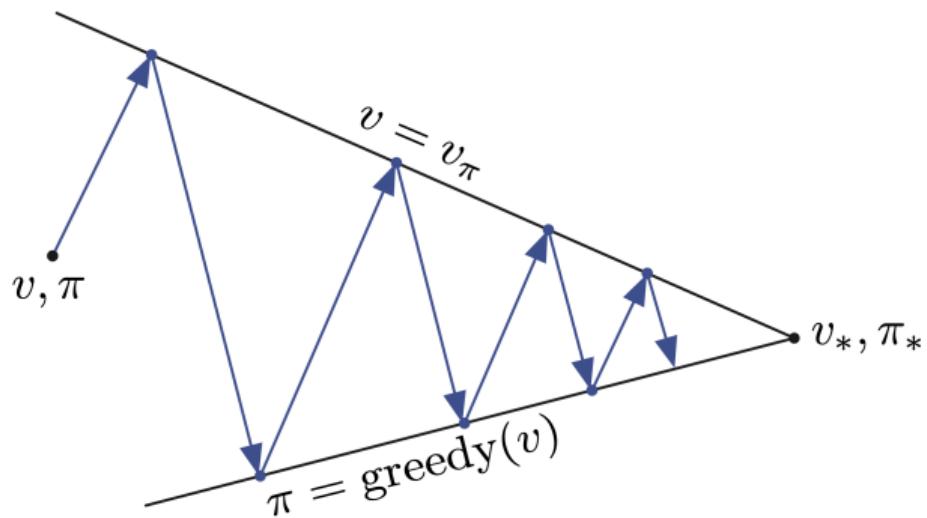
The greedy policy π' with respect to V^π is as good as or better than π ,

$$V^{\pi'}(\mathbf{s}) \geq V^\pi(\mathbf{s}),$$

for any $\mathbf{s} \in \mathcal{S}$.

Proof can be found in the Appendix.

Policy Iteration: Revisit



- **Policy Evaluation:** Compute \mathbf{V}^π via iterative policy evaluation
- **Policy Improvement:** Generate π' via greedy policy improvement

Policy Iteration: Pseudocode

- **Initialization:** $V(s) = 0$, $\pi(s) \in \mathcal{A}$ arbitrarily for any $s \in \mathcal{S}$
- **Repeat:**

$$\Delta \leftarrow 0$$

For each $s \in \mathcal{S}$

$$\nu \leftarrow V(s)$$

$$V(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s) \left[\mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a V(s') \right]$$

$$\Delta \leftarrow \max(\Delta, |\nu - V(s)|)$$

until $\Delta < \epsilon$

- **polystable** \leftarrow True

- For each $s \in \mathcal{S}$:

$$b \leftarrow \pi(s)$$

$$\pi(s) \leftarrow \arg \max_a [\mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a V(s')]$$

If $b \neq \pi(s)$ then **polystable** \leftarrow False

- If **polystable**, then Return π , else go to bullet point #2

Lecture Outline

1. Preliminaries

2. Planning: Dynamic Programming

- 2.1 Policy Iteration
- 2.2 Value Iteration

3. Learning: Monte Carlo Methods

- 3.1 MC Policy Evaluation (Prediction)
- 3.2 MC Policy Optimization (Control)

4. Learning: Temporal Difference Learning

- 4.1 TD Prediction
- 4.2 TD Control: SARSA

Value Iteration

- Policy iteration is **computationally inefficient**, as each iteration requires executing policy evaluation which requires multiple iterations
- According to the Bellman optimality equation,

$$V^{\pi^{\text{opt}}}(s) = \max_{a \in \mathcal{A}} \left[\mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a V^{\pi^{\text{opt}}}(s') \right].$$

- **Value iteration** idea: iteratively apply the above updates

$$V_{k+1}(s) = \max_{a \in \mathcal{A}} \left[\mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a V_k(s') \right].$$

- Drive the optimal deterministic policy

$$\pi^{\text{opt}}(s) = \arg \max_{a \in \mathcal{A}} \left[\mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a V^{\pi^{\text{opt}}}(s') \right].$$

- **Convergence** is guaranteed when γ is strictly smaller than 1 (more in Appendix), or eventual termination is guaranteed from all states.

Value Iteration: Pseudocode

- **Initialization:** $V(s) = 0$, $\pi(s) \in \mathcal{A}$ arbitrarily for any $s \in \mathcal{S}$
- **Repeat:**

$$\Delta \leftarrow 0$$

For each $s \in \mathcal{S}$

$$\nu \leftarrow V(s)$$

$$V(s) \leftarrow \max_{a \in \mathcal{A}} \left[\mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a V(s') \right]$$

$$\Delta \leftarrow \max(\Delta, |\nu - V(s)|)$$

until $\Delta < \epsilon$

- **Output:** optimal deterministic policy given by

$$\pi^{opt}(s) = \arg \max_{a \in \mathcal{A}} \left[\mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a V^{\pi^{opt}}(s') \right].$$

Example: Gambler's Problem

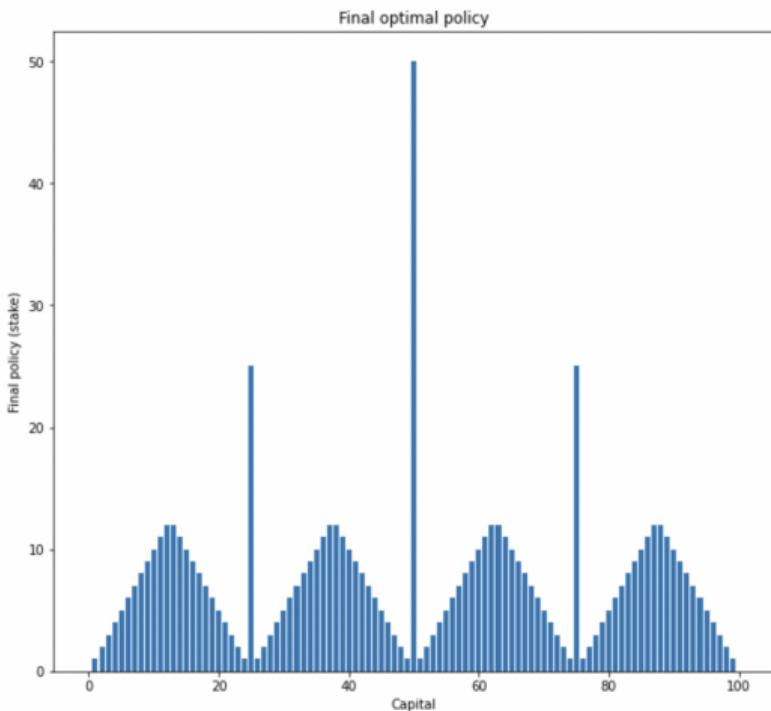


- A gambler makes bets on the outcomes of a sequence of coin flips
- The gambler must decide for each coin flip what proportion of capital to stake
- If **the outcome of the coin flip = heads**, then:
The gambler **wins** as much money as they have staked on this flip
- Else:
The gambler **loses** their stake
- The game ends when the gambler reaches the goal of **£100** or **runs out of money**

Example: Gambler's Problem (Cont'd)

- Undiscounted, episodic, finite MDP task
- \mathcal{S} : $\{0, 1, \dots, 99, 100\}$, termination states **0** and **100**
- $\mathcal{A}(s)$: $\{1, 2, \dots, \min(s, 100 - s)\}$, depends on the state
- $\Pr(\text{outcome of coin flip is heads}) = p$ (known parameter)

Example: Gambler's Problem, the Optimal Policy



Some Technical Questions

- How do we know that value iteration converges to $V^{\pi^{\text{opt}}}$?
- Or that iterative policy evaluation converges to V^π ?
- And therefore that policy iteration converges to $V^{\pi^{\text{opt}}}$?
- Is the solution unique?
- These questions are resolved by **Banach fixed-point theorem** (more in the appendix)

Lecture Outline

1. Preliminaries

2. Planning: Dynamic Programming

- 2.1 Policy Iteration
- 2.2 Value Iteration

3. Learning: Monte Carlo Methods

- 3.1 MC Policy Evaluation (Prediction)
- 3.2 MC Policy Optimization (Control)

4. Learning: Temporal Difference Learning

- 4.1 TD Prediction
- 4.2 TD Control: SARSA

MC Policy Evaluation

- Defined for **episodic** tasks
 - Value functions are updated upon completion of an episode
 - Different from **step-by-step** methods (e.g., temporal difference learning)
- **Objective:** estimate the value function V^π for a given policy π , from a set of episodes obtained by following π

$$S_0, A_0, R_0, \dots, S_T \sim \pi$$

- V^π is the expected return $\mathbb{E}^\pi(\sum_{0 \leq t \leq T} \gamma^t R_t | S_0 = s)$
- Monte Carlo idea: use **empirical mean** return to approximate **expected return**
- Types of MC methods:
 - **First-visit MC method:** $V^\pi(s)$ estimated by the average of returns following **each first visit** to s in a set of episodes
 - **Every-visit MC method:** $V^\pi(s)$ estimated by the average of returns following **each visit** to s in a set of episodes

First-Visit MC Policy Evaluation: Pseudocode

- **Initialization:**

N (counter), $N(s) \leftarrow 0$ for all $s \in \mathcal{S}$

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

- **Repeat:**

Generate an episode following policy π

For each distinct s appearing in the episode

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

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

$\text{Returns}(s) \leftarrow \text{Returns}(s) + G$

- **Output:**

For each distinct s

$N^{-1}(s)\text{Returns}(s)$

Every-Visit MC Policy Evaluation: Pseudocode

- **Initialization:**

$N \leftarrow$ counter, $N(s) \leftarrow 0$ for all $s \in \mathcal{S}$

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

- **Repeat:**

Generate an episode following policy π

For each s appearing in the episode

$G \leftarrow$ return following the occurrence of s

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

$\text{Returns}(s) \leftarrow \text{Returns}(s) + G$

- **Output:**

For each distinct s

$N^{-1}(s)\text{Returns}(s)$

Lecture Outline

1. Preliminaries

2. Planning: Dynamic Programming

- 2.1 Policy Iteration
- 2.2 Value Iteration

3. Learning: Monte Carlo Methods

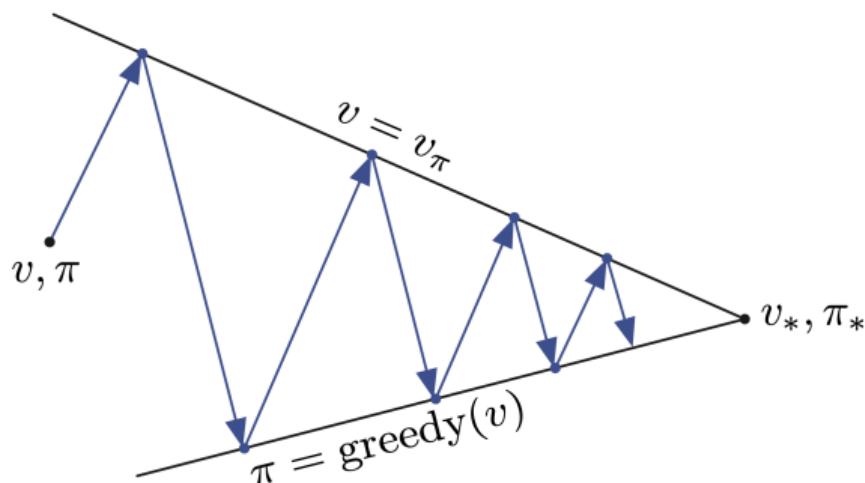
- 3.1 MC Policy Evaluation (Prediction)
- 3.2 MC Policy Optimization (Control)

4. Learning: Temporal Difference Learning

- 4.1 TD Prediction
- 4.2 TD Control: SARSA

MC Control with Generalized Policy Iteration

- **Objective:** use MC estimation to learn the optimal policy.
- Integrate policy iteration with MC methods



- **Policy Evaluation:** Compute \mathbf{V}^π via MC policy evaluation
- **Policy Improvement:** Generate π' via greedy policy improvement?

Policy Iteration Using State-Action Value Function

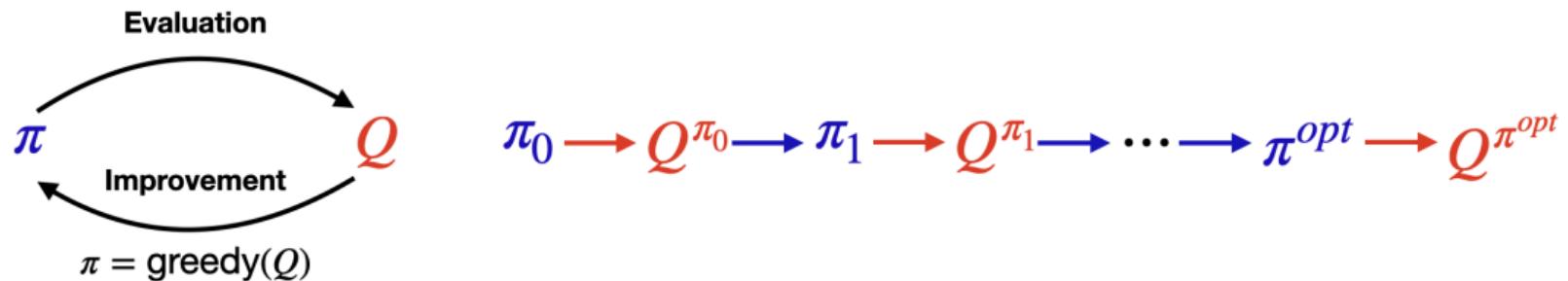
- Greedy policy improvement over V^π requires model of MDP

$$\pi'(s) = \arg \max_a [R_s^a + \gamma \sum_{s'} P_{ss'}^a V^\pi(s')]$$

- Greedy policy improvement over $Q^\pi(s, a)$ is model free

$$\pi'(s) = \arg \max_a Q^\pi(s, a)$$

MC Version of Policy Iteration



- **Policy Evaluation:** MC estimation of state-action value function
- **Policy Improvement:** Improve the policy wrt the current state-action value function

MC Estimation of State-Action Values

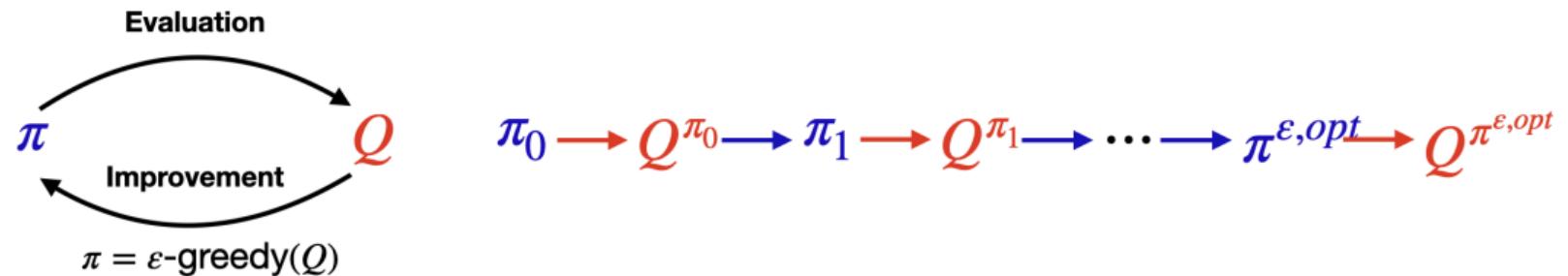
- Many state-action pairs may never be visited under a policy
 - Ex. if π is deterministic, only **one** state-action pair is observed for each distinct state
 - Need to ensure **exploration!**
- Two approaches for ensuring exploration:
 - **Exploring starts:** the first step of each episode starts at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

MC Control with ϵ -Greedy Exploration

- Simplest idea for ensuring continual exploration
- All m actions are tried with non-zero probabilities
- With probability $1 - \epsilon$ choose the **greedy** action
- With probability ϵ choose an action at **random**

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon, & \text{if } a = \arg \max_{a'} Q(s, a') \\ \epsilon/m, & \text{otherwise} \end{cases}$$

MC Control with ϵ -Greedy Exploration (Cont'd)



Pseudocode

- **Initialization:**

N (counter), $N(s, a) \leftarrow 0$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}$

Returns(s, a) \leftarrow empty lists, for all $s \in \mathcal{S}$, $a \in \mathcal{A}$

$\pi \leftarrow$ arbitrary ϵ -greedy policy

$Q \leftarrow$ arbitrary

- **Repeat:**

Generate an episode using exploring starts and policy π

For each distinct (s, a) appearing in the episode

$G \leftarrow$ return following the first occurrence of (s, a)

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

Returns(s, a) \leftarrow **Returns**(s, a) + G

$Q(s, a) \leftarrow \text{Returns}(s, a) / N(s, a)$

For each distinct s :

$\pi(a|s) \leftarrow \begin{cases} \epsilon/m + 1 - \epsilon, & \text{if } a = \arg \max Q(s, a) \\ \epsilon/m, & \text{otherwise} \end{cases}$

Lecture Outline

1. Preliminaries

2. Planning: Dynamic Programming

- 2.1 Policy Iteration
- 2.2 Value Iteration

3. Learning: Monte Carlo Methods

- 3.1 MC Policy Evaluation (Prediction)
- 3.2 MC Policy Optimization (Control)

4. Learning: Temporal Difference Learning

- 4.1 TD Prediction
- 4.2 TD Control: SARSA

TD Learning v.s. MC Methods v.s. DP Algorithms

(TD) Learning: a learning method that combines ideas from Monte Carlo (MC) methods and dynamic programming (DP)

Algorithms	DP	MC	TD
Planning	✓	✗	✗
Learning	✗	✓	✓
Model-free	✗	✓	✓
Step-by-step	✓	✗	✓
Episode-by-episode	✗	✓	✗
Episodic task	✓	✓	✓
Continuous task	✓	✗	✓

Lecture Outline

1. Preliminaries

2. Planning: Dynamic Programming

- 2.1 Policy Iteration
- 2.2 Value Iteration

3. Learning: Monte Carlo Methods

- 3.1 MC Policy Evaluation (Prediction)
- 3.2 MC Policy Optimization (Control)

4. Learning: Temporal Difference Learning

- 4.1 TD Prediction
- 4.2 TD Control: SARSA

MC Prediction

- **Objective:** learns \mathbf{V}^π from experience under π
- MC Policy Evaluation: $\mathbf{V}(\mathbf{s}) \leftarrow \text{average}[\text{Returns}(\mathbf{s})]$
- Incremental update for every-visit MC prediction:

$$\mathbf{V}(\mathbf{S}_t) \leftarrow \mathbf{V}(\mathbf{S}_t) + \alpha_t [\mathbf{G}_t - \mathbf{V}(\mathbf{S}_t)]$$

where α_t is $\frac{1}{\#[\text{Returns}(\mathbf{S}_t)]}$ at time t

- We may regard \mathbf{G}_t as a **target**
- The update can be performed after return \mathbf{G}_t is observed
- i.e. after the episode is completed

TD Prediction

- Unlike MC methods, TD methods wait only until **next time step**
- The simplest TD method (so called TD(0)) considers the update

$$V(S_t) \leftarrow V(S_t) + \alpha_t [R_t + \gamma V(S_{t+1}) - V(S_t)]$$

- This update rule has $R_t + \gamma V(S_{t+1})$ as the **target**
- Considered as a **bootstrap** method: update in part based on an existing estimate
- Different from “bootstrap” in statistics: a resampling method (e.g., sample with replacement) for **uncertainty quantification** of a given estimate

MC vs TD update

- Notice that under the MDP assumption

$$V^\pi(s) = \mathbb{E}^\pi(G_t | S_t = s) \quad (1)$$

$$\begin{aligned} &= \mathbb{E}^\pi\left(\sum_{k=0}^{\infty} \gamma^k R_{t+k} | S_t = s\right) \\ &= \mathbb{E}^\pi[R_t + \gamma V^\pi(S_{t+1}) | S_t = s] \end{aligned} \quad (2)$$

- MC methods use as the target the random variable in (1)
- TD methods use as the target the random variable in (2)
 - **Immediate reward** and estimate of the **future value**

TD: Pseudocode

- **Input:** π policy to be evaluated, step size α
- **Initialization:** V arbitrary
- **Repeat** for each episode:

Initialize state s

Repeat for each step of the episode:

$a \leftarrow$ action given by π for s

Take action a , observe reward r and next state s'

$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$

$s \leftarrow s'$

until s is a terminal state

Pros & Cons of MC vs TD

- MC must wait until the **end** of episode
- MC learns from **complete** sequences
- MC only works for **episodic** (terminating) environments
- TD can learn online after **each step**
- TD can learn from **incomplete** sequences
- TD works in **continuing** environments

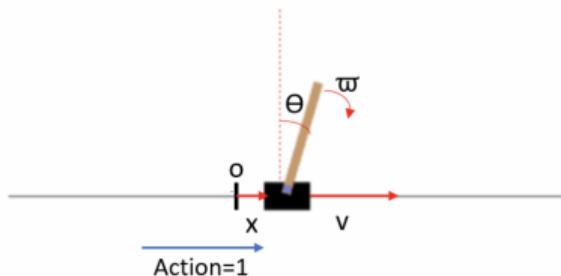
Pros & Cons of MC vs TD (Cont'd)

- Bias/Variance Trade-Off
- Return G_t is **unbiased** estimate of $V^\pi(S_t)$
- Oracle target $R_t + \gamma V^\pi(S_{t+1})$ is **unbiased** estimate of $V^\pi(S_t)$
- TD target $R_t + \gamma V(S_{t+1})$ is **biased** estimate of $V^\pi(S_t)$
- TD target has much lower **variance** than the return
 - Return depends on **many** random actions, transitions, rewards
 - TD target depends on **one** random action, transition, reward
- MC has **high variance, zero bias, insensitive to initialization**
- TD has **low variance, some bias, sensitive to initialization**

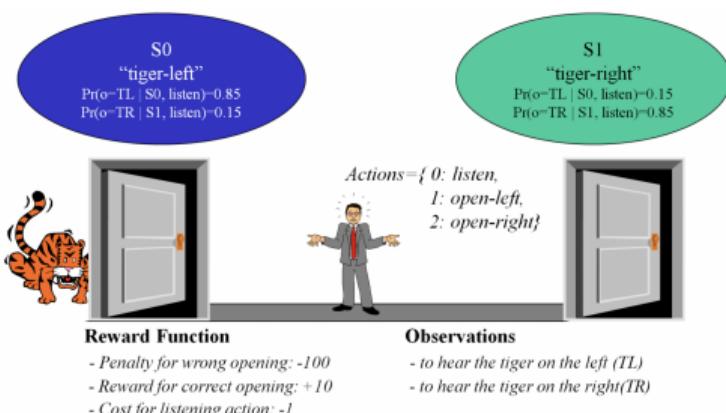
Pros & Cons of MC vs TD (Cont'd)

- TD exploits **Markov** & **stationary** properties
- Relies on the **Bellman equation**
- More **efficient** in MDP environments

frame: 53, Obs: (0.018, 0.669, 0.286, 0.618)
Action: 1.0, Cumulative Reward: 47.0, Done: 1



- MC **does** not exploit these properties
- More **flexible** in non-MDP environments (e.g., POMDP)



Rate of Convergence

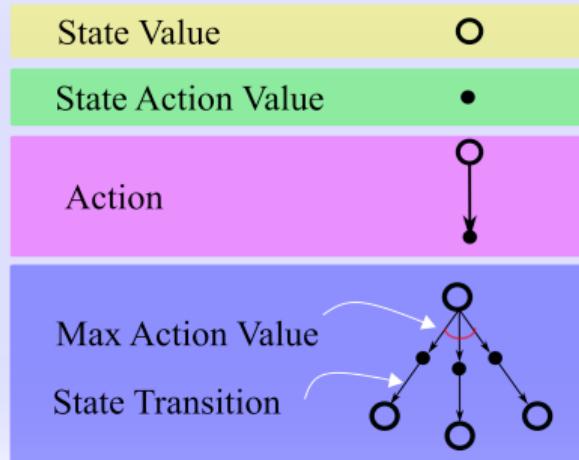
- For i.i.d. random variables X_1, \dots, X_n with mean μ and variance σ^2 ,

$$\sqrt{n}(\bar{X} - \mu) \rightarrow N(0, \sigma^2),$$

according to CLT.

- \bar{X} converges to μ at a rate of $n^{-1/2}$.
- For n episodes with T time points per episode, first-visit MC converges at a rate of $n^{-1/2}$.
- For n episodes with T time points per episode, TD converges at a rate of $(nT)^{-1/2}$, with proper choice of step sizes.
- First-visit MC requires $n \rightarrow \infty$ to be consistent
- TD requires either n or $T \rightarrow \infty$ to be consistent

Backup Diagram



Taken from <https://towardsdatascience.com/all-about-backup-diagram-fefb25aad804>

Backup Diagram (Cont'd)



TD



More states
and actions



Terminal state

MC

1. Preliminaries

2. Planning: Dynamic Programming

- 2.1 Policy Iteration
- 2.2 Value Iteration

3. Learning: Monte Carlo Methods

- 3.1 MC Policy Evaluation (Prediction)
- 3.2 MC Policy Optimization (Control)

4. Learning: Temporal Difference Learning

- 4.1 TD Prediction
- 4.2 TD Control: SARSA

SARSA: an On-Policy TD Control

- **SARSA**: a TD method for **policy optimisation**
 - Follows the pattern of policy iteration
 - Uses TD prediction method for **policy evaluation**
 - Uses ϵ -greedy exploration for **policy improvement**
- Similar to MC control, estimate state-action value $Q^\pi(s, a)$ (instead of the state value $V^\pi(s)$) for the control problem
- Different from MC control, update the state-value **every time step**

Bellman Equations

- Bellman equation for the (state) value function:

$$V^\pi(s) = \mathbb{E}[R_t + \gamma V^\pi(S_{t+1}) | S_t = s].$$

- Bellman equation for the state-action value function:

$$Q^\pi(s, a) = \mathbb{E} \left[R_t + \gamma \sum_{a'} \pi(a' | S_{t+1}) Q^\pi(S_{t+1}, a') | A_t = a, S_t = s \right],$$

or equivalently,

$$Q^\pi(s, a) = \mathbb{E}^\pi[R_t + \gamma Q^\pi(S_{t+1}, A_{t+1}) | A_t = a, S_t = s].$$

SARSA

- Incremental estimation of the state-action value function:

$$Q(\mathbf{S}_t, \mathbf{A}_t) \leftarrow Q(\mathbf{S}_t, \mathbf{A}_t) + \alpha [R_t + \gamma Q(\mathbf{S}_{t+1}, \mathbf{A}_{t+1}) - Q(\mathbf{S}_t, \mathbf{A}_t)],$$

for non-terminal state \mathbf{S}_{t+1}

- If \mathbf{S}_{t+1} is a terminal state, $Q(\mathbf{S}_{t+1}, \mathbf{A}_{t+1}) = 0$
- This update uses every element of the quintuple of variables:

$$(\mathbf{S}_t, \mathbf{A}_t, R_t, \mathbf{S}_{t+1}, \mathbf{A}_{t+1})$$

$S \ A \ R \ S \ A$

SARSA: Pseudocode

- **Initialization:** Q arbitrary
- **Repeat** for each episode:

Initialize state s

Choose action a from s using policy derived from Q (ε -greedy)

Repeat for each step of the episode:

 Take action a , observe reward r and next state s'

$a' \leftarrow$ action from s' using policy derived from Q (ε -greedy)

$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$

$s \leftarrow s'$, $a \leftarrow a'$

until s is a terminal state

Convergence of SARSA

Theorem

SARSA converges to the optimal Q -function, $\mathbf{Q}(\mathbf{s}, \mathbf{a}) \rightarrow \mathbf{Q}^{\pi^{\text{opt}}}(\mathbf{s}, \mathbf{a})$ for any \mathbf{s} and \mathbf{a} , if

- All state-action pairs are explored infinitely many times,

$$\sum_{t=0}^{\infty} \mathbb{I}(\mathbf{S}_t = \mathbf{s}, \mathbf{A}_t = \mathbf{a}) = \infty.$$

- The policy converges to a greedy policy,

$$\lim_{t \rightarrow \infty} \pi_t(\mathbf{a}|\mathbf{s}) = \mathbb{I}(\mathbf{a} = \arg \max_{\mathbf{a}'} \mathbf{Q}_t(\mathbf{s}, \mathbf{a}'))$$

- Robbins-Monro sequence of step-sizes (Robbins and Monro, 1951),

$$\sum_{t=0}^{\infty} \alpha_t = \infty \text{ and } \sum_{t=0}^{\infty} \alpha_t^2 < \infty$$

Convergence of SARSA (Cont'd)

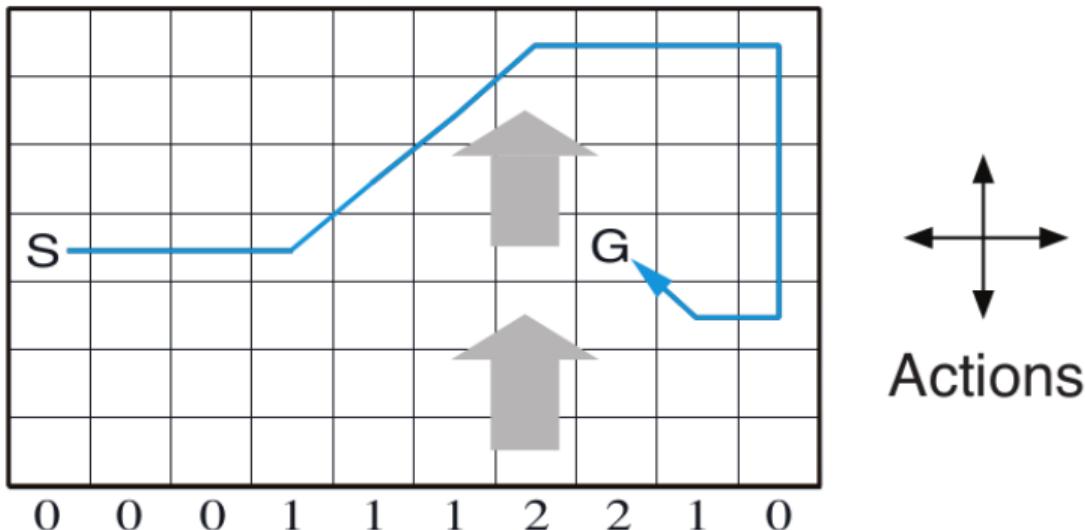
- Condition 1: All state-action pairs are **explored** infinitely many times
 $\Rightarrow \varepsilon$ to be strictly positive
- Condition 2: The policy converges to a **greedy** policy
 $\Rightarrow \varepsilon_t$ decays to zero as t grows to infinity
- Condition 3: Robbins-Monro sequence of step-sizes

$$\sum_{t=0}^{\infty} \alpha_t = \infty \text{ and } \sum_{t=0}^{\infty} \alpha_t^2 < \infty$$

$\Rightarrow \alpha_t$ proportional to t^{-c} for $1/2 < c \leq 1$.

$\sum_t t^{-c} = \infty$ when $c \leq 1$ and $\sum_t t^{-c} < \infty$ when $c > 1$

Windy Gridworld Example



- An **episodic** task
- Rewards of **-1** until **goal** is reached
- Strength of wind indicated by numbers

Windy Gridworld Example (Cont'd)

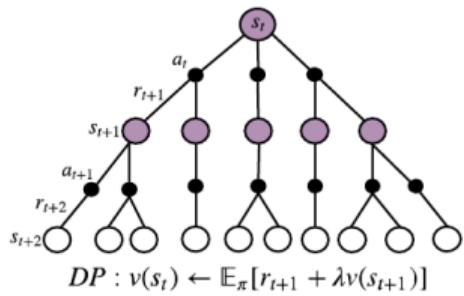
Optimal policy is:

```
[ 'R' 'D' 'R' 'R' 'R' 'R' 'R' 'R' 'R' 'R' 'D'  
[ 'R' 'R' 'U' 'L' 'R' 'R' 'R' 'U' 'R' 'D'  
[ 'L' 'U' 'R' 'R' 'R' 'R' 'R' 'R' 'R' 'D'  
[ 'R' 'R' 'R' 'R' 'R' 'R' 'U' 'G' 'R' 'D'  
[ 'R' 'D' 'D' 'R' 'R' 'R' 'U' 'D' 'L' 'L'  
[ 'D' 'D' 'D' 'R' 'R' 'U' 'U' 'D' 'R' 'U'  
[ 'R' 'R' 'U' 'R' 'U' 'U' 'U' 'U' 'R' 'L'  
[ '0' '0' '0' '1' '1' '1' '2' '2' '1' '0' ]
```

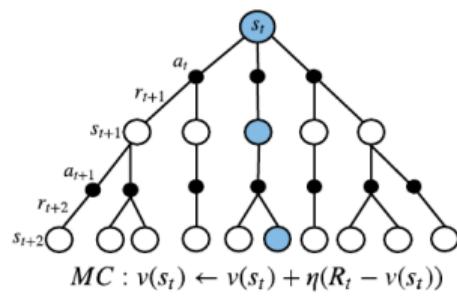
Summary

- Planning v.s. Learning
- Dynamic programming v.s. Monte Carlo v.s. Temporal Difference
- Policy Iteration v.s. Value Iteration
- Policy Evaluation v.s. Policy Improvement
- MC Prediction v.s. MC Control
- TD Prediction v.s. SARSA
- γ -Contraction, Banach Fixed Point Theorem

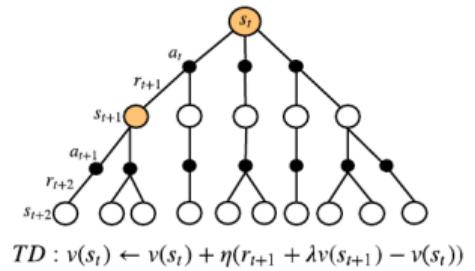
Summary (Cont'd)



Dynamic Programming (DP)



Monte Carlo (MC)



Temporal Difference (TD)

Appendix: Proof of Policy Improvement Theorem

Consider a sequence of policies:

- π_0 : a given stationary policy π
- π_k : a Markov policy that implements π' at the first k times and follows π afterwards
- π_∞ : the greedy policy π'

We show in the appendix

- **Step 1:** π_1 is no worse than π_0 , i.e., $Q^\pi(s, \pi'(s)) \geq V^\pi(s)$
- **Step 2:** π_{k+1} is no worse than π_k for any $k \geq 1$

This proves the policy improvement theorem

Appendix: Policy Improvement Theorem, Step 1

- π_0 : a given stationary policy π
- π_1 : a Markov policy that implements π' at the initial time and follows π afterwards
- By definition,

$$\pi'(\mathbf{s}) = \arg \max_{\mathbf{a}} Q^\pi(\mathbf{s}, \mathbf{a})$$

- This yields

$$Q^\pi(\mathbf{s}, \pi'(\mathbf{s})) = \max_{\mathbf{a}} Q^\pi(\mathbf{s}, \mathbf{a}) \geq \sum_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s}) Q^\pi(\mathbf{s}, \mathbf{a}) = V^\pi(\mathbf{s})$$

- i.e., π_1 is no worse than π_0

Appendix: Policy Improvement Theorem, Step 2

- π_k : a Markov policy that implements π' at the first k times and follows π afterwards
- The difference between two value functions is given by

$$V^{\pi_{k+1}}(s) - V^{\pi_k}(s) = \gamma^k \mathbb{E}^{\pi'}[Q^\pi(S_k, \pi'(S_k)) | S_0 = s] - \gamma^k \mathbb{E}^{\pi'}[V^\pi(S_k) | S_0 = s]$$

- Results in Step 1 yield $Q^\pi(S_k, \pi'(S_k)) \geq V^\pi(S_k)$, and hence $V^{\pi_{k+1}}(s) \geq V^{\pi_k}(s)$
- i.e., π_{k+1} is no worse than π_k

Appendix: Bellman Expectation Operator

Definition

Define the Bellman Expectation Operator T^π as a function that maps a given value function V into another value function $T^\pi V$ such that

$$T^\pi V(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left[\mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a V(s') \right], \quad \forall s \in \mathcal{S}.$$

- The Bellman equation can be rewritten as $V^\pi = T^\pi V^\pi$
- This operator is a **γ -contraction**, i.e. it makes value function closer by at least γ

$$\begin{aligned} \max_s |T^\pi V_1(s) - T^\pi V_2(s)| &= \gamma \max_s \left| \sum_{a,s'} \pi(a|s) \mathcal{P}_{ss'}^a [V_1(s') - V_2(s')] \right| \\ &\leq \gamma \max_s |V_1(s) - V_2(s)| \max_s \left| \sum_{a,s'} \pi(a|s) \mathcal{P}_{ss'}^a \right| = \gamma \max_s |V_1(s) - V_2(s)| \end{aligned}$$

- Iterative Policy Evaluation: $V_0 \rightarrow T^\pi V_0 \rightarrow T^\pi T^\pi V_0 \rightarrow \dots$

Appendix: Banach Fix Point Theorem

Theorem

Suppose T is a γ -contraction. Then under certain conditions,

- T admits a **unique** fix point V^* , i.e. $TV^* = V^*$;
- V^* can be found as follows: define a sequence $\{V_k\}_k$ such that $V_{k+1} = TV_k$. Then $V^* = \lim_k V_k$

- Proof can be found [here](#)
- T^π has a unique fix point
- V^π is the fix point, according to the Bellman equation
- Iterative policy evaluation converges to V^π

Appendix: Bellman Optimality Operator

Definition

Define the Bellman Expectation Operator \mathbf{T} as a function that maps a given value function \mathbf{V} into another value function \mathbf{TV} such that

$$\mathbf{TV}(\mathbf{s}) = \max_{\mathbf{a} \in \mathcal{A}} \left[\mathcal{R}_s^{\mathbf{a}} + \gamma \sum_{\mathbf{s}'} \mathcal{P}_{ss'}^{\mathbf{a}} \mathbf{V}(\mathbf{s}') \right], \quad \forall \mathbf{s} \in \mathcal{S}.$$

- The Bellman optimality equation can be rewritten as $\mathbf{V}^{\pi^{\text{opt}}} = \mathbf{TV}^{\pi^{\text{opt}}}$
- This operator is a **γ -contraction** as well

$$\begin{aligned} \max_{\mathbf{s}} |\mathbf{TV}_1(\mathbf{s}) - \mathbf{TV}_2(\mathbf{s})| &= \gamma \max_{\mathbf{s}, \mathbf{a}} \left| \sum_{\mathbf{s}'} \mathcal{P}_{ss'}^{\mathbf{a}} [\mathbf{V}_1(\mathbf{s}') - \mathbf{V}_2(\mathbf{s}')] \right| \\ &\leq \gamma \max_{\mathbf{s}'} |\mathbf{V}_1(\mathbf{s}') - \mathbf{V}_2(\mathbf{s}')| \end{aligned}$$

Appendix: Convergence of Dynamic Programming

- T has a unique fix point
- $V^{\pi^{\text{opt}}}$ is the fix point, according to the Bellman optimality equation
- According to the Banach fix point theorem, **value iteration** converges to $V^{\pi^{\text{opt}}}$
- **Policy iteration** (that integrates iterative policy evaluation & policy improvement) converges to π^{opt}

Questions