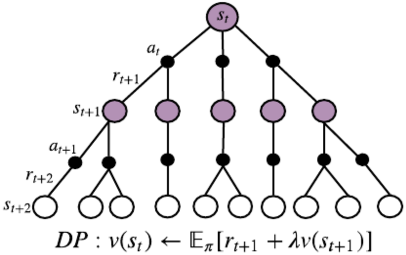
ST455: Reinforcement Learning

Lecture 3: Elementary Solution Methods
Dynamic Programming and Monte Carlo

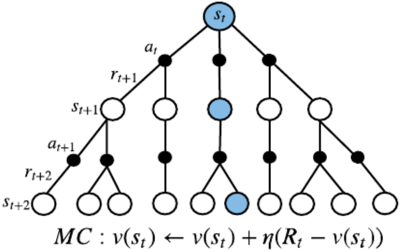
Chengchun Shi

Lecture Outline

Lecture Outline (Cont'd)



Dynamic Programming (DP)



Monte Carlo (MC)

Learning v.s. Planning

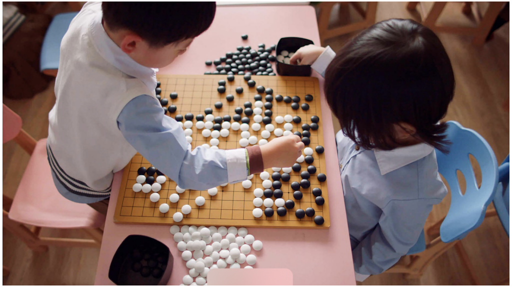
Two fundamental problems in sequential decision making

- **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
  - a.k.a. deliberation, reasoning, introspection, pondering, thought, search
  - 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**, **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 assumption: **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, $\mathcal{P}^{a}_{ss'} = \Pr(S_{t+1} = s' | A_t = a, S_t = s)$
- $\mathcal{R}$: reward function, $\mathcal{R}^{a}_{s} = \mathbb{E}(R_t | A_t = a, S_t = s)$
- γ: discounted factor $\in [0, 1]$, allowed to be 1 if all sequences terminate (e.g., finite horizons)
- Dynamic Programming (DP) and Monte Carlo methods (MC) 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,

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

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

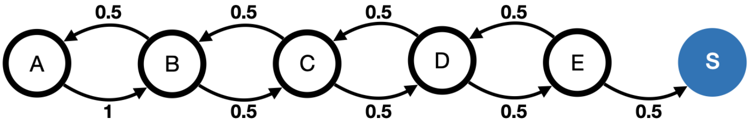
$$V^{\pi^{\text{opt}}}(s) = \max_{a} \mathbb{E}[R_t + \gamma V^{\pi^{\text{opt}}}(S_{t+1}) | A_t = a, S_t = s],$$

- or equivalently,

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

Bellman Equation: The Random Walk Example

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

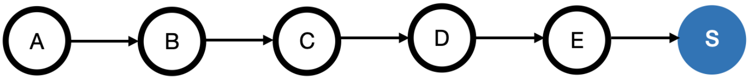


- Reward for transition to State $\boldsymbol{S}$ of value **1**, zero reward for other transitions
- Bellman equations:

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

Bellman Optimality Equation: Random Walk

- The **random walk** example:



- Reward for transition to State $\boldsymbol{S}$ of value $\mathbf{1}$, zero reward for other transitions
- Bellman optimality equations:

$$
\begin{aligned}
V^{\pi^{\text{opt}}}(A) &= \max_{a} \mathbb{E}[R_t + \gamma V^{\pi^{\text{opt}}}(S_{t+1}) | A_t = a, S_t = A] = \gamma V^{\pi^{\text{opt}}}(B) \\
V^{\pi^{\text{opt}}}(B) &= \max_{a} \mathbb{E}[R_t + \gamma V^{\pi^{\text{opt}}}(S_{t+1}) | A_t = a, S_t = B] = \gamma V^{\pi^{\text{opt}}}(C) \\
&\vdots \\
V^{\pi^{\text{opt}}}(S) &= \max_{a} \mathbb{E}[R_t + \gamma V^{\pi^{\text{opt}}}(S_{t+1}) | A_t = a, S_t = S] = 1
\end{aligned}
$$

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 $\gamma = 0$, $\mathcal{S} = \emptyset$

State-Action Value Function (Cont'd)

Relationships between V^π and Q^π

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

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

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

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

Dynamic Programming

Definition (Dynamic Programming)

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

- **Dynamic**: sequential or temporal component to the problem
- **Programming**: optimise a "program", i.e., a policy
- Dynamic programming (DP) is **rarely** used in practice (the environment is usually unknown)
- However, they provide a foundation for other solution methods

Dynamic Programming (Cont'd)

"Dynamic programming" is used to solve many other statistical learning problems

- Learning optimal **dynamic treatment regimes** (DTRs)
- Multi-scale **change point detection**
- De Boor algorithm for evaluating **B-spline** basis functions

Also used in bioinformatics, optimisation, control theory (see wiki page)

Dynamic Programming Methods

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

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

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

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

Policy Iteration: Policy Evaluation

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

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

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

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

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

Policy Evaluation (Cont'd)

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

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

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

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

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

Policy Evaluation: Algorithm

- **Iterative Policy Evaluation**: an iterative method that outputs a sequence of value functions $V_0, V_1, V_2, \cdots, V_k \to V^\pi$
- **Initial** value function V_0 is chosen arbitrarily subject to the **constraint** that at terminal state it has value $\mathbf{0}$
- **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 $\mathbf{1}$ (more in appendix), or eventual termination is guaranteed from all states under π

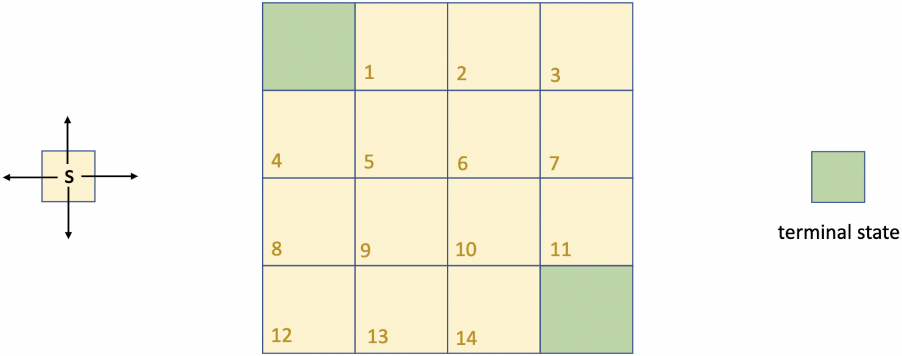
Policy Evaluation: Pseudocode

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

$$
\begin{aligned}
&\Delta \leftarrow 0 \\
&\textbf{For each } s \in \mathcal{S} \\
&\qquad \nu \leftarrow V(s) \\
&\qquad V(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s) \left[\mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a V(s') \right] \\
&\qquad \Delta \leftarrow \max(\Delta, |\nu - V(s)|) \\
&\textbf{until } \Delta < \epsilon
\end{aligned}
$$

- **Output** V

GridWorld Example



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

GridWorld Example (Cont'd)

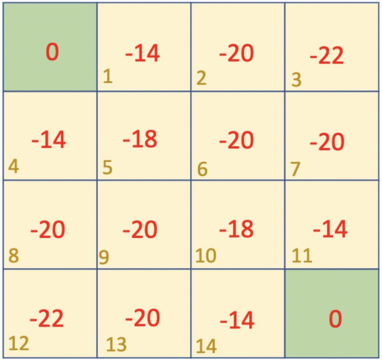


Figure: Values of uniform random policy

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

GridWorld Example (Cont'd)

By **symmetry** and Bellman equation,

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

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

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

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

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

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

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

GridWorld Example (Cont'd)

$k = 0$

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

$k = 2$

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

$k = 10$

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

$k = 1$

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

$k = 3$

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

$k = \infty$

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,

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

Theorem

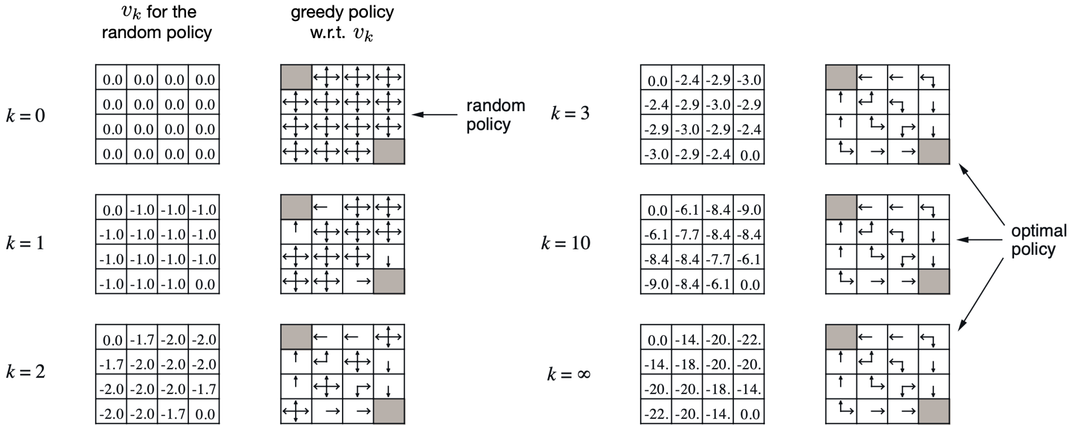
The greedy policy π' with respect to V^π is as good as or better than π,

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

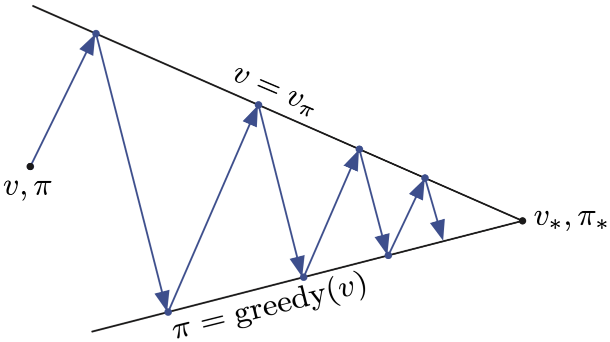
for any $s \in \mathcal{S}$.

Proof can be found in the Appendix.

GridWorld Example



Policy Iteration: Revisit



- **Policy Evaluation**: Compute $\boldsymbol{V^\pi}$ via iterative policy evaluation
- **Policy Improvement**: Generate $\boldsymbol{\pi}'$ 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 $s \in \mathcal{S}$

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

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

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

  until $\Delta < \epsilon$
- **policystable** $\leftarrow$ **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 policystable** $\leftarrow$ **False**
- **If policystable**, **then Return** π, **else** go to bullet point #2

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

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

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

Value Iteration: Pseudocode

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

  $\Delta \leftarrow \mathbf{0}$
  For each $s \in \mathcal{S}$
  $\nu \leftarrow V(s)$
  $V(s) \leftarrow \max_{a\in\mathcal{A}} \left[\mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a V(s')\right]$
  $\Delta \leftarrow \mathbf{max}(\Delta, |\nu - V(s)|)$
  until $\Delta < \epsilon$
- **Output**: optimal deterministic policy given by

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

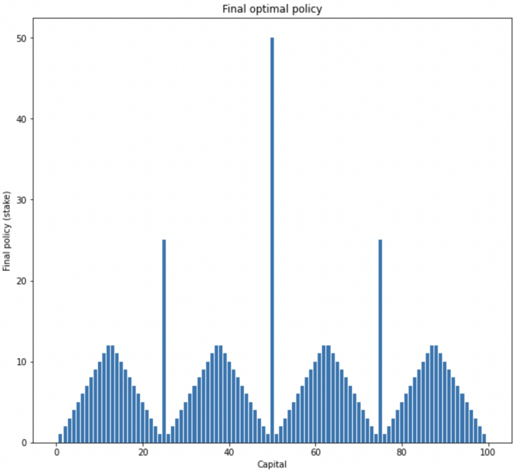
Example: Gambler's Problem

- 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
- $\mathcal{S}$: $\{0, 1, \cdots, 99, 100\}$, termination states $\mathbf{0}$ and $\mathbf{100}$
- $\mathcal{A}(s)$: $\{1, 2, \cdots, \min(s, 100 - s)\}$, depends on the state
- Pr(outcome of coin flip is heads) $= p$ (known parameter)
- Seminars:
  - Show the value function for different iterations
  - Show the optimal policy

Example: Gambler's Problem, the Optimal Policy



Some Technical Questions

- How do we know that value iteration converges to $\boldsymbol{V}^{\pi^{\text{opt}}}$?
- Or that iterative policy evaluation converges to $\boldsymbol{V}^{\pi}$?
- And therefore that policy iteration converges to $\boldsymbol{V}^{\pi^{\text{opt}}}$?
- Is the solution unique?
- These questions are resolved by **Banach fixed-point theorem** (or **contraction mapping theorem**), mentioned in Seminar 2 (more in the appendix)

Monte Carlo (MC) Methods

- Learning methods for solving the RL problem based on **averaging sample returns**
  - Estimating value functions and discovering optimal policies
  - Not assuming a model of the environment, based only on **experiences** (**model free**)
- Defined for **episodic** tasks
  - Value functions and policies are updated upon completion of an episode
  - Different from **step-by-step** methods (e.g., temporal difference learning)

MC Policy Evaluation

- **Objective**: estimate the value function V^π for a given policy π, from a set of episodes obtained by following π

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

- 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**
- Convergence is guaranteed by **law of large numbers**
- 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), $N(s) \leftarrow \mathbf{0}$ for all $s \in \mathcal{S}$

  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

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

  Returns$(s) \leftarrow$ **Returns**$(s) + G$

- **Output**:

  For each distinct s

  $N^{-1}(s)$**Returns**(s)

Every-Visit MC Policy Evaluation: Pseudocode

- **Initialization**:

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

  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

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

  Returns$(s) \leftarrow$ **Returns**$(s) + G$

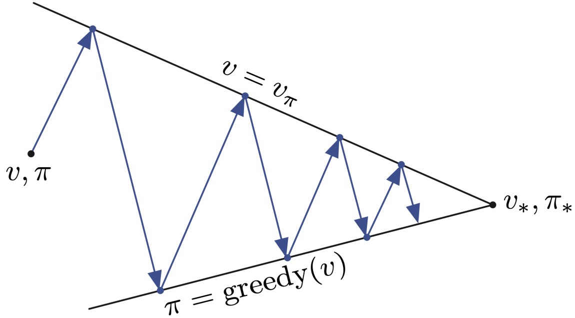
- **Output**:

  For each distinct s

  $N^{-1}(s)$**Returns**(s)

MC Control

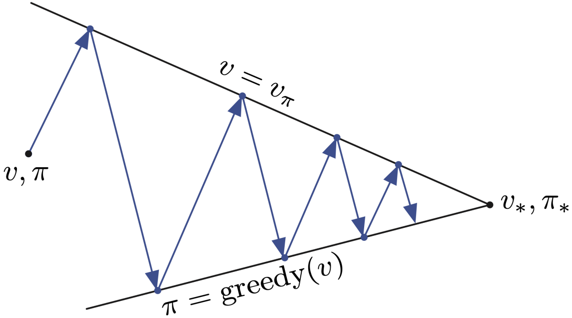
- **Objective**: use MC estimation to learn the optimal policy.
- Recall the policy iteration algorithm



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

MC Control with Generalized Policy Iteration

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



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

Policy Iteration Using State-Action Value Function

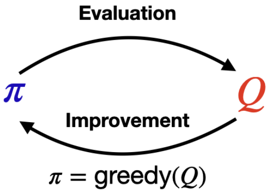
- Greedy policy improvement over V^π requires model of MDP

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

- 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



$$\pi_0 \rightarrow Q^{\pi_0} \rightarrow \pi_1 \rightarrow Q^{\pi_1} \rightarrow \cdots \rightarrow \pi^{opt} \rightarrow Q^{\pi^{opt}}$$

- **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 Exploring Starts

- **Initialization**:

  N (counter), $N(s, a) \leftarrow 0$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}$
  Returns$(s, a) \leftarrow$ an empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}$
  $\pi \leftarrow$ arbitrary
  $Q \leftarrow$ arbitrary

- **Repeat**:

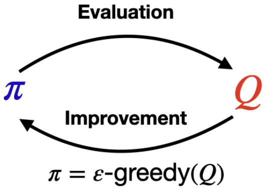
  Generate an episode using exploring starts and policy π
  For each distinct (s, a) appearing in the episode
  $\quad G \leftarrow$ return following the first occurrence of (s, a)
  $\quad N(s, a) \leftarrow N(s, a) + 1$
  $\quad \textbf{Returns}(s, a) \leftarrow \textbf{Returns}(s, a) + G$
  $\quad Q(s, a) \leftarrow \textbf{Returns}(s, a) / N(s, a)$
  $\pi(s) \leftarrow \arg\max_a Q(s, a)$ for all s

MC Control with ε-Greedy Exploration

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

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

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



$$\pi_0 \rightarrow Q^{\pi_0} \rightarrow \pi_1 \rightarrow Q^{\pi_1} \rightarrow \cdots \rightarrow \pi^{\varepsilon,opt} \rightarrow Q^{\pi^{\varepsilon,opt}}$$

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**$(s, a) + G$

  $Q(s, a) \leftarrow$ **Returns**$(s, a) / N(s, a)$

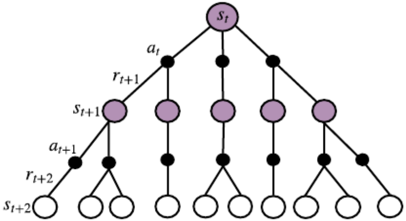
  For each distinct s:

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

Summary

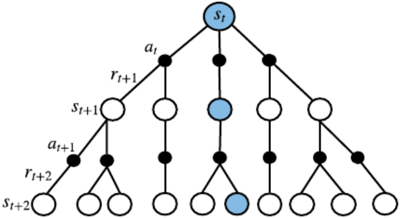
- Planning v.s. Learning
- Dynamic programming v.s. Monte Carlo Methods
- Policy Iteration v.s. Value Iteration
- Policy Evaluation v.s. Policy Improvement
- MC Policy Evaluation v.s. MC Control
- γ-Contraction, Banach Fixed Point Theorem

Summary (Cont'd)



$$DP : v(s_t) \leftarrow \mathbb{E}_\pi[r_{t+1} + \lambda v(s_{t+1})]$$

Dynamic Programming (DP)

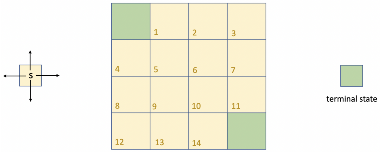


$$MC : v(s_t) \leftarrow v(s_t) + \eta(R_t - v(s_t))$$

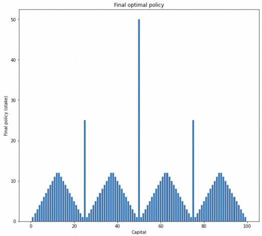
Monte Carlo (MC)

Seminar

- Solution to HW2 (due Wed 12pm)
- Iterative policy evaluation: Gridworld problem



- Value iteration: Gambler's problem



- Monte Carlo prediction & control: Black jack example

Questions

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,

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

- This yields

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

- 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: Value Function ∞-Norm

- Measure distance between two value functions $\boldsymbol{V_1}$ and $\boldsymbol{V_2}$ by the ∞-norm
- i.e., the **largest** difference between state values,

$$\|\boldsymbol{V}_1 - \boldsymbol{V}_2\|_\infty = \max_{s \in \mathcal{S}} |\boldsymbol{V_1}(s) - \boldsymbol{V_2}(s)|$$

- Given a sequence of values $\{\boldsymbol{V_k}\}_{\boldsymbol{k}}$, convergences requires $\|\boldsymbol{V_k} - \boldsymbol{V}^*\|_\infty \to \mathbf{0}$ for some $\boldsymbol{V}^*$ as $\boldsymbol{k} \to \infty$

Appendix: Bellman Expectation Operator

Definition

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

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

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

$$\max_s |\boldsymbol{T}^{\pi}\boldsymbol{V}_1(s) - \boldsymbol{T}^{\pi}\boldsymbol{V}_2(s)| = \gamma \max_s \left|\sum_{a,s'} \pi(a|s)\mathcal{P}_{ss'}^a [\boldsymbol{V}_1(s') - \boldsymbol{V}_2(s')]\right|$$

$$\leq \gamma \max_s |\boldsymbol{V}_1(s) - \boldsymbol{V}_2(s)| \max_s \left|\sum_{a,s'} \pi(a|s)\mathcal{P}_{ss'}^a\right| = \gamma \max_s |\boldsymbol{V}_1(s) - \boldsymbol{V}_2(s)|$$

- Iterative Policy Evaluation: $\boldsymbol{V}_0 \rightarrow \boldsymbol{T}^{\pi}\boldsymbol{V}_0 \rightarrow \boldsymbol{T}^{\pi}\boldsymbol{T}^{\pi}\boldsymbol{V}_0 \rightarrow \cdots$

Appendix: Banach Fix Point Theorem

Theorem

Suppose $\boldsymbol{T}$ is a γ-contraction. Then under certain conditions,

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

- Proof can be found here
- $\boldsymbol{T}^{\boldsymbol{\pi}}$ is has a unique fix point
- $\boldsymbol{V}^{\boldsymbol{\pi}}$ is the fix point, according to the Bellman equation
- Iterative policy evaluation converges to $\boldsymbol{V}^{\boldsymbol{\pi}}$

Appendix: Bellman Optimality Operator

Definition

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

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

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

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

Appendix: Convergence of Dynamic Programming

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