MDP: knows S, A, P, R
RL doesn't know P

SC3000/CZ3005
Artificial Intelligence

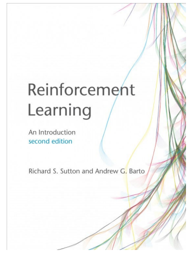
Reinforcement Learning

Prof Bo AN

Research area: artificial intelligence, computational game theory, reinforcement learning, optimization
www.ntu.edu.sg/home/boan
Email: boan@ntu.edu.sg
Office: N4-02b-55

Lesson Outline

- Some RL algorithms:
  - Monte-Carlo
  - Q-learning
  - Deep Q-Network



Reinforcement Learning

- Motivation
  - In last lecture, we compute the value function and find the optimal policy
  - But if without the transition function $P(s'|s,a)$?
  - We can learn the value function and find the optimal policy without transition
    - From experience

Experience $\xrightarrow{\text{learning}}$ Policy/Value

RL algorithms

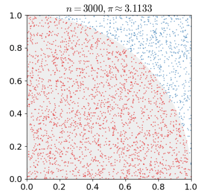
- Types
  - Monte Carlo
  - Q-Learning
  - DQN
  - …

Model-free *Learning*
→ Sampling: Monte Carlo
→ Bootstrapping: Q-Learning → DNN for Function Approx.: DQN

What is Monte Carlo

- Idea behind MC:
  - Just use randomness to solve a problem
- Simple definition:
  - Solve a problem by generating suitable random numbers and observing the fraction of numbers obeying some properties
- An example for calculating π (not policy in RL):
  - $S_{red} = \frac{1}{4}\pi r^2, S_{squre} = r^2$
  - putting dots on the square randomly for $n = 3000$ times
  - $\pi \approx 4 \times \frac{N_{red}}{n}$, N_{red} is the number of dots in the circle

$\pi \approx \frac{4 S_{red}}{S_{square}}$ random dots that estimate this



Monte Carlo in RL: Prediction

- Basic Idea: we run in the world randomly and gain experience to learn
- What experience? Many trajectories!
  - $(s_1, a_1, r_2, s_2, a_2, r_3, \ldots, s_T), \ldots$
- What we learn? Value function!
  - Recall that the return is the total discounted rewards:

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^n r_{t+n} + \cdots = \Sigma_i \gamma^i r_{t+i}$$

sum of reward for one trajectory

  - Recall that the value function is the expected return from s

$$V_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

so we cannot cal this

dk this

- How we learn?
  - Use experience to learn an empirical state value function $\tilde{V}_\pi(s) = \frac{1}{N}\Sigma_{i=1}^{N} G_{i,s}$

sample this many times

use Ave as approx of V-functions

$\mathbb{E}$ of G func

↳ total Reward of single traj

$\tilde{V}_\pi = \frac{1}{N}\sum_{i=1}^{N} G_{i,s}$

e.g want to find value of cell 2 → $V(S_2)$

use G_t which starts from Cell 2 & calc that for every trajectory → then Ave.

An Example

- One-dimensional grid world
  - A robot is in a 1x4 world
  - State: current cell $s \in [cell_1, cell_2, cell_3, cell_4]$
  - Action: left or right
  - Reward:
    - Move one step (-1)
    - Reach the destination cell (+10) (ignoring the one-step reward)

No transition probability.
- dk the prob of going to the next state.

$cell_1$	$cell_2$	$cell_3$	$cell_4$
	Start point		Destination

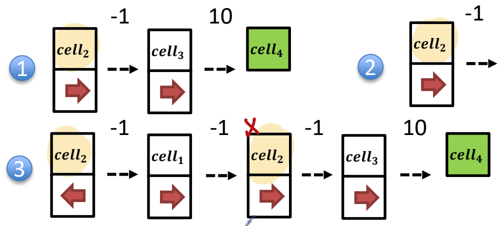
One-dimensional Grid World

$cell_1$	$cell_2$	$cell_3$	$cell_4$
	Start point		Destination

- Trajectory or episode:
  - The sequence of states from the staring state to the terminal state
  - Robot starts in $cell_2$, ends in $cell_4$
- The representation of the three episodes

First visit mc don't recalc this part

3 diff traj



can we estimate value of cell 1, cell 2, cell 3.

since cell 2 appears twice in this traj → for now we can use first visit MC

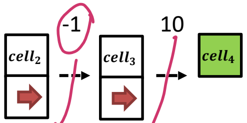
Compute Value Function

- Idea: Average return observed after visits to (s, a)
- First-visit MC: average returns only for first time (s, a) is visited in an episode
- Return in one episode (trajectory):

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^n r_{t+n} + \cdots = \Sigma_i \gamma^i r_{t+i}$$

- We calculate the return for $cell_2$ of first episode with $\gamma = 0.9$

Gt from this trajectory.



$$G_t = -1 \times 0.9^0 + 10 \times 0.9^1 = 8$$

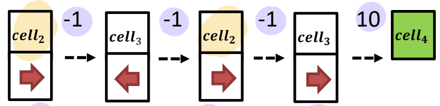
Reward

discount factor = 0.9

$cell_1$ $-1 \times -1 \times 0.9^0 + 10 \times 0.9^1$
=

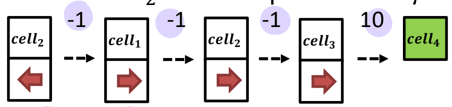
Compute Value Function (cont'd)

- Similarly the return for $cell_2$ of second episode with $\gamma = 0.9$



$$G_t = -1 \times 0.9^0 - 1 \times 0.9^1 - 1 \times 0.9^2 + 10 \times 0.9^3 = 4.58$$

- Similarly the return for $cell_2$ of third episode with $\gamma = 0.9$



$$G_t = -1 \times 0.9^0 - 1 \times 0.9^1 - 1 \times 0.9^2 + 10 \times 0.9^3 = 4.58$$

- The empirical value function for $cell_2$ is $\frac{8 + 4.58 + 4.58}{3} = 5.72$

Compute Value Function (cont'd)

Q(s,a)

- Given these three episodes, we compute the value function for all non-terminal state

6.2 $cell_1$	5.72 $cell_2$	8.73 $cell_3$

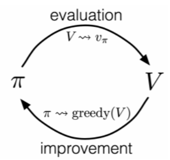
- We can get more accurate value function with more episodes

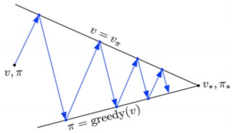
First Visit Monte Carlo Policy Evaluation

- Average returns only for the first time s is visited in an episode
- Algorithm
  - Initialize:
    - $\pi \leftarrow$ policy to be evaluated
    - $V \leftarrow$ an arbitrary state-value function
    - $Returns(s) \leftarrow$ an empty list, for all state s
  - Repeat many times:
    - Generate an episode using π
    - For each state s appearing in the episode:
      - $R \leftarrow$ return following the first occurrence of s
      - Append R to $Returns(s)$
      - $V(s) \leftarrow average(Returns(s))$

Monte Carlo in RL: Control

- Now, we have the value function of all states given a policy
- We need to improve policy to be better
- Policy Iteration
  - Policy evaluation
  - Policy improvement
- However, we need to know how good an action is





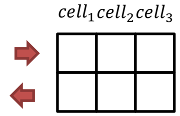
Q-value

- Estimate how good an action is when staying in a state
- Defined as the expected return starting from s, taking the action a and thereafter following policy π

$$Q^\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$$

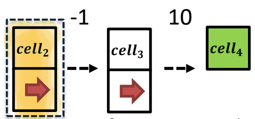
similar to ave

- Representation: A table
  - Filled with the Q-vale given a state and an action



Computing Q-value

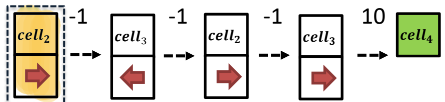
- MC for estimating Q:
  - A slight difference from estimating the value function
  - Average returns for state-action pair (s, a) is visited in an episode
- We calculate the return for $(cell_2, \text{right})$ of first episode with $\gamma = 0.9$



$$G_t = -1 \times 0.9^0 + 10 \times 0.9^1 = 8$$

Compute Q-Value (cont'd)

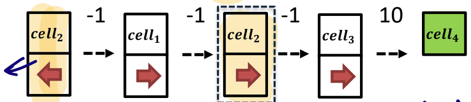
- Similarly the return for $(cell_2, \text{right})$ of second episode with $\gamma = 0.9$



$$G_t = -1 \times 0.9^0 - 1 \times 0.9^1 - 1 \times 0.9^2 + 10 \times 0.9^3 = 4.58$$

- Similarly the return for $(cell_2, \text{right})$ of third episode with $\gamma = 0.9$

don't consider left



$$G_t = -1 \times 0.9^0 + 10 \times 0.9^1 = 8$$

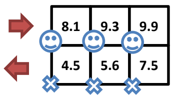
↳ consider this instead

- The empirical Q-value function for $(cell_2, \text{right})$ is $\frac{8 + 4.58 + 8}{3} = 6.86$

Q-Value for Control

- Filling the Q-table
  - By going through all state-action pairs, we get a complete Q-table with all the entries filled
  - A possible Q-table example

$cell_1 \, cell_2 \, cell_3$



- Selecting action

$$\pi'(s) = \text{argmax}_{a \in A} Q^{\pi}(s,a)$$

At $cell_1, cell_2$ and $cell_3$, we choose right

(unbiased Estimation of true value)

MC control algorithm

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
  $Q(s,a) \leftarrow$ arbitrary
  $Returns(s,a) \leftarrow$ empty list
  $\pi \leftarrow$ an arbitrary ε-soft policy

Repeat forever:
  (a) Generate an episode using π
  (b) For each pair s, a appearing in the episode:
    $R \leftarrow$ return following the first occurrence of s, a
    Append R to $Returns(s,a)$
    $Q(s,a) \leftarrow \text{average}(Returns(s,a))$
  (c) For each s in the episode:
    $a^* \leftarrow \arg\max_a Q(s,a)$
    For all $a \in \mathcal{A}(s)$:

$$\pi(s,a) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = a^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq a^* \end{cases}$$

Limitation of MC: Requires the entire trajectory. too slow

Need whole trajectory

Policy evaluation

Policy improvement

ε-Greedy trade off btw exploration & exploitation.

explore new

exploit best of all you know

e.g. $\varepsilon = 0.1$

a_1	a_2	a_3	a_4	a_5
0.02	0.02 + 0.9	0.02	0.02	0.02

$\underbrace{0.02}_{0.92}$

Q-Learning

- Previously, we need the whole trajectory
- In Q-Learning, we only need one-step trajectory: (s, a, r, s')
- The difference is the Q-value computing
  - Previously:

$$\tilde{Q}_{\pi}(s,a) = \frac{1}{N} \Sigma_{i=1}^{N} G_{i,s}$$

  - Now, updating rule:

$$Q_{new}(S_t, A_t) \leftarrow Q_{old}(S_t, A_t) + \alpha\big(\underline{R_{t+1} + \gamma \max_a Q_{old}(S_{t+1}, a)} - Q_{old}(S_t, A_t)\big)$$

new estimation

learning rate

new sample

old estimation

small #

Q-Learning

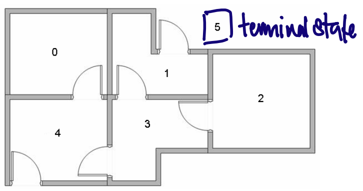
Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

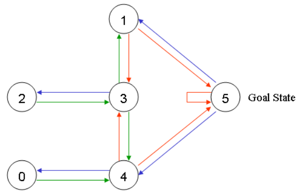
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
    $S \leftarrow S'$
  until S is terminal

A Step-by-step Example

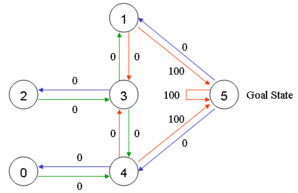
- 5-room environment as MDP
  - We'll number each room 0 through 4
  - The outside of the building can be thought of as one big room 5
  - End at room 5
  - Notice that doors at rooms 1 and 4 lead into the building from room 5 (outside)





A Step-by-step Example (cont'd)

- Goal
  - Put an agent in any room, and from that room, go outside (or room 5)
- Reward
  - The doors that lead immediately to the goal have an instant reward of 100
  - Other doors not directly connected to the target room have zero reward



action

$$R = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 100 \end{bmatrix} \end{array}$$

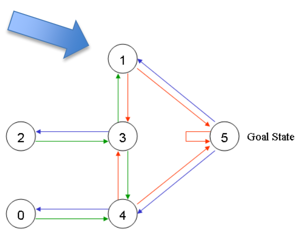
state

Q-Learning Step by Step

- Initialize matrix Q as a zero matrix

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

- $\alpha = 0.01, \gamma = 0.99$
- Loop for each episode until converge
  - Initial state: current we are in room 1 (1st outer loop)
  - Loop for each step of episode (until reach room 5)
    - … (Next slide)



Q-Learning Step by Step (cont'd)

- … (last slide)
  - Loop for each step of episode (until room 5)
    - By random selection, we go to 5
    - We get 100 reward
    - Update Q: $Q_{new}(S_t, A_t) \leftarrow Q_{old}(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q_{old}(S_{t+1}, a) - Q_{old}(S_t, A_t))$
      - At room 5, we have 3 possible actions: go to 1, 4 or 5; We select the one with max reward
      - $Q_{new}(1,5) \leftarrow Q_{old}(1,5) + \alpha\left(100 + \gamma \max_a Q_{old}(5,a) - Q_{old}(1,5)\right) = 0 + 0.01 \times (100 + 0.99 \times 0 - 0) = 1$

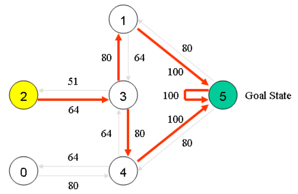
$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \Rightarrow Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Q-Learning Step by Step (cont'd)

- When we loop many episodes, we can get

$$
Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c}
\begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{bmatrix}
0 & 0 & 0 & 0 & 80 & 0 \\
0 & 0 & 0 & 64 & 0 & 100 \\
0 & 0 & 0 & 64 & 0 & 0 \\
0 & 80 & 51 & 0 & 80 & 0 \\
64 & 0 & 0 & 64 & 0 & 100 \\
0 & 80 & 0 & 0 & 80 & 100
\end{bmatrix}
\end{array}
$$

- According to this Q-table, we can select actions
  - E.g. We are at room 2
  - Greedily select based on maximun of Q value



An Example of Iteration Process

- A complex grid world example
- https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_td.html

NANYANG TECHNOLOGICAL UNIVERSITY | SINGAPORE

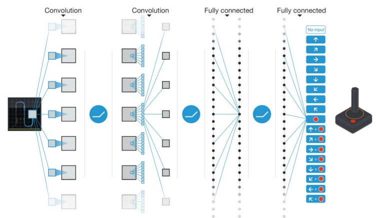
Deep Q-Network

- Previously, we represent the Q-value as a table
- However, tabular representation is insufficient
  - Many real world problems have enormous state and/or action spaces
  - Backgammon: 10^20 states
  - Computer Go: 10^170 states
  - Robots: continuous state space
- We use a neural network as a black box to replace the table
  - Input a state and an action, output the Q-value

NANYANG TECHNOLOGICAL UNIVERSITY | SINGAPORE

DQN in Atari

- Input state s is stack of raw pixels from last 4 frames
- Output is $q(s,a)$ for 18 button
- Reward is change in score for that step



NANYANG TECHNOLOGICAL UNIVERSITY | SINGAPORE

DQN in Atari (cont’d)

- Pong’s video
- https://www.youtube.com/watch?v=PSQt5KGv7Vk
- Beat human on many games