NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

AI6101 Introduction to AI and AI Ethics

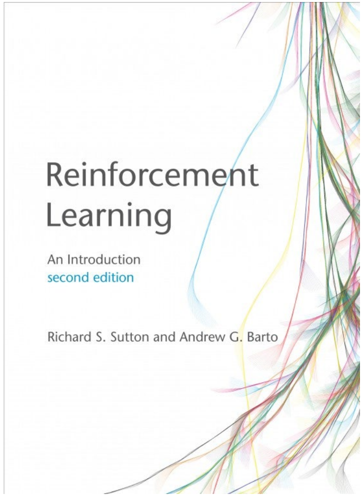
Reinforcement Learning

Assoc Prof Bo AN

www.ntu.edu.sg/home/boan
Email: boan@ntu.edu.sg
Office: N4-02b-55

Lesson Outline

- Intro of reinforcement learning
- Some RL algorithms
- Advanced materials:
  - Policy gradient methods
  - Exploitation vs exploration



Reinforcement Learning

What makes reinforcement learning different from other machine learning paradigms?

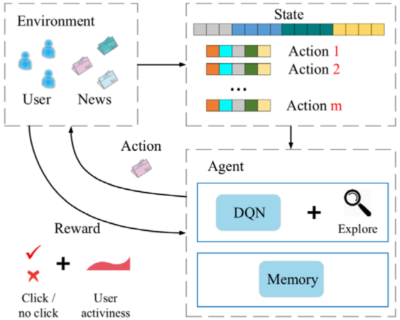
- There is no supervisor, only a reward signal
- Feedback is delayed, not instantaneous
- Time really matters (sequential, non i.i.d data)
- Agent's actions affect the subsequent data it receives

Examples of Reinforcement Learning

Robotic control

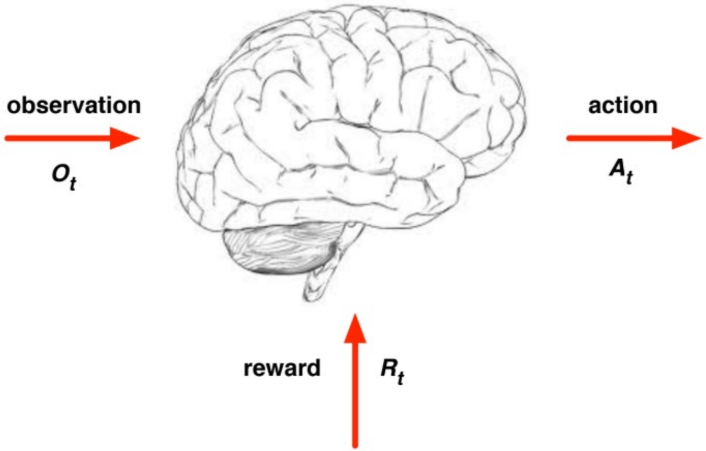
The Game of Go

Video games



Recommendation System

Agent and Environment



- The observation is the perception of the environment for agent
- The action will change the outside environment
- The reward is a scalar value indicates how well agent is doing at step t

The agent’s job is to maximize the cumulative reward

Major Components of an RL Agent

- Policy: agent’s behavior function
- Value function: how good is each state and/or action
- Model: agent’s representation of the environment

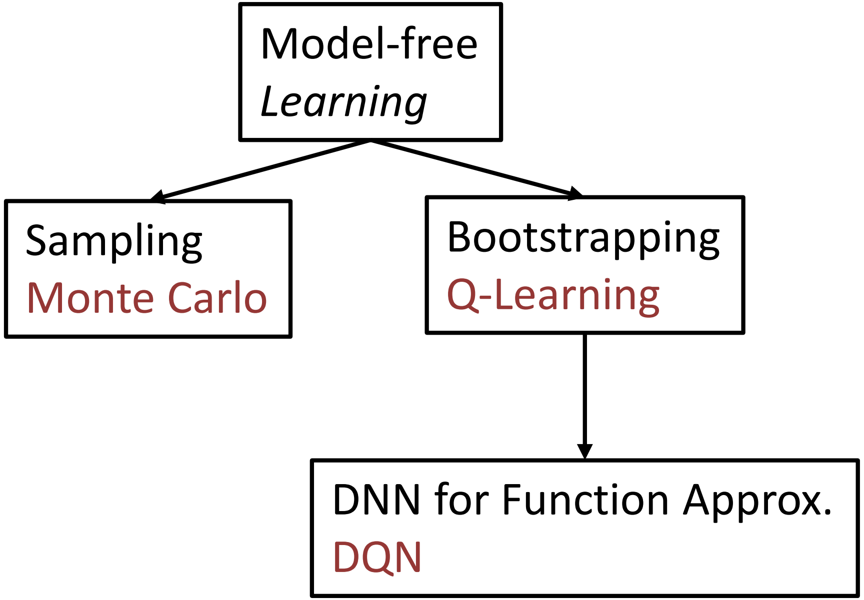
Reinforcement Learning

- 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

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

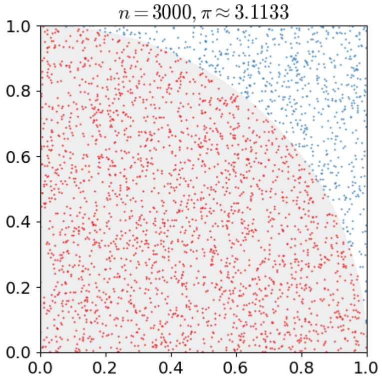
RL algorithms

- Types
  - Monte Carlo
  - Q-Learning
  - 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



Monte Carlo in RL: Prediction

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

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

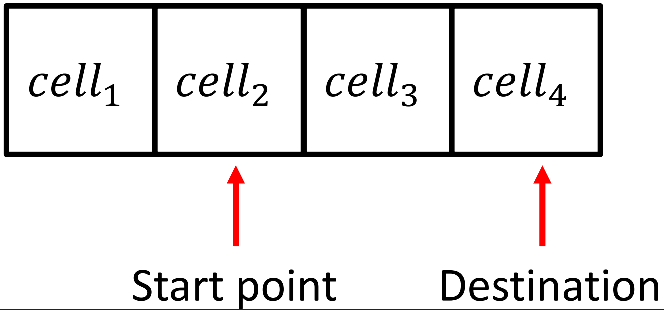
  - Recall that the value function is the expected return from s

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

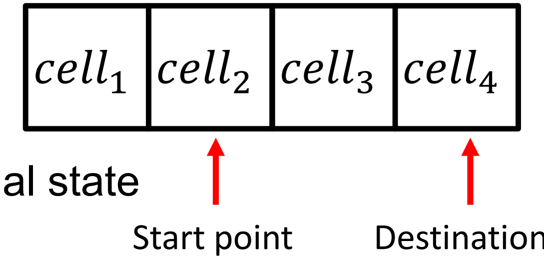
- How we learn?
  - Use experience to learn an empirical state value function $\tilde{V}_\pi(s) = \frac{1}{N}\Sigma_{i=1}^{N} G_{i,s}$

An Example

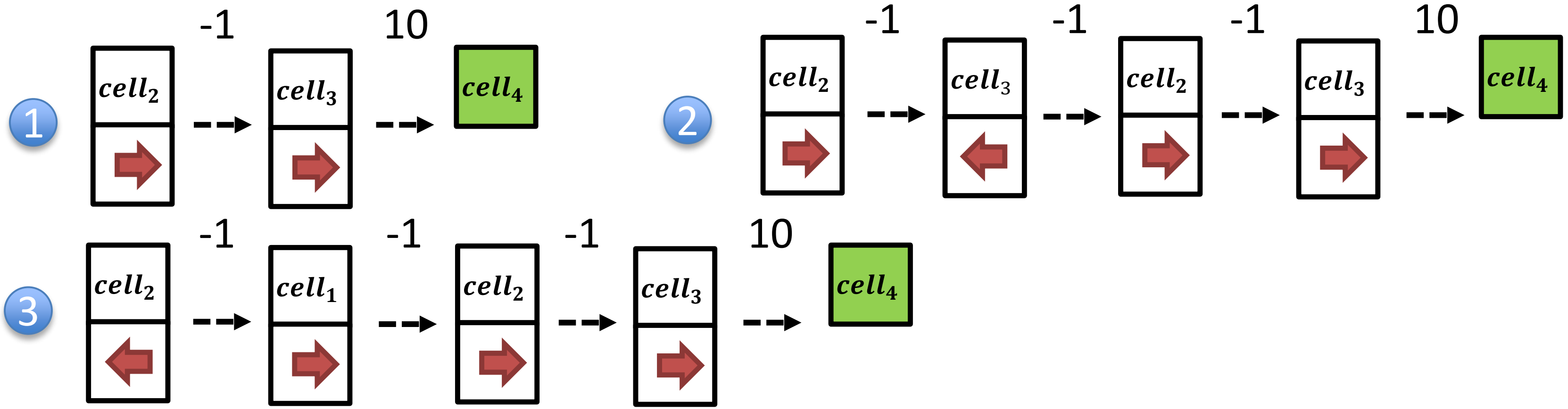
- One-dimensional grid world
  - A robot is in a 1x4 world
  - State: current cell $s \in [cell_1, cell_2, cell_3, cell_4]$
  - Action: left or right
  - Reward:
    - Move one step (-1)
    - Reach the destination cell (+10) (ignoring the one-step reward)



One-dimensional Grid World



- 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

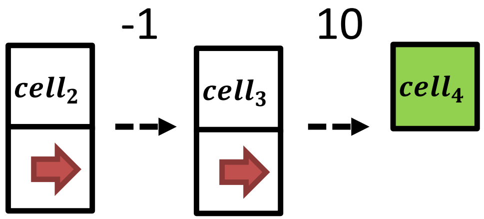


Compute Value Function

- 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):

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

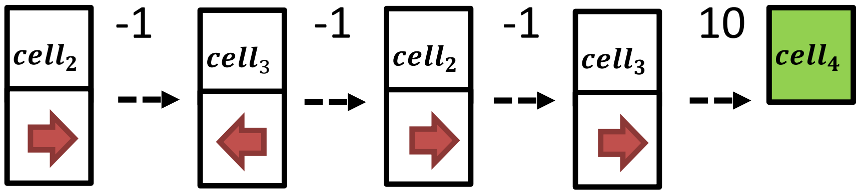
- We calculate the return for $cell_2$ of first episode with $\gamma = 0.9$



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

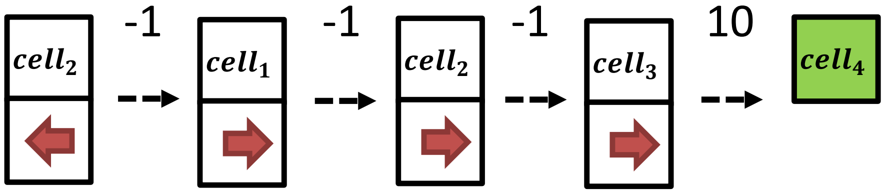
Compute Value Function (cont'd)

- 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)

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

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

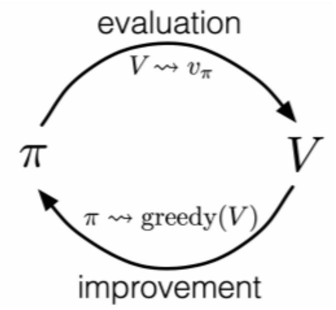
- We can get more accurate value function with more episodes

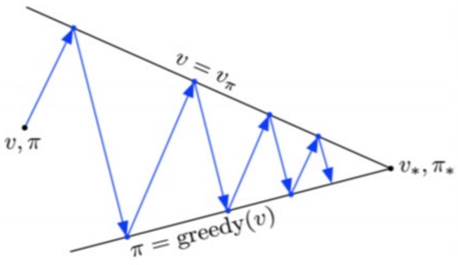
First Visit Monte Carlo Policy Evaluation

- 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



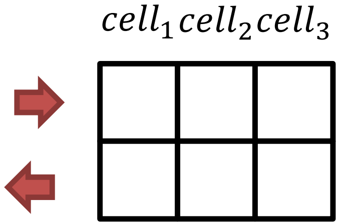


Q-value

- 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

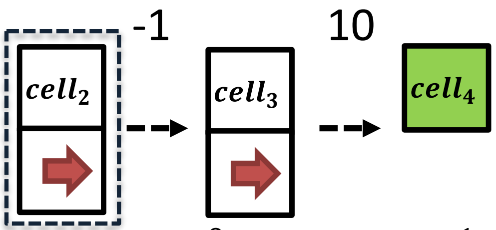
$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[G_t | S_t = s, A_t = a]$$

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



Computing Q-value

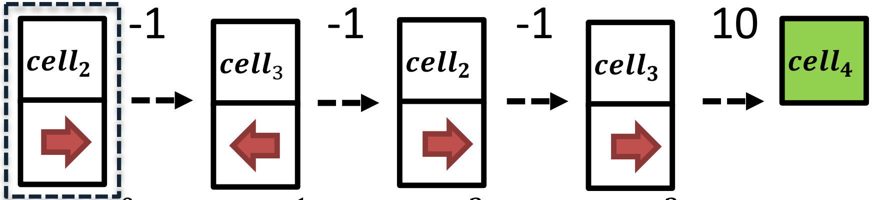
- MC for estimating Q:
  - A slight difference from estimating the value function
  - Average returns for state-action pair (s, a) is visited in an episode
- We calculate the return for $(cell_2, \text{right})$ of first episode with $\gamma = 0.9$



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

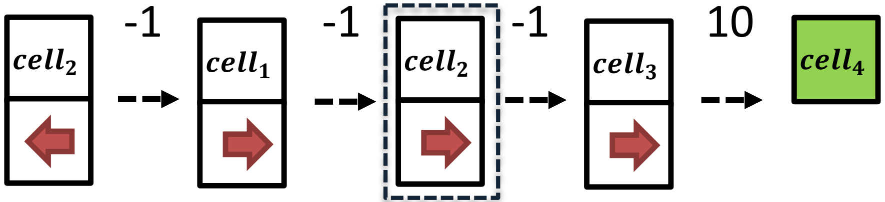
Compute Q-Value (cont'd)

- 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 $\gamma = 0.9$

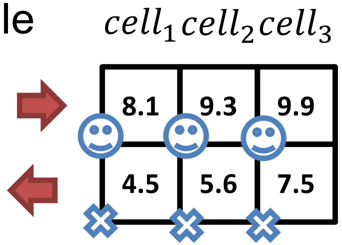


$$G_t = -1 \times 0.9^0 + 10 \times 0.9^1 = 8$$

- 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



- Selecting action

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

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

MC control algorithm

Policy evaluation

Policy improvement

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 $a \in \mathcal{A}(s)$:

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

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:

$$\underbrace{Q_{new}(S_t, A_t)}_{\text{new estimation}} \leftarrow Q_{old}(S_t, A_t) + \underbrace{\alpha}_{\text{learning rate}}(\underbrace{R_{t+1} + \gamma \max_a Q_{old}(S_{t+1}, a)}_{\text{new sample}} - \underbrace{Q_{old}(S_t, A_t)}_{\text{old estimation}})$$

Q-Learning

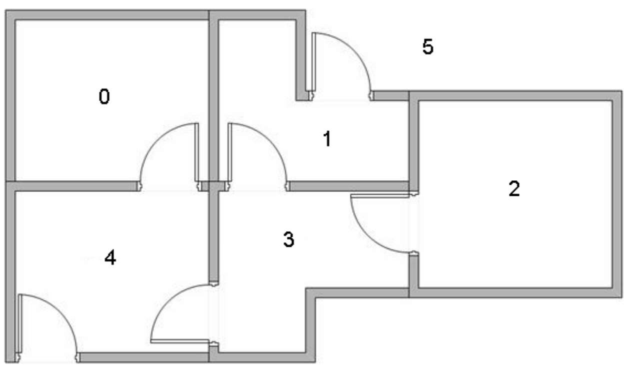
Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

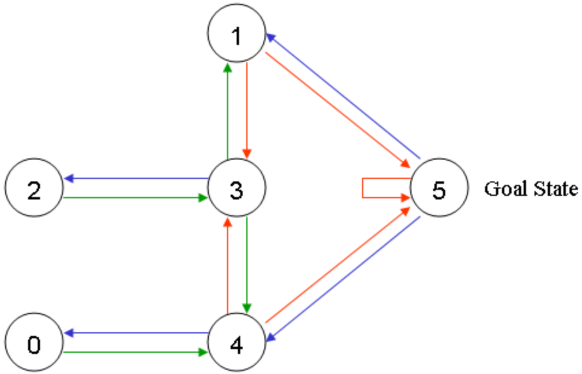
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

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'
    $Q(S, A) \leftarrow Q(S, \underline{A}) + \alpha\left[R + \gamma \underline{\max_a Q(S', a)} - Q(S, A)\right]$
    $S \leftarrow S'$
  until S is terminal

A Step-by-step Example

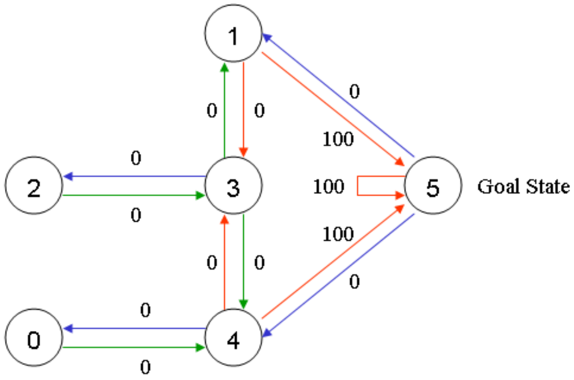
- 5-room environment as MDP
  - We'll number each room 0 through 4
  - The outside of the building can be thought of as one big room 5
  - End at room 5
  - Notice that doors at rooms 1 and 4 lead into the building from room 5 (outside)





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

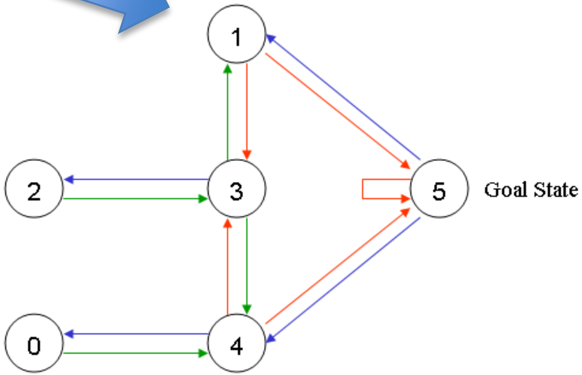


$$
R = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ \text{state} \end{array}
\begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \quad \text{action} \\
\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0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

Q-Learning Step by Step

- Initialize matrix Q as a zero matrix
- $\alpha = 0.01, \gamma = 0.99$
- 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)

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



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
      - $Q_{new}(1{,}5) \leftarrow Q_{old}(1{,}5) + \alpha\left(100 + \gamma \max_a Q_{old}(5, a) - Q_{old}(1{,}5)\right) = 0 + 0.01 \times (100 + 0.99 \times 0 - 0) = 1$

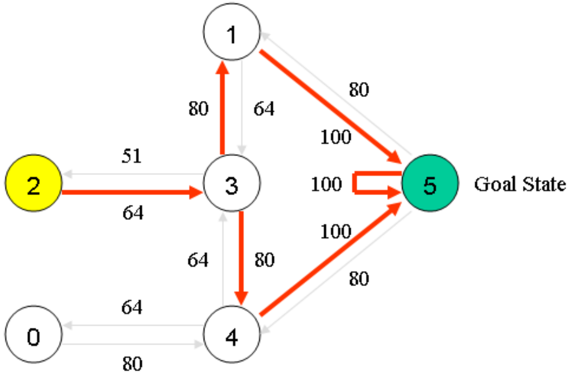
$$Q = \begin{array}{c|c} & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}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

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

Value-based Methods: SARSA

- SARSA Introduction
  - Similar to Q-learning with some differences
    - On-policy: update the Q-table with the (s, a, r, s') samples generate by the current policy

on-policy: updating the Q with current policy

$$Q(S,A) = Q(S,A) + \alpha \cdot [R + \gamma \cdot \cancel{\max_{A}}(Q(S',A')) - Q(S,A))]$$

The next state and next action in transition samples

  - Epsilon greedy can still be used to output actions like Q-learning

For code, refer to https://www.geeksforgeeks.org/sarsa-reinforcement-learning/

Value-based Methods: SARSA

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

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

Advanced Materials

Policy Gradient Methods: Background

- How do value function work as a policy?
  - Output actions with the best Q values
- Can we directly learn a policy mapping states to actions?
- Policy gradient methods
  - Learn a *parameterized* policy that can select actions without consulting a value function
  - Use $\pi(a|s,\boldsymbol{\theta}) = \Pr\{A_t = a \mid S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}\}$ with parameters $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ for the probability that action *a* is taken at time *t* given that the environment is in state *s* at time *t*
  - The value functions can also be parameterized as $\hat{v}(s,\mathbf{w})$ with parameters $\mathbf{w} \in \mathbb{R}^{d}$ (optional)
  - Update the parameters by gradient ascent given some performance measures $J(\boldsymbol{\theta})$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \widehat{\nabla J(\boldsymbol{\theta}_t)}$$

Policy Gradient Methods: Linear Example

- A linear function approximation example of policy gradient methods
  - Parameters of policy function of a linear function, soft-max policy

$$\theta = \{\theta_0, \theta_1, \theta_2\} \quad \text{action} = \{a_0, a_1, a_2\} \quad s = \{s_0, s_1\}$$

$$\pi(s, a_i) = \frac{e^{a_i\theta_0 + s_0\theta_1 + s_1\theta_2}}{\sum_{j=0}^{|A|} e^{a_j\theta_0 + s_0\theta_1 + s_1\theta_2}}$$

Sampling actions with these probability

  - We introduce optimization rules in the following slides
- Deep Neural networks can also be used as the approximation function
  - Deep Reinforcement Learning (DRL)

Policy Gradient Methods

- Policy gradient (PG) methods model and optimize the policy directly

$$\pi_\theta(s, a) = \mathbb{P}[a \mid s, \theta]$$

- By maximizing performance measure w.r.t π_θ

$$J(\theta) = V^{\pi_\theta} = E[R]$$

Policy Gradient Methods

- Policy gradient (PG) methods model and optimize the policy directly
- The policy is modeled with a parameterized function respect to θ, $\pi_\theta(a|s)$

Transition function

Performance measure

$$J(\theta) = \sum_{s\in\mathcal{S}} d^\pi(s) V^\pi(s) = \sum_{s\in\mathcal{S}} d^\pi(s) \sum_{a\in\mathcal{A}} \boxed{\pi_\theta(a|s)}\boxed{Q^\pi(s,a)}$$

Value function

Gradients

$$\begin{aligned}\nabla_\theta J(\theta) &= \nabla_\theta \sum_{s\in\mathcal{S}} d^\pi(s) \sum_{a\in\mathcal{A}} Q^\pi(s,a)\pi_\theta(a|s) \\ &\propto \sum_{s\in\mathcal{S}} d^\pi(s) \sum_{a\in\mathcal{A}} Q^\pi(s,a)\nabla_\theta \pi_\theta(a|s)\end{aligned}$$

Policy Gradient Methods: REINFORCE

- REINFORCE (Monte-Carlo policy gradient) relies on an estimated return by Monte-Carlo methods using episode samples to update the policy parameter θ

$$
\begin{aligned}
\nabla_\theta J(\theta) &\propto \sum_{s\in\mathcal{S}} d^\pi(s) \sum_{a\in\mathcal{A}} Q^\pi(s,a) \nabla_\theta \pi_\theta(a|s) \\
&= \sum_{s\in\mathcal{S}} d^\pi(s) \sum_{a\in\mathcal{A}} \pi_\theta(a|s) Q^\pi(s,a) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} \\
&= \mathbb{E}_\pi[Q^\pi(s,a)\nabla_\theta \ln \pi_\theta(a|s)] && \text{; Because } (\ln x)'=1/x \\
&= \mathbb{E}_\pi[G_t \nabla_\theta \ln \pi_\theta(A_t|S_t)] && \text{; Because } Q^\pi(S_t,A_t)=\mathbb{E}_\pi[G_t|S_t,A_t]
\end{aligned}
$$

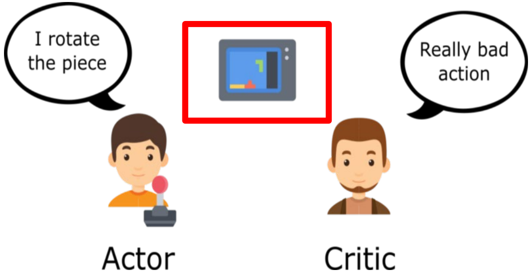
$$\boxed{G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}}$$

Policy Gradient Methods: REINFORCE

1. Initialize the policy parameter θ at random.
2. Generate one trajectory on policy π_θ: $S_1, A_1, R_2, S_2, A_2, \ldots, S_T$.
3. For t=1, 2, … , T:
    1. Estimate the the return G_t;
    2. Update policy parameters: $\theta \leftarrow \theta + \alpha\gamma^t G_t \nabla_\theta \ln \pi_\theta(A_t|S_t)$

Policy Gradient Methods: Actor-Critic

- Actor-critic methods consist of two models
  - Critic updates the value function parameters w and depending on the algorithm it could be action-value $Q_w(a|s)$ or state-value $V_w(s)$
  - Actor updates the policy parameters θ for $\pi_\theta(a|s)$, in the direction suggested by the critic



1. Initialize s, θ, w at random; sample $a \sim \pi_\theta(a|s)$.
2. For $t = 1 \dots T$:
   1. Sample reward $r_t \sim R(s, a)$ and next state $s' \sim P(s'|s, a)$;
   2. Then sample the next action $a' \sim \pi_\theta(a'|s')$;
   3. Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s, a) \nabla_\theta \ln \pi_\theta(a|s)$;
   4. Compute the correction (TD error) for action-value at time t:
      $\boxed{\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)}$
      and use it to update the parameters of action-value function:
      $\boxed{w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s, a)}$
   5. Update $a \leftarrow a'$ and $s \leftarrow s'$.

Exploitation vs Exploration

- Online decision-making involves a fundamental choice:
  - Exploitation Make the best decision given current information
  - Exploration Gather more information
- The best long-term strategy may involve short-term sacrifices
- Gather enough information to make the best overall decisions

Examples

Restaurant Selection

- Exploitation Go to your favorite restaurant
- Exploration Try a new restaurant

Online Banner Advertisements

- Exploitation Show the most successful advert
- Exploration Show a different advert

Oil Drilling

- Exploitation Drill at the best known location
- Exploration Drill at a new location

Game Playing

- Exploitation Play the move you believe is best
- Exploration Play an experimental move

Principles

Naive Exploration

- Add noise to greedy policy (e.g. ϵ-greedy)

Optimistic Initialization

- Assume the best until proven otherwise

Optimism in the Face of Uncertainty

- Prefer actions with uncertain values

Probability Matching

- Select actions according to probability they are best

Information State Search

- Lookahead search incorporating value of information