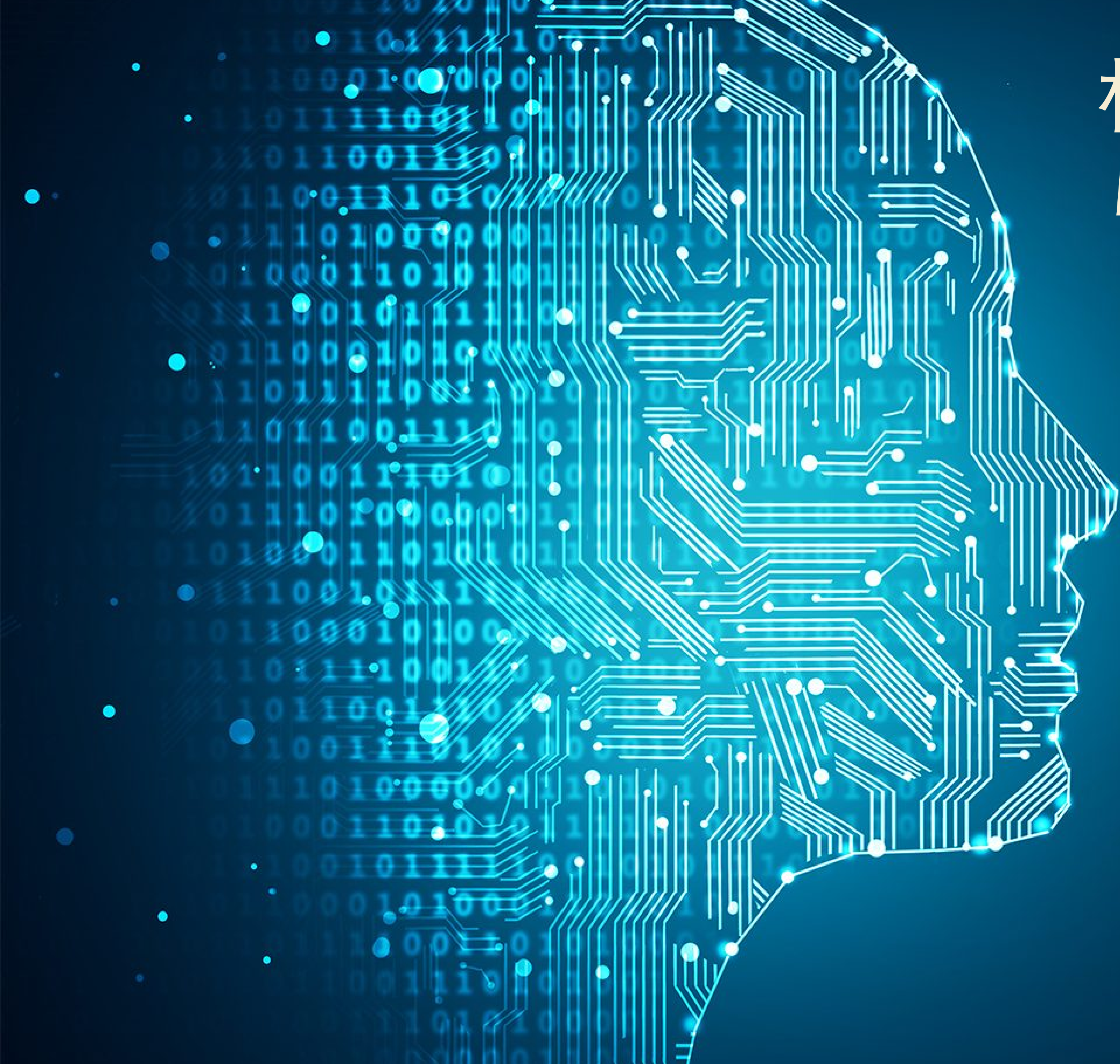


机器学习与人工智能
Machine Learning and Artificial Intelligence

Lecture 10 Reinforcement Learning

Yingjie Zhang (张颖婕)

Peking University

yingjiezhang@gsm.pku.edu.cn

2021 Fall

Agenda

- Overview
- Markov Decision Process
- Q-Learning

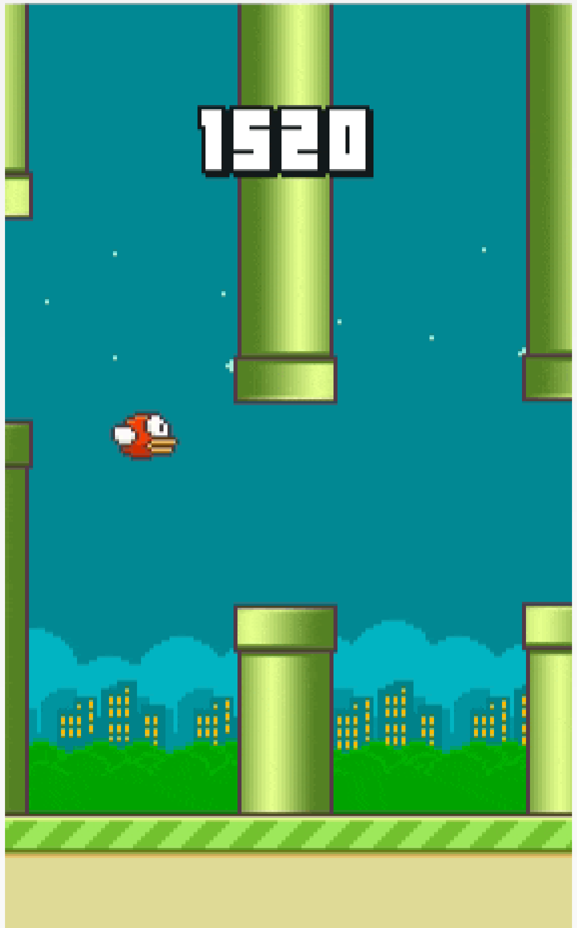
RL and ML

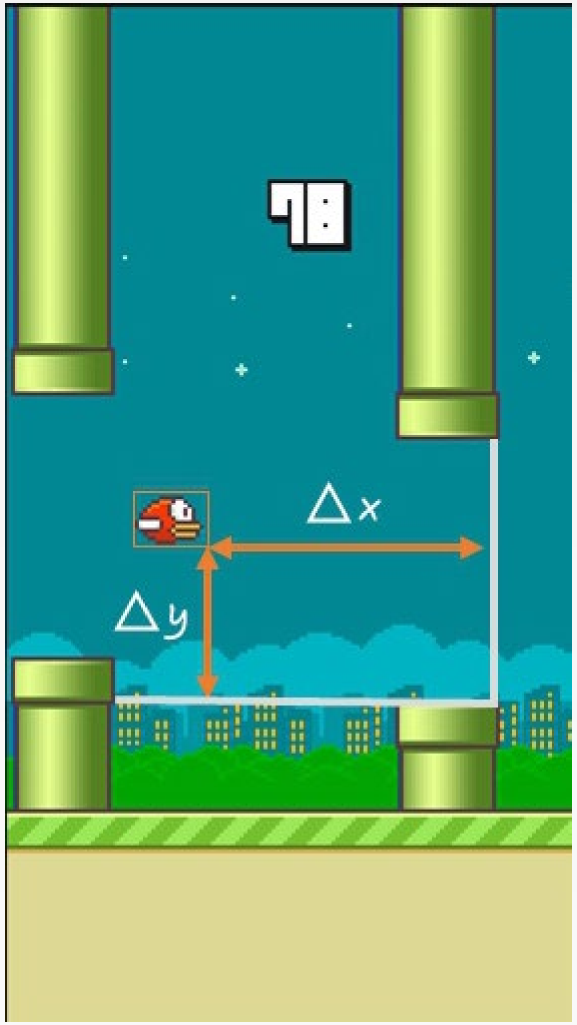
- 1. Supervised Learning (error correction)
  - learning approaches to regression & classification
  - learning from examples, learning from a teacher
- 2. Unsupervised Learning
  - learning approaches to dimensionality reduction, density estimation, recoding data based on some principle, etc.
- 3. Reinforcement Learning
  - learning approaches to sequential decision making
  - learning from a critic, learning from delayed reward

(Partial) List of Applications

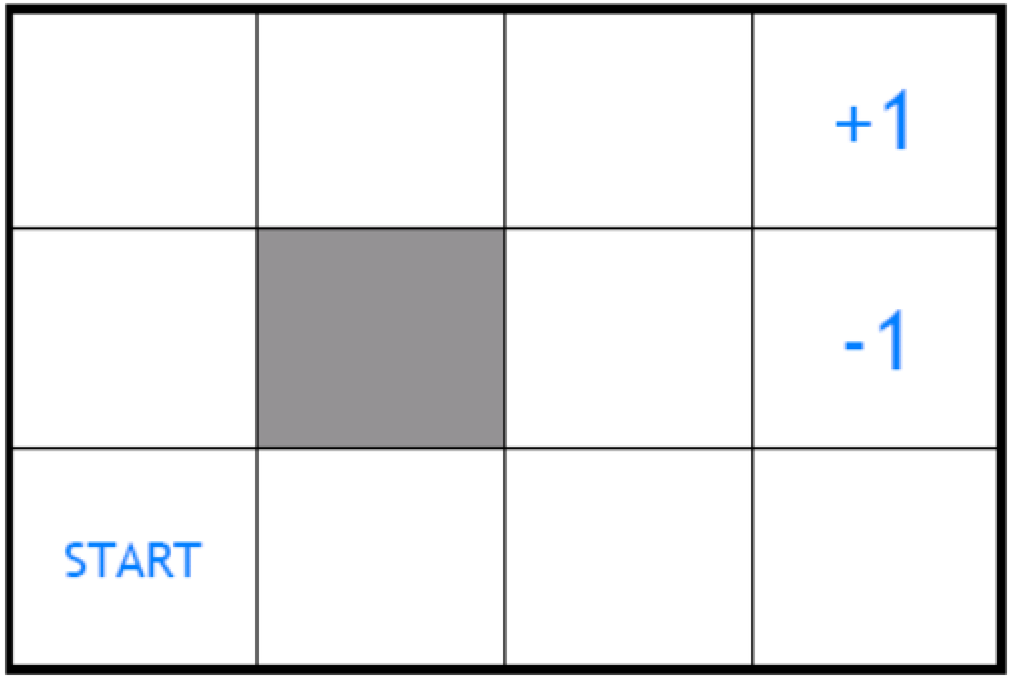
- Robotics:
  - Navigation, walking, juggling, ...
- Games:
  - Backgammon, Chess, ...
- Operation Research:
  - Warehousing, transportation, scheduling, ...
- Control:
  - Helicopters, elevators, admission control in telecom, ...

Flappy Birds





光华管理学院
Guanghua School of Management



actions: UP, DOWN, LEFT, RIGHT

UP

80% move UP
10% move LEFT
10% move RIGHT

- reward +1 at [4,3], -1 at [4,2]
- reward -0.04 for each step

- what's the strategy to achieve max reward?

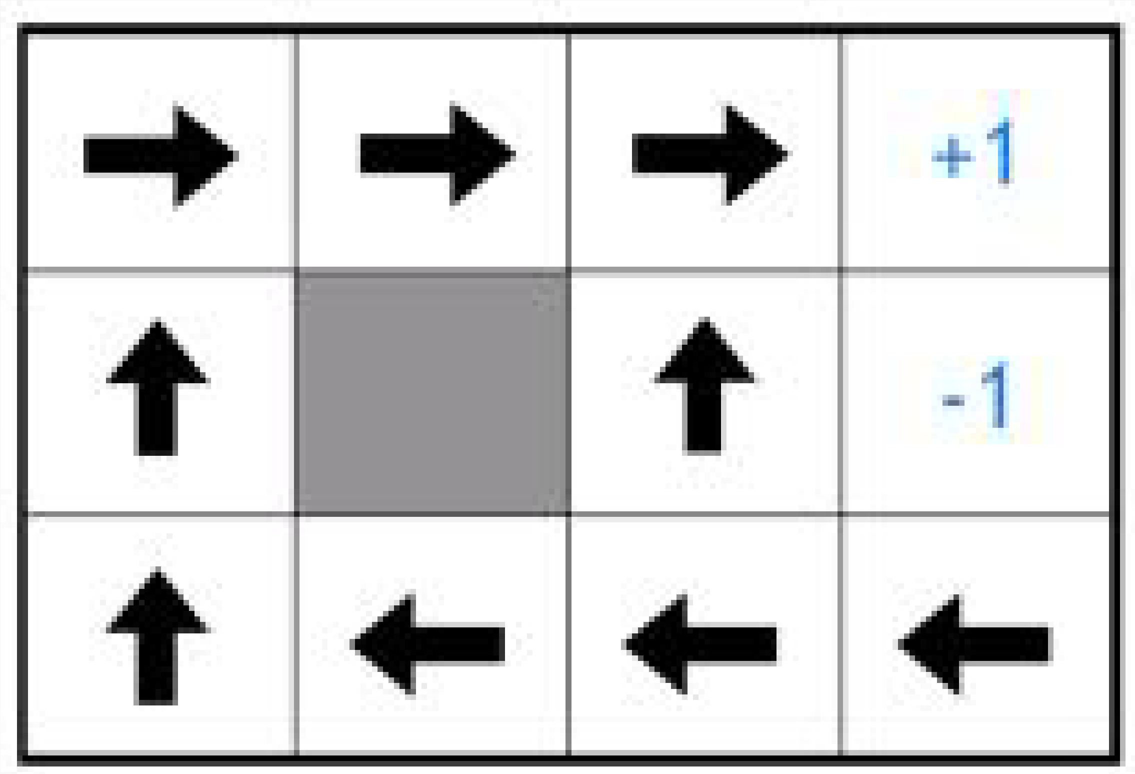
Characteristics of RL

- RL is learning how to map states to actions, so as to maximize a numerical reward over time.
- Unlike other forms of learning, it is a multistage decision-making process (often Markovian).
- Actions may affect not only the immediate reward but also subsequent rewards (Delayed effect)

Elements of RL

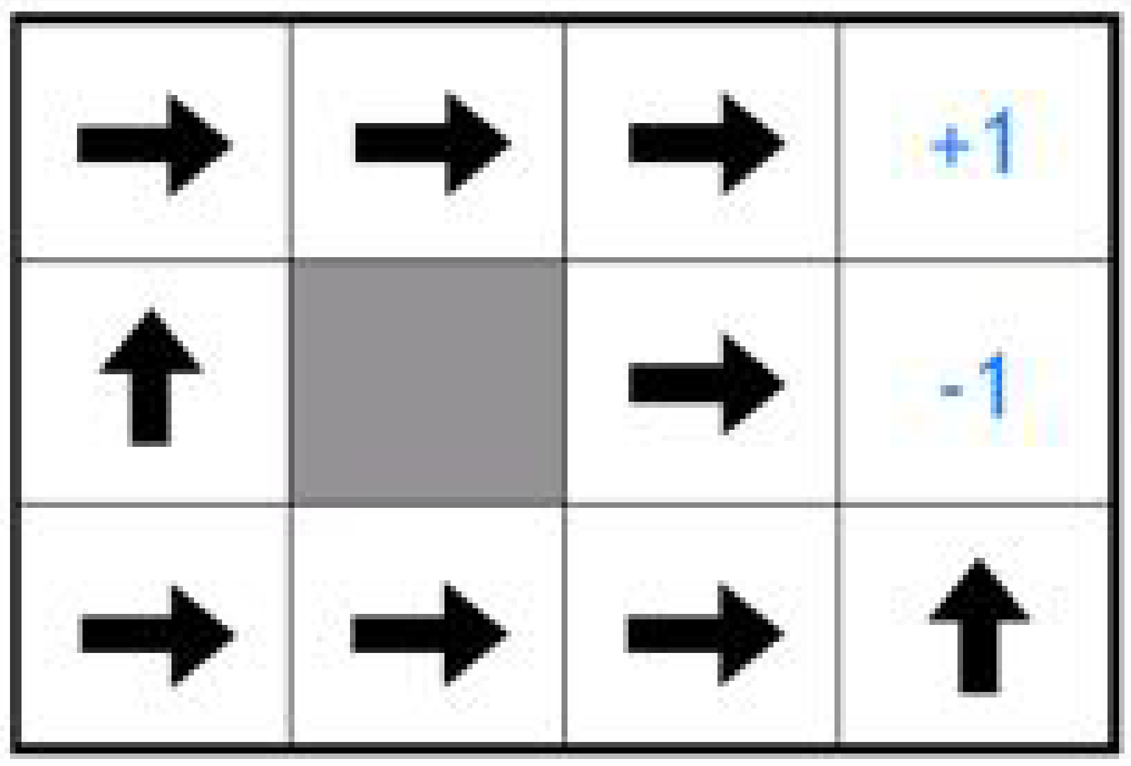
- Environment:
  - Physical world in which the agent operates
- State:
  - Current situation of the agent
- A policy
  - A map from state space to action space.
  - May be stochastic.
- A reward function
  - It maps each state (or, state-action pair) to a real number, called reward.
- A value function
  - Value of a state (or, state-action pair) is the total expected reward, starting from that state (or, state-action pair).

Is this policy optimal?



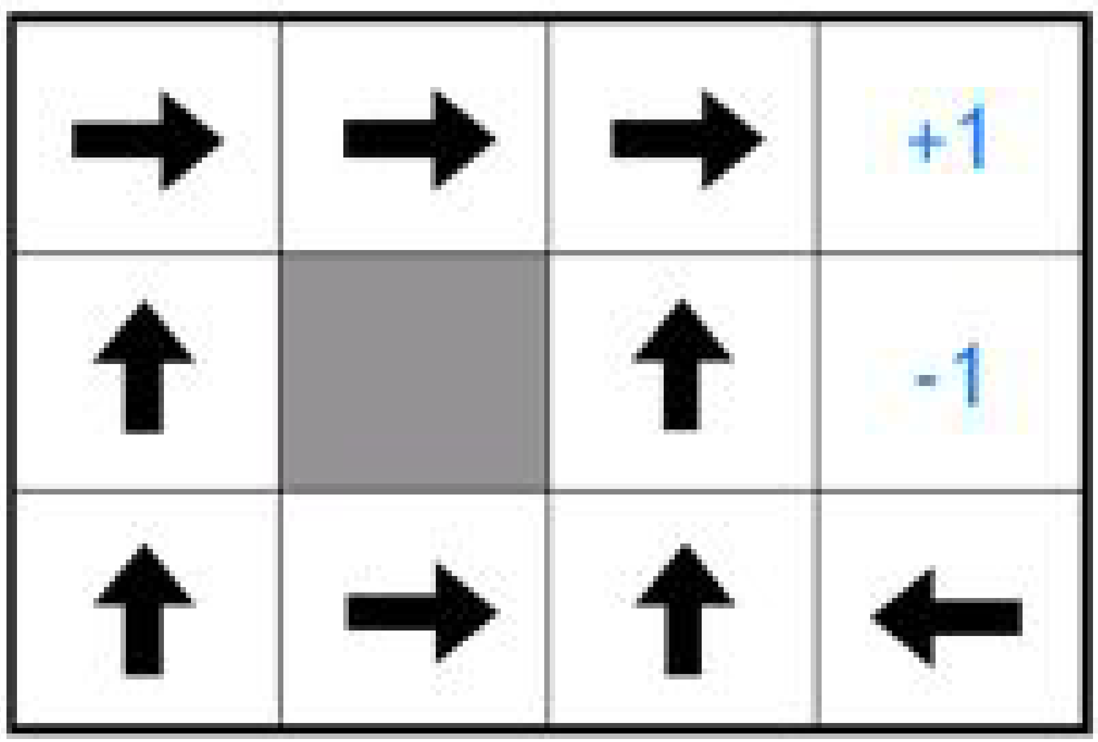
Source: Eric Xing @CMU

Reward for each step -2



Source: Eric Xing @CMU

Reward for each step -0.1



光华管理学院
Guanghua School of Management

Source: Eric Xing @CMU

The Precise Goal

- To find a **policy** that maximizes the **Value function**.
- There are different approaches to achieve this goal in various situations.

- **Markov Decision Process** (value/policy iteration):
  - Require state transition and reward function
- **Q-Learning**:
  - Unknown reward and transition function

Markov Decision Process (MDP)

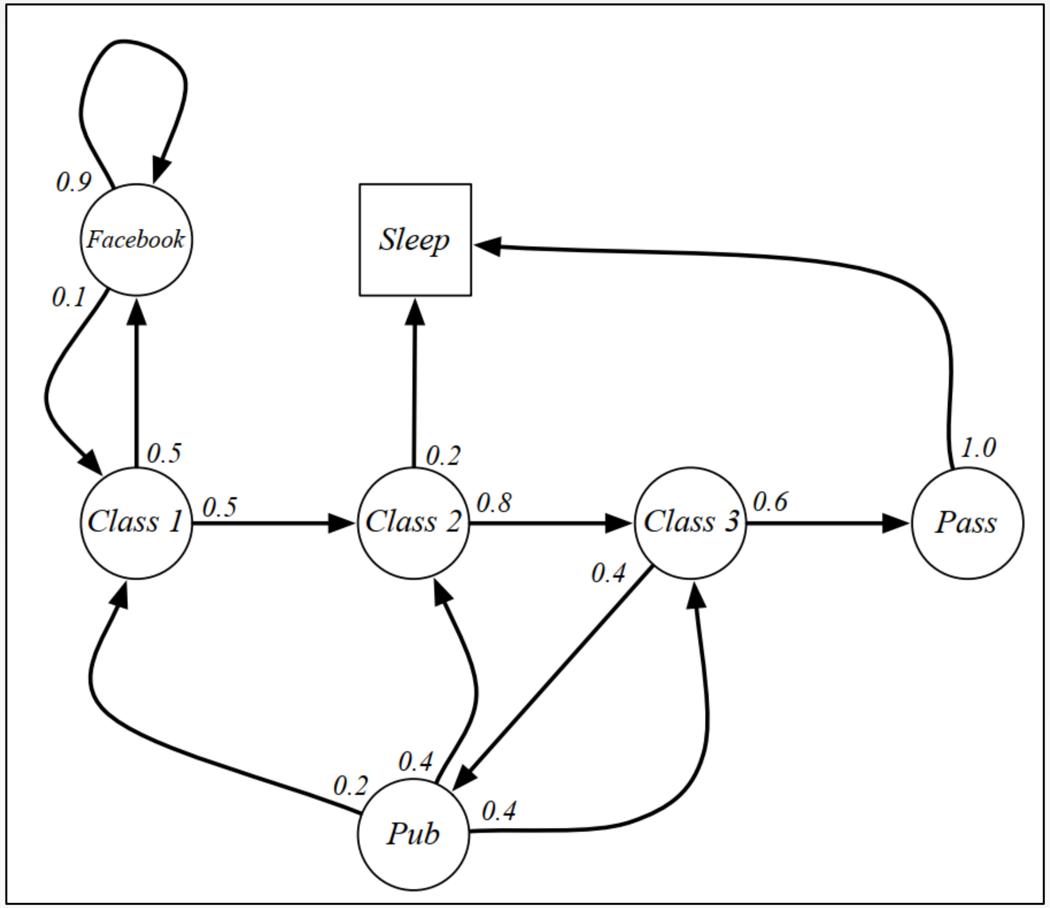
Components

- S: set of possible states
- A: set of possible actions
- $R(s, a)$: reward function
- $P(s'|s, a) = \mathrm{P}(S_{t+1} = s'|S_t = s. A_t = a)$: state transition probabilities

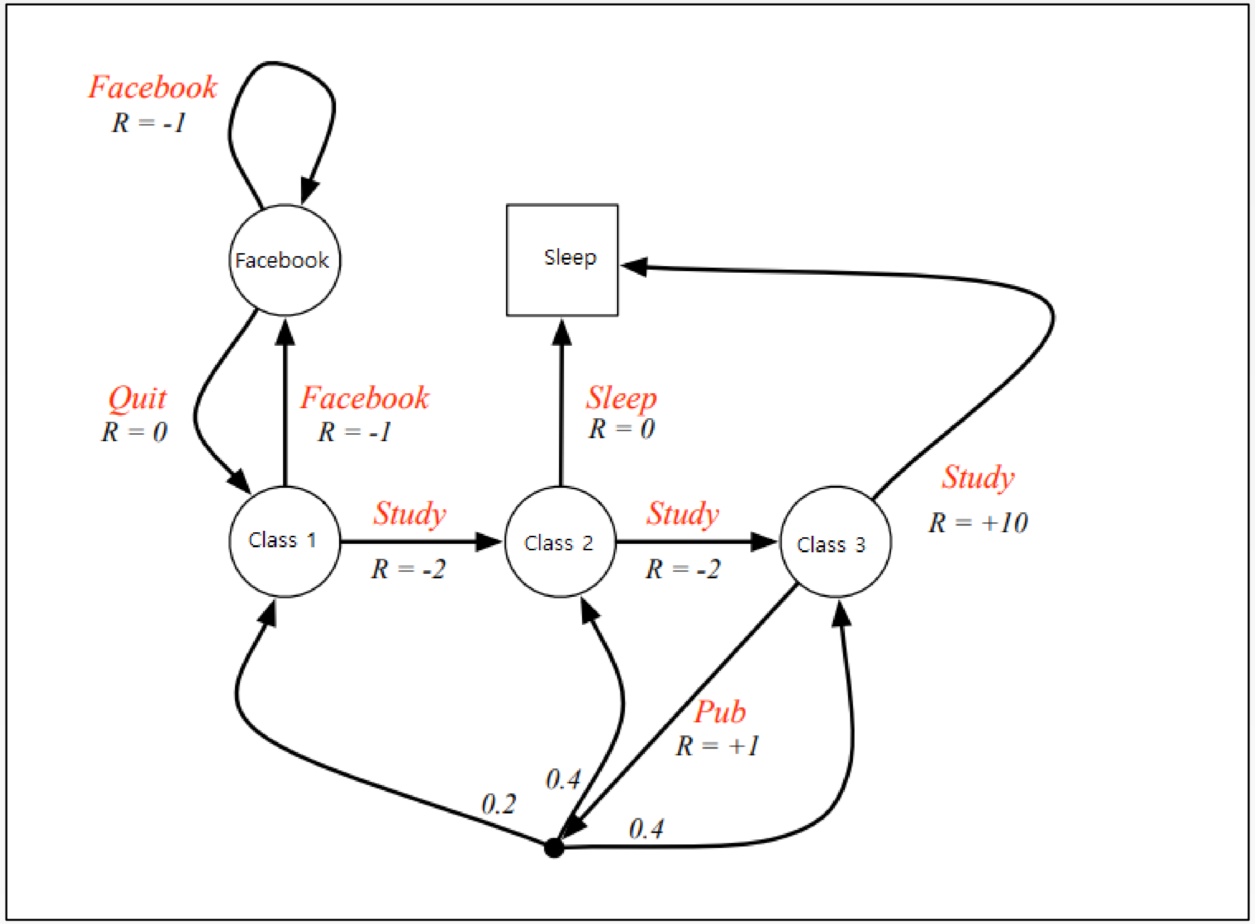
- Markovian Assumption

$$P(S_{t+1}|S_t, A_t, S_{t-1}, A_{t-1}, \dots, S_1, A_1) = P(S_{t+1}|S_t, A_t)$$

Student MDP



0.9
Facebook
0.1
Sleep
0.5
0.2
1.0
Class 1
0.5
Class 2
0.8
Class 3
0.6
Pass
0.4
0.4
0.2
0.4
Pub



Facebook
R = -1
Facebook
Sleep
Quit
R = 0
Facebook
R = -1
Sleep
R = 0
Study
R = +10
Class 1
Study
R = -2
Class 2
Study
R = -2
Class 3
Pub
R = +1
0.4
0.2
0.4

Model for our Data

- Start at state $s_0 \in S$
- At time t, agent observes $s_t \in S$,

  then chooses $a_t \in A$, $a_t = \Pi(s_t)$

  then receives $r_t \in R = R(s_t, a_t)$,

  and changes to $s_{t+1} \in S \sim P(\cdot\,|s_t, a_t)$
- Total payoff is: (γ is the discounted factor, $0 < \gamma < 1$)

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \cdots$$

光华管理学院
Guanghua School of Management

Goal

- Learn a policy $\Pi: S \rightarrow A$ for choosing actions to maximize the expected total payoff: $E[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots] = \sum_{t=0}^{\infty} \gamma^t E[r_t]$
- $E[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots]$: infinite-horizon discounted reward

Fixed Point Iteration for Optimization

$J(\boldsymbol{\theta})$

$$\frac{dJ(\boldsymbol{\theta})}{d\theta_i} = 0 = f(\boldsymbol{\theta})$$

$$0 = f(\boldsymbol{\theta}) \rightarrow \theta_i = g(\boldsymbol{\theta})$$

$$\theta_i^{(t+1)} = g\left(\boldsymbol{\theta}^{(t)}\right)$$

$$J(x) = \frac{x^3}{3} - 1.5x^2 + 2x$$

$$\frac{dJ(x)}{dx} = 0 = x^2 - 3x + 2 = f(x)$$

$$x = \frac{x^2+2}{3} = g(x)$$

$$x \leftarrow \frac{x^2+2}{3}$$

Fixed Point Iteration for Optimization

$$J(x) = \frac{x^3}{3} - 1.5x^2 + 2x$$

$$\frac{dJ(x)}{dx} = 0 = x^2 - 3x + 2 = f(x)$$

$$x = \frac{x^2+2}{3} = g(x)$$

$$x \leftarrow \frac{x^2+2}{3}$$

```python
def f(x):
    return x**2 - 3.*x + 2

def g(x):
    return (x**2 + 2)/3


def solveJ(x0,f,g,n):
    x = x0
    for i in range(n + 1):
        print("i = %2d x = %.4f f(x) = %.4f" %(i,x,f(x)))
        x = g(x)

if __name__ == "__main__":
    x = solveJ(0,f,g,20)
```

Fixed Point Iteration for Optimization

```python
def f(x):
    return x**2 - 3.*x + 2

def g(x):
    return (x**2 + 2)/3


def solveJ(x0,f,g,n):
    x = x0
    for i in range(n + 1):
        print("i = %2d x = %.4f f(x) = %.4f" %(i,x,f(x)))
        x = g(x)

if __name__ == "__main__":
    x = solveJ(0,f,g,20)
```

```
i =  0 x = 0.0000 f(x) = 2.0000
i =  1 x = 0.6667 f(x) = 0.4444
i =  2 x = 0.8148 f(x) = 0.2195
i =  3 x = 0.8880 f(x) = 0.1246
i =  4 x = 0.9295 f(x) = 0.0755
i =  5 x = 0.9547 f(x) = 0.0474
i =  6 x = 0.9705 f(x) = 0.0304
i =  7 x = 0.9806 f(x) = 0.0198
i =  8 x = 0.9872 f(x) = 0.0130
i =  9 x = 0.9915 f(x) = 0.0086
i = 10 x = 0.9944 f(x) = 0.0057
i = 11 x = 0.9963 f(x) = 0.0038
i = 12 x = 0.9975 f(x) = 0.0025
i = 13 x = 0.9983 f(x) = 0.0017
i = 14 x = 0.9989 f(x) = 0.0011
i = 15 x = 0.9993 f(x) = 0.0007
i = 16 x = 0.9995 f(x) = 0.0005
i = 17 x = 0.9997 f(x) = 0.0003
i = 18 x = 0.9998 f(x) = 0.0002
i = 19 x = 0.9999 f(x) = 0.0001
i = 20 x = 0.9999 f(x) = 0.0001
```

Value Function

- Bellman Equation

$$V^{\Pi}(s) = R(s,a) + \gamma \sum_{s_1 \in S} P(s_1|s,a) V^{\Pi}(s_1)$$

Value Function

$R(s,a) = -100$ if falling

$R(s,a) = 100$ if succeed

$R(s,a) = 0$ otherwise

$\gamma = 0.9$

Value Iteration

Initialize $V(s) = 0$ or randomly

while not converged:

for $s \in S$:

$$V(s) = \max_a R(s,a) + \gamma \sum_{s'} p(s'|s,a)V(s')$$

Initialize $Q(s,a) = 0$, $V(s) = 0$ or randomly

while not converged:

for $s \in S$

for $a \in A$

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} p(s'|s,a)V(s')$$

$$V(s) = \max_a Q(s,a)$$

Return $\Pi(s) = argmax_a Q(s,a), \forall s$

Policy Iteration

Algorithm 1 Policy Iteration

procedure PolicyIteration($R(s, a$ … n, $p(\cdot|s, a)$ transition probabilities)

Compute value function for fixed policy is easy

Initialize policy π randomly

System of |S| equations and |S| variables

while not converged **do**

Solve Bellman equations for fixed policy π

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, \pi(s)) V^{\pi}(s'), \ \forall s$$

Improve policy π using new value function

$$\pi(s) = \operatorname*{argmax}_{a} R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) V^{\pi}(s')$$

return π

Greedy policy w.r.t. current value function

Greedy policy might **remain the same** for a particular state if there is no better action

Source: Matt Gormley@CMU

Q-Learning

Exploration and Exploitation

- Exploration: find more information
- Exploitation: maximize the reward by exploiting already known information

K-Armed Bandits Problem

Motivation

Initialize $Q(s,a) = 0$, $V(s) = 0$ or randomly

while not converged:

  for $s \in S$

    for $a \in A$

      $Q(s,a) = R(s,a) + \gamma \sum_{s'} p(s'|s,a)V(s')$

    $V(s) = \max_{a} Q(s,a)$

What if we don't know $R(s,a)$ or $p(s'|s,a)$?

Motivation

- Let $Q^*(s,a)$ be the (true) expected discounted reward for taking a in s
- $V^*(s) = \max_a Q^*(s,a)$
- $Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) \max_{a'} Q^*(s',a')$
- $\Pi^*(s) = argmax_a Q^*(s,a)$

- Idea of Q-Learning: If we can learn Q^*, then we can define Π^* without $R(s,a)$ and $p(s'|s,a)$

Algorithm

1. Initialize $Q(s, a) = 0$
2. Do forever:
   a. Select action a and execute it
   b. Receive reward r from environment
   c. Observe new state s'
   d. Update $Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a')$, When the transition probability is *deterministic*.

      or update $Q(s, a) \leftarrow (1 - \alpha_n)Q(s, a) + \alpha_n \left(r + \gamma \max_{a'} Q(s', a')\right)$, when the transition probability is *stochastic*. $\alpha_n = \frac{1}{1+visit_n(s,a)}$

光华管理学院
Guanghua School of Management

ϵ-greedy Variant

- Choose hyperparameter ϵ
- New step 2a.
  - With prob. $(1 - \epsilon)$: select action $a = \max_{a'} Q(s, a')$
  - With prob. ϵ: select random action $a \in A$

Deep Q-Learning

Question: *What if our state space S is too large to represent with a table?*

Examples:

- s_t = pixels of a video game
- s_t = continuous values of a sensors in a manufacturing robot
- s_t = sensor output from a self-driving car

Answer: Use a parametric function to approximate the table entries