Reinforcement Learning: Introduction

COMP4211

THE DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
計算機科學及工程學系

Learning Paradigms

Supervised learning

- the learner is provided with a set of inputs together with the corresponding desired outputs

Unsupervised learning

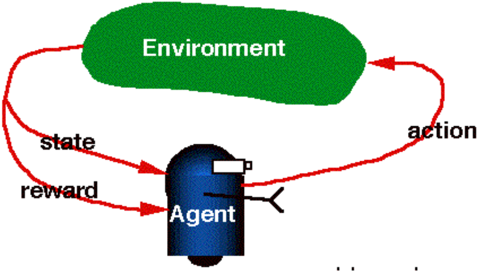
- training examples as input patterns, with no associated output patterns

Reinforcement learning

- Given: input and evaluative output only

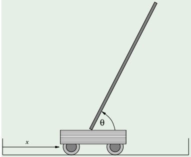
What is Reinforcement Learning (RL)?

Learning from interacting with an environment to achieve a goal



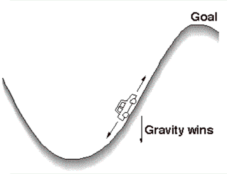
Learning a mapping from states to actions to maximize total reward

Example (Pole balancing)



- goal: balance the pole as long as possible
- states: dynamic states of cart-pole system
- actions: push left, push right
- rewards: always 0 unless pole falls or cart hits end of track, in which case -1

Example (Mountain car)



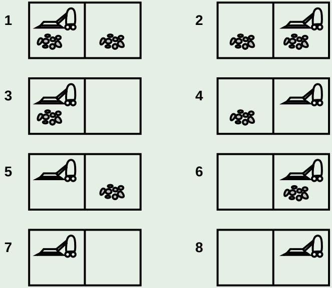
- goal: minimize time to the “goal”
- states: car’s position and velocity
- actions: forward, reverse, none
- rewards: always -1 until car reaches the goal

RL Framework

Given: a finite set of states S and a set of actions A

Example (Vacuum world)

Two locations, each location may or may not contain dirt, and the cleaner may be in one location or the other



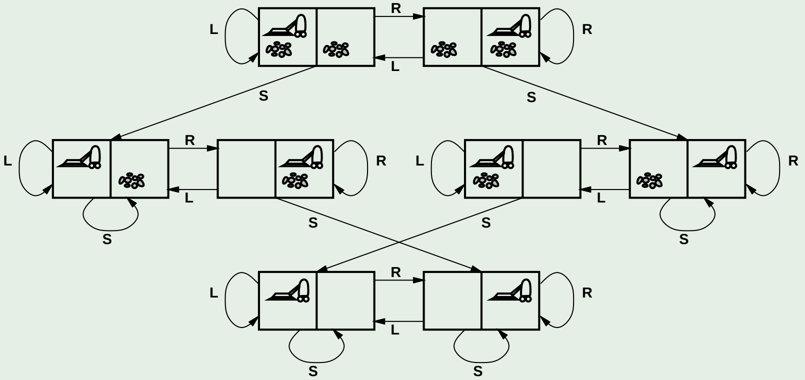
- 8 possible states
- Possible actions: left, right, and suck

RL Framework...

At each discrete time, agent

- observes state $s_t \in S$ and
- chooses action $a_t \in A$

Example (Vacuum world)



RL Framework...

At each discrete time, agent

- observes state $s_t \in S$ and
- chooses action $a_t \in A$

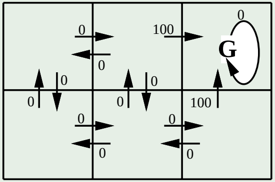
then

- receives immediate reward r_t and
- state changes to s_{t+1}

$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} \ldots$$

Example

States, actions, rewards, state changes



- G: absorbing state

Markov Assumption

$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} \ldots$$

s_{t+1} and r_t depend only on current state and action

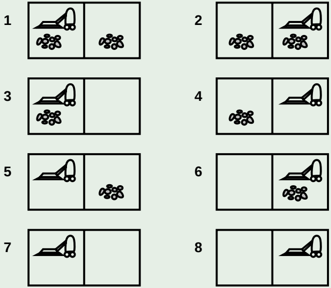
$$s_{t+1} = \delta(s_t, a_t) \quad \text{and} \quad r_t = r(s_t, a_t)$$

- Markov decision process (MDP)

Deterministic vs Non-Deterministic

Deterministic

Example



start in 1

- action "right" goes to 2

Non-deterministic: Actions may have uncertain outcomes

Example

action "suck" can dirty a clean carpet

- start in #4, action "suck" $\rightarrow$ reach $\{2,4\}$

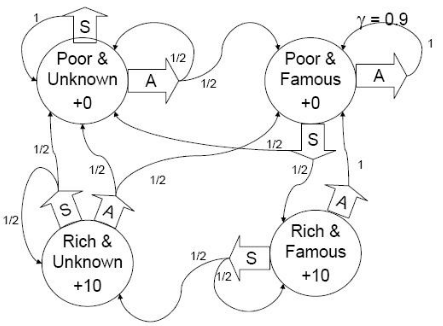
Non-Deterministic

Actions may have uncertain outcomes

- $P(s, s', a)$: probability of transition from s to s' given action a
- $R(s, s', a)$: expected reward on transition s to s' given action a

Example

You run a startup company. In every state you must choose between "Saving money" or "Advertising".

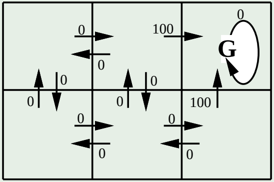


Learn a mapping from states to actions

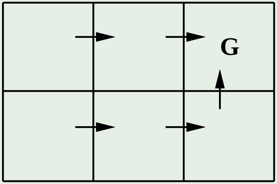
- action policy π: $S \rightarrow A$

Example (deterministic policy)

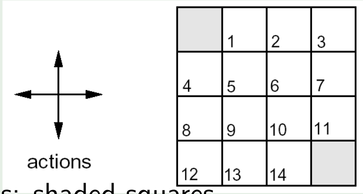
problem



(deterministic) policy

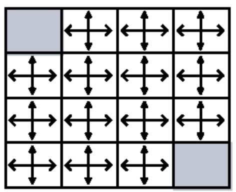


Example (nondeterministic policy)



- terminal states: shaded squares
- reward: -1 until the terminal state is reached
- actions that would take agent off the grid leave state unchanged

Random policy



Discounted Rewards

$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} \ldots$$

A reward (payment) in the future is not worth quite as much as a reward now

Example

Being promised \$10,000 next month is worth only 90% as much as receiving \$10,000 right now.

Assuming payment n months in future is worth only $(0.9)^n$ of payment now, what is the programmer's future discounted sum of rewards?

- (reward now) $+$ $(0.9)\times$(reward in 1 time step) $+$ $(0.9)^2\times$ (reward in 2 time steps) $+$ $(0.9)^3\times$ (reward in 3 time steps) $+$ (infinite sum)

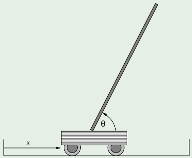
Discounted Return

γ: the discount factor for future rewards

$$\text{discounted return} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots$$

- $0 \leq \gamma < 1$
- shortsighted $0 \leftarrow \gamma \rightarrow 1$ farsighted

Example (Pole balancing)



- reward = -1 upon failure; 0 otherwise
- discounted return $= -\gamma^k$ for k steps before failure
- return is maximized by avoiding failure for as long as possible