Reinforcement Learning (I)

Wenjun Zeng

Reading Group of Ye Lab, UM

Nov. 18, 2018

Contents

I. Introduction

Machine learning:

1) Supervised learning

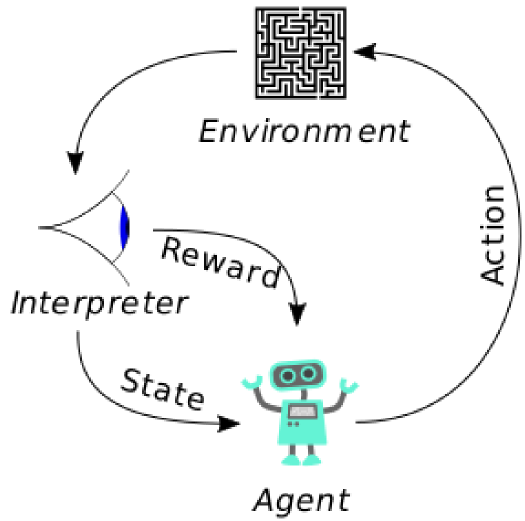
2) Unsupervised learning

3) Reinforcement learning (RL)

RL: learn a suitable behavioral policy from experience by *interacting* with environment.

- Sequential decision-making
- Prediction (supervised/unsupervised learning) $\Rightarrow$ Decision (RL)

R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 2017.

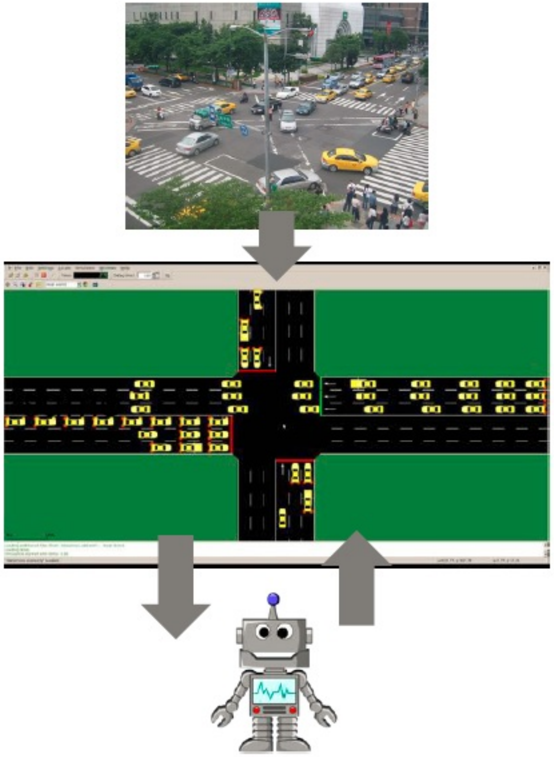


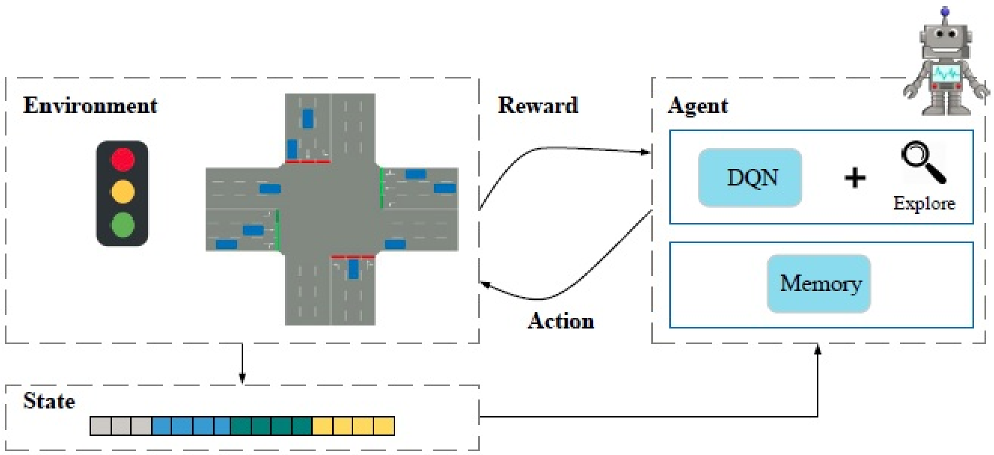
- Agent, Environment
- State
- Action
- Reward

An agent takes actions in an environment, which is interpreted into a reward and a representation of the state, which are fed back into the agent.

II. Applications

2.1 Intelligent Traffic Signal Control

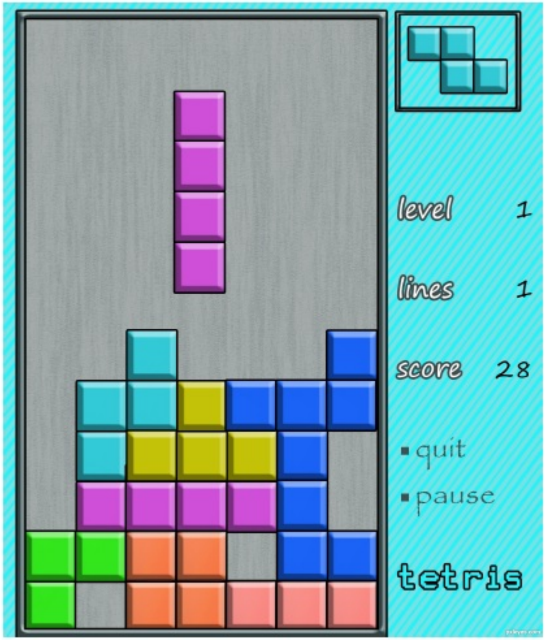




- Agent, Environment
- State: queue length, #cars, waiting time, traffic situations (image), signal
- Action: keep the signal or change the signal
- Reward: queue length, average waiting time, sum of delay

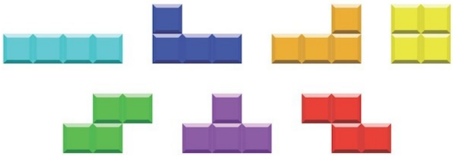
H. Wei, G. Zheng, H.Yao, and Z. Li, "IntelliLight: A reinforcement learning Approach for intelligent traffic light control," KDD'18.

2.2 Tetris



- Height: 12
- Width: 7
- Rotate and move the falling shape
- Gravity related to current height
- Score when eliminating an entire level
- Game over when reaching the ceiling

Decision Process Model of Tetris



- State: current board, current falling tile, prediction of next tile
- Termination State: When a tile reaches ceiling, game is over with no more future reward.
- Action: rotation and shift
- Transitional Reward: If a level is cleared by the current action, score 1; otherwise score 0.
- System Dynamics: Next board is determined by current board and player's placement of current tile. Future tiles are generated *randomly*.

Interesting facts on Tetris

- First released in 1984 by Alexey Pajitnov from the Soviet Union
- Has been proved to be NP-complete
- Game will be over with probability 1.
- For a 12×7 board, the number of possible states $2^{12\times 7} \approx 10^{25}$
- Highest score achieved by human ≈ 1 million
- Highest score achieved by RL algorithm ≈ 35 millions

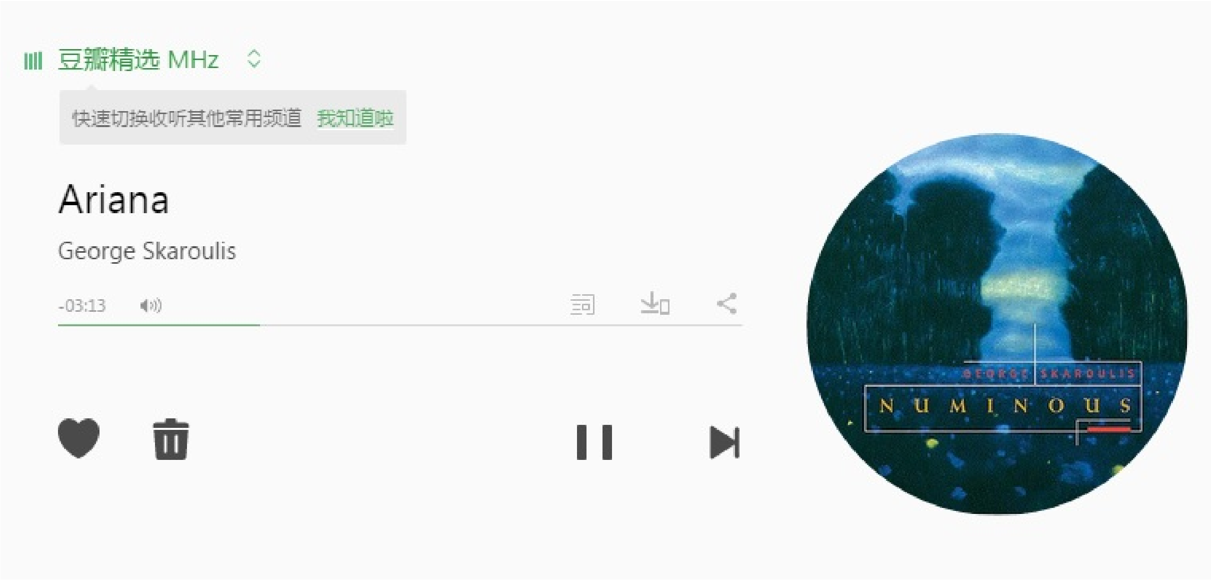
More on Games and Learning

- Tetris domain of 2008 Reinforcement Learning Competition
- 2013 Reinforcement Learning Competition (unmanned helicopter control)
- Since 2006, Annual Computer Poker Competition
- 2013, 2014, 2015 MIT Pokerbots
- Go (Weiqi): AlphaGo and AlphaGo Zero of DeepMind
- Many more...

Many Other Applications

Interactive Recommendation

- Douban.fm music recommend and feedback
- The machine needs to make decisions, not just prediction.

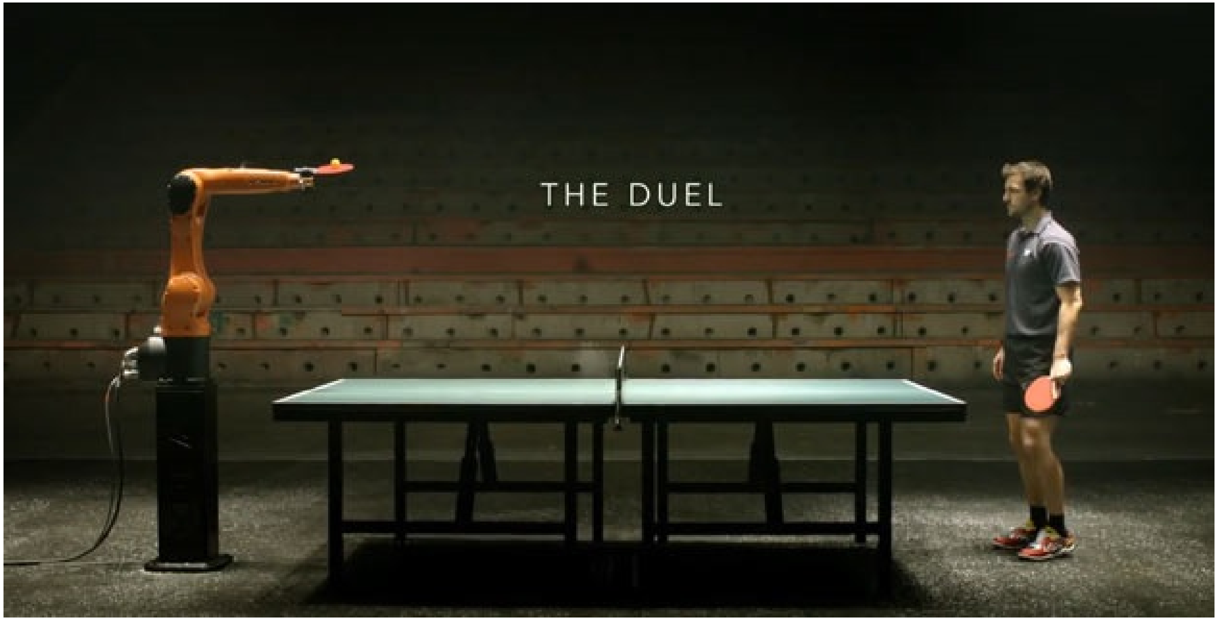


Robotics Control

Stanford Autonomous Helicopter

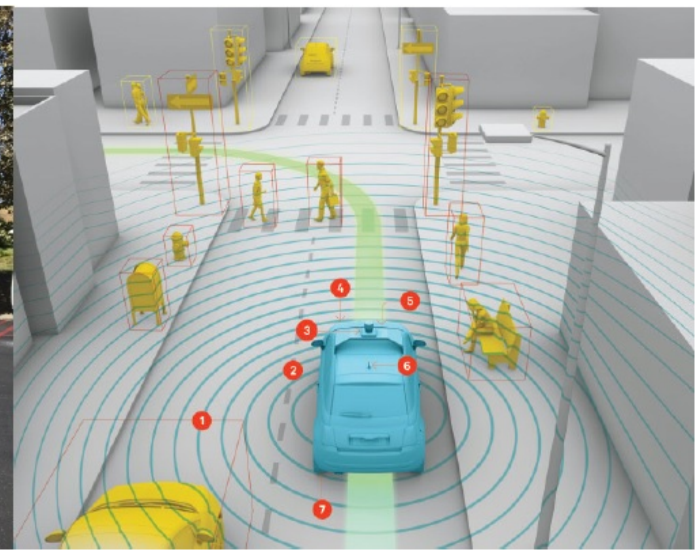
Robotics Control

Ping Pong Robot



Self-Driving Cars

Google Self-Driving Cars



2.3 Summary of Applications of RL

- Intelligent Traffic
- Games: Go, Chess, Tetris, Poker...
- Control of complex systems
  - Robotics
  - Unmanned vehicle/aircraft
  - Planning of power grid
  - Smart home solution
- Business
  - Inventory and supply chain
  - Dynamic pricing with demand learning
- Finance: Option pricing...

III. Dynamic Programming

- ❑ Why discuss dynamic programming (DP) here?
- ❑ Reinforcement Learning = Approximate Dynamic Programming (ADP)

DP: tool for solving certain types of optimization problems (multistage decision).

Basic idea is similar to recursion.

Example: Recursive manner to compute factorial $f(n) = n!$. Due to $n! = n \times (n-1)!$

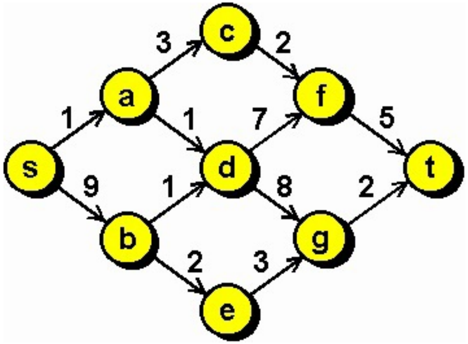
$$f(n) = \begin{cases} 1, & n = 1 \\ nf(n-1), & n > 1 \end{cases}$$

```
function y = factorial(n)
if n == 1
    y = 1;
else
    y = factorial(n);
end
```

Idea: To solve a big problem that is hard, first solve a few smaller but similar ones.

Shortest Path Problem

Find the shortest path from s to t on a graph.



d_{ij}: distance between node i and node j.

Greedy method is simple and it finds the path:

$$s \to a \to d \to f \to t$$

Distance: $1 + 1 + 7 + 5 = 14$.

DP Formulation of Shortest Path Problem

V_i: the shortest distance from i to t.

- We aim at computing V_s.
- It is hard to directly compute V_i in general.

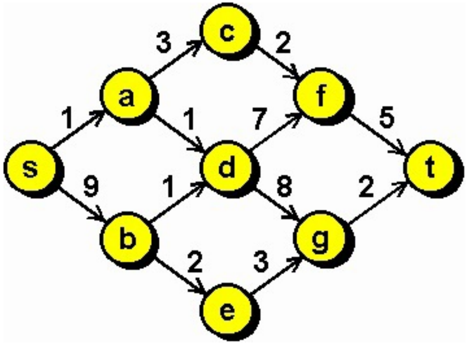
If the first step is to move from i to j, then the shortest distance must be $d_{ij} + V_j$.
To minimize the total distance, choose j to minimize $d_{ij} + V_j$:

$$V_i = \min_j \{d_{ij} + V_j\}, \text{ for all } i$$

which is the recursion formula for the shortest path problem.

- Boundary condition: $V_t = 0$.

Solve the DP



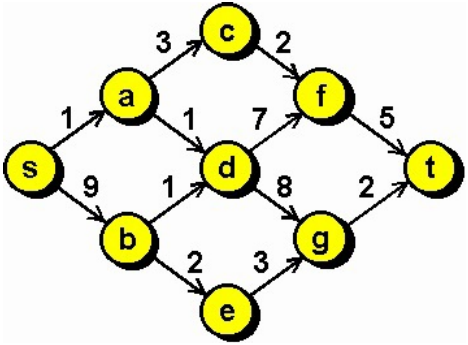
- At the destination, $V_t = 0$.
- At nodes f and g, $V_f = 5$ and $V_g = 2$.
- At nodes c, d, and e.
  - For c and e, there is only one path

    $V_c = d_{cf} + V_f = 2 + 5 = 7; \quad V_e = d_{eg} + V_g = 3 + 2 = 5$
  - For d, we have

    $V_d = \min\{d_{df} + V_f, d_{dg} + V_g\} = \min\{7 + 5, 8 + 2\} = 10$

    Optimal path at d is $d \to g \to t$.

Solve the DP (Continued)



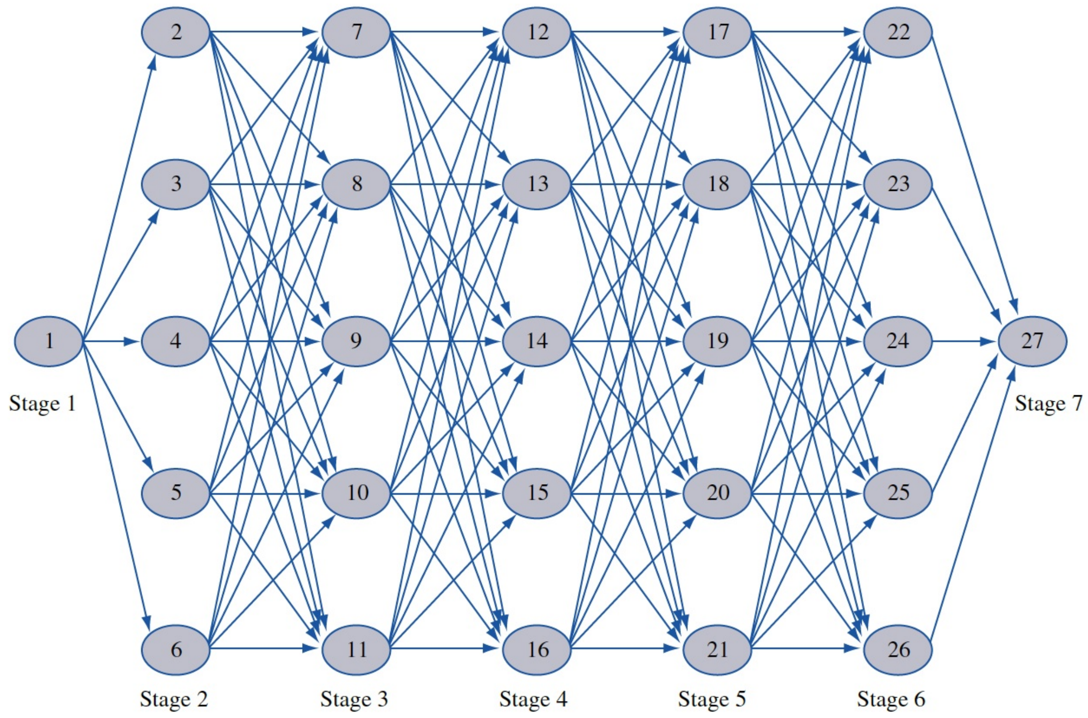
- At nodes a and b:
  - $V_a = \min\{d_{ac} + V_c, d_{ad} + V_d\} = \min\{3 + 7, 1 + 10\} = 10$
    Optimal path at a is $a \to c \to f \to t$.
  - $V_b = \min\{d_{bd} + V_d, d_{be} + V_e\} = \min\{1 + 10, 2 + 5\} = 7$
    Optimal path at b is $b \to e \to g \to t$.
- At node s:
  $V_s = \min\{d_{sa} + V_a, d_{sb} + V_b\} = \min\{1 + 10, 9 + 7\} = 11$
  Optimal path at s is $s \to a \to c \to f \to t$.

This optimal path is different from that one obtained by greedy method.

Computational Efficiency of DP

Compared with exhaustion, the complexity of DP is much lower when the network is large.

Take example of the following network:



Complexity analysis of DP:

- 6th stage: no additions
- 5th, 4th, 3th, and 2nd stage: $5 \times 5 = 25$ additions
- 1st stage: 5 additions

Total complexity of DP: $4 \times 25 + 5 = 105$ additions.

Complexity analysis of exhaustion:

- Number of possible paths: 5^5.
- For each path, it requires 5 additions.

Total complexity of exhaustion: $5 \times 5^5 = 15625 \gg 105$ additions.

Model Abstraction from Shortest Path Problem

In this example, we have those V_i's, the shortest distance to go from i to t.

- This V is called the value function.
- The nodes $s, a, d, \cdots, g, t$ are states.
- The value function is a function of the states.

A state summarizes all the (historical) information that is useful for (future) decisions. Conditioned on the state, the problem becomes Markov.

The recursion formula

$$V_i = \min_j \{d_{ij} + V_j\}, \quad \text{for all } i$$

connects the value function at different states. It is the Bellman equation.

Bellman Equation

Richard E. Bellman

- $i \leftarrow s$, $j \leftarrow s'$
- State transits from s to s' under action a
- Replace cost d_{ij} with reward $R_a(s, s')$

We get the Bellman equation for more general DP

$$V(s) = \max_a \{R_a(s, s') + V(s')\}$$

$R_a(s, s')$: immediate reward received by the agent taking action a.

IV. Markov Decision Processes (MDP)

Andrey Markov

- State space $\mathcal{S}$, $s \in \mathcal{S}$
- Action space $\mathcal{A}$, $a \in \mathcal{A}$
- Policy π, $a = \pi(s)$ for deterministic case or $a \sim \pi(a|s)$ for stochastic case
- Reward $R(s, a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
- Transition Dynamics $\mathcal{T}$: $\mathcal{S} \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}_+$
  For stochastic transition model, $\mathcal{T}(s'|s, a)$ is the probability that the agent reaches state s' after taking action a.

Objective of MDP

Denoting s_t, a_t, and $R_t = R(s_t, a_t)$ as the state, action, and reward of $t = 0, 1, 2, \cdots$, MDP aims at finding policy to maximize the expectation of

$$G_t = \sum_{t=0}^{\infty} \gamma^t R_t$$

with $\gamma \in [0, 1)$ the discount factor.

Why discount factor $\gamma < 1$?

- Future rewards have higher uncertainty, e.g., stock market.
- Future rewards do not provide immediate benefits.
- Devaluation in future.

State value function

$$V_\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \middle| s_0 = s, \pi\right]$$

has recursive relationship

$$V_\pi(s) = \mathbb{E}\left[R(s, a_0)\right] + \gamma \mathbb{E}_{s' \sim \mathcal{T}(s' | s, \pi(s))}\left[V_\pi(s')\right]$$

Q-function (state-action values)

$$Q_\pi(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \middle| s_0 = s, a_0 = a, \pi\right]$$

is the expected return for executing a particular action at a given state.

Policy Evaluation

- Compute the value function of a particular policy.

Defining $[\boldsymbol{V}_\pi]_i = V_\pi(s_i)$, $[\boldsymbol{T}_\pi]_{i,j} = \mathcal{T}(s_j|s_i, \pi(s_i))$, and $[\boldsymbol{R}]_i = R(s_i)$, it follows that

$$\boldsymbol{V}_\pi = \boldsymbol{R} + \gamma \boldsymbol{T}_\pi \boldsymbol{V}_\pi$$

and thus, yields the fixed-point iteration

$$\boldsymbol{V}_\pi \leftarrow \boldsymbol{R} + \gamma \boldsymbol{T}_\pi \boldsymbol{V}_\pi.$$

It realizes a contraction mapping and hence, convergence is guaranteed.

- $\boldsymbol{T}_\pi \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ leads to curse of dimensionality.
- How to solve the curse of dimensionality will be discussed in future talks.

Bellman's Principle of Optimality

A policy is optimal if it achieves the highest value at all states.

$$\exists \pi^* : \; V_{\pi^*}(s) \geq V_\pi(s), \; \forall \pi, s$$

- Whether such a π^* exists?
- Under mild conditions, optimal policies are indeed optimal everywhere [Puterman'94].

Optimal Q-function

$$Q^*(s,a) = \max_\pi Q_\pi(s,a)$$

It is easy to see

$$\pi^* = \arg\max_{a \in \mathcal{A}} Q^*(s,a)$$

Bellman's Optimality Equations

- Bellman equation on value function

$$V^*(s) = \max_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s'|s,a) V^*(s') \right\}$$

- Bellman equation on Q-function

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s'|s,a) \max_{a' \in \mathcal{A}} Q^*(s',a')$$

M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley, 1994.

V. Algorithms

- Model-based
- Model-free

Algorithms for solving MDP

- Value iteration
- Policy iteration
- Q-learning
- Policy gradient

5.1 Value Iteration

Solve the Bellman's optimality equation

$$V^*(s) = \max_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s'|s,a) V^*(s') \right\}$$

by fixed-point iteration:

1. For each state s, initialize $V(s) = 0$.
2. Repeat until convergence

   For each state, update

$$V(s) \leftarrow \max_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s'|s,a) V(s') \right\}$$

Define two operators

$$(T_\pi V)(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s'|s, \pi(s)) V^*(s')$$

and

$$(TV)(s) = \max_{a \in \mathcal{A}} \left\{ R(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s'|s, a) V^*(s') \right\}$$

Bellman equation becomes $T\boldsymbol{V}^* = \boldsymbol{V}^*$.

Theorem: *For all bounded initial $V^0(s)$, we have*

$$V^*(x) = \lim_{k \to \infty} \left(T^k V^0 \right)(s)$$

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

The key point proof of the the theorem is that the operators T and T_π are contraction mappings.

5.2 Policy Iteration

Initialize π randomly.

Repeat until convergence

1. Policy Evaluation:

$$V_\pi(s) \leftarrow R(s) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s'|s, \pi(s)) V_\pi(s')$$

2. Policy Improvement:

   For each state, update

$$\pi(s) \leftarrow \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{T}(s'|s, \pi(s)) V_\pi(s')$$

Understand "policy improvement": one step lookahead using the evaluated value function.

Thank you for your attention!