Foundations of Artificial Intelligence
(人工智慧導論)

Lecture 6: Reinforcement Learning

Shang-Tse Chen & **Hsuan-Tien Lin**

htlin@csie.ntu.edu.tw

Department of Computer Science
& Information Engineering

National Taiwan University
(國立台灣大學資訊工程系)

from supervised to reinforcement

Reinforcement Learning

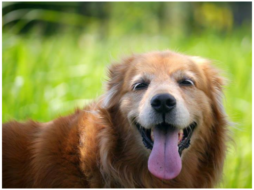
a 'very different' but natural way of learning

Teach Your Dog: Say 'Sit Down'

The dog pees on the ground.

BAD DOG. THAT'S A VERY WRONG ACTION.

- cannot easily show the dog that y_n = sit when $\mathbf{x}_n =$ 'sit down'
- but can 'punish' to say $\tilde{y}_n$ = pee is wrong



reinforcement: learn with **'partial/implicit information'** (often sequentially)

Reinforcement Learning

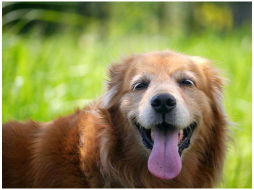
a 'very different' but natural way of learning

Teach Your Dog: Say 'Sit Down'

The dog sits down.

Good Dog. Let me give you some cookies.

- still cannot show y_n = sit when $\mathbf{x}_n$ = 'sit down'
- but can 'reward' to say $\tilde{y}_n$ = sit is good



reinforcement: learn with **'partial/implicit information'** (often sequentially)

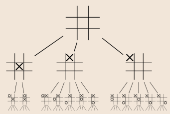
Application 16 (?): Go Playing Agent

AlphaGo

(Public Domain, from Wikipedia; used here for education purpose; all other rights still belong to Google DeepMind)

Non-ML Techniques

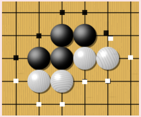
Monte C. Tree Search ≈ **move simulation** in brain



(CC-BY-SA 3.0 by Stannered on Wikipedia)

ML Techniques

Deep Learning ≈ **board analysis** in human brain



(CC-BY-SA 2.0 by Frej Bjon on Wikipedia)

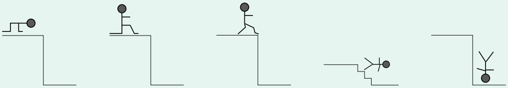
Reinforcement Learn. ≈ **(self)-practice** in human training

(Public Domain, from Wikipedia)

good AI: important to use the **right** techniques—ML **& others, including human**

Supervised versus Reinforcement

- supervised relies on **teacher** with (almost) correct examples
- not typically how we learned to walk with **trial-and-error** (or trial-and-reward)

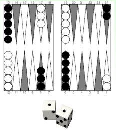


(illustrative figures courtesy of Prof. Malik Magdon-Ismail)

- supervised learning ≈ learning to **decide**
  reinforcement learning ≈ learning to **control**
- two essences: **try** $\tilde{y}_n$ (often called **action** a_n) & **graded** with goodness (often called **reward**) r_n

goal: learn **best** decision from history data of **trial-and-reward**

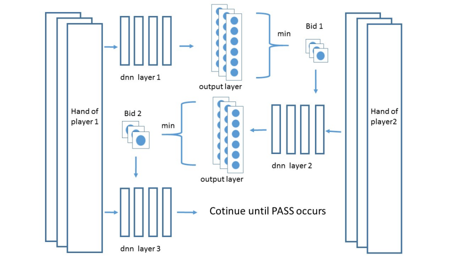
Reinforcement Learning to Play Backgammon

data $\longrightarrow$ ML $\longrightarrow$ 

- **target** function 'f': $\mathbf{x}$ (board state, also noted s) $\rightarrow$ optimal action a
- data: (state s_1, action a_1, reward $r_1 = ?$),
  $(s_2, a_2, r_2 = ?)$, ..., $(s_T, a_T, r_T = 1)$
- **hypothesis** g (often also called agent **policy** π) that hopefully $\approx f$
- issues:
  - sparse (possibly delayed) rewards
  - credit assignment
  - data collection (next states) depends on chosen actions a_t
  - sometimes multi-agent (competition/collaboration)

if games (one strategy vs another) **simulated** properly, agents can be strong enough

Application 17: Bridge Bidding



(figure from Yeh et al., Automatic Bridge Bidding using Deep Reinforcement Learning, 2018)

non-‘standard’ RL task: partial information, collaborative

RL: rich opportunities for different kinds of control/interactive tasks

Application 18: Computer-Assisted Diagnosis

data → ML → AI

By DataBase Center for Life Science;
licensed under CC BY 4.0 via Wikimedia Commons

for computer-assisted diagnosis, with RL

- state s_t: queried patient symptoms
- action a_t: query another symptom or make a diagnosis
- reward r_t: whether the diagnosis is correct
- learned policy π: dialogue system that **efficiently identifies disease of patient**

my student's earlier work
as intern @ HTC DeepQ

Efficient Diagnosis: More than Plain RL

REFUEL: Exploring Sparse Features in Deep Reinforcement Learning for Fast Disease Diagnosis

Yu-Shao Peng
HTC Research & Healthcare
ys_peng@htc.com

Kai-Fu Tang
HTC Research & Healthcare
kevin_tang@htc.com

Hsuan-Tien Lin
Department of CSIE,
National Taiwan University
htlin@csie.ntu.edu.tw

Edward Y. Chang
HTC Research & Healthcare
edward_chang@htc.com

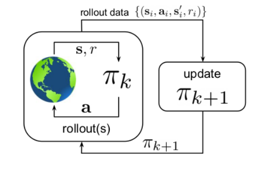
some symptoms
$\Longrightarrow$ all symptoms
idea: **autoencoder-like error term for reconstruction**

positive symptoms
$\Longrightarrow$ efficient dialogue
idea: **auxiliary reward for positive symptom**

successful 'application intelligence' sometimes need **beyond-textbook** ideas

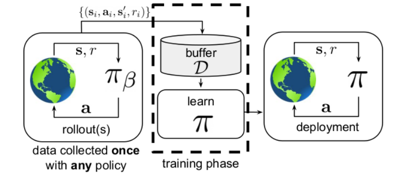
Online RL versus Offline RL

online RL



- textbook scenario
- 'easier' to analyze mathematically

offline (batch) RL



- practical scenario
- data quality depends on 'collector' policy

truely 'online learning' is **luxurious in practice**

Online + Batch for Real-World Applications

model re-trained with historical daily batch data, incrementally or completely

→ **online** prediction for each request

↓ **batch** labeled data collected daily

← model re-trained

purely online

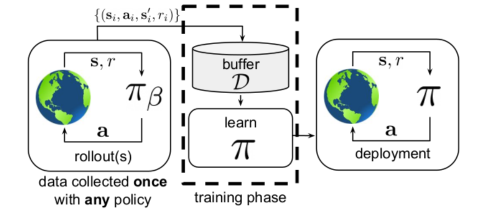
- incremental update costly online
- delayed labels hard to handle properly

purely batch

- cannot capture drifts/trends well
- complete re-training possibly costly

real-world ML system
different from **textbook settings**

From Offline RL to Imitation Learning



offline RL

- data collected by 'any policy'
- data often more noisy ($\times$)
- data often with wider coverage ($\bigcirc$)

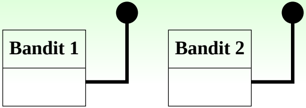
imitation learning

- data collected by 'expert policy' (demonstration)
- data often more clean ($\bigcirc$)
- data often with more biased coverage ($\times$)

imitation learning can be **more data efficient** in building proof-of-concept system

stateless reinforcement learning: bandit learning

A Simple Trial-and-Reward Environment



(illustrative figures courtesy of Prof. Malik Magdon-Ismail)

$\mathcal{D} = (a_1, r_1), (a_2, r_2), \ldots, (a_T, r_T)$

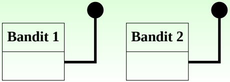
- action $a_t \in \{1, 2\}$
- $r_t = 1$ with probability p_{a_t}, and 0 otherwise
- policy π specifies a_t for $t = 1, 2, \ldots, T$
- evaluation: expected γ-discounted reward (with $\gamma < 1$) after ∞ rounds

$$V(\pi) = \mathbb{E}\left((1-\gamma)\sum_{t=1}^{\infty} \gamma^{t-1} r_t\right)$$

as **early rewards** more preferred

what are the possible policies?

Some Naive Policies



Constant π_1

$\pi_1(t) = 1$: expected reward

$$\mathbb{E}\left((1-\gamma)\sum_{t=1}^{\infty}\gamma^{t-1}r_t\right) = p_1$$

Constant π_2

$\pi_2(t) = 2$: expected reward

$$\mathbb{E}\left((1-\gamma)\sum_{t=1}^{\infty}\gamma^{t-1}r_t\right) = p_2.$$

Greedy π_{grd}

$$\pi_{\text{grd}}(t) = \max_{a\in\{1,2\}} \hat{p}_a(t)$$

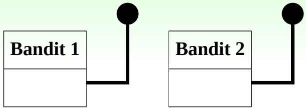
with $\hat{p}_a(t)$ estimating p_a

Random π_{rnd}

$$\pi_{\text{rnd}}(t) = \text{(random fair coin flip)}$$

key question: assume $p_2 > p_1$ without loss of generality, can we be **'nearly as good as'** π_2?

Blindspot in Pure Exploitation



Greedy π_{grd}

$$\pi_{\text{grd}}(t) = \max_{a \in \{1,2\}} \hat{p}_a(t)$$

with $\hat{p}_a(t)$ estimating p_a.

doomed when $(a_1 = 1, r_1 = 1)$, $(a_2 = 2, r_2 = 0)$

Exploitation and Exploration

Greedy π_{grd} (exploitation)

$$\pi_{\text{grd}}(t) = \max_{a \in \{1,2\}} \hat{p}_a(t)$$

with $\hat{p}_a(t)$ estimating p_a

Random π_{rnd} (exploration)

$$\pi_{\text{rnd}}(t) = \text{(random fair coin flip)}$$

Greedy + Random

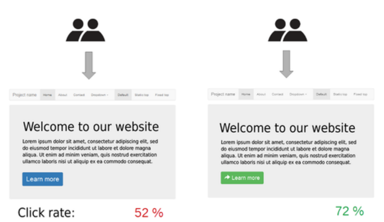
- run Greedy with prob. $(1 - \epsilon)$
  run Random with prob. ϵ
- called ϵ-greedy

Greedily-Random

- randomly execute a with prob. $\propto$ converted $\hat{p}_a(t)$
- e.g. Exponential-weight for Exploration & Exploitation

theoretically: you would not **regret**
doing **a little** exploration

Application 19: Smart A/B Testing



(CC-BY-SA 4.0 by Maxime Lorant on Wikimedia)

Greedy + Random

start with some 'small' traffic for option B

Greedily-Random

tune A/B percentage by performance

smart A-B testing allows **continuous improvement** in evolving AI products

Exploration-Exploitation Tradeoff

Principle

start with exploration, gradually switching to exploitation, always doing enough but not too much, exploration

More Exploitation

- more stable
- reacts more slowly

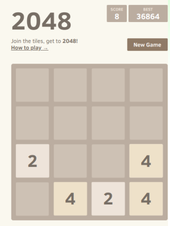
More Exploration

- less stable
- reacts faster

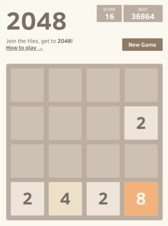
often needs **other criteria**
to determine the right tradeoff

stateful reinforcement learning

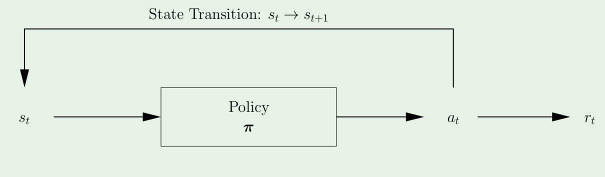
Dynamic Environment

$s_t =$ 

$a_t =$ down, $r_t = 8$

$s_{t+1} =$ 

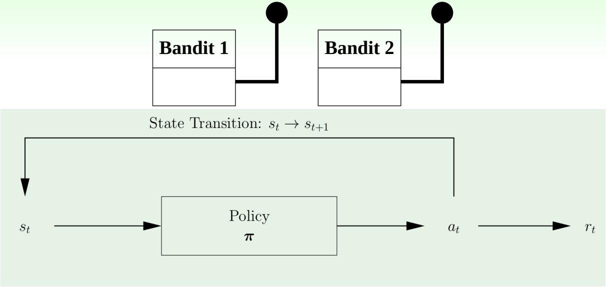
(https://play2048.co/)



(illustrative figures courtesy of Prof. Malik Magdon-Ismail)

passive reactive environment: reacts dynamically by **fixed but unknown** way

Modeling Dynamic Environment with Different States



different states $\Rightarrow$ different (probabilities of) rewards, e.g.

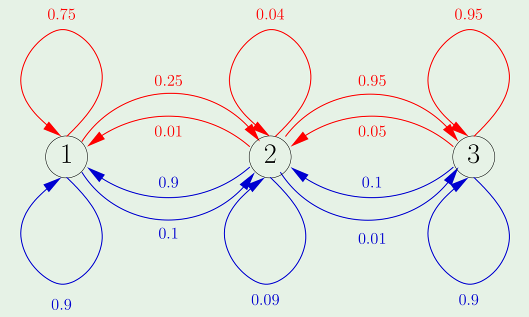
s	1	2	3
p_1	0.9	0.9	0.1
p_2	0.5	0.5	0.1

want: policy π such that $\pi(s) \approx$ optimal action

Modeling Dynamic Environment with State Transitions

T_1	s_{t+1}		
s_t	1	2	3
1	0.75	0.25	0
2	0.01	0.04	0.95
3	0	0.05	0.95

T_2	s_{t+1}		
s_t	1	2	3
1	0.90	0.10	0
2	0.90	0.09	0.01
3	0	0.10	0.90



(illustrative figures courtesy of Prof. Malik Magdon-Ismail)

s	1	2	3
p_1	0.9	0.9	0.1
p_2	0.5	0.5	0.1

Markov Decision Process (MDP):
'automata' with probabilistic transitions

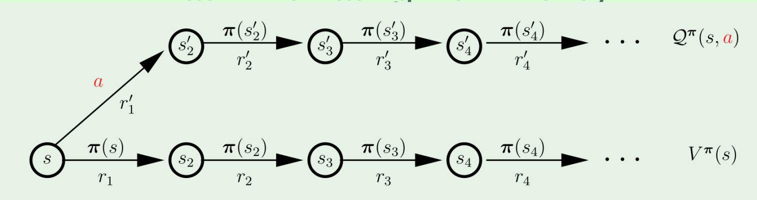
Value Function of Policy for Known MDP

$$
\begin{array}{cccccccccc}
s & \xrightarrow{\pi} & s' & \xrightarrow{\pi} & s'' & \xrightarrow{\pi} & s''' & \xrightarrow{\pi} & \ldots \\
 & \downarrow & & \downarrow & & \downarrow & & \downarrow & \\
\text{rewards} & r & & r' & & r'' & & r''' & \\
 & \cdot & & \cdot & & \cdot & & \cdot & \\
\text{discounts} & (1-\gamma) & & (1-\gamma)\gamma & & (1-\gamma)\gamma^2 & & (1-\gamma)\gamma^3 &
\end{array}
$$

$$V^\pi(s) = (1-\gamma)\sum_{t=1}^{\infty} \gamma^{t-1}\mathbb{E}[r_t | s, \pi]$$

value function $V^\pi(s)$: performance of π
when starting from s in MDP

What If Deviating from Policy?



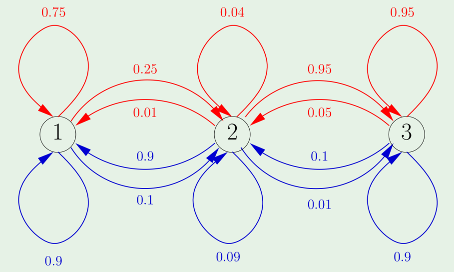
(illustrative figures courtesy of Prof. Malik Magdon-Ismail)

$Q^\pi(s, a)$: rewards from s when taking a first, and follow π later

$$
\begin{aligned}
Q^\pi(s, a) &= (1-\gamma)r(s, a) + \gamma \sum_{s'} \mathrm{T}_a(s, s') V^\pi(s') \\
&= (1-\gamma)r(s, a) + \gamma \sum_{s'} \mathrm{T}_a(s, s') Q^\pi(s', \pi(s'))
\end{aligned}
$$

best policy π^* must satisfy
$\pi^*(s) = \text{argmax}_a Q^{\pi^*}(s, a)$

Policy Iteration Algorithm for Calculating π^*



s	1	2	3
p_1	0.9	0.9	0.1
p_2	0.5	0.5	0.1

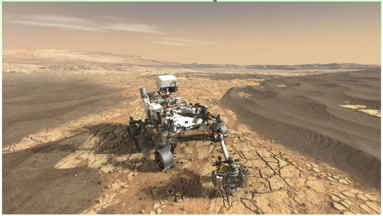
given **known** MDP, and any initial policy π, repeat

- (re-)compute V^π and $Q^\pi(s, a)$ [dynamic programming helps!]
- for every state s with $\pi(s) \neq \text{argmax}_a Q^\pi(s, a)$, change $\pi(s)$ to $\text{argmax}_a Q^\pi(s, a)$

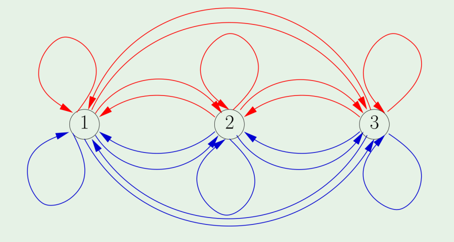
until no change of π

feasible if MDP is **known**
and (s, a) somewhat 'enumerable' (Q-table)

Unknown MDP: Explore & Estimate



(Mars exploration with Perseverance Rover, free image from NASA)



s	1	2	3
p_1	?	?	?
p_2	?	?	?

after enough exploratory actions, T_a and $r(s, a)$ can be estimated to calculate Q^π for any π.

Q-Learning: Updating $\hat{Q}$ Directly on the Fly

$$
\begin{aligned}
\text{ideal: } Q^{\pi^*}(s, a) &= (1-\gamma) r(s, a) + \gamma \sum_{s'} \mathrm{T}_a(s, s') V^{\pi^*}(s') \\
&= (1-\gamma) r(s, a) + \gamma \sum_{s'} \mathrm{T}_a(s, s') \max_{a'} Q^{\pi^*}(s', a')
\end{aligned}
$$

$$
\text{one-example: } \hat{Q}(s_t, a_t) = r_t + \gamma \max_{a'} \hat{Q}(s_{t+1}, a')
$$

deep Q-learning (with ‘any’ exploration):

- represent $\hat{Q}$ with NNet
- many techniques to stabilize
- $\pi(s) = \operatorname{argmax}_a \hat{Q}(s, a)$

Application 20: Data Center Cooling

(from Deepmind)

deep reinforcement learning: new opportunity to **control complicated systems**

Summary

Lecture 6: Reinforcement Learning

- from supervised to reinforcement
  trial-and-reward, instead of duck-fed with examples
- stateless reinforcement learning: bandit learning
  explore possible actions and exploit better-reward ones
- stateful reinforcement learning
  exploration + Q-learning + best action from Q