Single-Agent Reinforcement Learning

Xihan Li
Department of Computer Science,
University College London
xihan.li@cs.ucl.ac.uk
https://snowkylin.github.io

Feb 2025

Contents

1. Basic Game Theory and Nash Equilibrium
2. Potential Games and Best Respond Dynamics
3. Repeated Games and the Theory of Cooperation
4. Adversarial and Minimax Games
5. Bi-level Games and General-Sum Games and L-H Algorithms
6. **Single-agent Reinforcement Learning and MDP**
7. Learning Stochastic Games
8. Non-regret Learning and Correlated Equilibrium
9. Counterfactual Regret Minimisation
10. Learning with a Population of Agents

We are here

Outline

Cornerstones

- Markov Decision Process (MDP)
- Bellman Equation

Value-based RL - Model-based Learning

Dynamic Programming

- Policy Iteration
- Value Iteration

Value-based RL – Model-Free Learning

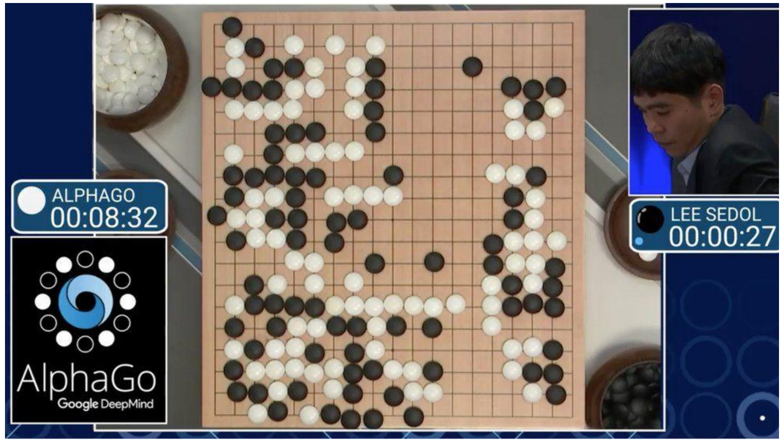
Temporal-Difference Learning

- SARSA
- Q-Learning

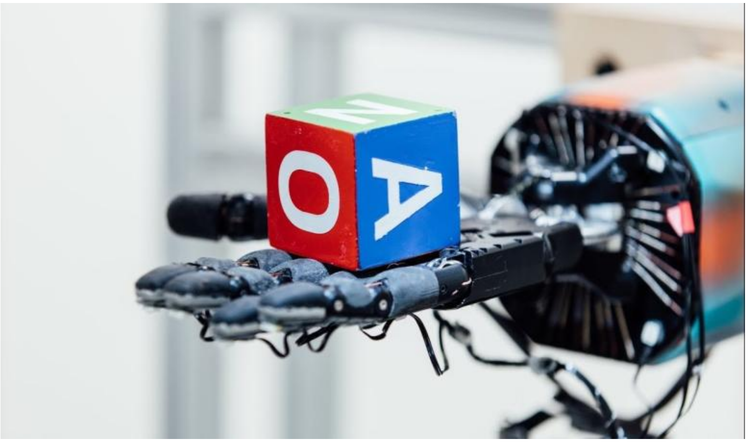
Policy-based RL

- Policy Gradient

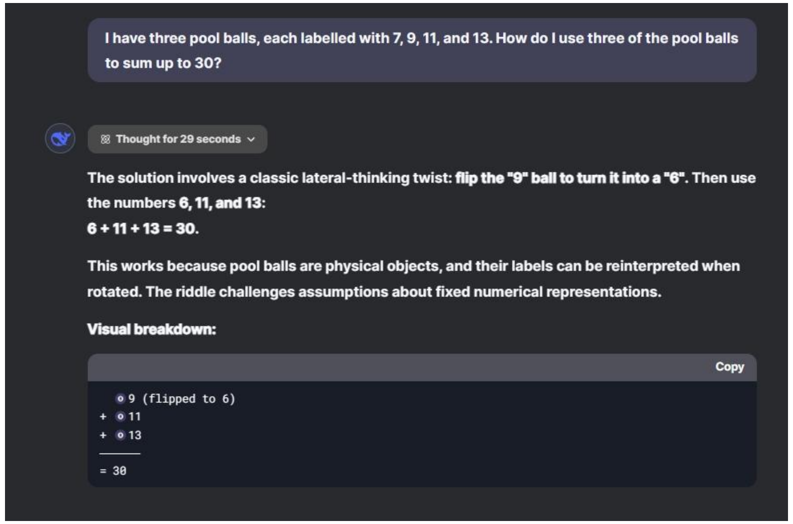
Reinforcement Learning (RL)



AlphaGo in playing Go (MCTS+RL)

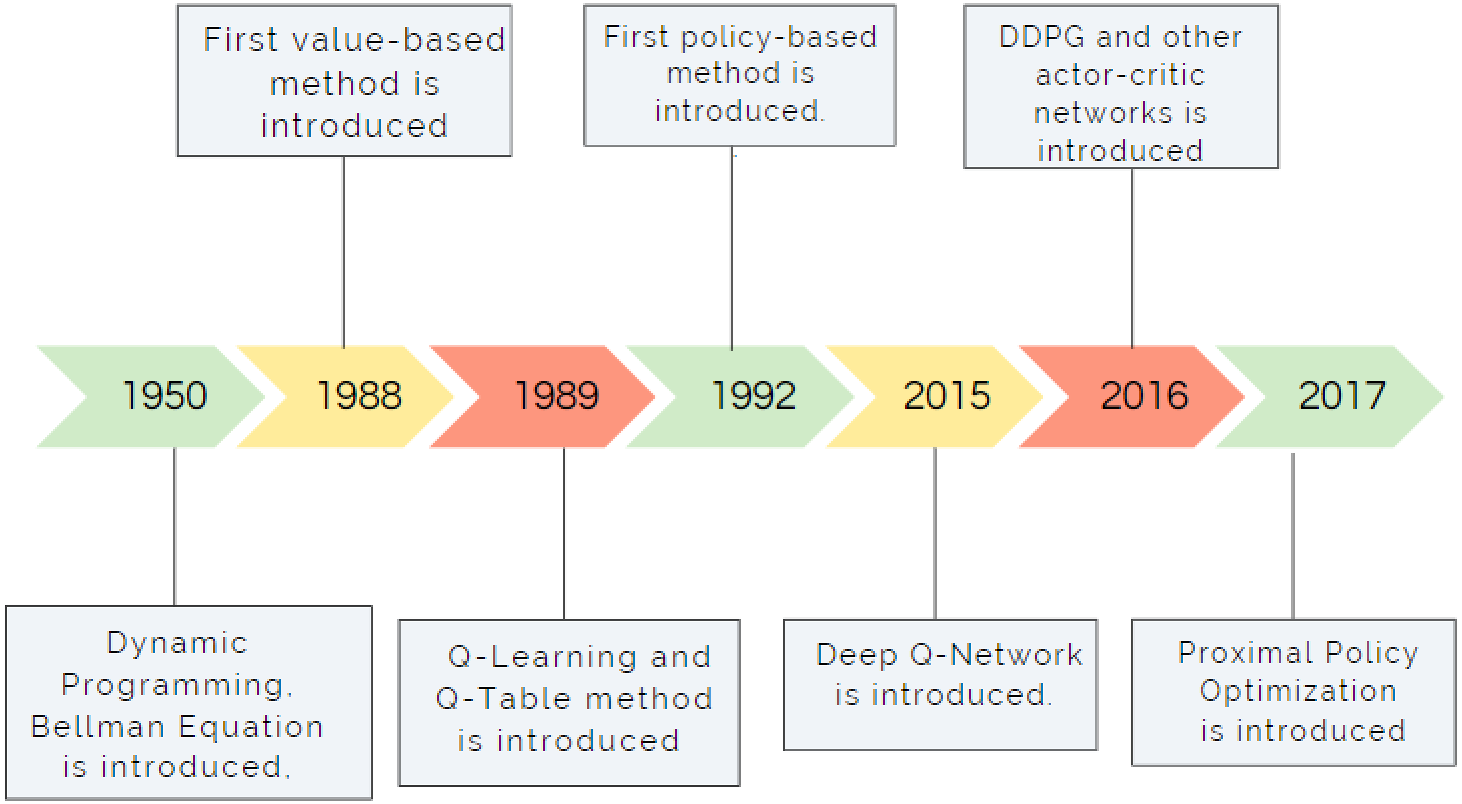


OpenAI's Dactyl in solving Rubik's Cube with a robot hand (Robotic+RL)



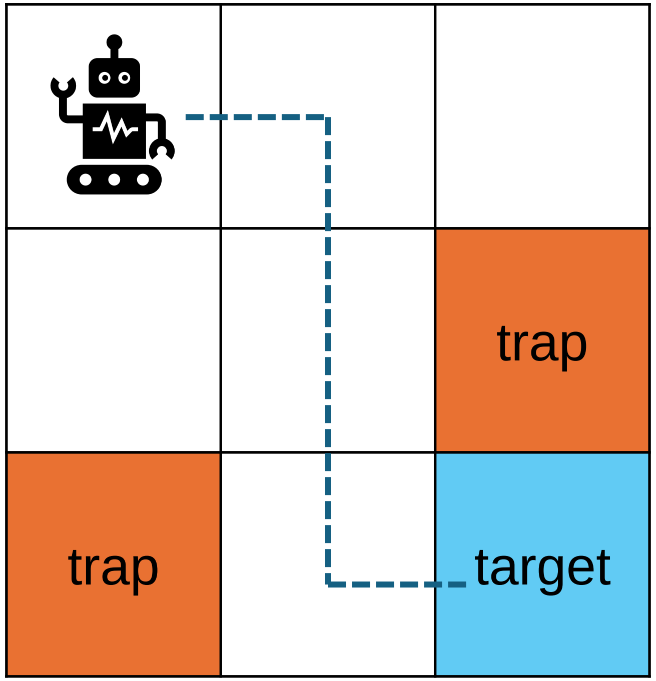
DeepSeek R1 in solving math problems (LLM+RL)

History of Reinforcement Learning



https://medium.com/@sanghaviharsh666/navigating-the-evolution-of-reinforcement-learning-a-historical-perspective-6fbbaa010351

Markov Decision Process: A Deterministic Example

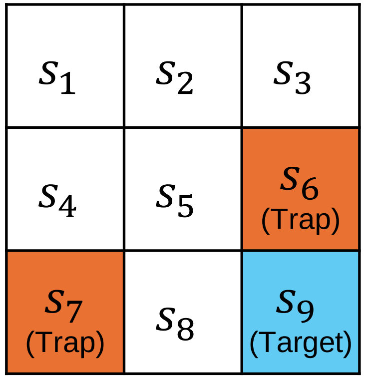


- A robot moves in a grid world
- The robot (called agent) can move across adjacent cells in the grid.
- Target: find a “good” policy that enables it to reach the target cell, when starting from any initial cell. The agent should reach the target without
  - entering any trap cells
  - taking unnecessary detours
  - colliding with the boundary of the grid

Zhao, Shiyu. *Mathematical Foundations of Reinforcement Learning*. Singapore: Springer Nature Singapore, 2025. https://doi.org/10.1007/978-981-97-3944-8.

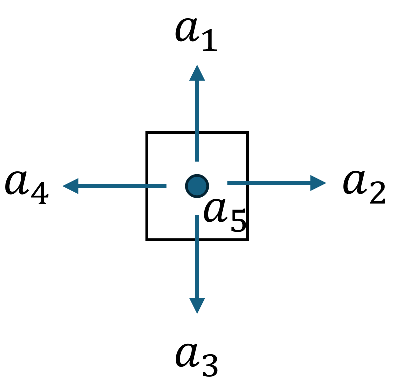
Markov Decision Process: State, Action and State Transition

State



9 cells → 9 states
State space: all states in the problem, $S = \{s_1, \dots, s_9\}$

Action



Move up, right, down, left and stay still
Action space: all possible actions in the problem, $A = \{a_1, \dots, a_5\}$

State Transition

Given a state s and an action a, what will be the next state $P(s, a)$?
e.g., $P(s_1, a_1) = s_2$, $P(s_5, a_3) = s_8$

State \ Action	a_1 (↑)	a_2 (→)	a_3 (↓)	a_4 (←)	a_5 (o)
s_1	s_1	s_2	s_4	s_1	s_1
s_2	…	…	…	…	…
s_3	…	…	…	…	…
s_4	…	…	…	…	…
s_5	…	…	…	…	…
s_6	…	…	…	…	…
s_7	…	…	…	…	…
s_8	…	…	…	…	…
s_9	s_6	s_9	s_9	s_8	s_9

Markov Decision Process: Reward and Agent's Policy

s_1	s_2	s_3
s_4	s_5	s_6 (Trap)
s_7 (Trap)	s_8	s_9 (Target)

- If the agent attempts to exit the boundary, let r = −1.
- If the agent attempts to enter a forbidden cell, let r = −1.
- If the agent reaches the target state, let r = +1.
- Otherwise, the agent obtains a reward of r = 0.

Reward

After executing an action a at a state s, the agent obtains a reward, denoted as $r(s, a)$, as feedback from the environment.

State \ Action	a_1 (↑)	a_2 (→)	a_3 (↓)	a_4 (←)	a_5 (o)
s_1	-1	0	0	-1	0
s_2	-1	0	0	0	0
s_3	-1	-1	-1	0	0
s_4	0	0	-1	-1	0
s_5	0	-1	0	0	0
s_6	0	-1	+1	0	-1
s_7	0	0	-1	-1	-1
s_8	0	+1	-1	-1	0
s_9	-1	-1	-1	0	+1

Agent's Policy

Which actions to take at every state s. E.g., the robot can take two different policies $\pi_1(s)$ and $\pi_2(s)$ are as follows

State	$\pi_1(s)$	$\pi_2(s)$
s_1	→	↓
s_2	↓	↓
s_3	←	↓
s_4	→	↓
s_5	↓	↓
s_6	↓	↓
s_7	→	→
s_8	→	→
s_9	o	o

π_1

→	↓	←
→	↓	↓
→	→	o

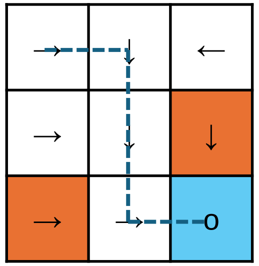
π_2

↓	↓	↓
↓	↓	↓
→	→	o

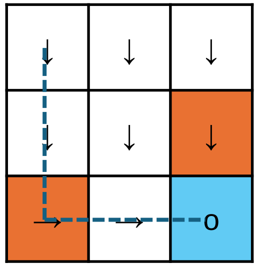
Markov Decision Process: Trajectory and Return

Trajectory

π_1



π_2



Given an initial state s_{init} and the agent's policy π, the agent can start from s_{init} and recurrently follow π to take actions.

$s_0 = s_{init}$
For $t = 0,1, \ldots$
- take an action $a_t \leftarrow \pi(s_t)$
- obtain a reward $R_t \leftarrow r(s_t, a_t)$
- transit to a new state $s_{t+1} \leftarrow P(s_t, a_t)$

For example:

π_1, start from s_1: $s_1 \xrightarrow[r=0]{a_2} s_2 \xrightarrow[r=0]{a_3} s_5 \xrightarrow[r=0]{a_3} s_8 \xrightarrow[r=1]{a_2} s_9$

π_2, start from s_1: $s_1 \xrightarrow[r=0]{a_3} s_2 \xrightarrow[r=-1]{a_3} s_5 \xrightarrow[r=0]{a_3} s_8 \xrightarrow[r=1]{a_2} s_9$

Return (value) of a trajectory

Given an initial state s and the agent's policy π, the return (value) $V_\pi(s)$ is the sum of all rewards on the trajectory

E.g., for the two trajectories, their return is

$$V_{\pi_1}(s_1) = 0 + 0 + 0 + 1 = 1$$
$$V_{\pi_2}(s_1) = 0 - 1 + 0 + 1 = 0$$

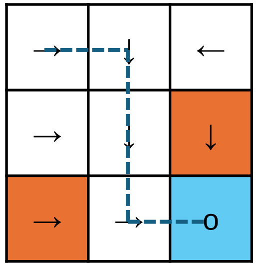
We would like to find a policy for the agent that maximize its return. Therefore, return can be used to evaluate policies.
- For the example, $V_{\pi_1}(s_1) > V_{\pi_2}(s_1)$ so policy π_1 is better than policy π_2

Policy Evaluation and Bellman Equation

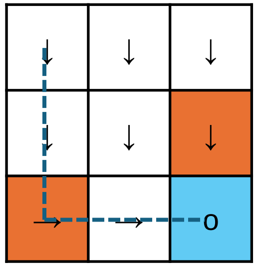
Given the agent's policy π, policy evaluation is to compute $V_\pi(s)$ for all $s \in S$

For a given π, this can be done by listing the trajectory for each state and add up the rewards

π_1



π_2



State	$V_{\pi_1}(s)$	$V_{\pi_2}(s)$
s_1	1	0
s_2	1	1
s_3	1	0
s_4	1	0
s_5	1	1
s_6	1	1
s_7	1	1
s_8	1	1
s_9	0	0

However, we have a more general way to evaluate the policy with **Bellman equation**.

Note that each return consists of an immediate reward and future rewards.

$$\overbrace{\underbrace{s \xrightarrow[R=r(s,a)]{a=\pi(s)}}_{r(s,\pi(s))} \underbrace{s' \xrightarrow[R'=r(s',a')]{a'=\pi(s')} s'' \xrightarrow[R''=r(s'',a'')]{a''=\pi(s'')} \cdots}_{V_\pi(s')}}^{V_\pi(s)}$$

- **Immediate reward**: the reward obtained after taking an action $a = \pi(s)$. That is, $r(s, \pi(s))$
- **Future rewards**: the return of the trajectory starting at the next state $s' = P(s, \pi(s))$

Therefore,

$$V_\pi(s) = r(s, \pi(s)) + V_\pi(s')$$

Which is the Bellman equation

Policy Evaluation with Bellman Equation

π_2

↓	↓	↓
↓	↓	↓
→	→	o

Given the Bellman equation

$$V_\pi(s) = \underbrace{r(s,\pi(s))}_{\text{Immediate reward}} + \underbrace{V_\pi(s')}_{\text{return starting from next state}}$$

We take π shown on the left as an example:

$$V_\pi(s_1) = 0 + V_\pi(s_4)$$
$$V_\pi(s_2) = 0 + V_\pi(s_5)$$
$$V_\pi(s_3) = -1 + V_\pi(s_6)$$
$$V_\pi(s_4) = -1 + V_\pi(s_7)$$
$$V_\pi(s_5) = 0 + V_\pi(s_8)$$
$$V_\pi(s_6) = 1 + V_\pi(s_9)$$
$$V_\pi(s_7) = 0 + V_\pi(s_8)$$
$$V_\pi(s_8) = 1 + V_\pi(s_9)$$
$$V_\pi(s_9) = 0$$

Let $V_\pi = \left(V_\pi(s_1), \ldots, V_\pi(s_9)\right)^\top$, we have the following matrix form of the equations:

$$V_\pi = AV_\pi + b$$

In which $A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$ and $b = \begin{bmatrix} 0 \\ 0 \\ -1 \\ -1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$

This is a type of fixed-point equation

$x = f(x)$ in which $f(x) = \mathrm{A}x + \mathrm{b}$

that can be solved by fixed-point iteration as follows:

```
Randomly initialize x
while |x − f(x)| > ϵ:
      x ← f(x)
```

Its convergence can be proved with contraction mapping theorem

Policy Improvement

π

↓	↓	↓
↓	↓	↓
→	→	o

V_π

0	1	0
0	1	1
1	1	0

π'

→	↓	←
→	↓	↓
→	→	o

Policy improvement: update $\pi(s)$ for a larger $V_\pi(s)$

Recall the Bellman equation

$$V_\pi(s) = r\big(s, \pi(s)\big) + V_\pi(s') = r\big(s, \pi(s)\big) + V_\pi(P(s, \pi(s)))$$

How should we update $\pi(s)$ to enlarge $V_\pi(s)$?

$\pi(s_1)$	$V_\pi(s_1) = r\big(s_1, \pi(s_1)\big) + V_\pi(P(s_1, \pi(s_1)))$
$\boldsymbol{a_1}$ (↑)	$r(s_1, a_1) + V_\pi(P(s_1, a_1)) = -1 + V_\pi(s_1) = -1$
$\boldsymbol{a_2}$ (→)	$r(s_1, a_2) + V_\pi(P(s_1, a_2)) = 0 + V_\pi(s_2) = \mathbf{1}$
$\boldsymbol{a_3}$ (↓)	$r(s_1, a_3) + V_\pi(P(s_1, a_3)) = 0 + V_\pi(s_4) = 0$
$\boldsymbol{a_4}$ (←)	$r(s_1, a_4) + V_\pi(P(s_1, a_4)) = -1 + V_\pi(s_1) = -1$
$\boldsymbol{a_5}$ (o)	$r(s_1, a_5) + V_\pi(P(s_1, a_5)) = 0 + V_\pi(s_1) = 0$

The largest!

Let

$$q_\pi(s, a) = r(s, a) + V_\pi(P(s, a))$$

Then we update $\pi(s)$ by

$$\pi'(s) = \arg\max_a q_\pi(s, a)$$

for each $s \in S$

For the grid world example, we improve

- $\pi'(s_1) \leftarrow \arg\max_a q_\pi(s_1, a) = a_2$
- $\pi'(s_3) \leftarrow \arg\max_a q_\pi(s_3, a) = a_4$
- $\pi'(s_4) \leftarrow \arg\max_a q_\pi(s_4, a) = a_2$

(marked in red)

The policy π is optimal when we cannot do any further improvement. I.e.,

$$\pi(s) = \arg\max_a q_\pi(s, a) \text{ for all } s \in S$$

Policy Iteration

We iteratively do policy evaluation and improvement, until no further improvement can be done.

```
Randomly initialize π(s) and V_π(s)
While True:
    While |V_π(s) − (R + V_π(s′))| > ϵ:
        V_π(s) ← R + V_π(s′) for all s ∈ S
    π′(s) ← argmax_a q_π(s,a) for all s ∈ S
    if π′(s) = π(s): break
    π(s) ← π′(s)
```

Policy evaluation: compute $V_\pi(s)$ for the current policy $\pi(s)$ by fixed-point iteration

Policy improvement: improve $\pi(s)$ by selecting the action a with maximal $q(s,a)$

$R = r\big(s, \pi(s)\big)$ is the immediate reward
$s' = P(s, \pi(s))$ is the next state
$q_\pi(s,a) = r(s,a) + V_\pi(P(s,a))$

Value Iteration

Recall that the policy π is optimal when we cannot do any further improvement. I.e.,

$$\pi^*(s) = \arg\max_a q(s,a) \text{ for all } s \in S$$

So for an optimal policy π^*, its value function

$$V_{\pi^*}(\mathrm{s}) = q_{\pi^*}(s, \pi(s))$$

will always equal to the maximal one among $q(s,a)$. I.e.,

$$V_{\pi^*}(\mathrm{s}) = \max_a \; q_\pi(s,a)$$

$$V_{\pi^*}(\mathrm{s}) = \max_a \left(r(s,a) + V_{\pi^*}(P(s,a))\right)$$

This is also a fixed-point equation that can be solved by fixed-point iteration.

Randomly initialize $V_\pi(s)$

While $\left|V_\pi(s) - \max_a \left(r(s,a) + V_\pi(P(s,a))\right)\right| > \epsilon$:

$\quad V_\pi(s) \leftarrow \max_a \left(r(s,a) + V_\pi(P(s,a))\right)$ for all $s \in S$

$\pi(s) \leftarrow \arg\max_a q_\pi(s,a)$ for all $s \in S$

Its convergence can be proved with contraction mapping theorem

Value Iteration

s_1	s_2	s_3
s_4	s_5	s_6 (Trap)
s_7 (Trap)	s_8	s_9 (Target)

State \ Action	a_1 (↑)	a_2 (→)	a_3 (↓)	a_4 (←)	a_5 (o)
s_1	-1	0	0	-1	0
s_2	-1	0	0	0	0
s_3	-1	-1	-1	0	0
s_4	0	0	-1	-1	0
s_5	0	-1	0	0	0
s_6	0	-1	+1	0	-1
s_7	0	0	-1	-1	-1
s_8	0	+1	-1	-1	0
s_9	-1	-1	-1	0	+1

Example:

$$V_{\pi^*}(s) = \max_{a \in \{a_1, \dots, a_5\}} \big(r(s,a) + V_{\pi^*}(P(s,a))\big)$$

$$V_{\pi^*}(s_1) = \max\big(-1 + V_{\pi^*}(s_1),\ 0 + V_{\pi^*}(s_2),\ 0 + V_{\pi^*}(s_4), -1 + V_{\pi^*}(s_1),\ 0 + V_{\pi^*}(s_1)\big)$$
$$V_{\pi^*}(s_2) = \max\big(-1 + V_{\pi^*}(s_2),\ 0 + V_{\pi^*}(s_3),\ 0 + V_{\pi^*}(s_5),\ 0 + V_{\pi^*}(s_1),\ 0 + V_{\pi^*}(s_2)\big)$$
$$V_{\pi^*}(s_3) = \max\big(-1 + V_{\pi^*}(s_3), -1 + V_{\pi^*}(s_3), -1 + V_{\pi^*}(s_6),\ 0 + V_{\pi^*}(s_2),\ 0 + V_{\pi^*}(s_3)\big)$$
$$V_{\pi^*}(s_4) = \max\big(\ 0 + V_{\pi^*}(s_1),\ 0 + V_{\pi^*}(s_5), -1 + V_{\pi^*}(s_7), -1 + V_{\pi^*}(s_4),\ 0 + V_{\pi^*}(s_4)\big)$$
$$V_{\pi^*}(s_5) = \max\big(\ 0 + V_{\pi^*}(s_2), -1 + V_{\pi^*}(s_6),\ 0 + V_{\pi^*}(s_8),\ 0 + V_{\pi^*}(s_4),\ 0 + V_{\pi^*}(s_5)\big)$$
$$V_{\pi^*}(s_6) = \max\big(\ 0 + V_{\pi^*}(s_3), -1 + V_{\pi^*}(s_6), +1 + V_{\pi^*}(s_9),\ 0 + V_{\pi^*}(s_3), -1 + V_{\pi^*}(s_6)\big)$$
$$V_{\pi^*}(s_7) = \max\big(\ 0 + V_{\pi^*}(s_4), -1 + V_{\pi^*}(s_8), -1 + V_{\pi^*}(s_7),\ 0 + V_{\pi^*}(s_7), -1 + V_{\pi^*}(s_7)\big)$$
$$V_{\pi^*}(s_8) = \max\big(\ 0 + V_{\pi^*}(s_5), +1 + V_{\pi^*}(s_9), -1 + V_{\pi^*}(s_8), -1 + V_{\pi^*}(s_2),\ 0 + V_{\pi^*}(s_8)\big)$$
$$V_{\pi^*}(s_9) = 0$$

Let $V_{\pi^*} = \big(V_{\pi^*}(s_1), \dots, V_{\pi^*}(s_9)\big)^{\top}$, we have

$$V_{\pi^*} = f(V_{\pi^*})$$

in which $f(V_{\pi^*}) = \max(A_1 V_{\pi^*} + b_1, A_2 V_{\pi^*} + b_2, A_3 V_{\pi^*} + b_3, A_4 V_{\pi^*} + b_4, A_5 V_{\pi^*} + b_5)$

Its convergence can be proved with contraction mapping theorem

Stochastic Cases

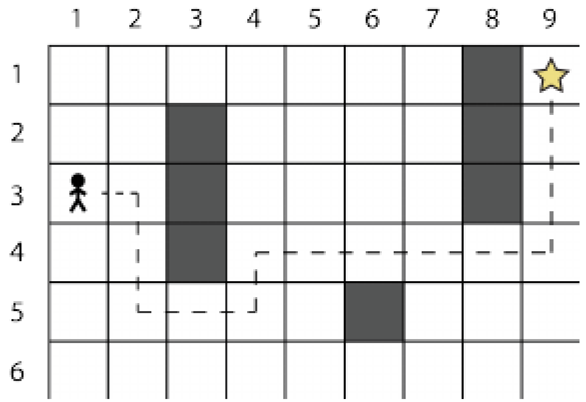
	Deterministic	Stochastic
State Transition	$P(s,a) = s'$	$P(s'\|s,a)$ is the conditional probability of transition from (s,a) to s'. $\sum_{s'\in S} P(s'\|s,a) = 1$
Agent's Policy	$\pi(s) = a$	$\pi(a\|s)$ is the conditional probability from s to a, $\sum_{a\in A} \pi(a\|s) = 1$
Return (Value)	$V_\pi(s) = R_1 + R_2 + \cdots$	$V_\pi(s) = \mathbb{E}[R_1 + R_2 + \cdots]$ (the expectation of total rewards)
Bellman Equation	$V_\pi(s) = r\big(s,\pi(s)\big) + V_\pi(s')$ in which $s' = P(s,\pi(s))$	$V_\pi(s) = \sum_{a\in A} \pi(a\|s)[r(s,a) + \sum_{s'\in S} P(s'\|s,a)\, V_\pi(s')]$
Action-Value Function	$q(s,a) = r(s,a) + V_\pi(P(s,a))$ $V_\pi(s) = q(s,\pi(s))$	$q(s,a) = r(s,a) + \sum_{s'\in S} P(s'\|s,a)\, V_\pi(s')$ $V_\pi(s) = \sum_{a\in A} \pi(a\|s) q(s,a)$

Decay factor γ

- Future rewards should be "less important"
- Sometimes critical for convergence
- γ is typically a number a bit less than 1 (e.g., 0.99)

	Without decay factor	With decay factor
Return (Value)	$V_\pi(s) = \mathbb{E}[R_1 + R_2 + R_3 + \cdots]$	$V_\pi(s) = \mathbb{E}[R_1 + \boldsymbol{\gamma} R_2 + \boldsymbol{\gamma}^2 R_3 + \cdots]$
Bellman Equation	$V_\pi(s) = \sum_{a \in A} \pi(a\|s)[r(s,a) + \sum_{s' \in S} P(s'\|s,a) V_\pi(s')]$	$V_\pi(s) = \sum_{a \in A} \pi(a\|s)[r(s,a) + \boldsymbol{\gamma} \sum_{s' \in S} P(s'\|s,a) V_\pi(s')]$
Action-Value Function	$q(s,a) = r(s,a) + \sum_{s' \in S} P(s'\|s,a) V_\pi(s')$	$q(s,a) = r(s,a) + \boldsymbol{\gamma} \sum_{s' \in S} P(s'\|s,a) V_\pi(s')$

Model-Based and Model-Free Setting



In a **model-based** setting, you have full access to the state transition $P(s'|s,a)$ and reward function $r(s,a)$

Model = state transition + reward function

Game developer's view

In a **model-free** setting, you don't have access to $P(s'|s,a)$ and $r(s,a)$, but can interact with the environment in an episodic way

$$s \xrightarrow[R]{a=\pi(s)} s' \xrightarrow[R']{a'=\pi(s')} s'' \xrightarrow[R'']{a''=\pi(s'')} \ldots$$

Player's view

For most complex, real-world settings

Learning in Model-Free Setting

Option 1: Learn a model (reduce the problem to model-based learning)
Estimate the state transition $P(s'|s,a)$ and reward function $r(s,a)$ of the given environment by episodic interaction.

Representative work: Dyna-Q
More detailed introduction can be found in
https://snowkylin.github.io/blogs/introduction-to-model-based-rl.html

A mathematician is pursuing a work as a firefighter.

"What do you do if you pass a Dumpster, and it's on fire?"

"Easy, I'd just put out the fire."

"Okay, what do you do if you pass a Dumpster, and it's not on fire?"

"Easy! I'd set it on fire!"

“Are you an idiot?!”

"No! I've just reduced the problem to one I've already solved!"

Learning in Model-Free Setting

Option 2: Learn a policy without a model

Improve agent's policy π

- without explicit involvement of $P(s'|s,a)$ (we don't know the exact probability)
- only involve $r(s,a)$ in an episodic setting. I.e., the agent is at state s and actually performed action a

Is it possible? Let's review the policy iteration method

Randomly initialize $\pi(s)$ and $V_\pi(s)$

While True:

> While $|V_\pi(s) - \left(R + V_\pi(s')\right)| > \epsilon$:
>
> $\qquad V_\pi(s) \leftarrow R + V_\pi(s')$ for all $s \in S$
>
> $\pi'(s) \leftarrow \arg\max_a q_\pi(s,a)$ for all $s \in S$
>
> if $\pi'(s) = \pi(s)$: break
>
> $\pi(s) \leftarrow \pi'(s)$

Policy evaluation: compute $V_\pi(s)$ for the current policy $\pi(s)$

Policy improvement: improve $\pi(s)$ by selecting the action a with maximal $q(s,a)$

$R = r\left(s, \pi(s)\right)$ is the immediate reward

$s' = P(s, \pi(s))$ is the next state

$q_\pi(s,a) = r(s,a) + V_\pi(P(s,a))$

Learning in Model-Free Setting

Option 2: Learn a policy without a model

Can we solve the problem if we have a way to estimate $V_\pi(s)$ without a model? Better not.
In policy improvement step, we need to compute

$$q_\pi(s,a) = r(s,a) + V_\pi(P(s,a))$$

Which require us to know the state transition function.

```
Randomly initialize π(s) and V_π(s)
While True:
    Estimate V_π(s) by episodic interaction
    with the environment using π(s)
    π'(s) ← arg max_a q_π(s,a) for all s ∈ S
    if π'(s) = π(s): break
    π(s) ← π'(s)
```

Policy evaluation: estimate $V_\pi(s)$ for the current policy $\pi(s)$

Policy improvement: improve $\pi(s)$ by selecting the action a with maximal $q(s,a)$

$$q_\pi(s,a) = r(s,a) + V_\pi(P(s,a))$$

Learning in Model-Free Setting

Option 2: Learn a policy without a model

How about estimate $q_\pi(s,a)$ directly instead of $V_\pi(s)$?

It works!

Randomly initialize $\pi(s)$ and $V_\pi(s)$

While True:

Estimate $q_\pi(s,a)$ by episodic interaction with the environment using $\pi(s)$

$\pi'(s) \leftarrow \arg\max_a q_\pi(s,a)$ for all $s \in S$

if $\pi'(s) = \pi(s)$: break

$\pi(s) \leftarrow \pi'(s)$

Policy evaluation: estimate $q_\pi(s,a)$ for the current policy $\pi(s)$

Policy improvement: improve $\pi(s)$ by selecting the action a with maximal $q(s,a)$

$$q_\pi(s,a) = r(s,a) + V_\pi(P(s,a))$$

Policy Evaluation - Estimating $\boldsymbol{q_{\pi}(s, a)}$ in an episodic setting

- $q_{\pi}(s, a)$ is the expectation of total rewards when starts from state s and performs action a.
- We can interact with the environment many times, to obtain lots of trajectories
- Then we do some simple statistics on the sampled trajectories:

$q_{\pi}(s, a) \approx$ the average return of (partial) trajectories
starting from s with action a

- Monte Carlo approach

Policy Evaluation - Estimating $q_\pi(s,a)$ with Temporal Difference Approach

- We can do incremental update on $q_\pi(s,a)$, even if the episode does not finish
- Assume we have a partial trajectory …, $s \xrightarrow[r]{a} s' \xrightarrow[r']{a'}, \ldots$
- We update $q_\pi(s,a)$ by

$$q_\pi(s,a) \leftarrow q_\pi(s,a) + \alpha\Big(\underbrace{r + q_\pi(s',a')}_{\text{New estimation based on the trajectory}} - \underbrace{q_\pi(s,a)}_{\text{Previous estimation}}\Big)$$

A small “update rate”, e.g., 0.01

Policy Improvement - Including Exploration in Agent's Policy

- We are estimating $Q_\pi(s, a)$ instead of precisely computing it, which means it can be inaccurate
- If we completely trust the estimated result to improve our policy, it may fall into suboptimality
  - Some (s, a) pair may never occur during the estimation
- We want to guarantee that every (s, a) has a positive probability during estimation.
- ϵ-greedy approach:
  - With probability 1-ε, choose the greedy action
  - With probability ε, choose an action at random

Left:
20% Reward = 0
80% Reward = 5

Right:
50% Reward = 1
50% Reward = 3

What if the first action is to choose the left door and observe reward=0?

SARSA Algorithm

Sarsa: An on-policy TD control algorithm

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using $\boxed{\text{policy derived from } Q \text{ (e.g., } \epsilon\text{-greedy)}}$
    $\boxed{Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]}$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Policy improvement with exploration

Policy evaluation with temporal difference

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 $Q(terminal, \cdot) = 0$

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

Similar to Value Iteration: not only estimate q_π but also optimize q_π towards optimal q_{π^*}

Off-policy: the policy that interacts with the environment and the policy in TD update can be different

Policy Gradient

- Assume the agent's policy $\pi(a|s)$ is parameterized by θ (denoted by π_θ) and differentiable
- Our target is to find an optimal policy π_θ^* that maximize the (expected) return $V_{\pi_\theta}(s)$
- Can we achieve this by gradient descend (ascend)?
- Let the objective be

$$J(\theta) = \mathbb{E}_{s_0}[V_{\pi_\theta}(s_0)]$$

in which s_0 is the initial state

- The gradient is

$$\frac{\partial}{\partial\theta}J(\theta) = \mathbb{E}_{\pi_\theta}[q_{\pi_\theta}(s,a)\frac{\partial}{\partial\theta}\log\pi_\theta(a|s)]$$

Thank you!

Xihan Li
Department of Computer Science,
University College London
xihan.li@cs.ucl.ac.uk
https://snowkylin.github.io

Feb 2025