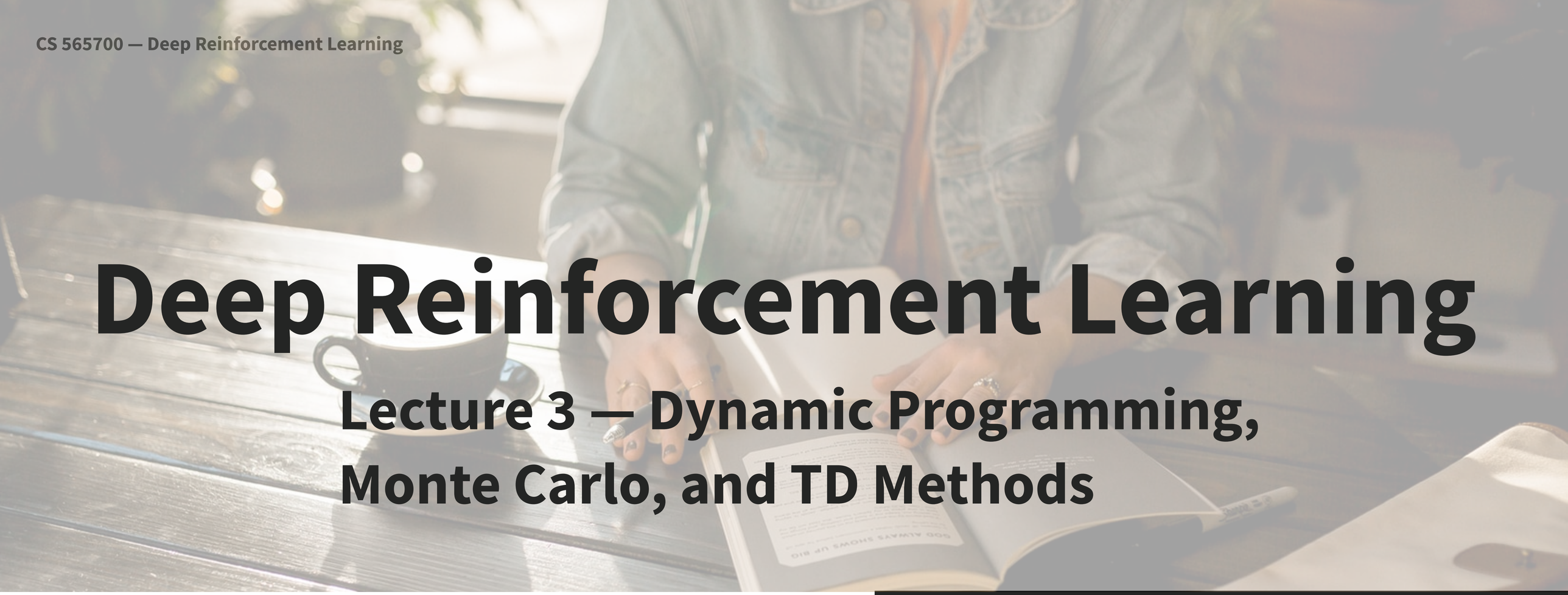


Deep Reinforcement Learning

Lecture 3 — Dynamic Programming, Monte Carlo, and TD Methods

國立清華大學
NATIONAL TSING HUA UNIVERSITY

National Tsing Hua University
Department of Computer Science

Prof. Chun-Yi Lee

Outline

- **Policy Iteration / Value Iteration**
- MC and TD method
- Sarsa
- Q-learning

Solving an MDP

Introduction

- Dynamic programming method
  - For solving a known MDP

- Model free method
  - For solving an unknown MDP

Dynamic Programming

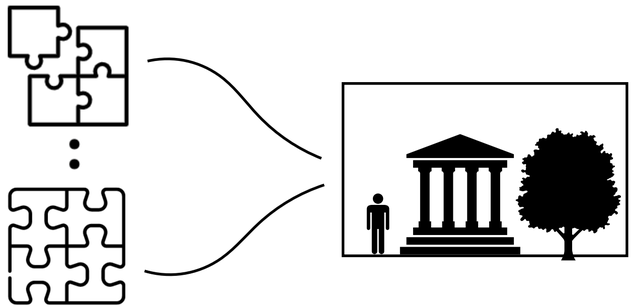
What is dynamic programming (DP)

- A general method for solving complex problems.
- Break a complex problem into easy sub-problems and solve the sub-problems.
- Commonly used in Mathematics, Computer Science, Biology, Economics, etc.

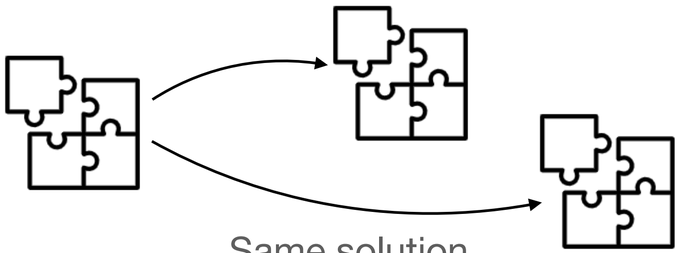
Dynamic Programming

Property

- Optimal sub-structure : An optimal solution can be decomposed into optimal sub solutions.



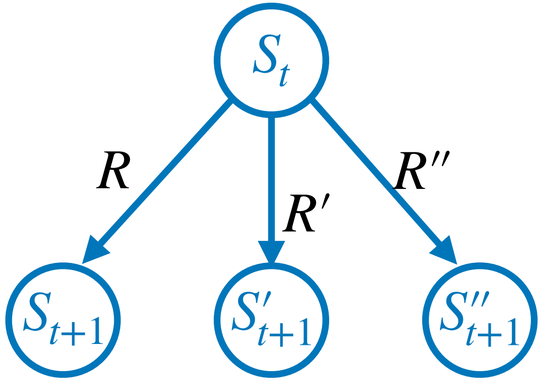
- Overlapped sub-problems : A recursive algorithm visits the same subproblems repeatedly, such that their solutions can be reused.



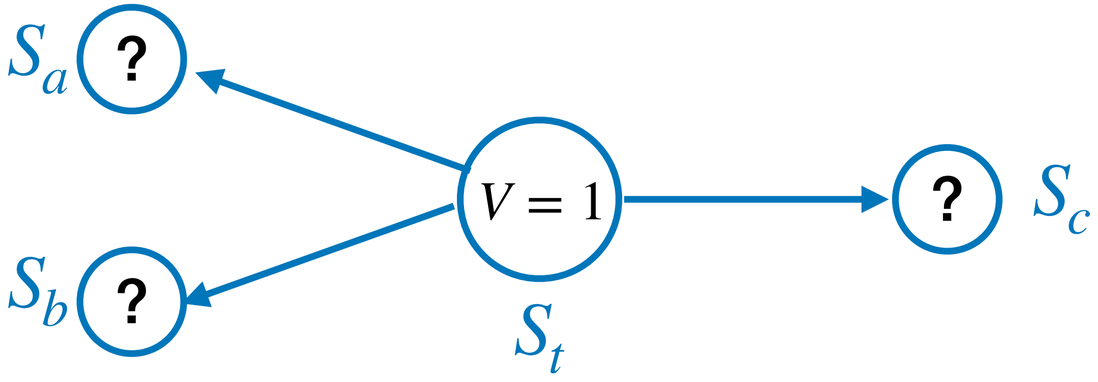
Why a Known MDP can Solved Using DP

MDP can be decomposed into sub-problems using the Bellman equation

- **First step**: Decompose a problem by the Bellman equation



- **Second step**: Computed state values can be stored and reused

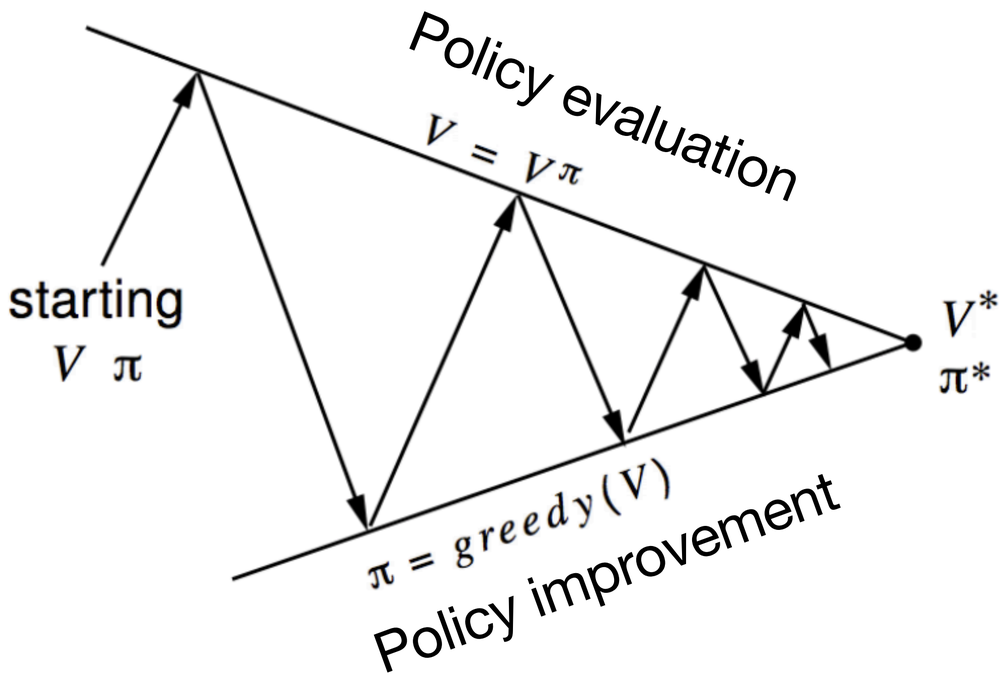


Dynamic Programming Method

Policy Iteration

DP is a way to find the optimal policy, and consists of two iterative steps:

- Policy evaluation
- Policy improvement



Policy Iteration

Policy evaluation — evaluate a given policy

Using Bellman expectation backup

$$V_{k+1}(s) \leftarrow \mathbb{E}[R_t + \gamma V_k(S_{t+1}) \,|\, S_t = s]$$
$$= \sum_a \pi(a \,|\, s) \sum_{s'} p(s' \,|\, s, a)[R_t + \gamma V_k(s')]$$

Policy Iteration

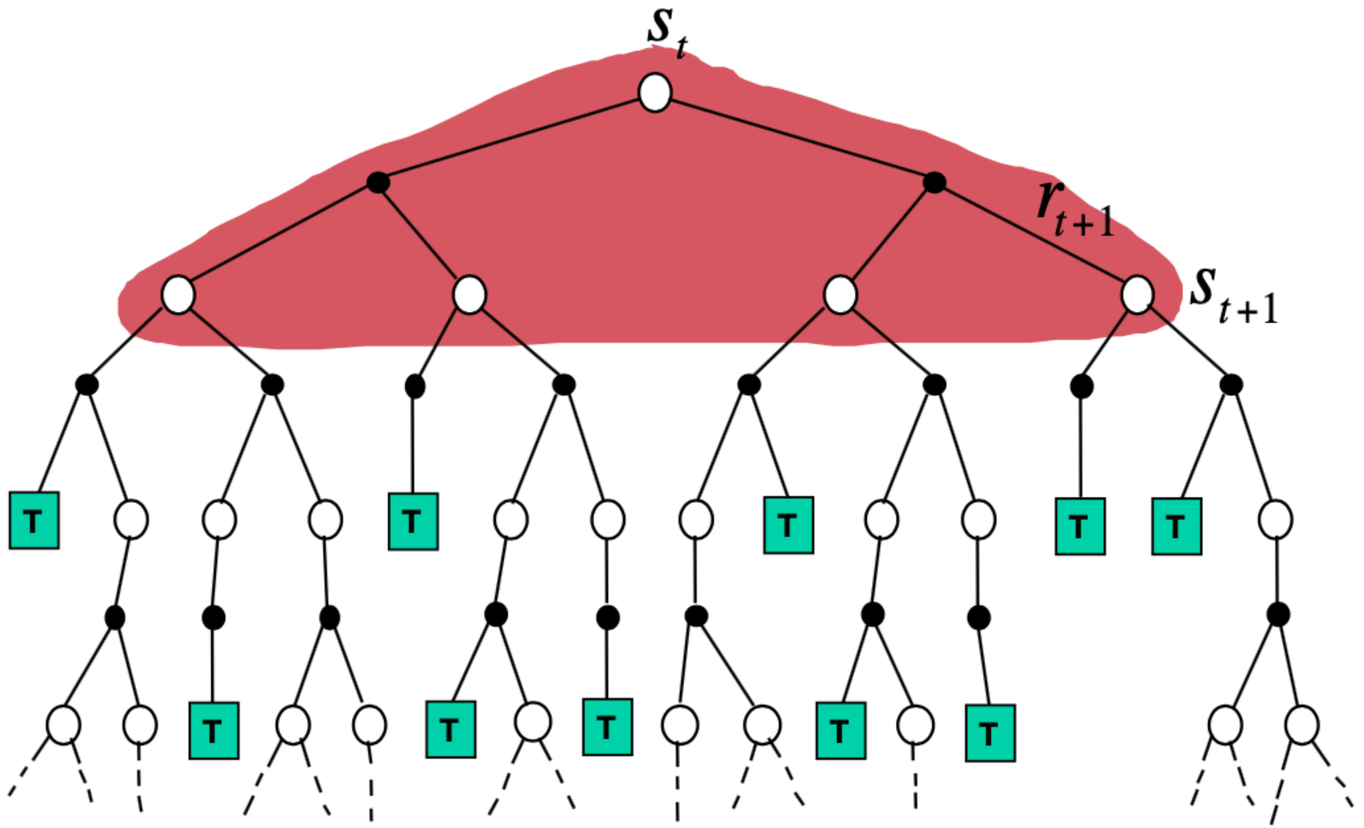
Policy evaluation in pseudo-code

Iterative policy evaluation

Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

Policy Iteration

The concept of policy evaluation using DP

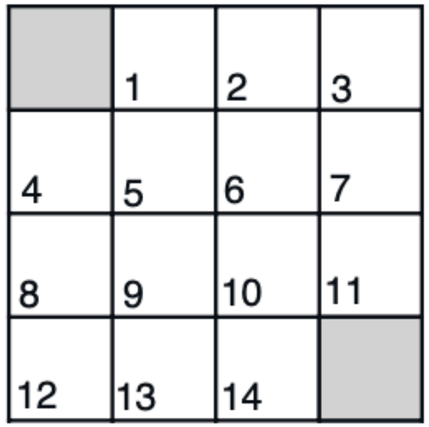


Policy Iteration

Evaluate a random policy in an example grid world

- Normal state {1~14} : 

- 4 action {→, ↑, ←, ↓} with equal probability
- All transitions get r = -1
- Terminal state : 



Policy Iteration

Evaluate a random policy in an example grid world

	1	2	3	4
1	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0

Iter 0 ~ Iter 1

	1	2	3	4
1	0.0	-1.0	-1.0	-1.0
2	-1.0	-1.0	-1.0	-1.0
3	-1.0	-1.0	-1.0	-1.0
4	-1.0	-1.0	-1.0	0.0

$$V_1(s) = (-1 + 0.0) * 0.25 + (-1 + 0.0) * 0.25 + (-1 + 0.0) * 0.25 + (-1 + 0.0) * 0.25$$
$$= -1$$

$$V_1(s) = (-1 + 0.0) * 0.25 + (-1 + 0.0) * 0.25 + (-1 + 0.0) * 0.25 + (-1 + 0.0) * 0.25$$
$$= -1$$

Policy Iteration

Evaluate a random policy in an example grid world

	1	2	3	4
1	0.0	-1.0	-1.0	-1.0
2	-1.0	-1.0	-1.0	-1.0
3	-1.0	-1.0	-1.0	-1.0
4	-1.0	-1.0	-1.0	0.0

Iter 1 ~ Iter 2

	1	2	3	4
1	0.0	-1.75	-2.0	-2.0
2	-1.75	-2.0	-2.0	-2.0
3	-2.0	-2.0	-2.0	-1.75
4	-2.0	-2.0	-1.75	0.0

$$V_1(s) = (-1 + -1) * 0.25 + (-1 + -1) * 0.25 + (-1 + -1) * 0.25 + (-1 + -1) * 0.25$$
$$= -2$$

$$V_1(s) = (-1 + -1) * 0.25 + (-1 + -1) * 0.25 + (-1 + 0.0) * 0.25 + (-1 + -1) * 0.25$$
$$= -1.75$$

Policy Iteration

Evaluate a random policy in an example grid world

	1	2	3	4
1	0.0	-13.11718	-18.69182	-20.53607
2	-13.11718	-16.84757	-18.70057	-18.69182
3	-18.69182	-18.70057	-16.84757	-13.11718
4	-20.53607	-18.69182	-13.11718	0.0

Iter 50 ~ Iter 51

	1	2	3	4
1	0.0	-13.16414	-18.76141	-20.61395
2	-13.16414	-16.90888	-18.7697	-18.76141
3	-18.76141	-18.7697	-16.90888	-13.16414
4	-20.61395	-18.76141	-13.16414	0.0

	1	2	3	4
1	0.0	-13.99894	-19.99843	-21.99824
2	-13.99894	-17.99861	-19.99844	-19.99843
3	-19.99843	-19.99844	-17.99861	-13.99894
4	-21.99824	-19.99843	-13.99894	0.0

Iter 200 ~ Iter 201

Converge

	1	2	3	4
1	0.0	-13.99894	-19.99843	-21.99824
2	-13.99894	-17.99861	-19.99844	-19.99843
3	-19.99843	-19.99844	-17.99861	-13.99894
4	-21.99824	-19.99843	-13.99894	0.0

Policy Iteration

Policy improvement — enhance through the value function

Improve the policy by argmax operation: $\pi'(s) = \arg\max_a Q_\pi(s, a)$,

The new policy π' causes $V_\pi(s) \leq Q_\pi(s, \pi'(s))$,

$$
\begin{aligned}
V_\pi(s) &\leq Q_\pi(s, \pi'(s)) = E_{\pi'}[R_t + \gamma V(s_{t+1}) \,|\, S_t = s] \\
&\leq E_{\pi'}[R_t + \gamma Q_\pi(s_{t+1}, \pi'(S_{t+1})) \,|\, S_t = s] \\
&\leq E_{\pi'}[R_t + \gamma R_{t+1} + \gamma^2 Q(S_{t+1}, \pi'(S_{t+2})) \,|\, S_t = s] \\
&\leq E_{\pi'}[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \ldots \,|\, S_t = s] = V_{\pi'}(s)
\end{aligned}
$$

Policy Iteration

Policy improvement — enhance through the value function

If the new policy π' is as good as π for all states, then:

$$V_\pi(s) = Q_\pi(s, \pi'(s)) = \max_a Q_\pi(s, a)$$

In such cases, π the optimal policy

Policy Iteration

Policy Iteration

According to the above derivation, it is then possible to obtain a monotonically improving sequence for π and V as follows:

$$\pi_0 \rightarrow V_{\pi_0} \rightarrow \pi_1 \rightarrow V_{\pi_1} \rightarrow \pi_2 \ldots\ldots\ldots \rightarrow \pi_* \rightarrow V_*$$

In a finite MDP, this process would converge to the optimal policy and the optimal value function within a finite number of iterations.

Value Iteration

Value Iteration through the Bellman Optimality Equation

- Iterative application of the Bellman optimality backup as follows:

$$V_{k+1}(s) \leftarrow max_a(\sum_{s'} p(s' \mid s, a)[R_t + \gamma V_k(s')])$$

- Obtain a sequence of improving sequence that gradually converge to V_*

$$V_0 \rightarrow V_1 \rightarrow V_2 \ldots\ldots\ldots \rightarrow V_*$$

- No explicit policy is required
- Intermediate value functions may not correspond to any policy

Value Iteration

Value Iteration

Value iteration

Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi \approx \pi_*$, such that
  $\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$

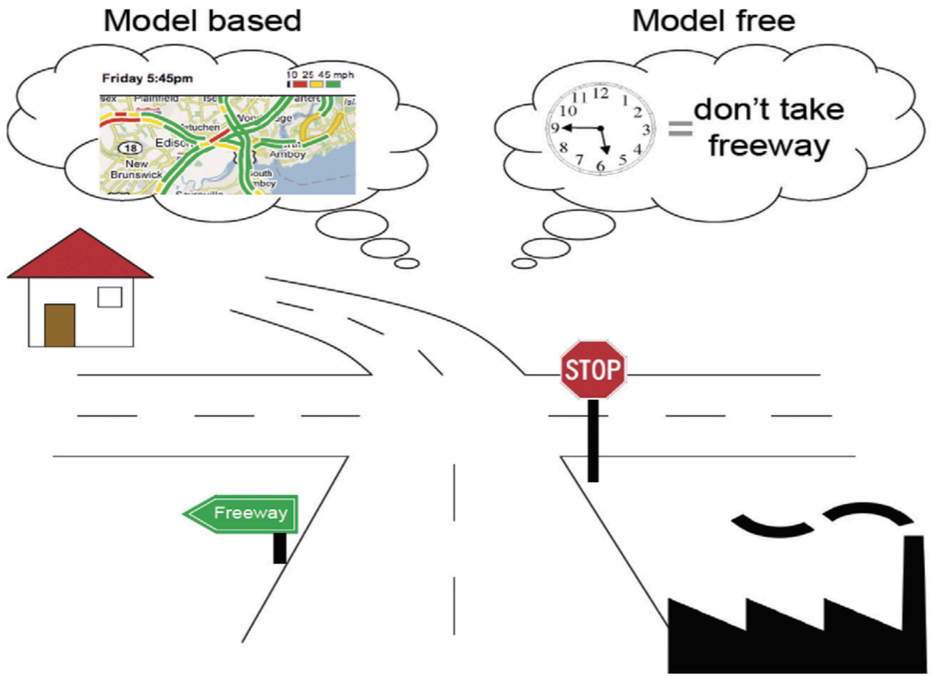
Outline

- Policy Iteration / Value Iteration
- **MC and TD method**
- Sarsa
- Q-learning

Model Free Method

What is a model free method?

- No need of the full knowledge of the environment
- **Only** requires **experiences** — sequences of S, A, R



Model Free Method

Prediction / Control

- Model free prediction
  - Estimate the value function of an unknown MDP

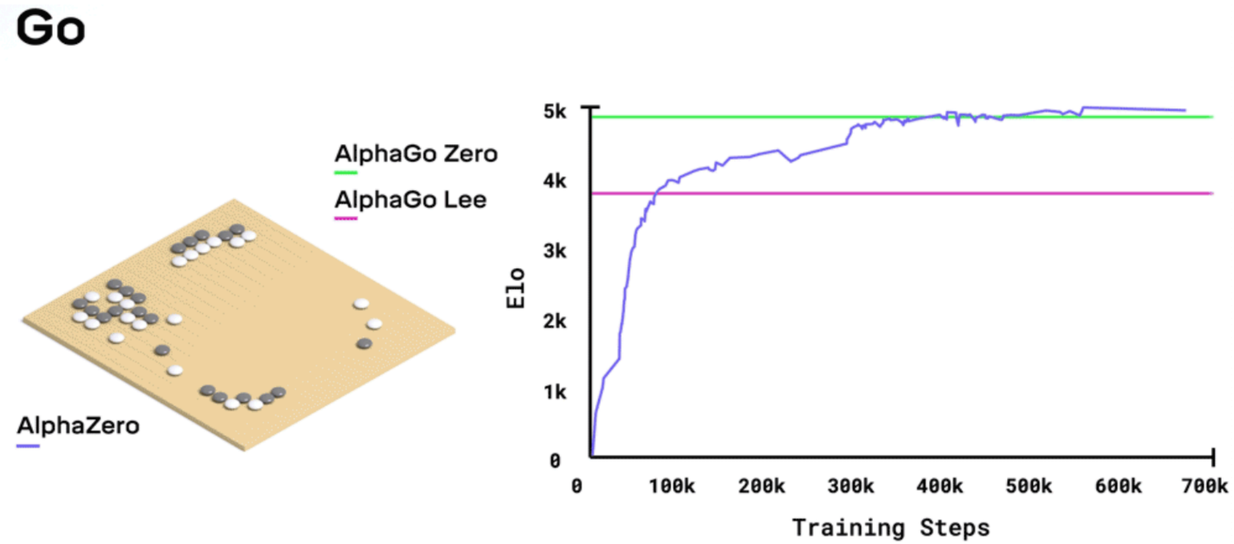
- Model free control
  - Optimize the policy of an unknown MDP

Monte Carol Method

Monte Carlo (MC) Prediction

- MC is only used for episodic tasks
- MC learning is based on averaged sample returns
- MC samples trajectories, where each one consists of samples (S_1, A_1, R_1), $(S_2, A_2, R_2), \ldots, (S_T, A_T, R_T) \sim \pi$. T is the terminal step of an episode.

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



Monte Carol Method

First-Visit Monte Carlo (MC)

- First-Visit MC — Average of the returns following the first visits to s

First-visit MC prediction, for estimating $V \approx v_\pi$

Initialize:
    $\pi \leftarrow$ policy to be evaluated
    $V \leftarrow$ an arbitrary state-value function
    $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:
    Generate an episode using π
    For each state s appearing in the episode:
        $G \leftarrow$ return following the first occurrence of s
        Append G to $Returns(s)$
        $V(s) \leftarrow \text{average}(Returns(s))$

Monte Carol Method

Every-Visit Monte Carlo (MC)

- Every-Visit MC — Average of the returns following all visits to s

Initialize:
$\pi \leftarrow$ policy to be evaluated
$V \leftarrow$ an arbitrary state-value function
$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:
Generate an episode using π
For each state s appearing in the episode: all s in this episode
$G \leftarrow$ return following ~~the first occurrence of s~~
Append G to $Returns(s)$
$V(s) \leftarrow \text{average}(Returns(s))$

Monte Carol Method Example

Every-Visit v.s. First-Visit

$$(A, r = +3) \rightarrow (A, r = +2) \rightarrow (B, r = -4) \rightarrow (B, r = +2) \rightarrow (A, r = +4) \rightarrow END$$

$$(B, r = -3) \rightarrow (A, r = +2) \rightarrow (B, r = -4) \rightarrow END$$

- **First-Visit** : $A = \frac{1}{2}[(3 + 2 - 4 + 2 + 4) + (2 - 4)] = \frac{5}{2}$

$$B = \frac{1}{2}[(-4 + 2 + 4) + (-3 + 2 - 4)] = -\frac{3}{2}$$

- **Every-Visit** : $A = \frac{1}{4}[7 + 4 + 4 + (-2)] = \frac{13}{4}$

$$B = \frac{1}{4}[2 + 6 + (-5) + (-4)] = -\frac{1}{4}$$

Incremental Monte Carlo Update

Incremental Mean

- The mean $\mu_1, \mu_2 \ldots$ of sequence $x_1, x_2 \ldots$ can be computed incrementally as follows:

$$\begin{aligned}\mu_k &= \frac{1}{k}\sum_{i=1}^{k} x_i = \frac{1}{k}(x_k + \sum_{i=1}^{k-1} x_i) \\ &= \frac{1}{k}(x_k + (k-1)\mu_{k-1}) \\ &= \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})\end{aligned}$$

Incremental Monte Carlo Update

Incremental Monte Carlo Update

- For each s_t in an episode with G_t ,

$$N(s_t) \leftarrow N(s_t) + 1$$

$$V(s_t) \leftarrow V(s_t) + \frac{1}{N(s_t)}(G_t - V(s_t))$$

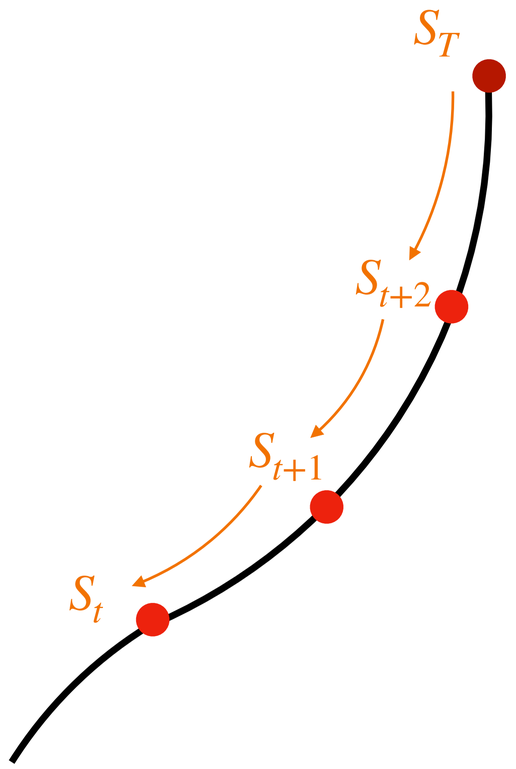
- In non-stationary problems, it can be useful to track a running mean, i.e., forget old episodes:

$$V(s_t) \leftarrow V(s_t) + \alpha(G_t - V(s_t))$$

Temporal Difference Method

Temporal Difference (TD) Prediction

- TD can learn from incomplete episodes
- TD updates by bootstrapping itself
- Learning from a guessed $V(s)$



Temporal Difference Method

Temporal Difference (TD)

- Updates $V(s_t)$ by the estimated the return $R_t + \gamma V(s_{t+1})$, rather than the actual return G_t:

$$V(s_t) \leftarrow V(s_t) + \alpha(R_t + \gamma V(s_{t+1}) - V(s_t))$$

- $R_t + \gamma V(s_{t+1})$ is called the ***TD target***
- $R_t + \gamma V(s_{t+1}) - V(s_t) = \delta_t$ is called the ***TD error***

Temporal Difference Method

Temporal Difference (TD) Algorithm

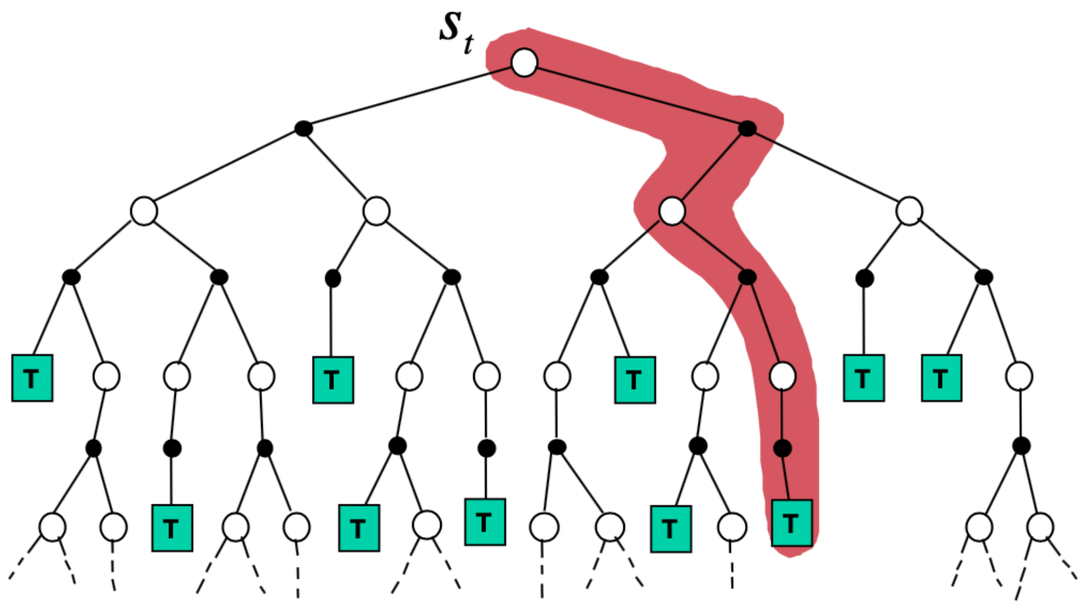
Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0, \forall s \in \mathcal{S}^+$)
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        $A \leftarrow$ action given by π for S
        Take action A, observe R, S'
        $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
        $S \leftarrow S'$
    until S is terminal

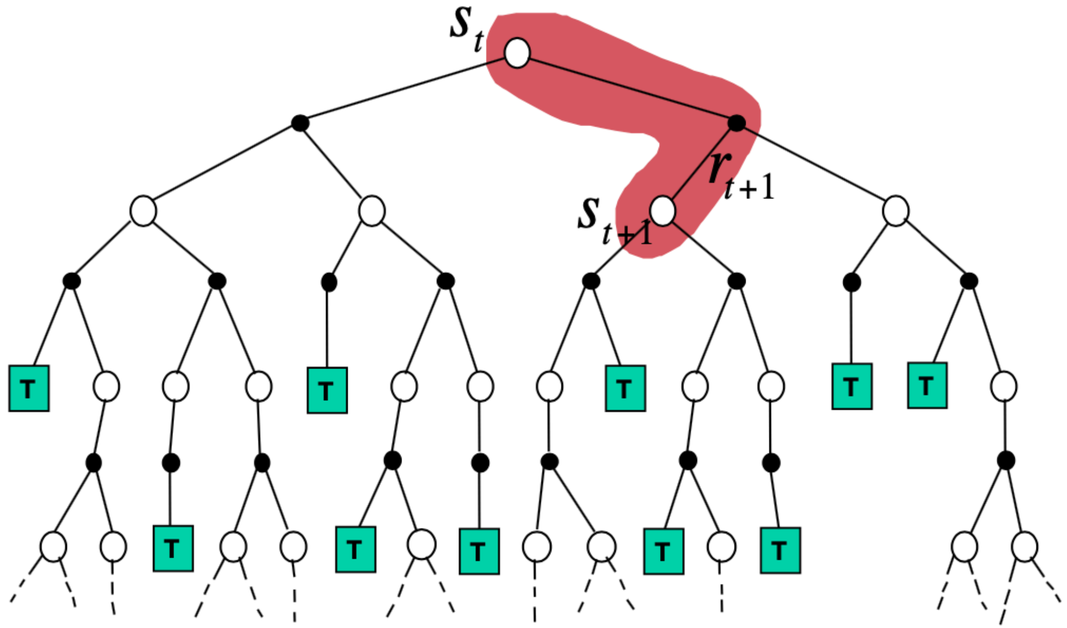
Monte Carol v.s. Temporal Difference

Comparison of the difference between MC and TD

Monte Carlo

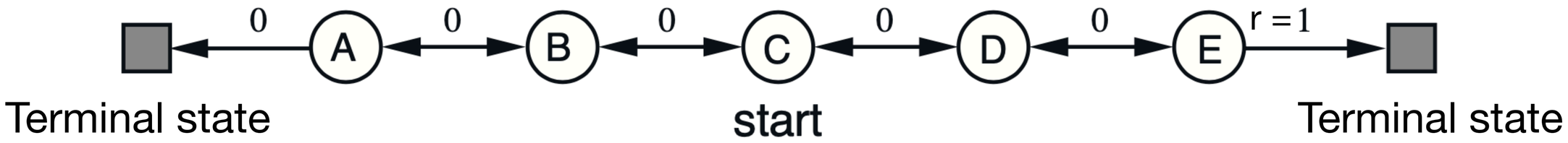


TD



Random Walk Example

Predict Returns using MC and TD

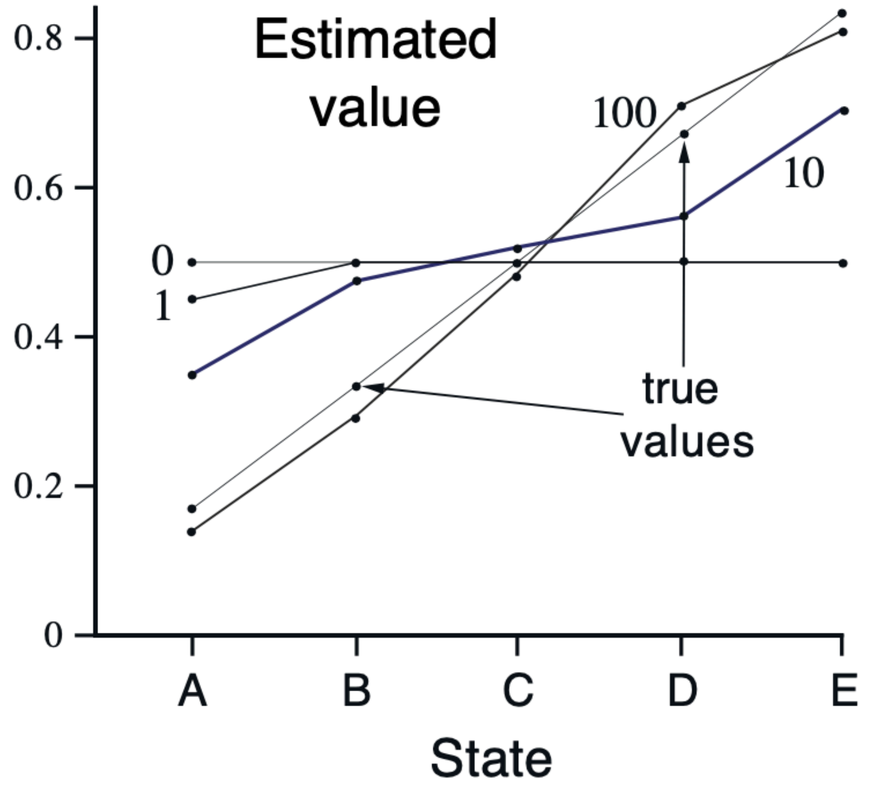


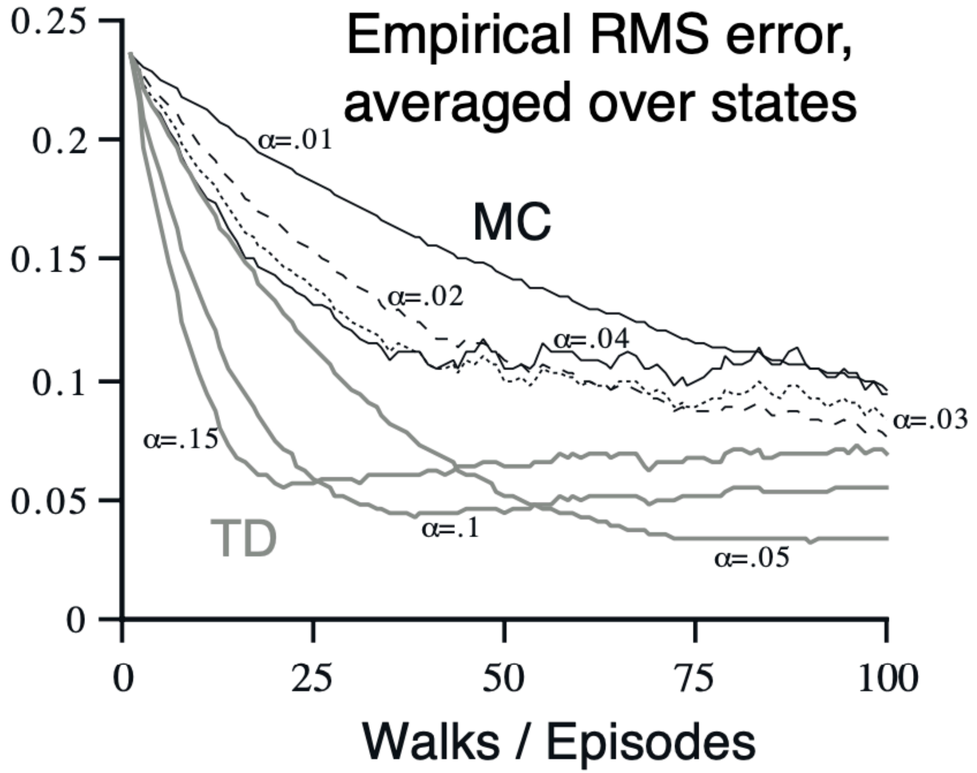
Given a random policy π : 50% Left, 50% Right

Assume that $\gamma = 1$, the true values from E to A : $\frac{5}{6}, \frac{4}{6}, \frac{3}{6}, \frac{2}{6}, \frac{1}{6}$

Random Walk Example

Predict Returns using MC and TD





Monte Carol versus Temporal Difference

Bias/Variance Trade-Off

- MC Method
  - G_t is obtained based on actual returns (unbiased)
  - G_t depends on many (s,a,r) pairs (higher variance)

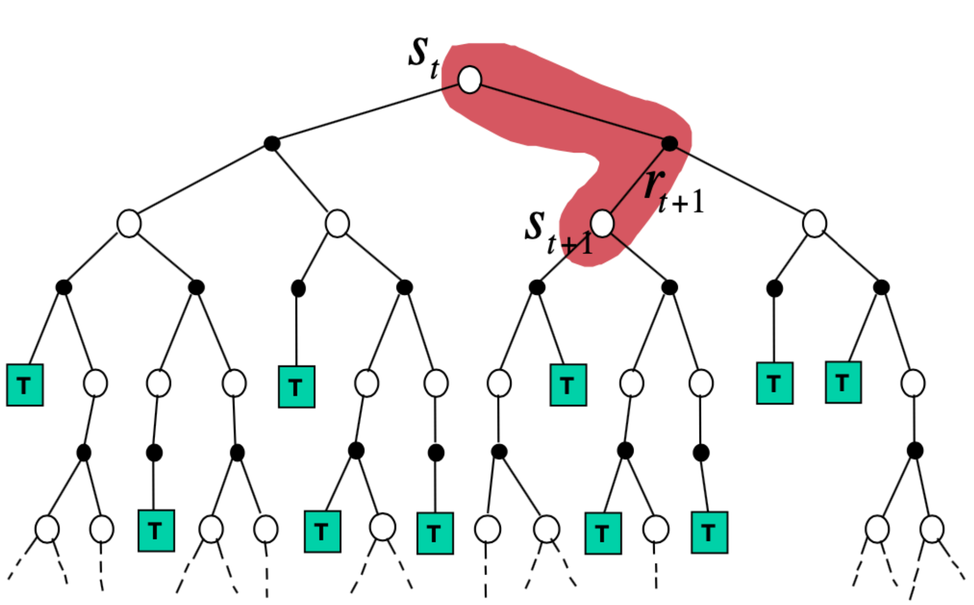
- TD Method
  - TD target $V(s)$ is based on estimated values (biased)
  - TD target only depends on one (s,a,r) pair (lower variance)

Multi-Step TD Method

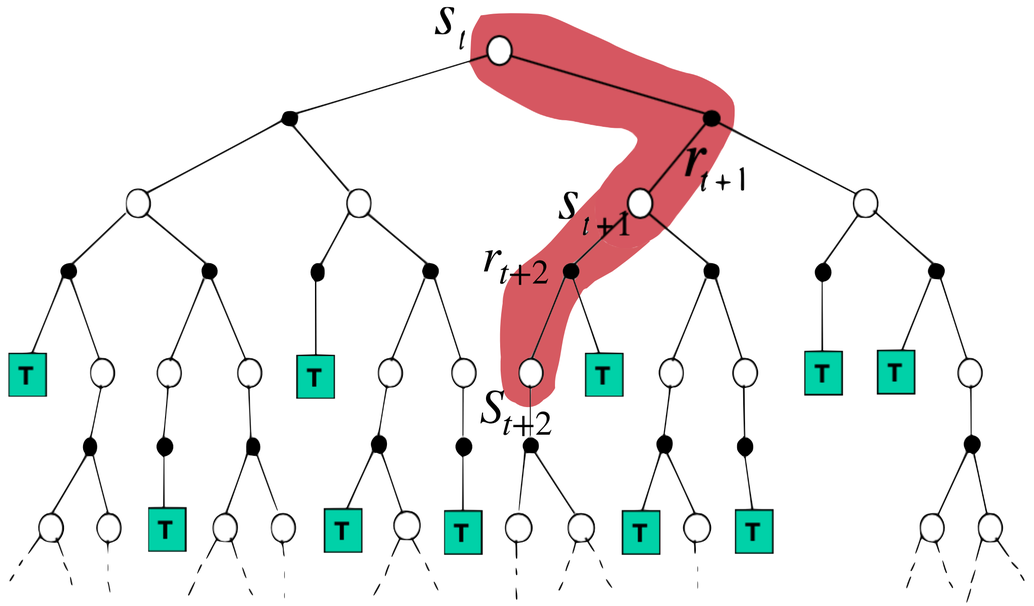
N-step Return

- Multi-step method — An intermediate method between MD and TD

1-step TD



2-step TD



Multi-Step TD Method

N-step return

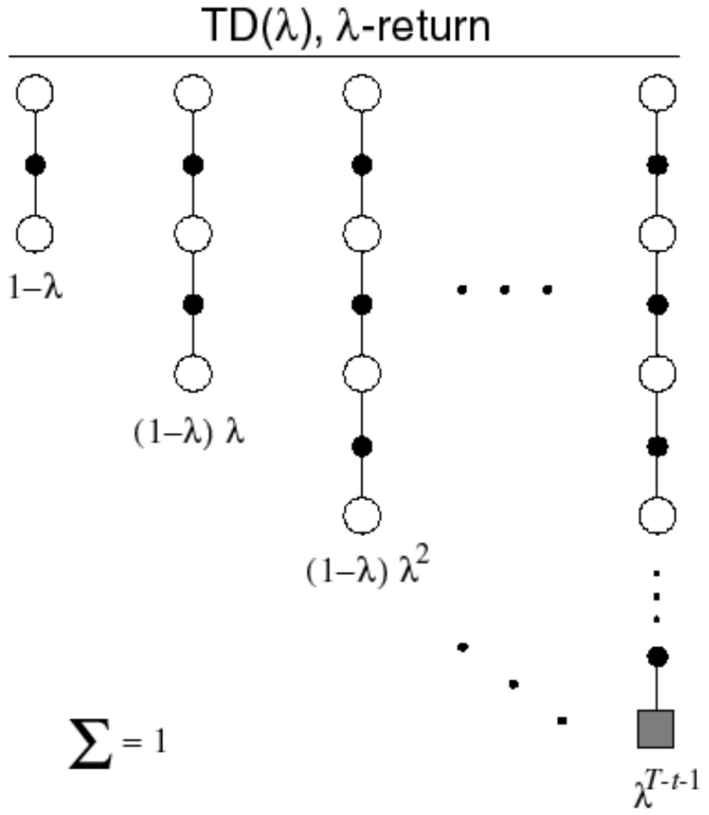
- N-step return : $G_t^n = \sum_{j=0}^{n} \gamma^j r_{t+j} + \gamma^n V(S_{t+n})$
  - E.g. 3-step return : $G_t^3 = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 V(S_{t+3})$
  - E.g. ∞-step return : $G_t^\infty = r_t + \gamma r_{t+1} + \ldots + \gamma^{T-1} R_T =$ MC return
- N-step TD learning uses n-step return as the TD target

λ-Return

Exponentially weighted multi-step return

- λ-return G_t^λ combines all G_t^n by using exponential weights :

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^n$$



- Wait for future step returns to compute G_t^λ
- Can only be computed from a complete episode
- Use λ-return to update :

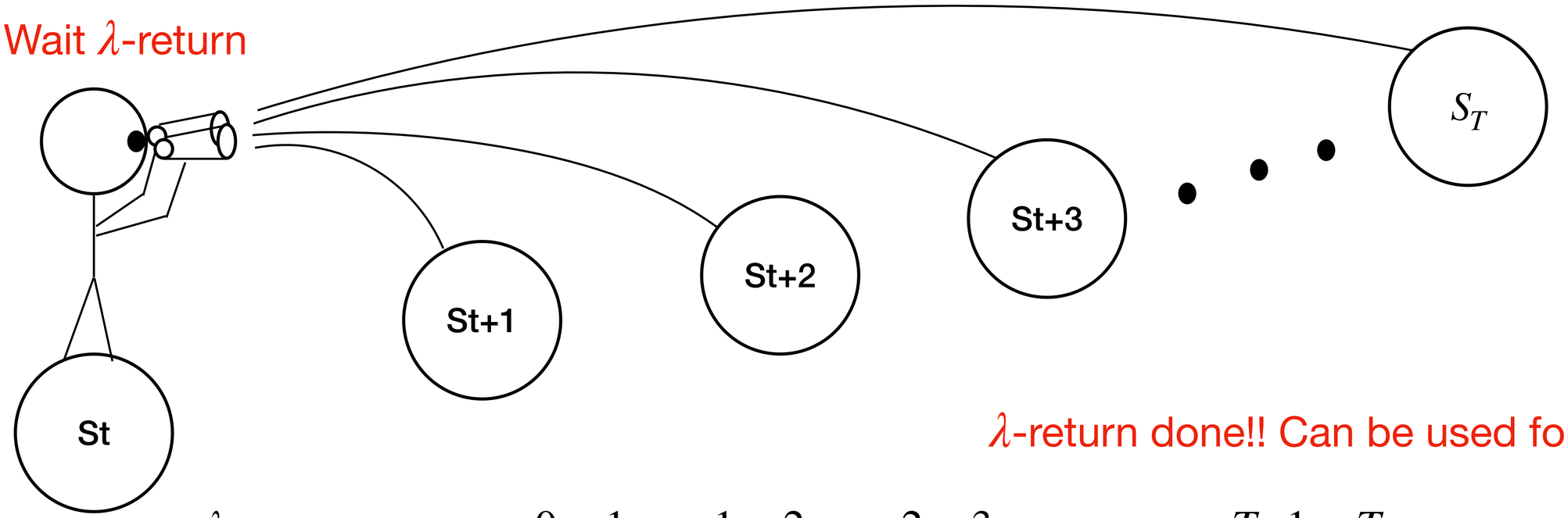
$$V(s_t) \leftarrow V(s_t) + \alpha(G_t^\lambda - V(s_t))$$

λ-Return

Exponentially weighted multi-step return

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^n$$

Wait λ-return

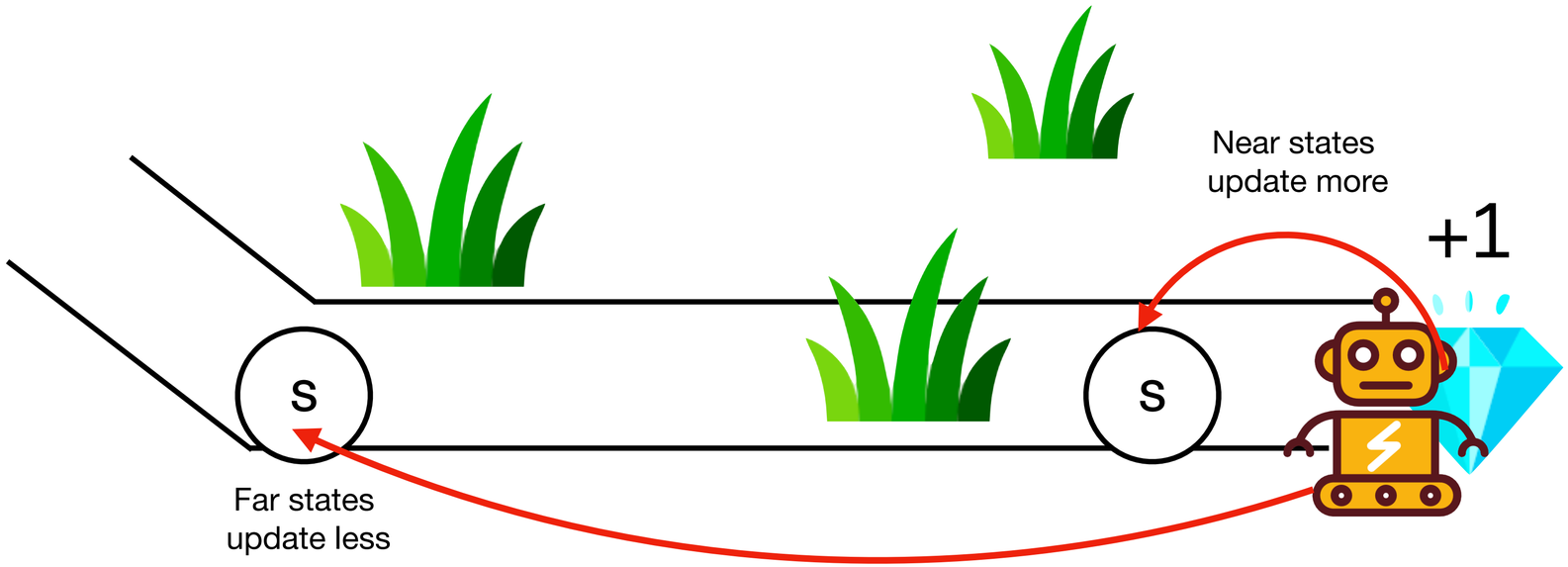


λ-return done!! Can be used for updates

$$G_t^\lambda = (1-\lambda)(\lambda^0 G_t^1 + \lambda^1 G_t^2 + \lambda^2 G_t^3 + \ldots + \lambda^{T-1} G_t^T)$$

λ-Return (Backward View)

Eligibility Trace



- Assign credits to most recent states
- Assign credits to most frequent states

λ-Return (Backward View)

Eligibility Trace

- Provide a mechanism to update by λ-return at every step from an incomplete sequence
- Creates $E(s)$ or $E(s, a)$ to hold the decayed values of value functions
- Updates the value function by :

$$\delta_t = R_t + \gamma V(s_{t+1}) - V(s_t)$$

$$V(s_t) \leftarrow V(s_t) + \alpha \delta_t E_t(s_t)$$

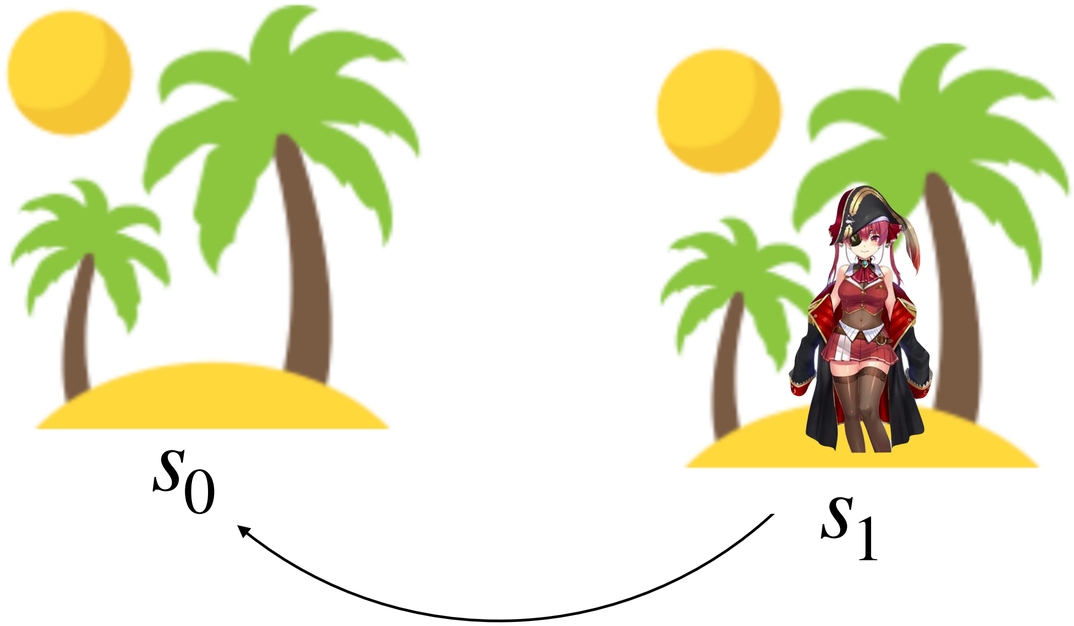
- Updates E by : $E_0(s) \leftarrow 0$

For the currently visiting state : $E_t(s) \leftarrow \gamma \lambda E_{t-1}(s) + 1$

For other states : $E_t(s) \leftarrow \gamma \lambda E_{t-1}(s)$

λ-Return (Backward View)

Eligibility Trace

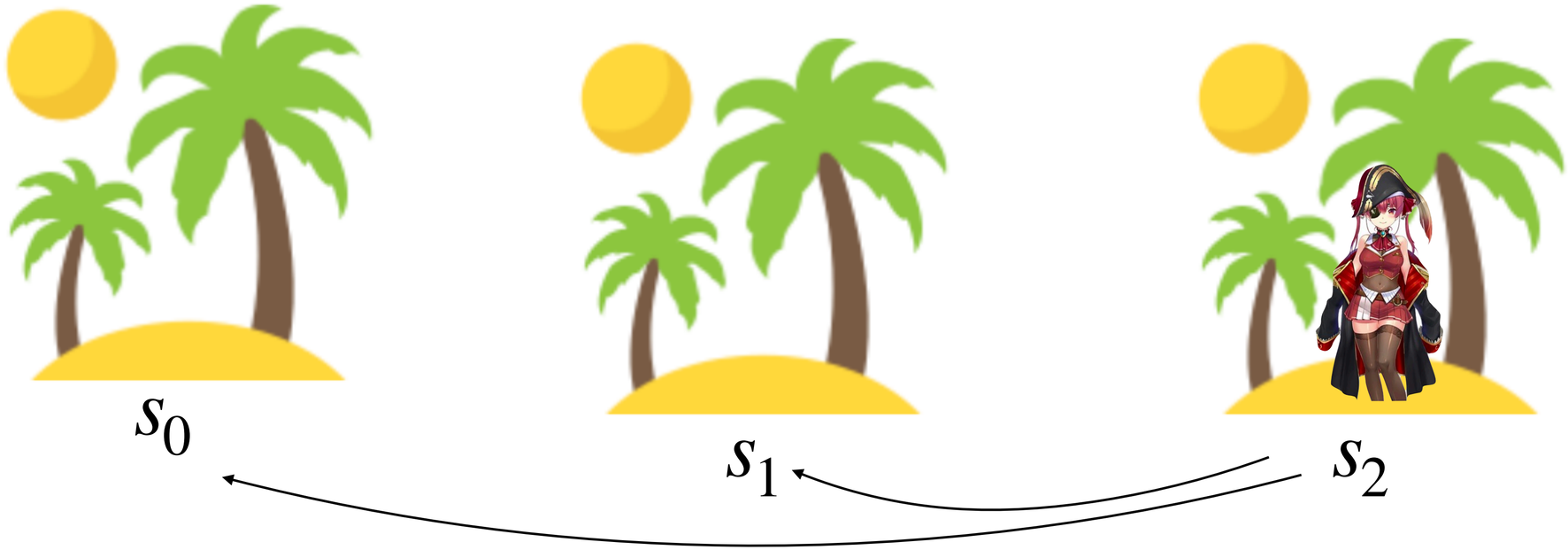


$$E_1(s_0) = \gamma\lambda E_0(s_0)$$
$$V(s_0) \leftarrow V(s_0) + \alpha\delta_1 E_1(s_0)$$

$$E_1(s_1) = \gamma\lambda E_0(s_1) + 1$$
$$V(s_1) \leftarrow V(s_1) + \alpha\delta_1 E_1(s_1)$$

λ-Return (Backward View)

Eligibility Trace



$$E_2(s_0) = \gamma\lambda E_1(s_0)$$
$$V(s_0) \leftarrow V(s_0) + \alpha\delta_2 E_2(s_0)$$

$$E_2(s_1) = \gamma\lambda E_1(s_1)$$
$$V(s_1) \leftarrow V(s_1) + \alpha\delta_2 E_2(s_1)$$

$$E_2(s_2) = \gamma\lambda E_1(s_2) + 1$$
$$V(s_2) \leftarrow V(s_2) + \alpha\delta_2 E_2(s_2)$$

λ-Return (Forward View v.s. Backward View)

Expansion of the TD (λ) Error Formula (Forward View)

$$G_t^\lambda - V(S_t) = -V(S_t) + (1-\lambda)(R_t + \gamma V(S_{t+1}))$$
$$+(1-\lambda)\lambda(R_t + \gamma R_{t+1} + \gamma^2 V(S_{t+2}))$$
$$+(1-\lambda)\lambda^2(R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 V(S_{t+3})) + \dots$$

$$R_t + \gamma V(S_{t+1}) - \cancel{\lambda R_t} - \lambda\gamma V(S_{t+1})$$
$$+\cancel{\lambda R_t} + \lambda\gamma R_{t+1} + \lambda\gamma^2 V(S_{t+2}) - \cancel{\lambda^2 R_t} - \cancel{\lambda^2\gamma R_{t+1}} - \lambda^2\gamma^2 V(S_{t+2})$$

$$G_t^\lambda - V(S_t) = -V(S_t) + (R_t + \gamma V(S_{t+1}) - \gamma\lambda V(S_{t+1}))$$
$$+\gamma\lambda(R_{t+1} + \gamma V(S_{t+2}) - \gamma\lambda V(S_{t+2}))$$
$$+(\gamma\lambda)^2(R_{t+2} + \gamma V(S_{t+3}) - \gamma\lambda V(S_{t+3})) \dots$$

λ-Return (Forward View v.s. Backward View)

Expansion of the TD (λ) Error Formula (Forward View)

$$
\begin{aligned}
G_t^\lambda - V(S_t) = &- V(S_t) + (R_t + \gamma V(S_{t+1}) - \gamma\lambda V(S_{t+1})) \\
&+\gamma\lambda(R_{t+1} + \gamma V(S_{t+2}) - \gamma\lambda V(S_{t+2})) \\
&+(\gamma\lambda)^2(R_{t+2} + \gamma V(S_{t+3}) - \gamma\lambda V(S_{t+3})) \ldots
\end{aligned}
$$

$$
\downarrow
$$

$$
\begin{aligned}
G_t^\lambda - V(S_t) = &(R_t + \gamma V(S_{t+1}) - V(S_t)) \\
&+\gamma\lambda(R_{t+1} + \gamma V(S_{t+2}) - V(S_{t+1})) \\
&+(\gamma\lambda)^2(R_{t+2} + \gamma V(S_{t+3}) - V(S_{t+2})) \ldots
\end{aligned}
$$

$$
\downarrow \quad \delta_t = R_t + \gamma V(s_{t+1}) - V(s)
$$

$$
= \boxed{\delta_t + \gamma\lambda\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \ldots}
$$

Compare this to the formula in page 41.

λ-Return (Forward View v.s. Backward View)

Simple Example

For Forward view

$$V(s_0) \leftarrow V(s_0) + \alpha(G_t^\lambda - V(s_0))$$

$$V(s_0) \leftarrow V(s_0) + \alpha(\delta_0 + \gamma\lambda\delta_1 + (\gamma\lambda)^2\delta_2)$$

For Backward view

T=0

$$E_0(s_0) = 1$$

$$V(s_0) \leftarrow V(s_0) + \alpha\delta_0 E_0(s_0)$$

T=1

$$E_1(s_0) = \gamma\lambda E_0(s_0) = \gamma\lambda$$

$$V(s_0) \leftarrow V(s_0) + \alpha\delta_1 E_1(s_0) = V(s_0) + \alpha\gamma\lambda\delta_1$$

T=2

$$E_2(s_0) = \gamma\lambda E_1(s_0) = (\gamma\lambda)^2$$

$$V(s_0) \leftarrow V(s_0) + \alpha\delta_2 E_2(s_0) = V(s_0) + \alpha(\gamma\lambda)^2\delta_2$$

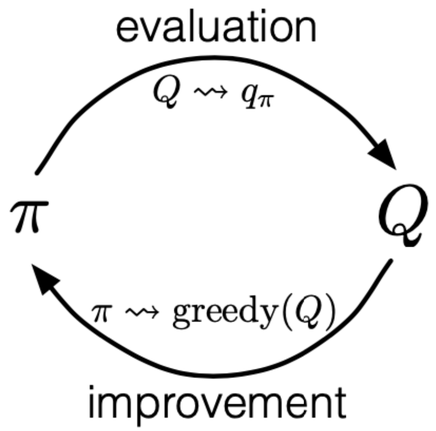
$$V(s_0) \leftarrow V(s_0) + \alpha(\delta_0 + \gamma\lambda\delta_1 + (\gamma\lambda)^2\delta_2)$$

Control Problem

Generalized Policy Iteration

Generalized Policy Iteration

- Policy evaluation : Any method for estimating V_π
- Policy improvement : Any method for improving π

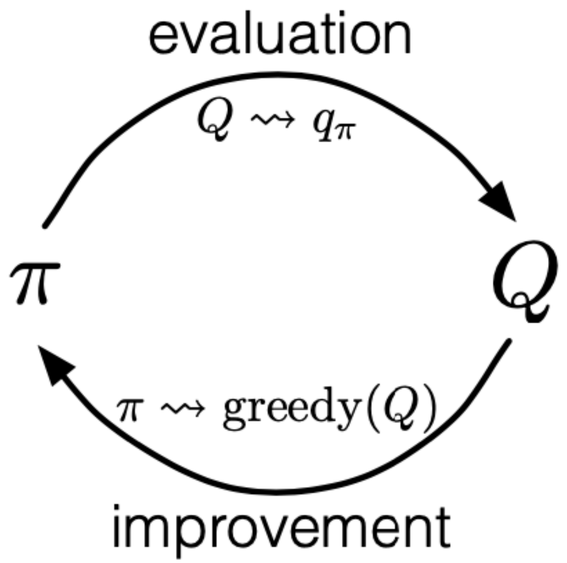


$$\pi_0 \rightarrow Q_{\pi_0} \rightarrow \pi_1 \rightarrow Q_{\pi_1} \rightarrow \pi_2 \ldots\ldots \rightarrow \pi_* \rightarrow Q_*$$

Monter Carlo Control

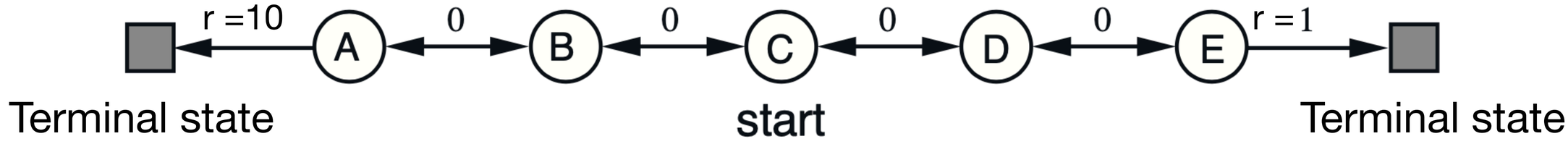
Monter Carlo Control

- Policy evaluation — **Monte-Carlo evaluation**
- Policy improvement — Greedy policy improvement ?



Greedy Policy Improvement

Problem



- Assume that in the first time, the agent reaches the right terminal state and gets r = 1, then updates the value estimate. Under such an scenario, the value function becomes:

$$V_{left} \leq V_{right}$$

- According to the greedy policy improvement scheme, the agent will always go to the right
- However, it's not the optimal policy !

ϵ-Greedy Exploration

Sometimes random

- Choose the greedy action a^* with probability $1 - \epsilon$
- Choose an random action a with probability ϵ

$$\pi(a \mid s) = \begin{cases} 1 - \epsilon & \text{if } a^* = \text{greedy}(Q(s, a)) \\ \epsilon & \text{otherwise} \end{cases}$$

ϵ-Greedy Exploration

Policy Improvement

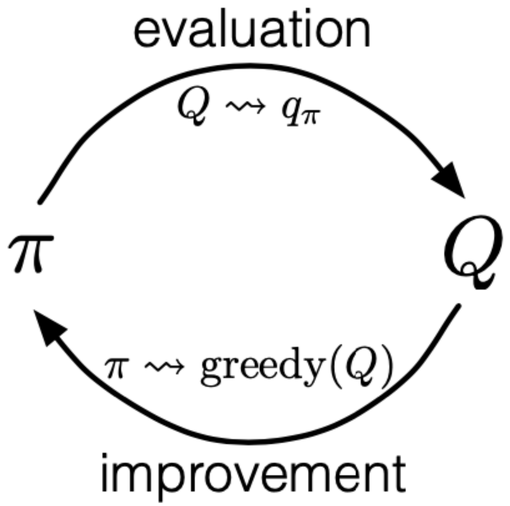
- For any ϵ-greedy policy π, the ϵ-greedy policy π' with respect to Q_π will lead to an improvement, $V_{\pi'}(s) \geq V_\pi(s)$

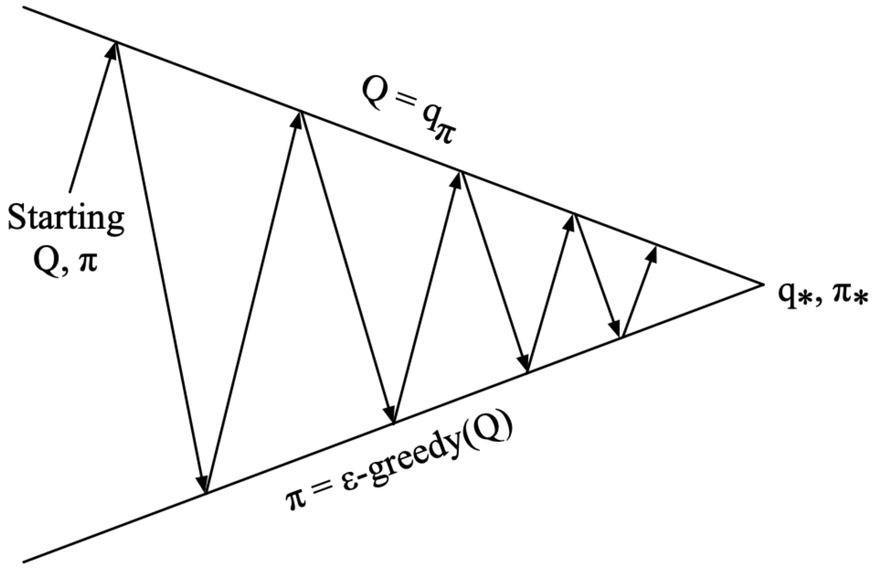
$$
\begin{aligned}
V_{\pi'}(s) &= \sum_{a \in A} \pi'(a \mid s) Q_\pi(s, a) \\
&= \frac{\epsilon}{m} \sum_{a \in A} Q_\pi(s, a) + (1 - \epsilon) max_{a \in A} Q_\pi(s, a) \quad \text{, m is the number of actions} \\
&\geq \frac{\epsilon}{m} \sum_{a \in A} Q_\pi(s, a) + (1 - \epsilon) \sum_{a \in A} \frac{\pi(a \mid s) - \frac{\epsilon}{m}}{1 - \epsilon} Q_\pi(s, a) \quad \text{(LINK)} \\
&= \sum_{a \in A} \pi(a \mid s) Q_\pi(s, a) = V_\pi(s)
\end{aligned}
$$

Monter Carlo Control

Monter Carlo Control

- Policy evaluation — Monte-Carlo evaluation
- Policy improvement — **ϵ-greedy policy improvement**



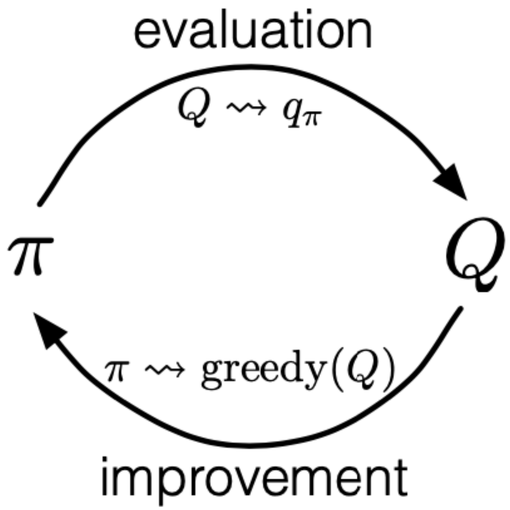


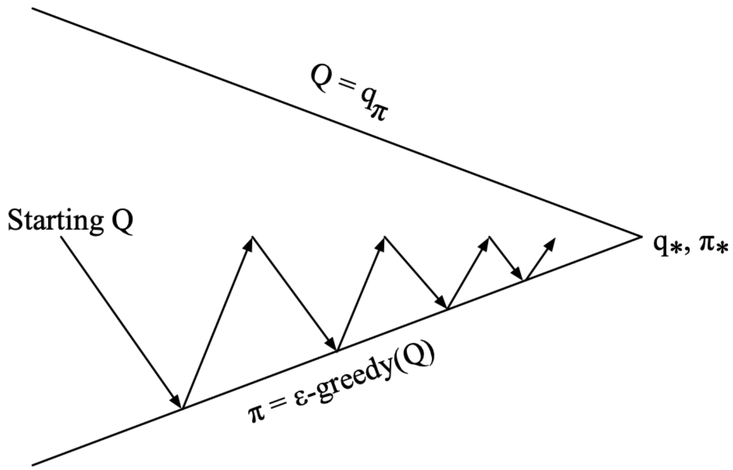
Monter Carlo Control

Monter Carlo Control

For every episode

- Policy evaluation — Monte-Carlo evaluation : $Q \sim q_\pi$
- Policy improvement — **ϵ-greedy policy improvement**





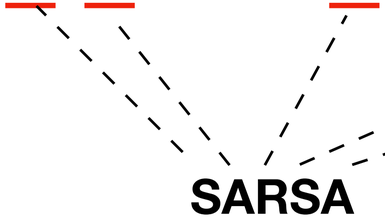
Outline

- Policy Iteration / Value Iteration
- MC and TD method
- **Sarsa**
- Q-learning

Sarsa

Temporal Difference Control

- Use TD method in policy evaluation
- Use ϵ-greedy policy improvement
- **On-policy** update
- Use incomplete sequences
- Update rule : $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$



Sarsa

Sarsa Algorithm

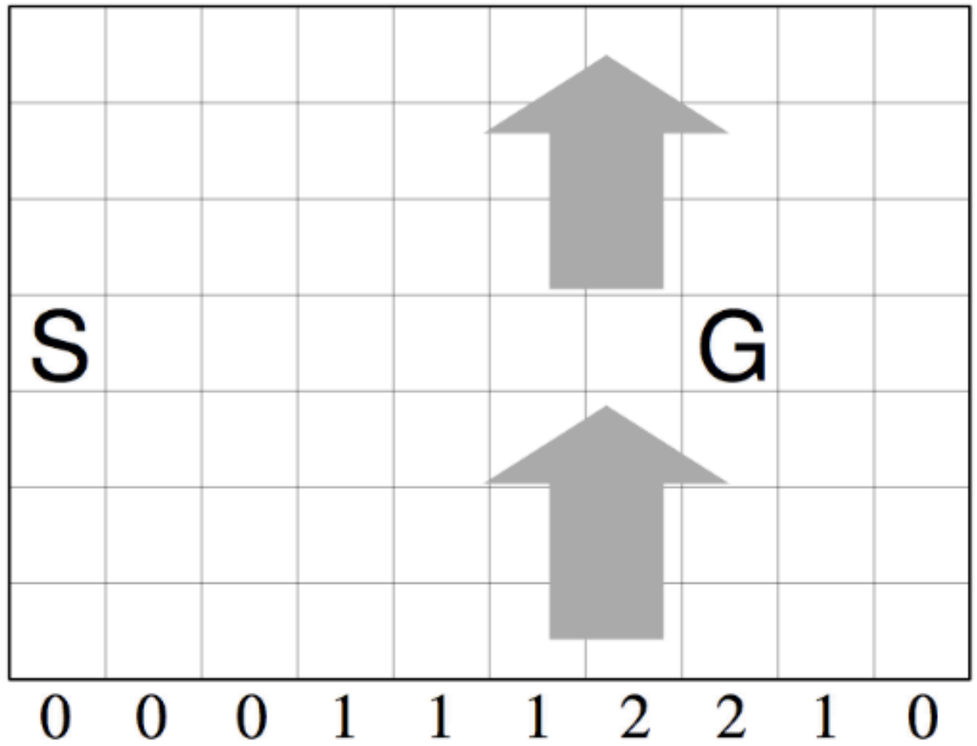
Sarsa (on-policy TD control) for estimating $Q \approx q_*$

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 policy derived from Q (e.g., ϵ-greedy)
        $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until S is terminal

Windy Grid World

Sample

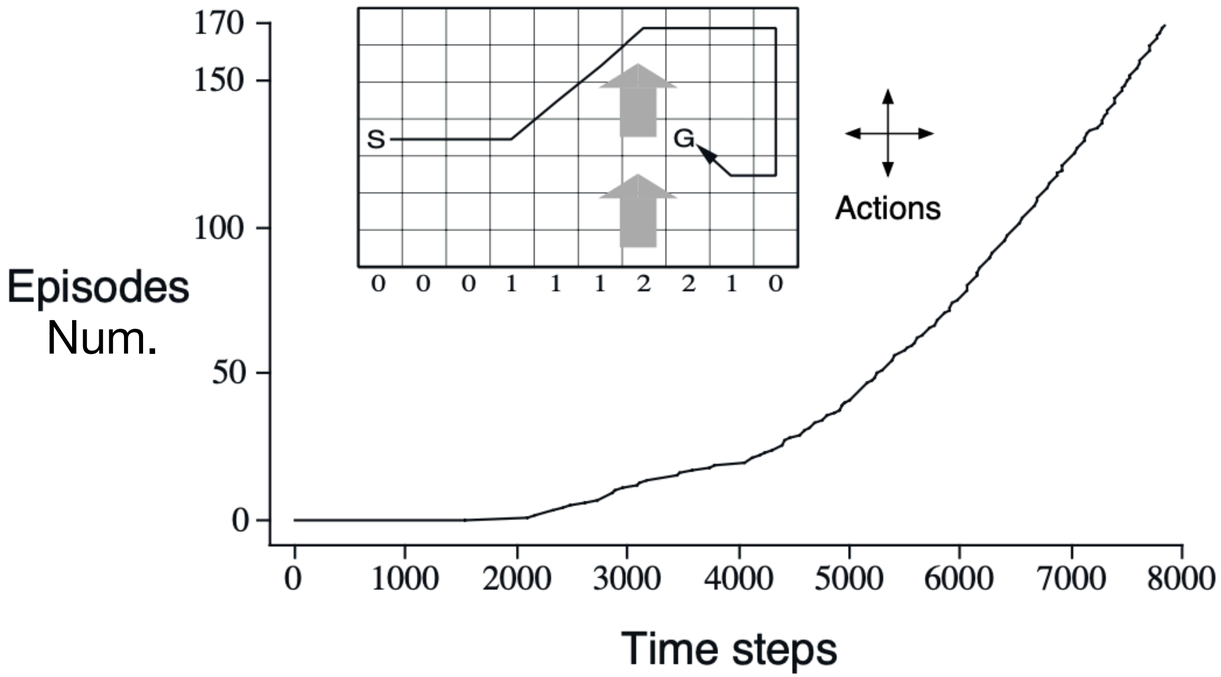
- 4 action {→, ↑, ←, ↓}
- All transitions get r = -1
- S : Start state
- G : Terminal state



Windy Grid World

Sample

- Monte Carlo methods cannot easily be used on this task because termination is not guaranteed for all policies.
- Need Step-by-Step backup



On-Policy v.s. Off-Policy

Difference

- Target Policy π— a policy that an agent is trying to learn
- Behavior Policy μ— a policy used by an agent for action selection
- On-policy learning — $\pi = \mu$
- Off-policy learning — $\pi \neq \mu$

Off-Policy Learning

Why off-policy learning is important

- Use old experience data generated by an old policy
- Can learn the optimal policy π while following a more exploratory policy μ
- Can learn from human replay data

Importance Sampling

What is Importance Sampling

Estimating the expected values of distribution p by samples from distribution q

$$\begin{aligned}\mathbb{E}_{x\sim p}[f(x)] &= \sum p(x)f(x) \\ &= \sum q(x)\frac{p(x)}{q(x)}f(x) \\ &= \mathbb{E}_{x\sim q}[\frac{p(x)}{q(x)}f(x)]\end{aligned}$$

Importance Sampling for Off-Policy

Monte Carlo target

- Use returns sample from μ to evaluate π
- G_t in Monte Carlo for off-policy update should be corrected as :

$$G_t^{\pi/\mu} = \frac{\pi(a_t \mid s_t)}{\mu(a_t \mid s_t)} \frac{\pi(a_{t+1} \mid s_{t+1})}{\mu(a_{t+1} \mid s_{t+1})} \cdots \frac{\pi(a_T \mid s_T)}{\mu(a_T \mid s_T)} G_t$$

- Modify update target : $V(s_t) \leftarrow V(s_t) + \alpha(G_t^{\pi/\mu} - V(s_t))$
- Variance become super large due to the importance sampling

Importance Sampling for Off-Policy

TD Target

- Use returns sampled from μ to evaluate π
- TD target $R_t + \gamma V(s_{t+1})$ for off-policy only need simple corrected as :

$$TD_{target} = \frac{\pi(a_t \mid s_t)}{\mu(a_t \mid s_t)}(R_t + \gamma V(s_{t+1}))$$

- Modify update target : $V(s_t) \leftarrow V(s_t) + \alpha(TD_{target} - V(s_t))$
- Variance is lower than Monte Carlo importance sampling

Outline

- Policy Iteration / Value Iteration
- MC and TD method
- Sarsa
- **Q-learning**

Q-Learning

Off-Policy Control Method

- **Off-policy** updates, no importance sampling needed
- Choose the next action as the greedy policy in next state
- Update rule :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

Q-Learning

Why Off-policy

- **Off-policy**

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

From Behavior Policy μ
(e.g. ϵ-greedy)

Choose max a
from Target Policy π

Q-Learning

Why no importance sampling

- **Although it is Off-policy, but we can**

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

No matter these collect from what policy, IT WILL BE SAME

Choose max a from Target Policy π

Assume it from target policy

Q-Learning

Q-learning Algorithm

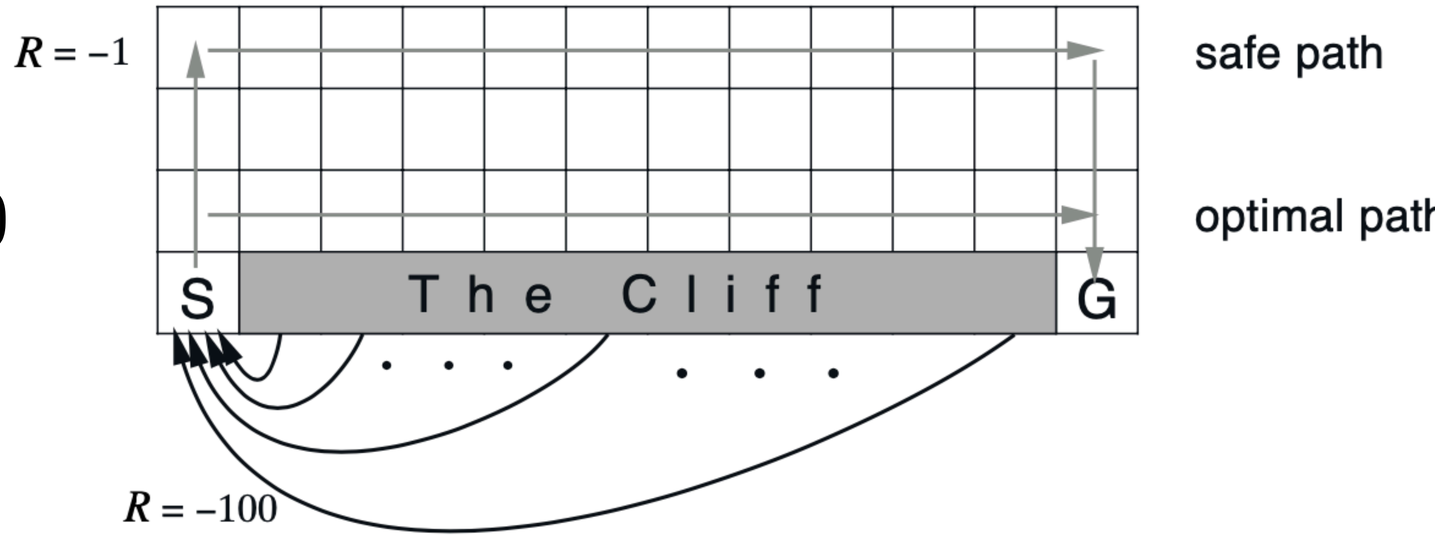
Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

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

Cliff Walking

Sample

- 4 action {→, ↑, ←, ↓}
- All transitions get r = -1
- Special cliff state : r = -100



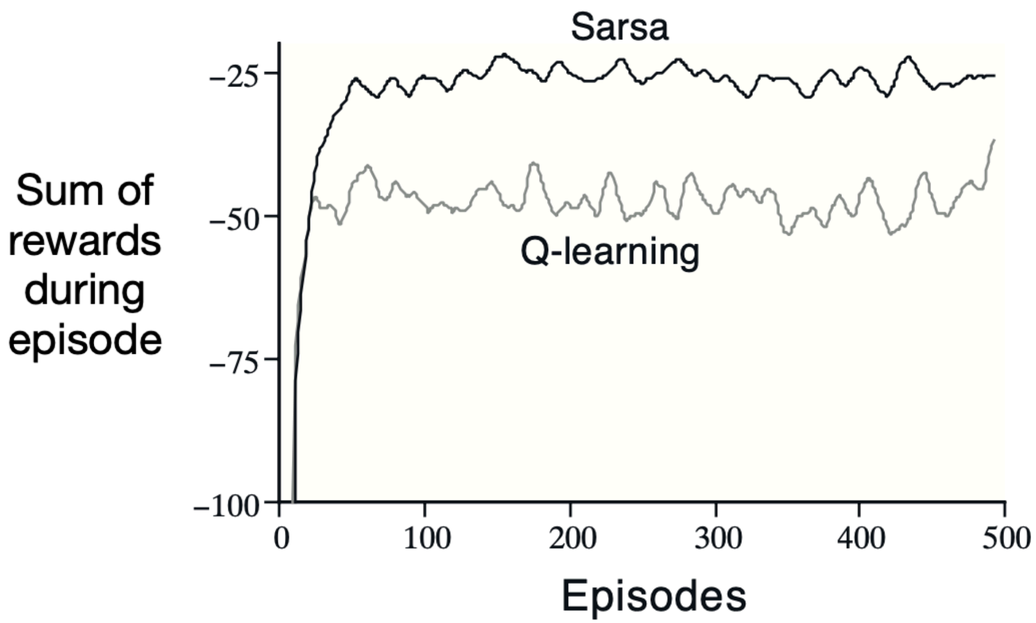
Cliff Walking

Result

- Q-learning :

  learn a policy for the optimal path; however, it occasionally fall off the cliff.

- Sarsa : learn the safe path.



Summary

- DP method to solved MDP : Policy Iteration, Value Iteration
- Model free method
  - Prediction : MC, TD
  - Control : Sarsa, Q-learning


すらめん：お疲れ様ー
fumiya。：どうだった？
Naoyuki Oya：。Ahoy！
THANK YOU!!!
#フリン航海記 でトゥイート