Model-free Prediction

Blink Sakulkueakulsuk

Four main Bellman Equations

Bellman Expectation Equation

value | Policy π

$$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \,|S_t = s, A_t \sim \pi(s)]$$
$$q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \,|S_t = s, A_t = a]$$

Bellman Optimality Equation

$$v^*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v^*(S_{t+1}) \,|S_t = s, A_t = a]$$
$$q^*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q^*\,(S_{t+1}, a') \,\middle|\, S_t = s, A_t = a)\right]$$

Prediction and Control

Prediction *

- A problem when we perform policy evaluation
- Estimating v_π or q_π try to ตีค่า policy

Control

- A problem when we perform policy optimization
- Estimating v^* or q^* try to have process to optimize policy

Evaluating Policy

$$\pi \geq \pi' \leftrightarrow v_{\pi}(s) \geq v_{\pi'}(s) \quad ,\forall s$$

One policy is better than or equal to another **if and only** if its value function is greater or equal to another.

Policy Evaluation

Given a policy, we want to estimate this

$$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \,|s, \pi]$$

We initialize all value to zero, and we iterate the equation above as an update

$$v_{k+1}(s) = \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \,|s, \pi] \quad , \quad \forall s$$

Note that

$$v_\pi(s) = \sum_a \pi(a|s) \sum_r \sum_{s'} p(r, s'|s, a)(r + \gamma v_\pi(s'))$$

If $v_{k+1}(s) = v_k(s), \forall s$, we solve v_π.

Policy Improvement

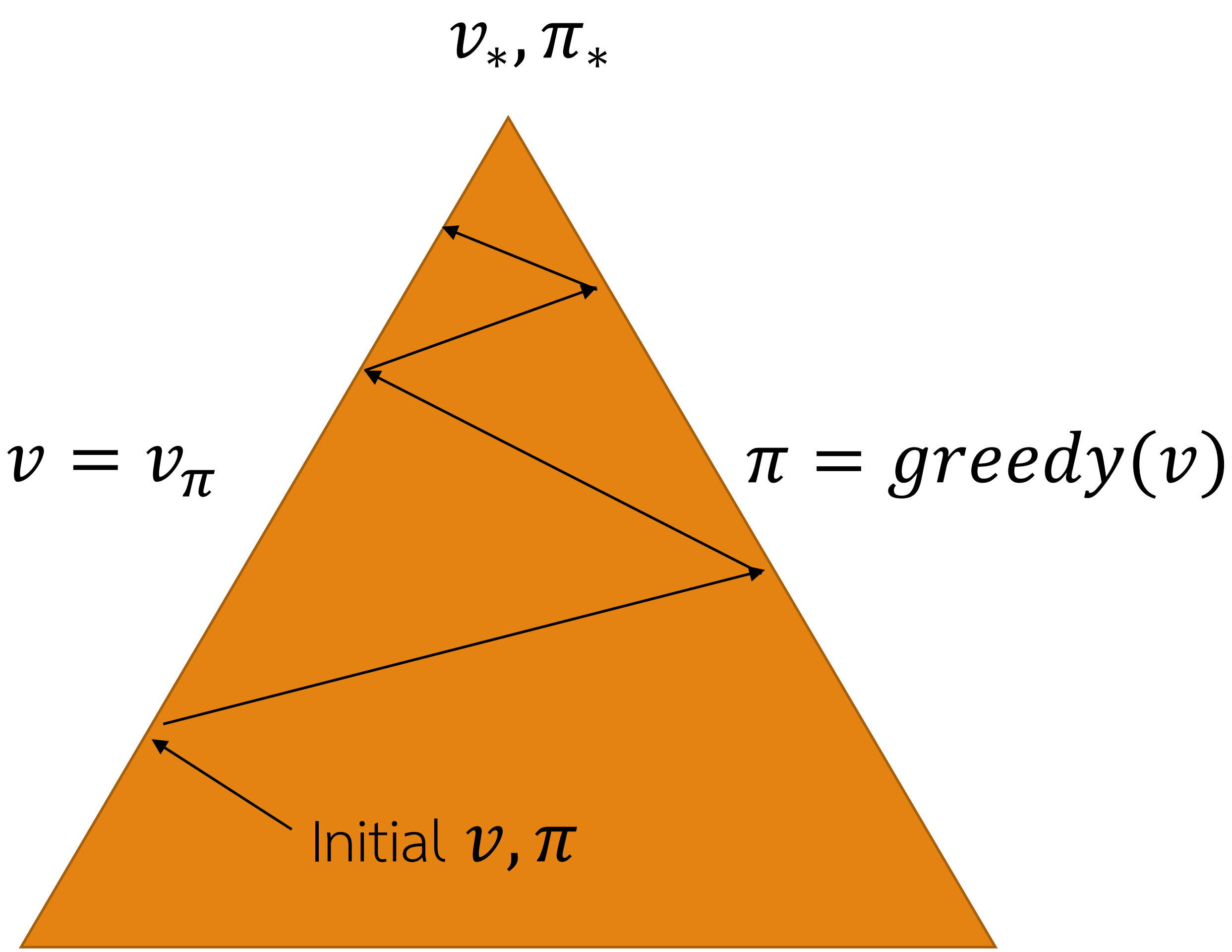
The example shows that we can improve a policy when we learn the value.

- In the example, we do not improve any policy

$$
\begin{aligned}
\forall s: \pi_{new}(s) &= argmax_a \, q_\pi(s,a) \\
&= argmax_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a]
\end{aligned}
$$

We use the new policy to evaluate, and repeat.

Policy Iteration



Starting with initial v, π

Loop do:

Perform policy evaluation,

$$v = v_\pi$$

Perform policy improvement,

$$\pi' \geq \pi$$

Value Iteration

Or we can learn the policy on the fly

- We update the policy every iteration

We can take the Bellman optimality equation as an update

$$v_{k+1}(s) \leftarrow \max_{a} \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \,|S_t = s, A_t = a] \quad , \quad \forall s$$

It is the same as **policy iteration** with k = 1. Basically, we improve the policy every iteration.

Model-Free Learning

Dynamic Model

What will happen if we don't know dynamic Model

$$v_\pi(s) = \sum_a \pi(a|s) \sum_r \sum_{s'} p(r, s'|s, a)(r + \gamma v_\pi(s'))$$

The value function above contains the dynamic of the problem

- $p(r, s'|s, a)$ tells the transition model of the problem. This is essentially the dynamic of the problem.

ถ้ารู้สิ่งนี้จะคำนวณทุกอย่างได้ ?

Key Question: What if we do not know that? Can the agent still learn?

- Spoiler Alerted: Yes, it can. (Otherwise, we wouldn't have this class)

Our Plan

Last Lecture

Markov

- We solved a **known MDP** by dynamic programming

This Lecture

- **Model-free prediction** to **estimate values** in an **unknown MDP**
- A brief touch of Deep Reinforcement Learning

Next Lecture

- **Model-free control** to **optimize values** in an **unknown MDP**

Monte Carlo Algorithm

Monte Carlo Algorithm

รอให้ run 1 ครั้งจนจบ แล้วมาดูว่าเรียนรู้อะไรบ้าง

To learn without a model, we can **sample the experience**

- Let's just act until the end and learn from that

Monte Carlo is a sampling method

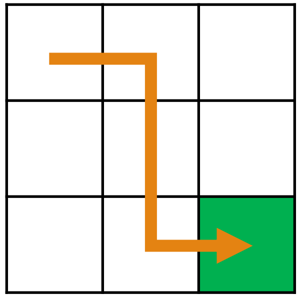
- Direct sampling of episodes
- Model-free
- No knowledge of MDP required

Monte Carlo Example

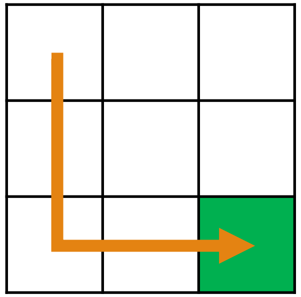
To sample the experience, given a policy π, the agent just plays out until termination.

For example, given a grid world and a robot that can move up, down, left, right, the samples are:

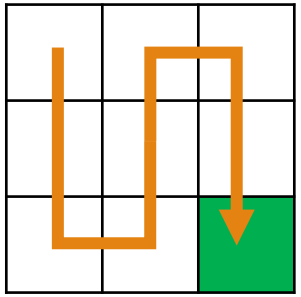
Episode 1 : run รอบ 1



Episode 2



Episode 3



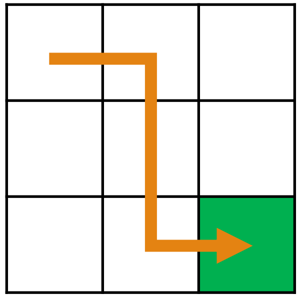
การที่ทำงานจนถึง goal state ได้แปลว่า รู้ Return

Monte Carlo Example

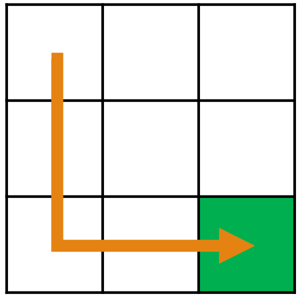
To sample the experience, given a policy π, the agent just plays out until termination.

For example, given a grid world and a robot that can move up, down, left, right, the samples are:

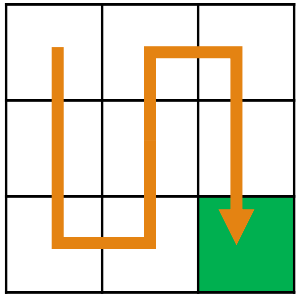
Episode 1



Episode 2



Episode 3



Monte-Carlo Policy Evaluation

Given the sequential decision problem, MC learns v_π from episodes under policy π

$$S_1, A_1, R_2, \dots, S_k \sim \pi$$

The return is calculated given an ending time T

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-1} R_T$$

เอา return ย้อนกลับมาคำนวณ state value ปัจจุบัน

The value function is then calculated from the expected return

$$v_\pi(s) = \mathbb{E}[G_t | S_t = s, \pi]$$

We can use **sample average return** instead of **expected return**

- This is called **Monte Carlo policy evaluation**

กำหนด sat of ๆ

ควร block timestep เพื่อป้องกัน ∞ timestep แล้วไม่ถึง goal

Part 1 Summary: Monte Carlo

Agent sampling ไปเรื่อยๆ
โดย reward จะถูก avg เพื่อให้ลู่เข้า
True value

ควรจะเดินเยอะไว้ก่อนให้ถึง goal

* เริ่มต้นจาก
การสุ่ม reward
ตาม DL ทั่วไป

Input: policy π, $num\_episodes$

Initialize $N(s, a) = 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$

Initialize $returns\_sum(s, a) = 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$

for $i \leftarrow 1$ ***to*** $num\_episodes$ **do**

  Generate an episode $S_0, A_0, R_1, \ldots, S_T$ using π → เก็บค่าทุกๆ timestep เพื่อคำนวณ

  timestep

  for $t \leftarrow 0$ ***to*** $T - 1$ **do**

    if (S_t, A_t) *is a first visit (with return G_t)* **then**

      $N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$ เพิ่ม $N(S_t, A_t)$

      $returns\_sum(S_t, A_t) \leftarrow returns\_sum(S_t, A_t) + G_t$

  end

end

$Q(s, a) \leftarrow returns\_sum(s, a) / N(s, a)$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$

return Q

อาจจะใช้หรือไม่
ใช้ Policy เหมือนกัน
ในทุก episode ก็ได้

Why first visit ?
• เราจะไม่คำนวณจังหวะที่กลับมา
ช่องเดิม

G_t + state 0 ≠ G_t + state 1
bc reward not equal except reward = 0

Agent ควรจะรู้ว่า target คืออะไร

value
or
q ก็ได้

หารจน. ครั้งที่ถูก visit

* มีโอกาสเดินวนลูป

https://miro.medium.com/v2/resize:fit:1400/1*bwwT1Jk-fz-n18m9iw9qRw.png

Blackjack Example

เฉลย HW 2

200 States

- Current hand (12-21)
- Dealer's hand (A-10)
- Our usable ace (yes, no)

Actions: stick, draw

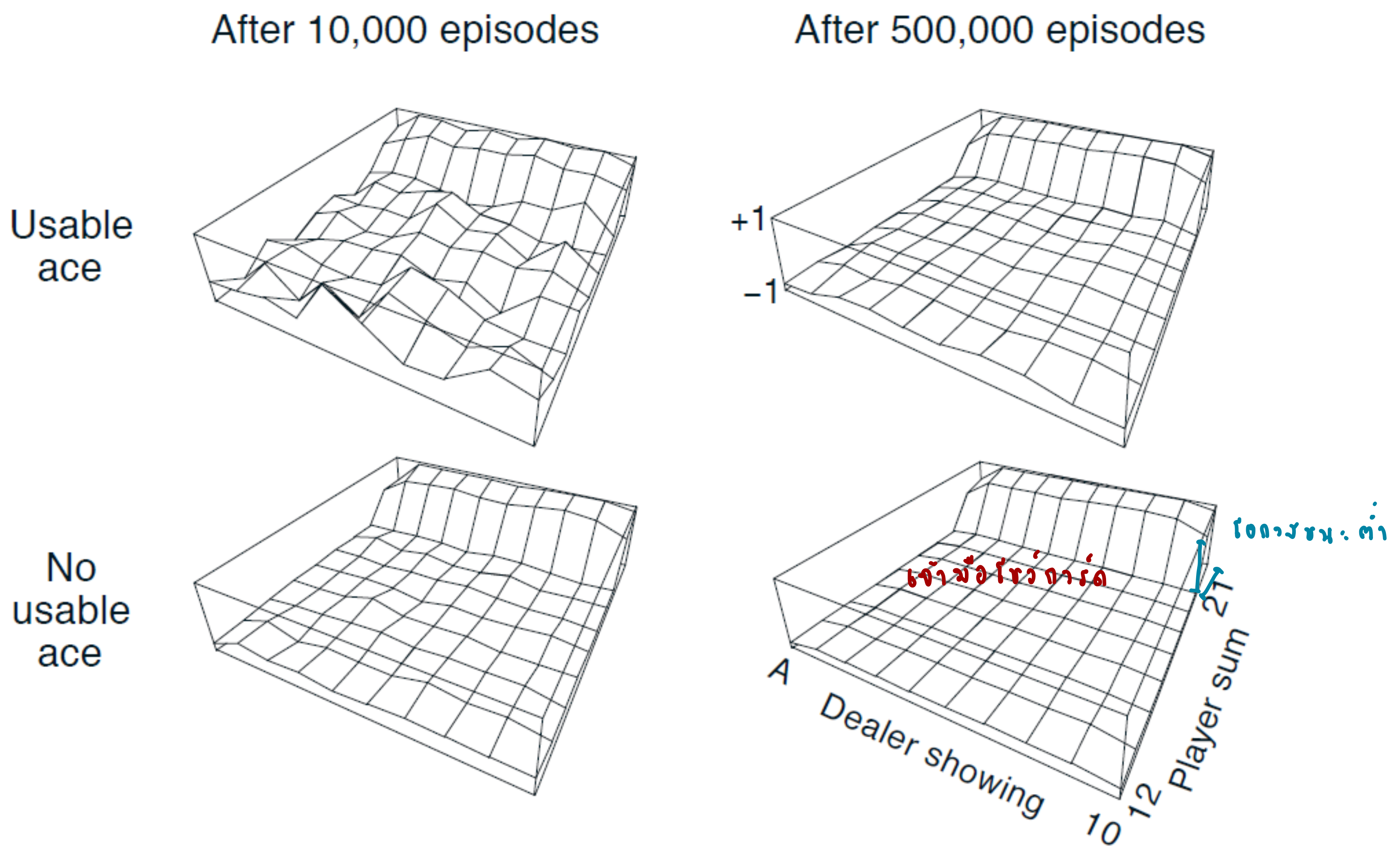
Reward

- Stick: +1 if win, 0 if draw, -1 if lose
- Draw: -1 if lose, 0 otherwise

Transitions: draw if sum < 12

กฎ

Goal of HW 2


After 10,000 episodes
After 500,000 episodes
Usable ace
No usable ace
+1
−1
A
Dealer showing
10
12
21
Player sum
เจ้ามือโชว์การ์ด
โอกาสชนะต่ำ

Function Approximation

DL + RL

from Monte carlo

Q (s , a) always add s , a

Value Function Approximation

In dynamic programming with MDP, we use **lookup tables** to learn value function.

- A mapping of state s to a value $v(s)$
- A mapping of state-action pair s, a to a state-action value $q(s, a)$

But we cannot do that for a large MDP

- Infeasible to store all information due to the curse of dimensionality
- **Too slow to learn** the value of each state **individually**
- States are often **not fully observable**

Value Function Approximation

For large MDP, we approximate the value function instead

$$v_{\mathbf{w}}(s) \approx v_{\pi}(s)$$
$$q_{\mathbf{w}}(s,a) \approx q_{\pi}(s,a)$$

แทนที่จะหาทางขวา

หาทางซ้ายแทน

The agent will learn the value and update (learnable) parameter $\mathbf{w}$. vector

- Monte Carlo Algorithm
- Temporal Difference Learning

เอาค่าที่ได้ (goal, return) มาเพื่อปรับให้ __ ?

Agent State Update

For large MDP, if the environment state is not fully observable, we can update the agent state by

$$S_t = u_\omega(S_{t-1}, A_{t-1}, O_t)$$

with parameters $\omega \in \mathbb{R}^n$

Linear Value Function Approximation

Let’s start with a simple, useful, and special case: a linear function

A state is represented by a **feature vector**

$$\mathbf{x}(s) = \begin{bmatrix} x_1(s) \\ \vdots \\ x_m(s) \end{bmatrix}$$

ใส่ s ไป เผิ่ง มี size m

$\mathbf{x} : S \Rightarrow \mathbb{R}^m$ is a mapping from an agent state to features

Linear Value Function Approximation

One way to approximate a value function is to find a linear combination of features.

$$v_{\mathbf{w}}(s) = \mathbf{w}^{\mathrm{T}}\mathbf{x}(s) = \sum_{j=1}^{n} w_j x_j(s)$$

ไ ถูก Model จนฐาน:

linear Regression

To learn the function, we can find an objective loss.

$$L(\mathbf{w}) = \mathbb{E}_{S\sim d}\left[\left(v_\pi(S) - \underbrace{\mathbf{w}^{\mathrm{T}}\mathbf{x}(s)}_{\text{things we want to learn}}\right)^2\right]$$

Linear Value Function Approximation

$$L(\mathbf{w}) = \mathbb{E}_{S\sim d}\left[\left(v_\pi(S) - \mathbf{w}^{\mathrm{T}}\mathbf{x}(s)\right)^2\right]$$ → เทียบ function ที่ ถึง Return

Then we can use stochastic gradient descent to update the parameter

- Stochastic gradient descent is a gradient descent algorithm that calculates loss of 1 input at a time
- Since we sample the problem, we can update the parameter using SGD

$$\nabla_{\mathbf{w}} v_{\mathbf{w}}(S_t) = \mathbf{x}(S_t) = \overset{\text{feature vector}}{\mathbf{x}_t} \Rightarrow \qquad \Delta\mathbf{w} = \overset{\text{step size}}{\alpha}\underbrace{\left(v_\pi(S_t) - v_{\mathbf{w}}(S_t)\right)}\mathbf{x}_t$$

Update = step-size x prediction error x feature vector

Function Approximation Example

Let's consider a bandit problem. If we want to learn the value function, we can find it through the objective loss function.

- q is a parametric function with parameters $\mathbf{w}$. For example, q could be a neural network.

vπ Parametic eq.

$$L(\mathbf{w}) = \frac{1}{2}\mathbb{E}[(R_{t+1} - q_{\mathbf{w}}(S_t, A_t))^2]$$

ใน loss ก็คือ

Then we update the weight by:

พจน์เกี่ยวกับ gradiant

$$\begin{aligned}
\mathbf{w}_{t+1} &= \mathbf{w}_t - \alpha \nabla_{\mathbf{w}_t} L(\mathbf{w}_t) \\
&= \mathbf{w}_t - \alpha \nabla_{\mathbf{w}_t} \frac{1}{2}\mathbb{E}[(R_{t+1} - q_{\mathbf{w}}(S_t, A_t))^2] \\
&= \mathbf{w}_t + \alpha \mathbb{E}\left[(R_{t+1} - q_{\mathbf{w}}(S_t, A_t)) \nabla_{\mathbf{w}_t} q_{\mathbf{w}}(S_t, A_t)\right]
\end{aligned}$$

Function Approximation Example

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\mathbb{E}\left[(R_{t+1} - q_{\mathbf{w}}(S_t, A_t))\nabla_{\mathbf{w}_t} q_{\mathbf{w}}(S_t, A_t)\right]$$

If we use a linear function approximation,

$$q(s, a) = \mathbf{w}^{\mathrm{T}}\mathbf{x}(s, a)$$

ฟังๆ

สรุป : prediction problem in model free en. so we sampling until get return , change

The gradient becomes

$$\nabla_{\mathbf{w}_t} q_{\mathbf{w}}(S_t, A_t) = \mathbf{x}(s, a)$$

Then the SGD update is

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha(R_{t+1} - q_{\mathbf{w}}(S_t, A_t))\mathbf{x}(s, a)$$

Linear Update = step-size x prediction error x ~~feature vector~~

gradiant

Non-linear Update = step-size x prediction error x ~~feature vector~~

Monte Carlo → Problem : รอ Agent ทำงานจนจบ , ระหว่าง run เรียนรู้อะไรไม่ได้เลย

MC algorithm can be used to learn value function

- However, the algorithm must wait until the end of an episode to learn something
- Return can have high variance

Alternatives?

- Yes! Temporal Difference Learning → เรียนรู้ผ่านช่วงว่างของเวลา

  ช่วงเกี่ยวกับเวลา

Temporal Difference Learning

Temporal Difference Learning

Given a Bellman equation,

$$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \,|S_t = s, A_t \sim \pi(S_t)]$$

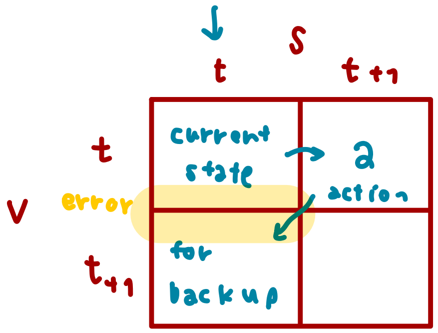
can change form to update value

We can update values by iterating using the following update function,

$$v_{k+1}(s) = \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \,|S_t = s, A_t \sim \pi(S_t)]$$

Temporal Difference Learning

Monte carlo - ทำจนจบ ค่อยเรียนรู้



$$v_{k+1}(s) = \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \,|S_t = s, A_t \sim \pi(S_t)]$$

Instead of calculating the expected return, we can sample the value.

if แทนตรงๆ noise จะสูง จึงเอามาทำ TDL, we avg out for ค่อยๆเรียนรู้

เช่น if ถูก 10 ล้าน and not using TDL ทุก state = 10 ล้าน. คำสาป curse

we want to know value of S_t after we action sth

$$v_{t+1}(S_t) = R_{t+1} + \gamma v_t(S_{t+1})$$

But instead of updating everything on the noisy value, we can update the value a little bit instead.

คูณกับ factor sth

value that sampling - Vt

$$v_{t+1}(S_t) = v_t(S_t) + \alpha_t \underbrace{(R_{t+1} + \gamma v_t(S_{t+1}) - v_t(S_t))}_{\text{หา error between timestep}}$$

MC Prediction vs TD Prediction

พยายามจะเรียนรู้ v_π จาก policy π

Prediction problem: learn v_π online from experience under policy π

Monte Carlo:

- Update value $v_n(S_t)$ with respect to sample return G_t

จบ episode ค่อย update

$$v_n(S_t) = v_n(S_t) + \alpha(G_t - v_n(S_t))$$

v เดิม

return (เริ่ม - จบ)

Temporal-difference learning:

- Update value $v_t(S_t)$ with respect to estimated return $R_{t+1} + \gamma v(S_{t+1})$

$$v_{t+1}(S_t) = v_t(S_t) + \alpha_t(R_{t+1} + \gamma v_t(S_{t+1}) - v_t(S_t))$$

update ทุก timestep

Temporal Difference Learning

$$v_{t+1}(S_t) = v_t(S_t) + \alpha(R_{t+1} + \gamma v_t(S_{t+1}) - v_t(S_t))$$

Target

Temporal Difference Learning

$$v_{t+1}(S_t) = v_t(S_t) + \alpha(R_{t+1} + \gamma v_t(S_{t+1}) - v_t(S_t))$$

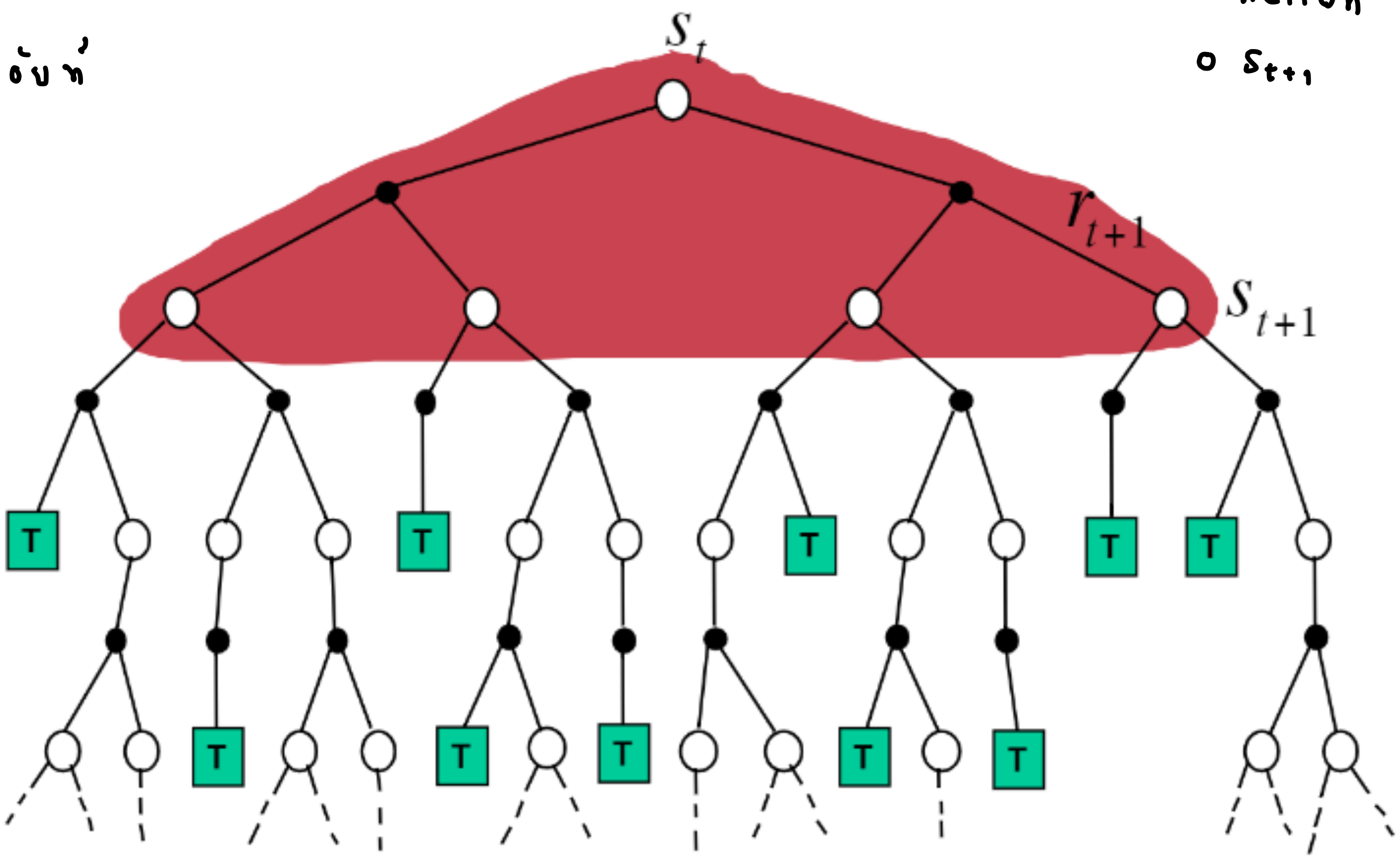
TD Error

Dynamic Programming Backup

$$v(S_t) \leftarrow \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid A_t \sim \pi(S_t)\right]$$

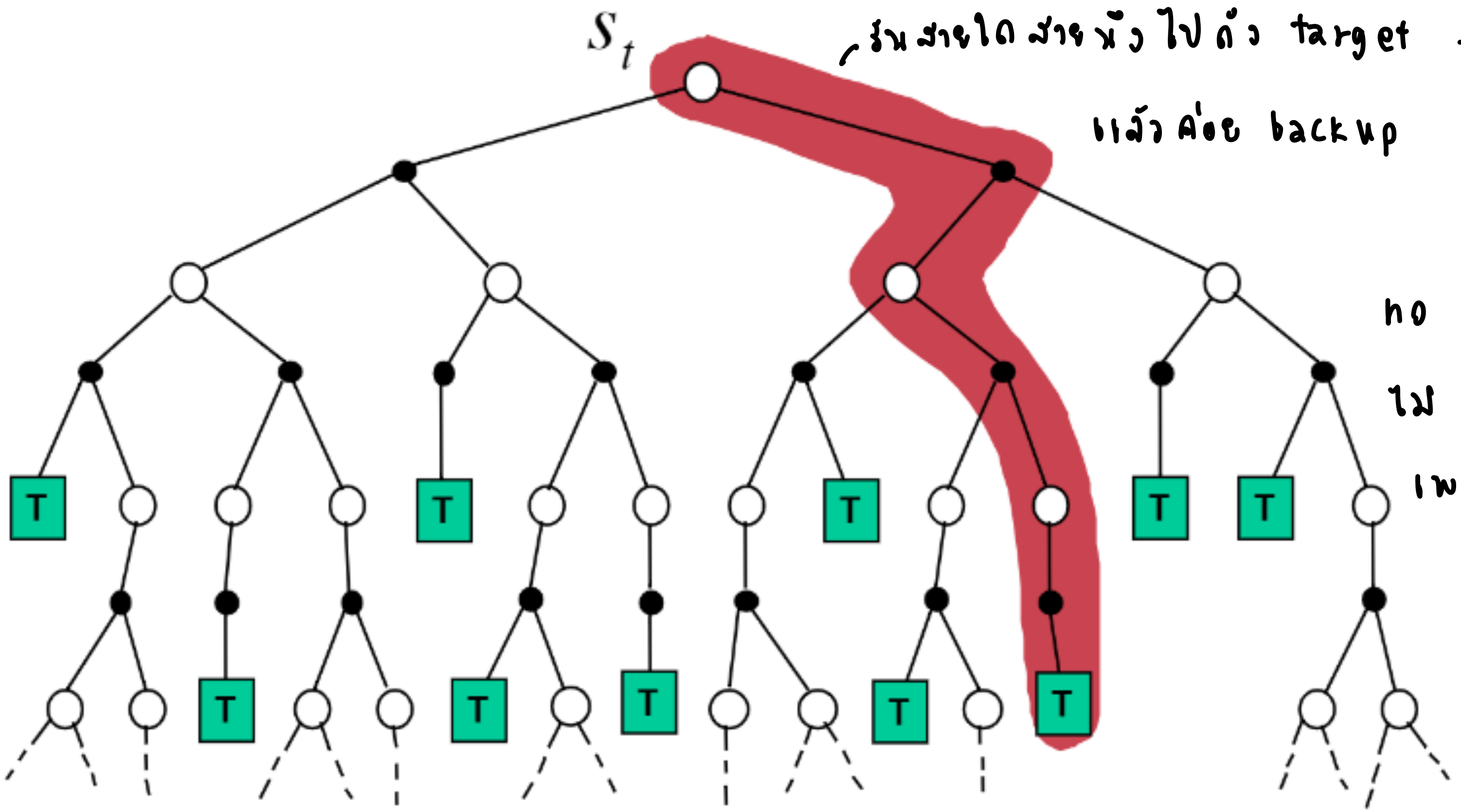
ทำครอบคลุมทุกอันที่
เป็นไปได้ **

• Action
○ S_{t+1}



Monte-Carlo Backup

$$v(S_t) \leftarrow v(S_t) + \alpha\left(G_t - v(S_t)\right)$$

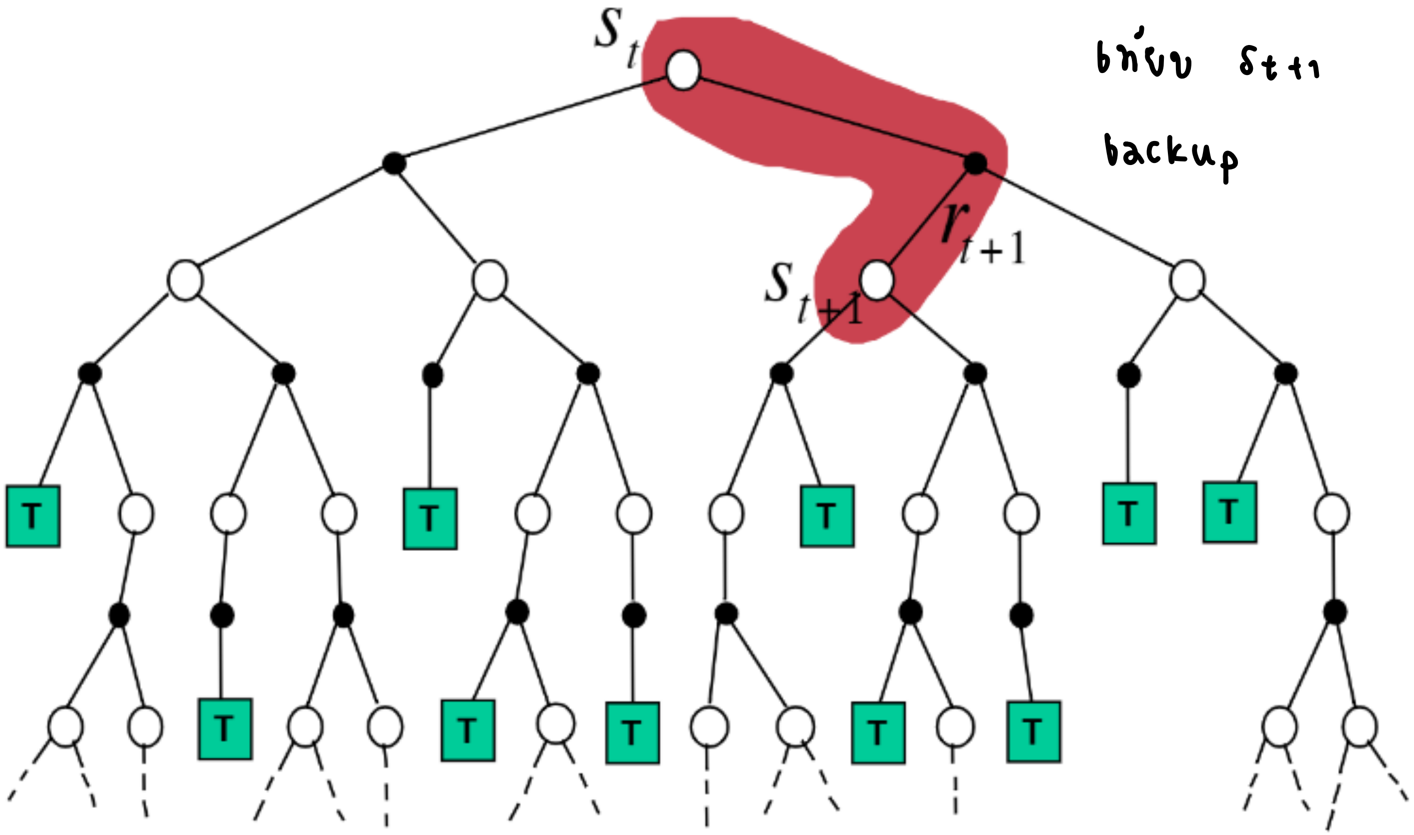


รันสายใดสายหนึ่งไปถึง target
แล้วค่อย backup } เพื่อ goal

หา S_{t+1}
ไม่ Bootstrap
เพราะใช้ ___ เลย

Temporal-Difference Backup

$$v(S_t) \leftarrow v(S_t) + \alpha \left(R_{t+1} + \gamma v(S_{t+1}) - v(S_t) \right)$$



เท่ยบ S_{t+1}

backup

Bootstrapping and Sampling

Bootstrapping → การสรุปข้อมูล

- Use the estimate of the next state to update
- DP and TD use
- MC does not

Sampling

- Update samples an expectation
- MC and TD sample
- DP does not

Temporal Difference Learning

We can also learn action values by updating value $q_t(S_t, A_t)$ with respect to estimated return $R_{t+1} + \gamma v(S_{t+1}, A_{t+1})$

$$q_{t+1}(S_t, A_t) = q_t(S_t, A_t) + \alpha_t(R_{t+1} + \gamma q_t(S_{t+1}, A_{t+1}) - q_t(S_t, A_t))$$

This algorithm is called SARSA, because it uses $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$

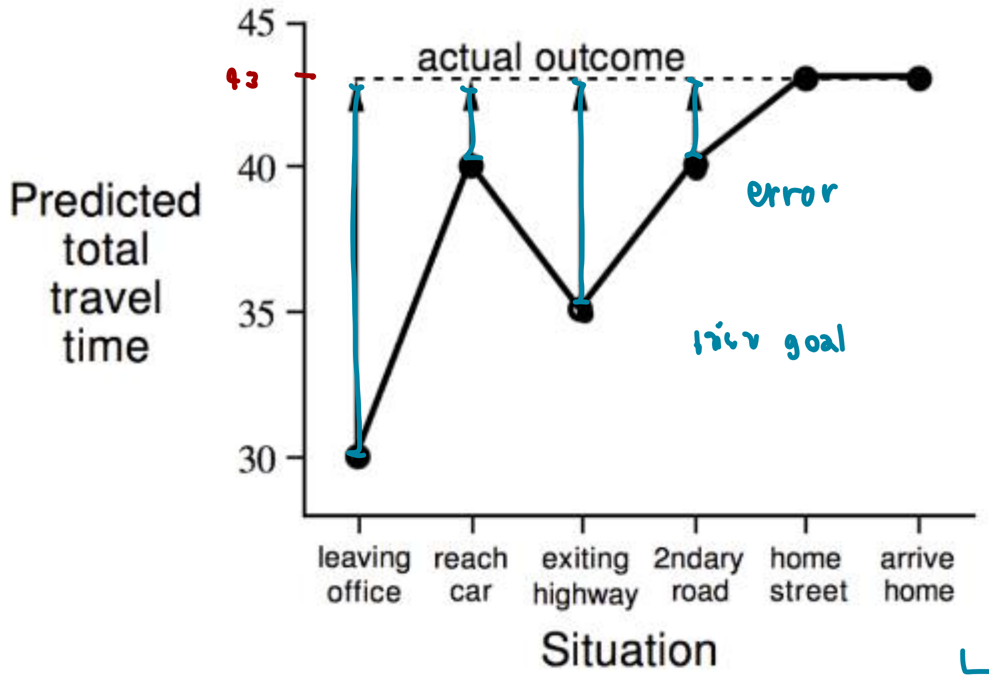
p_{TDL}

listen !!

State	Elapsed Time (minutes)	Predicted Time to Go	Predicted Total Time
leaving office	0	30	30
reach car, raining	5	35	40
exit highway	20	15	35
behind truck	30	10	40
home street	40	3	43
arrive home	43	0	43

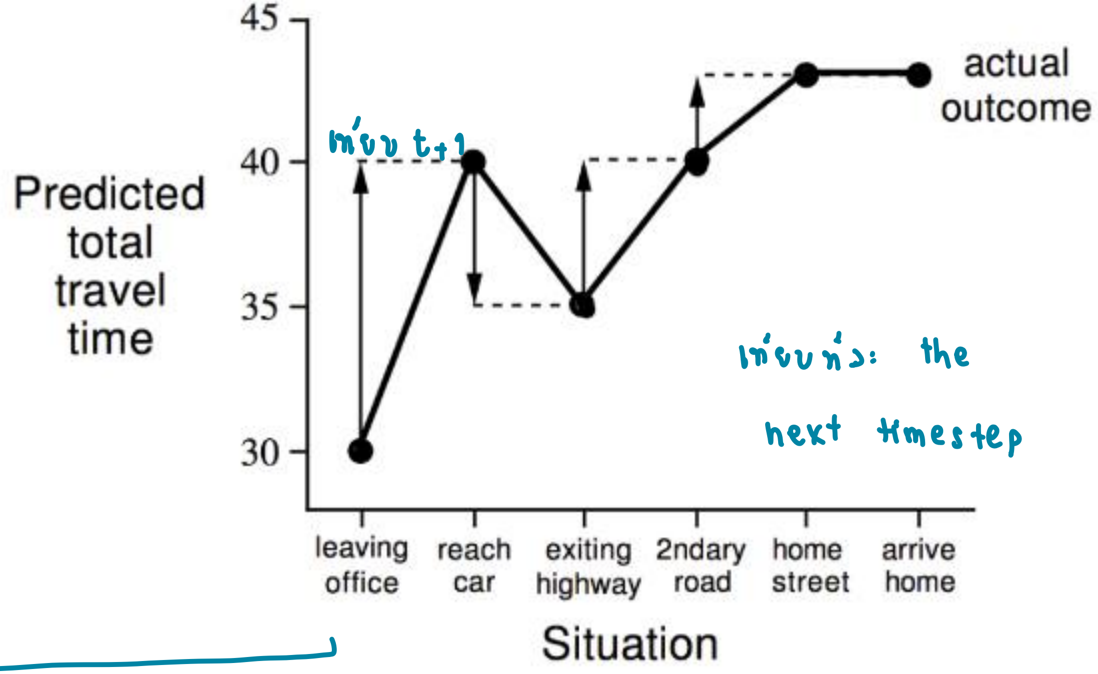
Changes recommended by Monte Carlo methods (α=1)

error from monte carlo



Changes recommended by TD methods (α=1)

error from tbL given $\alpha = 1$



ดูเขา true value

TD vs MC

TD can learn **before** knowing the final outcome

- TD learns online every step.
- MC must wait until the end of the episode.

สามารถใช้ในการเรียนรู้แบบ Continous environment ได้ แต่ MC มีจุดสิ้นสุดเพราะ.

TD can learn **without** the final outcome ต้อง terminate ก่อนเรียนรู้

- TD can learn even if the full interaction sequence is incomplete. MC must use the complete sequence.
- TD can be used in a continuing environment. MC environment must terminate.

Bias-Variance Trade-off

bias น้อย

Given MC return,

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-1} R_T$$

This is an **unbiased** estimation of $v_\pi(S_t)$

bias มาก ต่อ timestep ถัดไป

Given TD target,

$$R_{t+1} + \gamma v_t(S_{t+1})$$

This is a **biased** estimation of $v_\pi(S_t)$

MC : bias ↓ variance ↑

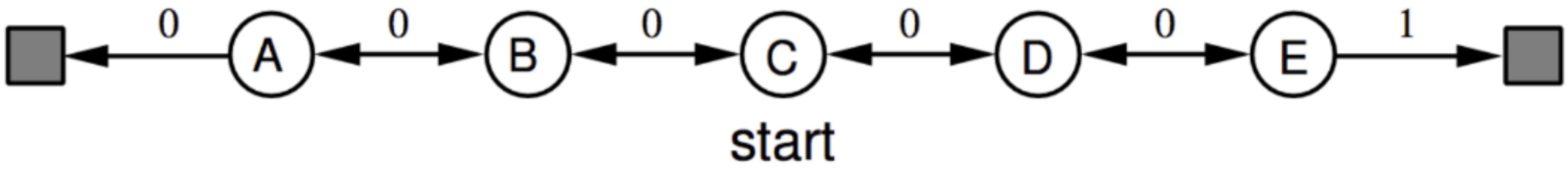
TD : " ↑ " ↓

Bias-Variance Trade-off

However, TD has **lower variance**

- MC return estimates from many random actions, transitions, and rewards
- TD target estimates from one random action, transition, and reward
- Variation of action, transition, and reward makes the TD target have less variance

Random Walk Example



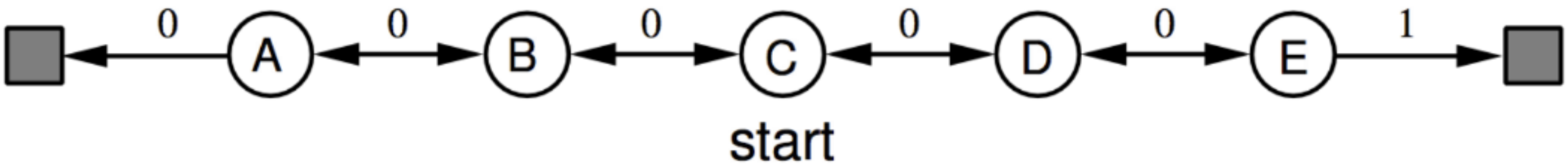
This game has 7 states, with uniform random transitions (50% left, 50% right)

เท่ากัน

We can see that the true values of all states are:

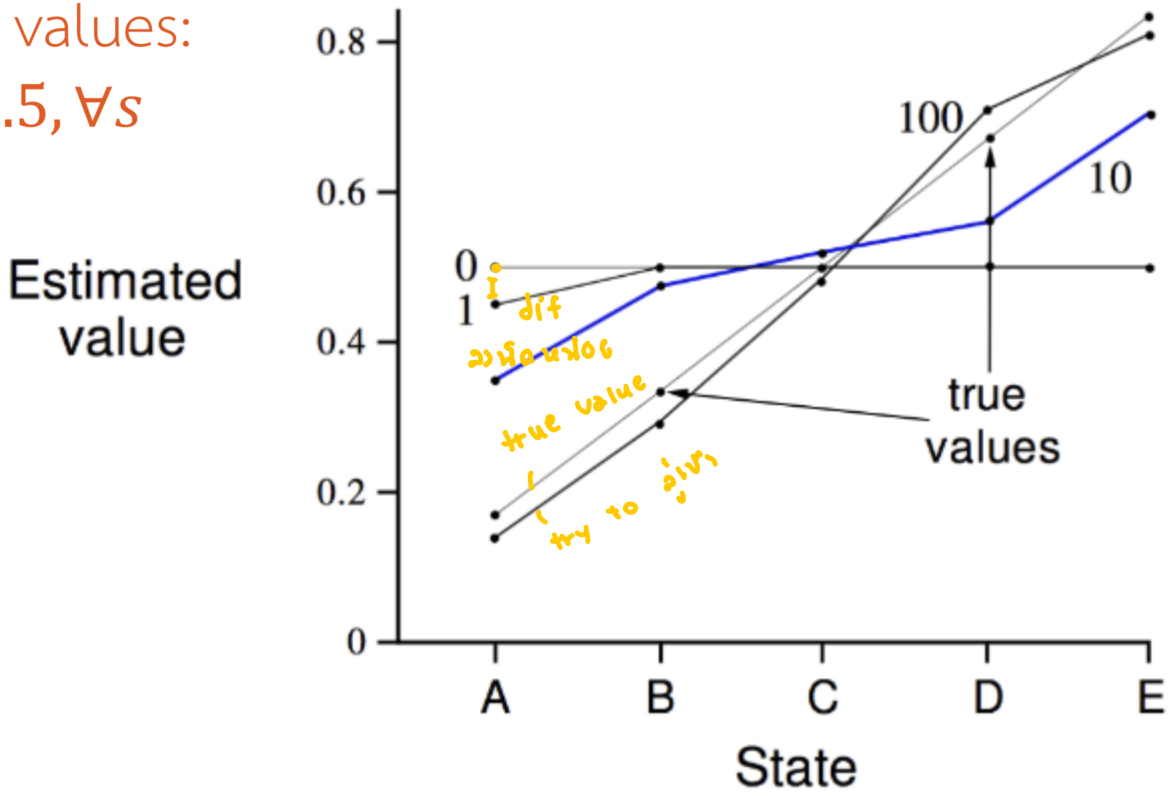
$$v(A) = \frac{1}{6}, v(B) = \frac{2}{6}, v(C) = \frac{3}{6}, v(D) = \frac{4}{6}, v(E) = \frac{5}{6}$$

Random Walk Example



Given that initial values:

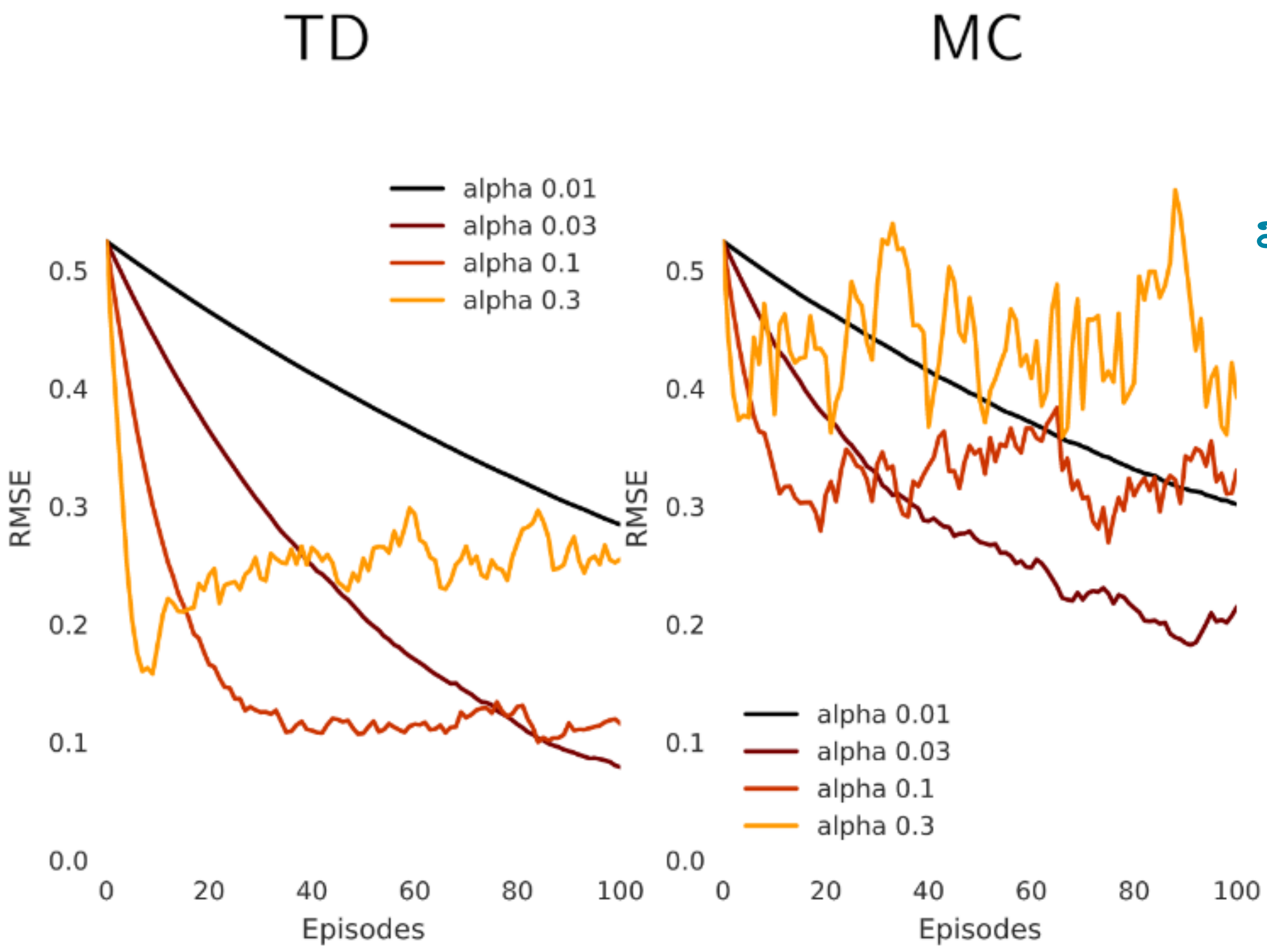
$$v(s) = 0.5, \forall s$$



Random Walk Example

HW 2 :

try this



alpha ↑

ไม่ลู่เข้าาวย

TD vs MC

TD exploits Markov property each step

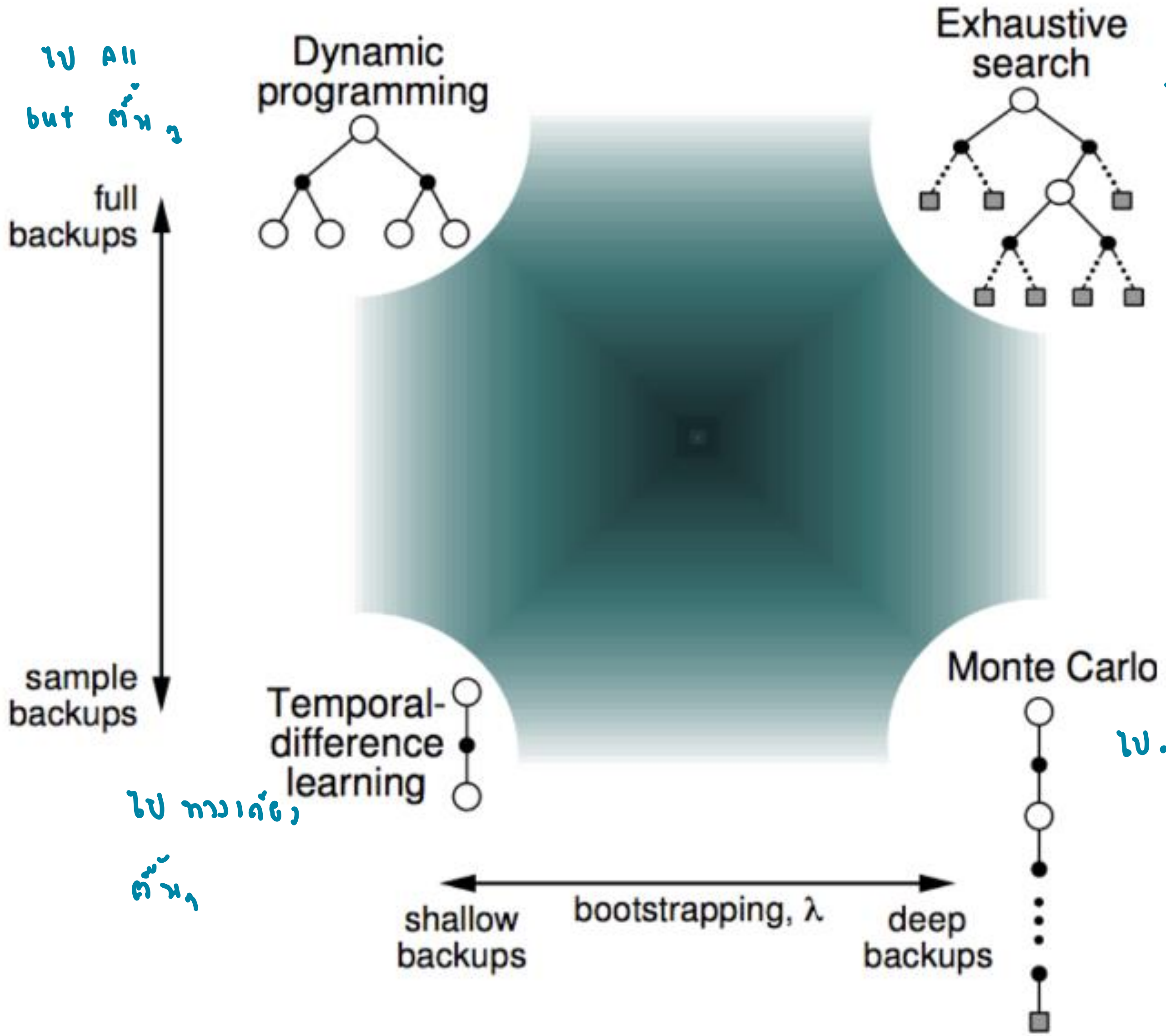
- TD can help in fully-observable environments

MC does not exploit Markov Property all step

- MC can help in partially-observable environments

ไป All
but ตื้นๆ

มา All that possible
backup, full backup



ไปสุดทางเดียว

ไป ทางเดียว
ตื้นๆ

Multi-Step Updates

Multi-Step Updates

TD and MC tackle the problem from different angles

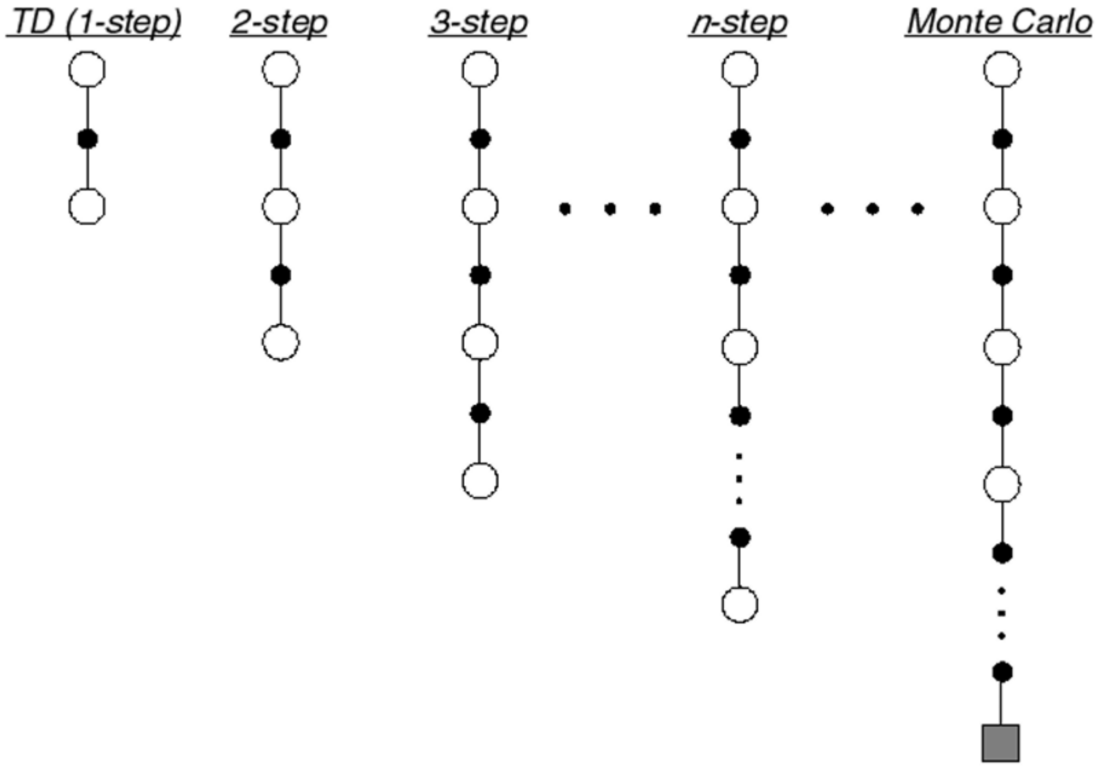
- TD estimates, and bootstrap
- MC use true return, but is noisy

Can we do something between?

- Multi-Step Prediction

Multi-Step Prediction

We can modify TD target to look n steps into the future



Multi-Step Returns

We can arrange TD and MC as:

เครื่องประทับ 1 : want to predict 1 step ล่วงหน้า

TD $\quad n = 1 \quad G_t^{(1)} = R_{t+1} + \gamma v(S_{t+1})$

$\quad n = 2 \quad G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 v(S_{t+2})$

sample 2 timestep แล้วค่อย Bootstrap

$\vdots$

MC $\quad n = \infty \quad G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-t-1} R_T$

sample จนจบ (feel like MC)

In general, we can formalize n-step return as

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ \ldots \ + \gamma^{n-1} v(S_{t+n})$$

Multi-Step Returns

In general, we can formalize n-step return as

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ \dots \ + \gamma^{n-1} v(S_{t+1})$$

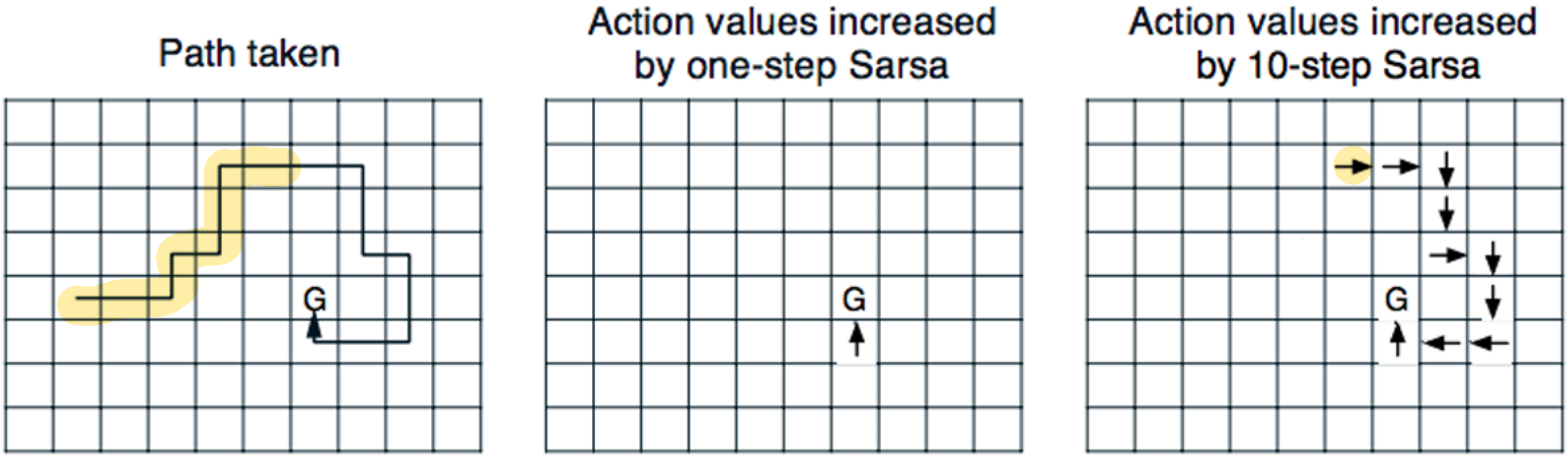
We can define multi-step temporal-difference learning as

if $G_t = v(s_t)$ สมการ = 0 และค่า $v(s_t)$ จะไม่อัพเดทแล้ว

$$v(S_t) \leftarrow v(S_t) + \alpha(\overbrace{G_t^{(n)} - v(S_t)})$$

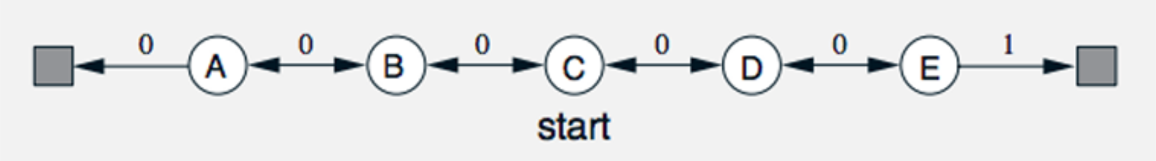
TD's target - current value

Example

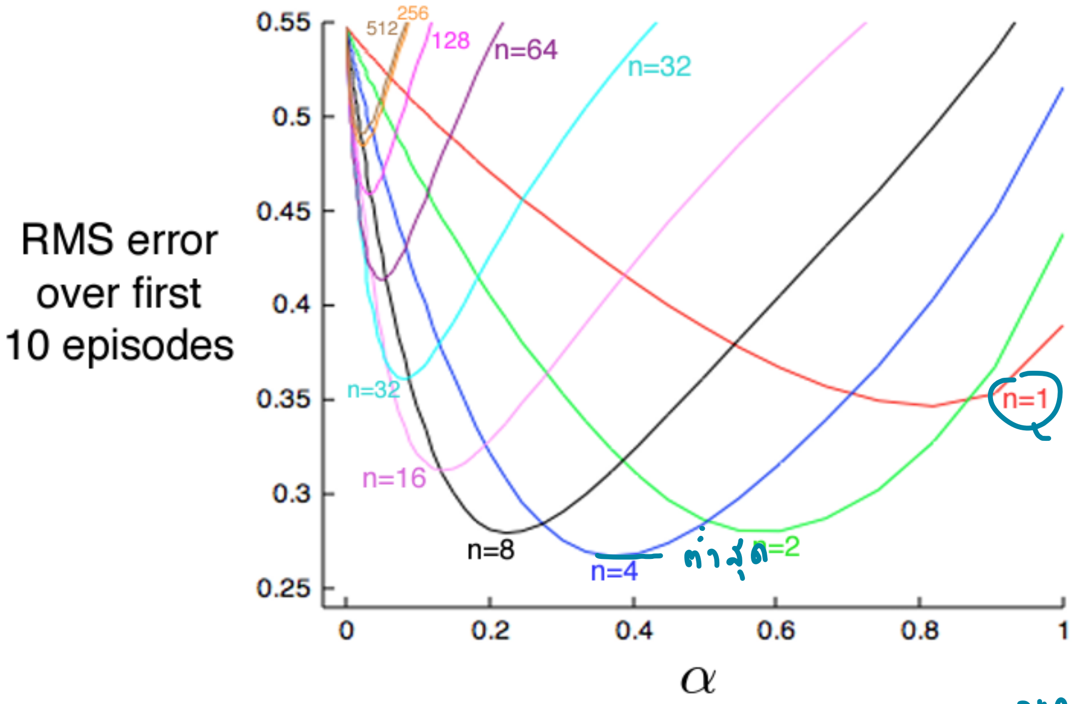


(Reminder: SARSA is TD for action values $q(s, a)$)

Example



..., but with 19 states, rather than 5



α ที่ n ต่างๆ

จน. step

α, n are hyperparameter ต้องจูน

Mixing Multi-Step Returns

Consider a multi-step return equation,

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ \dots \ + \gamma^{n-1} v(S_{t+1})$$

Multi-step returns bootstrap on one state, $v(S_{t+n})$. However, we can modify the equation so you can bootstrap a little bit on many states

Reward + decaying factor $(1-\lambda)v(S_{t+1})$ + decay (return)

$$G_t^{\lambda} = R_{t+1} + \gamma((1-\lambda)v(S_{t+1}) \ + \lambda G_{t+1}^{\lambda})$$

จบสมการ

Multi-Step Returns

$$G_t^{\lambda} = R_{t+1} + \gamma((1-\lambda)v(S_{t+1}) + \lambda G_{t+1}^{\lambda})$$

hyperparameter แบบมหาศาล

If we consider:

$\lambda = 0$ $\qquad G_t^{\lambda=0} = R_{t+1} + \gamma v(S_{t+1})$ (TD)

$\lambda = 1$ $\qquad G_t^{\lambda=1} = R_{t+1} + \gamma G_{t+1}$ (MC)

Multi-Step Returns

to balance

Using multi-step returns,

- We can utilize benefits of TD and MC
- Bootstrapping may cause bias
- Monte Carlo involves high variance

Generally, using multi-step returns are good

Summary

No Dynamic Model of Problem
cannot find Expectation value
then we sampling by using
Monte carlo run all to learn
and TD run step+1 to learn
but they are all have curse
then we merge MC & TD to
Multi-step Returns.