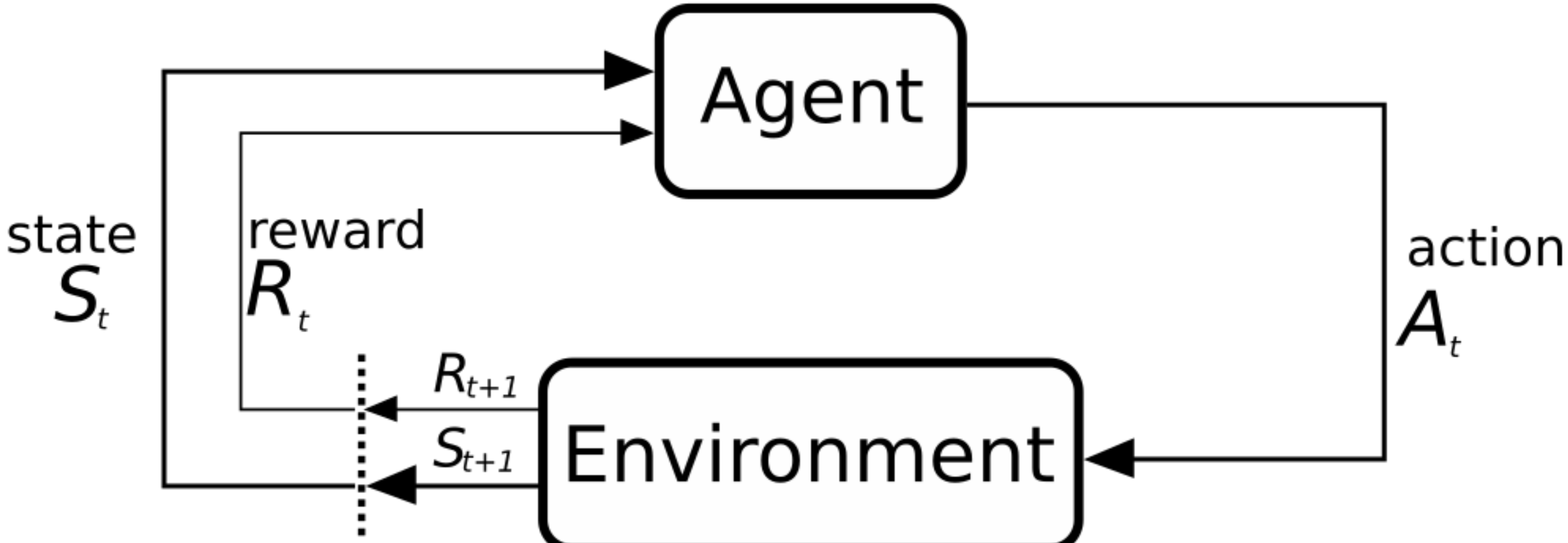
Markov Decision Process

Blink Sakulkueakulsuk

Recap



Rewards

It means, at time t, how good is the agent action

The ultimate goal of the agent is to maximize cumulative reward

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \ldots$$

This cumulative reward is called **return**

Values

We define the expected cumulative reward of a state s, the **value.**

$$
\begin{aligned}
v(s) &= \mathbb{E}[G_t \mid S_t = s] \\
&= \mathbb{E}[R_{t+1} + R_{t+2} + R_{t+3} + \ldots \mid S_t = s]
\end{aligned}
$$

Action values

We can map value to both action and state

$$\begin{aligned} q(s,a) &= \mathbb{E}[G_t \mid S_t = s, A_t = a] \\ &= \mathbb{E}[R_{t+1} + R_{t+2} + R_{t+3} + \ldots | S_t = s, A_t = a] \end{aligned}$$

Markov decision processes

$$p(r, s \mid S_t, A_t) = p(r, s \mid H_t, A_t)$$

The equation above essentially means “the future state is independent of the past state given the present state”

MDP provides baseline for reinforcement learning.

Policy

Given a state, a policy $\pi(S)$ defines an agent's action

- Mapping from agent state to action

Deterministic policy: $\pi(S) = A$

Stochastic policy: $\pi(A|S) = p(A|S)$

Value function

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

- $A_t \sim \pi(s)$ means that we choose action A from policy π
- $\gamma \in [0,1]$ is a discount factor

This is called a **Bellman equation**

If we want to find the optimal value, we can modify the equation to

$$v^*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v^*(S_{t+1}) \mid S_t = s, A_t = a]$$

Model

A **model** predicts what will happen next in the environment

For example, a model P predicts the next state

$$P(s, a, s') \approx p(S_{t+1} = s' \mid S_t = s, A_t = a)$$

For example, a model R predicts the next reward

$$R(s, a) = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$$

Exploration vs Exploitation

Exploration: gain knowledge

~~Exploration~~: gain maximum reward
exploitation

But do we have enough knowledge to know the action to gain maximum reward?

- It is a trade-off. We need to learn, but not too much.

Option 1: epsilon-greedy

A simple, yet powerful, algorithm

- $\epsilon \in [0,1]$ is a variable for exploration
- We select random action with probability ϵ
  - $a = random(\boldsymbol{A})$
- We select the best action with probability $1 - \epsilon$
  - $a = argmax_{a \in \boldsymbol{A}} Q_t(a)$

Option 2: Upper Confidence Bounds

$$a_t = argmax_{a \in A} Q_t(a) + c\sqrt{\frac{\log t}{N_t(a)}}$$

Basically, when we haven't explored some actions much, the value is high, and that action is selected and updated. Eventually, the value will converge to the true value for each action. Thus we can find an optimal action.

Markov Decision Process

This Lecture

Last time: Multi-arm bandit

- Single state problem with multiple actions
- No model

Today: Full sequential problem formulation

- Assume that true model is given
- Assume that the environment is fully observable

Next: Full sequential problem formulation

- What if true model is NOT given

Markov Decision Process

Markov decision processes (MDPs) tells us formally about the environment

, Should ตัดสินใจได้

State ปัจจุบัน represent อดีต

"The future is independent of the past given the present"

MDPs can be used to formalize many RL applications

- Continuous MDPs (optimal control)
- Partially observable problems
- One state MDPs (bandits)

Markov Decision Process

MDPs can be formalized as a tuple $\{S, A, p, \gamma\}$

- S is the set of all possible states
- A is the set of all possible actions
- $p(r, s'|s, a)$ is the joint probability of a reward r and the next state s', given a state s and an action a

  if now อยู่ s จะ take a, ค.เป็นไปได้ของ next state s' ได้ reward r เช่น robot เดินหน้า or แบตหมด

  ล้ม เลือก ล้มคนละท่า ผลไม่เท่ากัน

  s'' r-10   s'' r-100
- $\gamma \in [0,1]$ is a discount factor

  γ grammar

Noted that p defines the dynamics of the problem

Markov Decision Process

We can use $p(r, s'|s, a)$ to find state transition

sum of ทุก random variable

$$\text{Action เดี่ยว} \rightarrow p(s'|s,a) = \sum_{r} p(r, s'|s, a)$$

บวกหมด all reward < r - 10 + r - 100

$r \backslash s'$	a	b
1	0.1	0.1
2	0.2	0.2
3	0.3	0.1

=>

r	
1	0.2
2	0.4
3	0.4

Or to find expected reward

loop every state
↓
loop each action reward ?

$$\mathbb{E}[R|s, a] = \sum_{r} r \sum_{s'} p(r, s'|s, a)$$

idea coding for loop

มองมุม sum over sth. (r)

บวกหมด s'

$1(0.2) + 2(0.4) + 3(0.4)$
$= 0.2 + 0.8 + 1.2$
$= 2.2$ ✓

To code this, we just loop through the states and rewards

Markov Property

"The future is independent of the past given the present"

Consider a sequence of random variables, $\{S_t\}_{t\in\mathbb{N}}$, indexed by time. A state s has the Markov property when

$\forall s' \in S,$ for all s' ใน state s

2 ตัวนี้เท่ากัน ดัง Markov

every history

is no effect to Model

เงิคฟัง

$$p(S_{t+1} = s' | S_t = s) = p(S_{t+1} = s' | h_{t-1}, S_t = s)$$

cause all in State s

For all possible histories $h_{t-1} = \{S_1, \ldots, S_{t-1}, A_1, \ldots, A_{t-1}, R_1, \ldots, R_{t-1}\}$

Example: cleaning robot

Given a cleaning robot

- It has two states: high and low battery
- It has two actions in high state: wait, and search
- It has three actions in low state: wait, search, and recharge
- Transition probability are described as followed
  - $p(S_{t+1} = high \mid S_t = high, A_t = search) = \alpha$
  - $p(S_{t+1} = low \mid S_t = high, A_t = search) = 1 - \alpha$

battery of robot

s	a	s'	$p(s' \mid s, a)$	$r(s, a, s')$
high	search	high	α	$r_{\texttt{search}}$
high	search	low	$1 - \alpha$	$r_{\texttt{search}}$
low	search	high	$1 - \beta$	-3
low	search	low	β	$r_{\texttt{search}}$
high	wait	high	1	$r_{\texttt{wait}}$
high	wait	low	0	$r_{\texttt{wait}}$
low	wait	high	0	$r_{\texttt{wait}}$
low	wait	low	1	$r_{\texttt{wait}}$
low	recharge	high	1	0
low	recharge	low	0	0

may be พลังงานเฮือกสุดท้าย

Cleaning Robot MDPs

$MDP = \{S, A, p, \gamma\}$
$S = \{high, low\}$
$A = \{search, wait, recharge\}$

p = this look up table

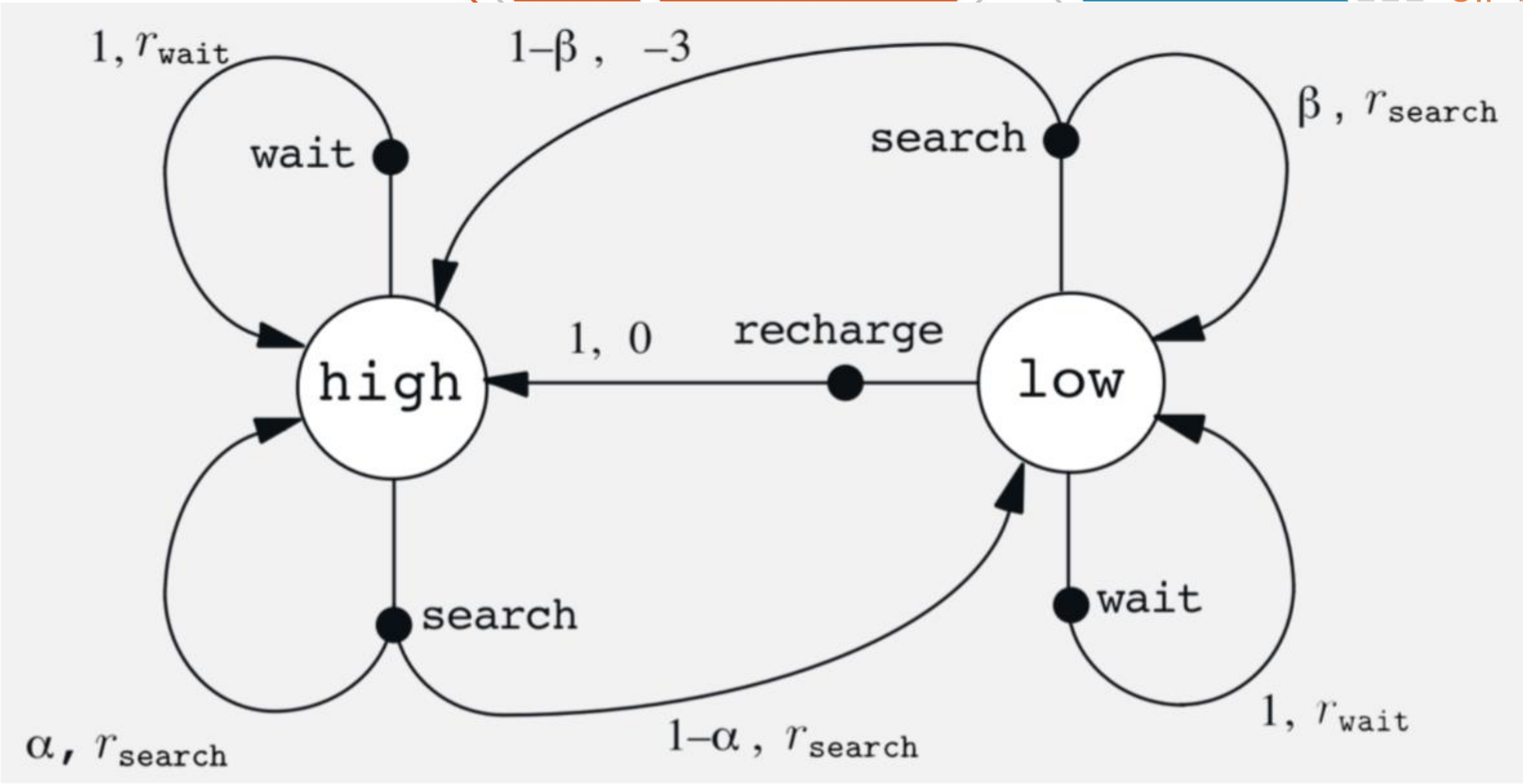
s	a	s'	$p(s' \mid s, a)$	$r(s, a, s')$
high	search	high	α	r_{search}
high	search	low	$1 - \alpha$	r_{search}
low	search	high	$1 - \beta$	-3
low	search	low	β	r_{search}
high	wait	high	1	r_{wait}
high	wait	low	0	r_{wait}
low	wait	high	0	r_{wait}
low	wait	low	1	r_{wait}
low	recharge	high	1	0
low	recharge	low	0	0

Let's go one more step further. When we talk about the value function, we need to find the expected reward. So, let's find some!

$$q(s = high, a = search) = \mathbb{E}[R|s = high, a = search]$$
$$q(s = high, a = search) = \alpha r_{search} + (1 - \alpha) r_{search} = r_{search}$$

value of 1 state

$$q(s = low, a = search) = \mathbb{E}[R|s = low, a = search]$$
$$q(s = low, a = search) = (1 - \beta)(-3) + \beta r_{search} = (r_{search} + 3)\beta - 3$$


1, r_{wait}
1–β , –3
β , r_{search}
wait
search
1, 0
recharge
high
low
search
wait
α, r_{search}
1–α , r_{search}
1, r_{wait}

Return Revisited

reward ใน return

return ใน value function

try to Maximize return

Recall that

return reward

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- $\gamma \in [0,1]$ is a discount factor

- The discount represents the trade-off between short-term reward and long-term planning
- We can set the upper bound of the sum to T, the horizon of the problem, to bound the problem

Policy Revisited

รู้ value ของ each state $\rightarrow$ RL Agent try to maximize return $\rightarrow$ state s จะไปไหน; ไปฝั่ง

Remind that

The Goal of an RL agent

Is to $\underbrace{\text{find a behavior policy}}_{\pi}$ that maximizes the $\underbrace{\text{expected return}}_{\text{Value}}$

A policy is a mapping $\pi: S \times A \Rightarrow [0,1]$

- It means that, for all states, and all action, there is a probability of taking an action on that state

Value function revisited

now consider policy π

The value function denotes the expected return of state s

RL try to learn

value of next state

ภายใต้ Policy π

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

given อยู่ state ปัจจุบัน

The equation above (hard to code) can be rewritten as:

3 loop

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

Propobiliy ที่ state นั้นจะเกิดขึ้น

over r

sum over s'

Action Value revisited

We can apply the same concept to action value equation

$$
\begin{aligned}
q_\pi(s,a) &= \mathbb{E}[G_t \mid S_t = s, A_t = a, \pi] \\
&= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \mathbb{E}[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]
\end{aligned}
$$

ฯ เลือก best action ที่หาได้ concept : maximize reward

The equation above (hard to code) can be rewritten as:

action value

$$q_\pi(s,a) = \sum_r \sum_{s'} p(r, s' | s, a)[r + \gamma \sum_{a'} \pi(a' | s')\, q_\pi(s', a')]$$

if ระบบเป็น continuos ตัว Σ ทุกตัว จะเป็น integral

Note that

state value

$$v_\pi(s) = \sum_a \pi(a|s) q_\pi(s,a) = \mathbb{E}[q_\pi(S_t, A_t | S_t = s, \pi]\,, \forall s$$

กำหนดด้วย Policy π

Optimal Value Function

ถ้า Agent เจอ Policy π ที่ the best
แล้ว อันนั้นคือ optimal value function.

The optimal value function is the maximum value of the value function of all policies

optimal แล้ว

$$v^{*}(s) = \max_{\pi} v_{\pi}(s)$$

1 2 3

each Policy get v(s) เท่าไหร่

The optimal action value function is the maximum value of the action-value function of all policies

$$q^{*}(s,a) = \max_{\pi} q_{\pi}(s,a)$$

If we can find these value, we can find the best policy. Thus, **we solve the problem.**

Prediction and Control

Prediction

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

Policy นั้น ดี or not

Control

- A problem when we perform policy optimization
- Estimating v^* or q^*

best Policy

Four main Bellman Equations

Bellman Expectation Equation

หา state value มีค่าเท่าไหร่

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

Bellman Optimality Equation

หา best value

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

Evaluating Policy

ex. π จะดีกว่า π′ ก็ต่อเมื่อ value function ที่เกิดจาก π สูงกว่า value function ที่เกิดจาก π′

$$\pi \geq \pi' \overset{\text{ก็ต่อเมื่อ}}{\leftrightarrow} \underbrace{v_\pi(s)}_{\text{value function ภายใต้ } \pi} \geq \underbrace{v_{\pi'}(s)}_{\text{value function ภายใต้ } \pi'} \quad , \underbrace{\forall s}_{\text{given ทุก state}}$$

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

Policy เกิดจากการเรียนรู้จาก agent

single agent

take 1 action

Policy เปลี่ยน

* เปรียบเทียบ Policy
เพื่อถ้าได้ π แย่กว่าอดีต
ก็ไม่ควรเก็บไว้ *

Optimal Policy

Theorem

For any Markov decision process

- There exists an optimal policy π^* that is better than or equal to all other policies, $\pi^* \geq \pi, \forall\pi$
- All optimal policies achieve the optimal value function,
$$v_{\pi^*}(s) = v^*(s)$$
- All optimal policies achieve the optimal action-value function,
$$q_{\pi^*}(s,a) = q^*(s,a)$$

Finding an Optimal Policy

สามารถมีหลาย Optimal Policy

We can use action value to find the optimal policy

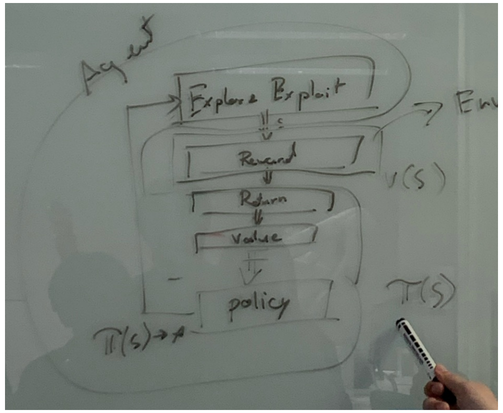
Recall that there are many actions in one state. The action with highest action-value is the optimal action.

when มั่นใจใน Policy

reward ลงทุน สั้นปัจจุบัน

return ลงทุนระยะยาว

$$\pi^*(s,a) = \begin{cases} 1 & \text{if } a = \mathit{argmax}_{a \in A} q_*(s,a) \\ 0 & \text{otherwise} \end{cases}$$



π : S X A

Noted that there can be multiple optimal policies

Solving the Bellman Optimality Equation

Using models (dynamic programming)

- Value iteration
- Policy iteration

Sampling Method

- Monte Carlo
- Q-learning
- Sarsa

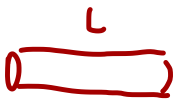
Dynamic Programming

Technique using for solve RL Problem

In general, set up a solution as a recursion, solve with iteration from the base case

In Reinforcement Learning, we iterate and update the value. There are two parts of the process.

- Policy evaluation — check good or not
- Policy Improvement — improve to make it better



L	P
1	1
2	2
3	4
4	5
5	8

$$P(L) = \max \begin{cases} 1 + P(L-1) \\ 2 + P(L-2) \\ 4 + P(L-3) \\ 5 + P(L-4) \\ 8 + P(L-5) \end{cases}$$

$$P(1) = 1$$

$$P(2) = \max \begin{cases} P(1) + 1 \\ P(2) + 2 \end{cases} = \begin{cases} 1+1 \\ 2 \end{cases}$$

ตัด 1 หน่วย

$$P(3) = \max \begin{cases} P(2) + 1 \\ P(1) + 2 \\ P(0) + 4 \end{cases} = \begin{cases} 2+1 \\ 1+2 \\ 0+4 \end{cases}$$

ตัว 3 หน่วย

BIG ASSUMPTION

We are doing a synchronous dynamic programming ***

- It means we update every state at the same time (kind of)

For an asynchronous dynamic programming, we can tweak the methods a little bit

Policy Evaluation

Given a policy, we want to estimate this

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

value function

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

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

จำนวน timestep ที่จะ update ไปเรื่อยๆ

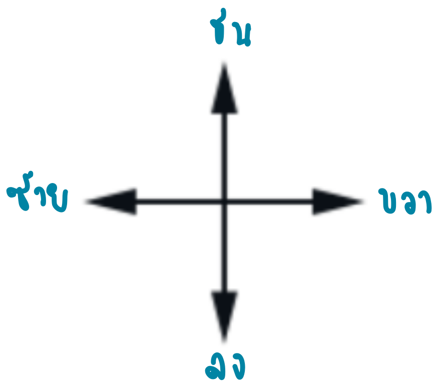
Note that

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

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

Policy Evaluation

don't care to update policy , focus on value function



actions

goal	1	2	3
4	5	6	7
8	9	10	11
12	13	14	goal

$$R_t = -1$$

$\frac{-1 \times 4}{4}$ action ที่ทำได้

เฉลี่ย

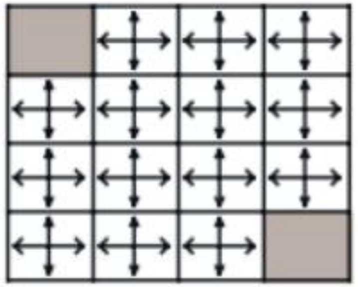
on all transitions

except goal state

Policy Evaluation

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



random policy

Prob

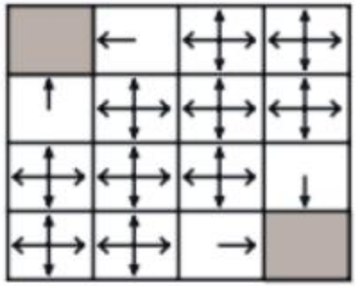
value

$$V(s) = \frac{1}{4}(-1 + \gamma(0)) + \frac{1}{4}(-1 + \gamma(0)) + \frac{1}{4}(-1 + \gamma(0)) + \frac{1}{4}(-1 + \gamma(0))$$

$$= -1$$

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



Prob

return

$$V(s) = \frac{1}{4}(-1 + -1) + \frac{1}{4}(-1 + -1) + \frac{1}{4}(-1 + 0) + \frac{1}{4}(-1 + -1)$$

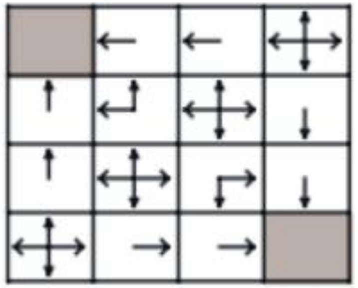
$$= -\frac{7}{4} = -1.75$$

grammar simpified = 1

why

$k = 2$

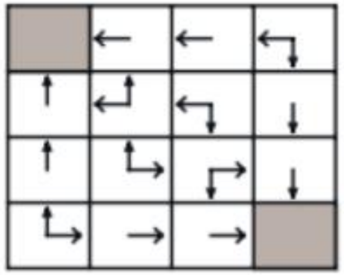
0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



Policy Evaluation

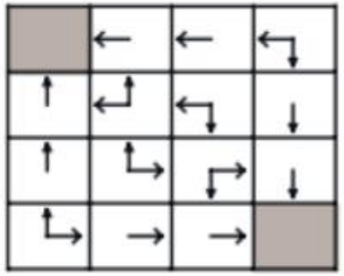
$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0



$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0



optimal policy

k=3,10,∞
policy เท่ากันเป๊ะ

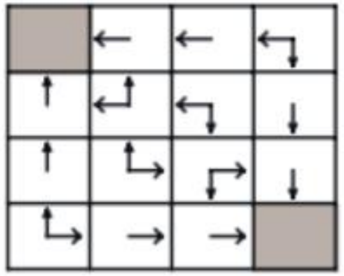
Model Policy for reach goal but search Algorithm doesn't

cannot control with complicate problem.

$k = \infty$

รู้ได้ไงว่าเข้าตอนไหน

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



Policy Improvement

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

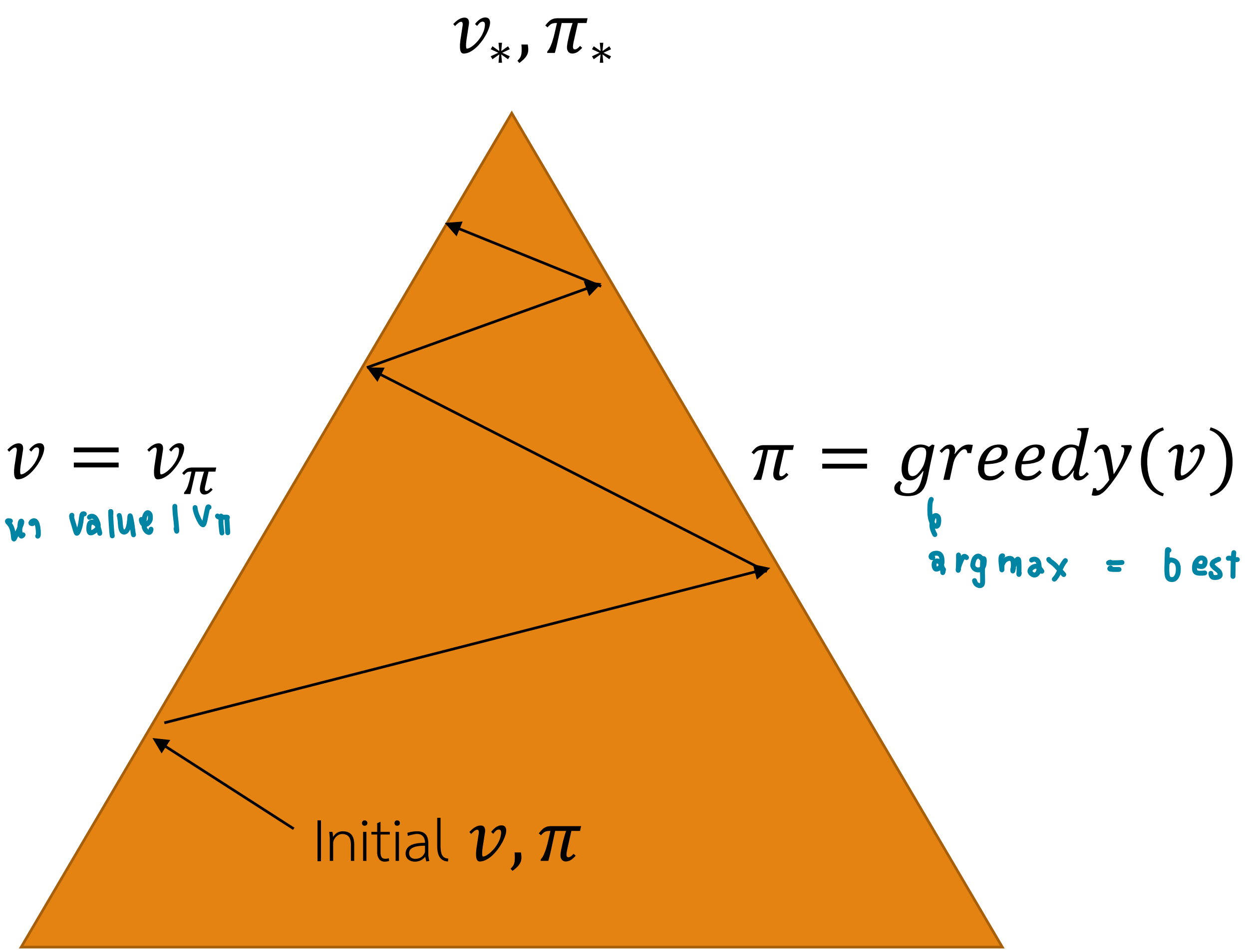
- In the example, we do not improve any policy

เอา max value a in s

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

We use the new policy to evaluate, and repeat.

Policy Iteration



argmax = best action

Starting with initial v, π

Loop do:

Perform policy evaluation,

$$v = v_\pi$$

Perform policy improvement,

$$\pi' \geq \pi$$

Policy Improvement

prove ของ policy movement

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

Since we improve the policy, $v_{\pi_{new}}(s) \geq v_\pi(s)$ for all s

It means that

$$v_{\pi_{new}}(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi_{new}}(S_{t+1}) | S_t = s]$$

max over action

Which is the Bellman optimality equation!

So, π_{new} must be either an improvement, or be optimal

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.

Shortest Path Example

g			

Problem

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

V_2

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

V_3

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

V_4

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-4
-3	-4	-4	-4

V_5

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

V_6

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

Summary

Prediction

- Bellman Expectation Equation
- Iterative Policy Evaluation

Control

- Bellman Expectation Equation + Greedy Policy Improvement
- Policy Iteration

Or

- Bellman Optimality Equation
- Value Iteration

Asynchronous Dynamic Programming

So far, we use **synchronous** dynamic programming

- We update all states in parallel at the same time (kind of)

However, we do not have to back up everything in parallel

- **Asynchronous Dynamic Programming** backs up states individually

Asynchronous Dynamic Programming

Three general approaches for asynchronous dynamic programming

- In-place dynamic programming
- Prioritized sweeping
- Real-time dynamic programming

In-place Dynamic Programming

During the updates, synchronous value iteration stores two copies of value function

$$\forall s \in S: v_{new}(s) \leftarrow \max_a \mathbb{E}[R_{t+1} + \gamma v_{old}(S_{t+1}) | S_t = s]$$
$$v_{old} \leftarrow v_{new}$$

don't want to update every state

However, we can store the value **in-place** if we do asynchronous dynamic programming

$$\forall s \in S: v(s) \leftarrow \max_a \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) | S_t = s]$$

Problem : bias variance trend of noise , value swing

Prioritized Sweeping

We will pick and update states with a lot of Bellman error first

- Intuitively, such states have a lot to learn
- For example, we can use

ค่าสูงสุด ... - ค่าเฉลี่ย } should adjust policy more ↑

= น้อย , ไม่ต้องปรับ

best action

from expectation

$$\left|\max_a \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) | S_t = s] - v(s)\right|$$

ค่าเฉลี่ย

We back up states with large Bellman error, then we update those states

if we at state s , then check value

Real-Time Dynamic Programming

* in Simulation

ใช้เยอะ

Only update states that are relevant to the agent

For example, if an agent is in S_t, maybe we can just update that state value, and couple other relevant states

Full-Width Backups → เก่งค่าหมด

Standard DP uses full-width backups

- It means that, to backup, the algorithm considers **all successor states and actions** using true transition model and reward function

DP is effective for medium-sized problem

- Million states ok
- Curse of Dimensionality can be a problem here