Model-Free Control

Blink Sakulkueakulsuk

Four main Bellman Equations

Bellman Expectation Equation

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

Bellman Optimality Equation

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

Prediction and Control

Prediction

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

Control

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

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

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

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

Policy Improvement

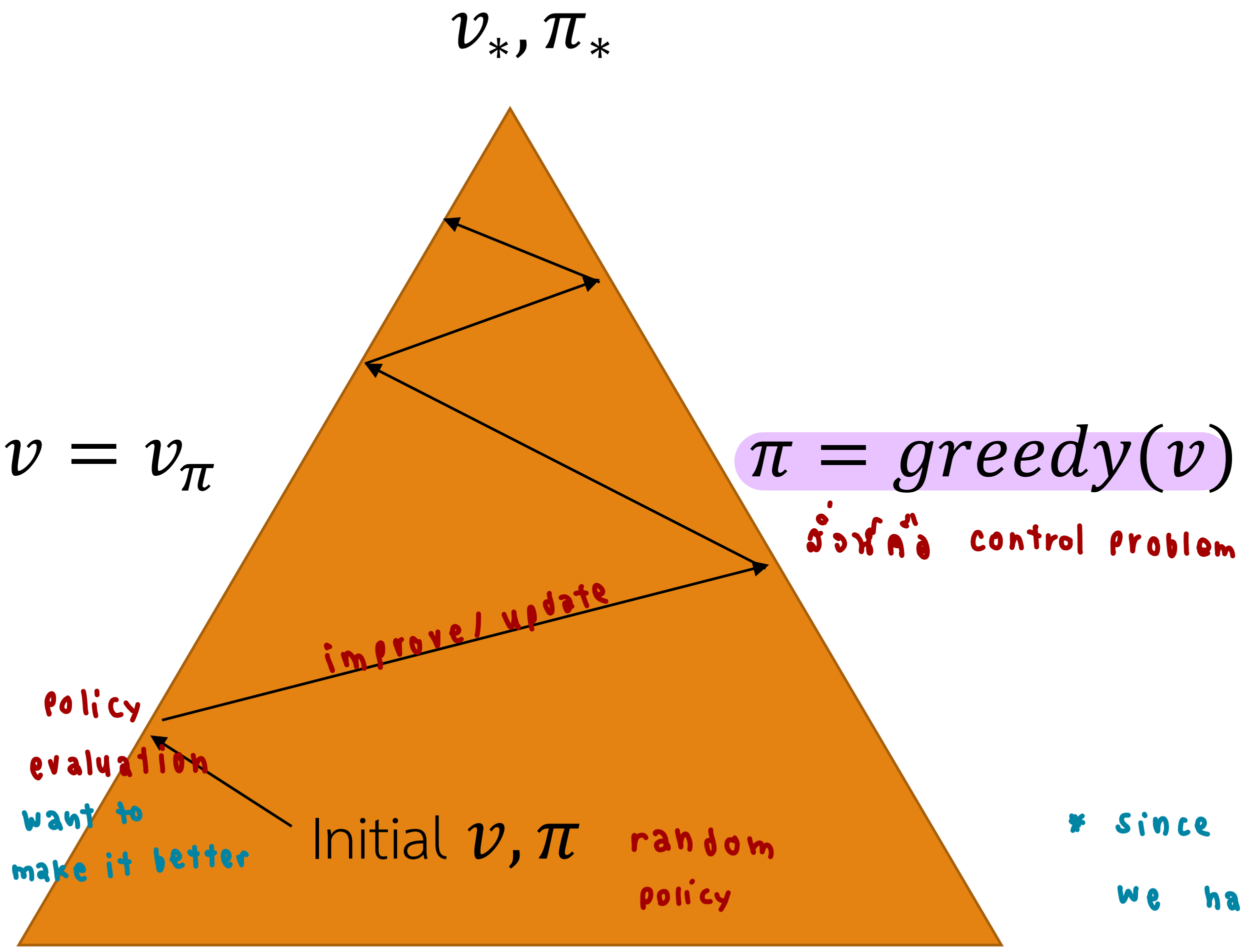
The example shows that we can improve a policy when we learn the value.

- 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,

$$\pi' \geq \pi$$

เกิดจาก value ที่ ≥ value เดิม

not optimal

bc found good policy then fix prob to 1 other 0

* since we greedify the action value

we have no exploration at all. แก้โดย $\pi = \epsilon$ - greedy (a)

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.

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.

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)

Monte-Carlo Policy Evaluation

$\pi'(s) = \arg\max_a \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) | S_t = s, A_t = a]$

but MC want import now not waiting for other

$\pi's = \arg\max \square$?

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

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

The return is calculated given an ending time T

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

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**

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**

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,

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

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

Non-linear Update = step-size x prediction error x gradient

Temporal Difference Learning

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

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

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

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

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

MC Prediction vs TD Prediction

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

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

Temporal-difference learning:

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

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

Bootstrapping and Sampling

Bootstrapping

- 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 $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$

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

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

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

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

Model-free Control

Our Plan

Last Lecture

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

This Lecture

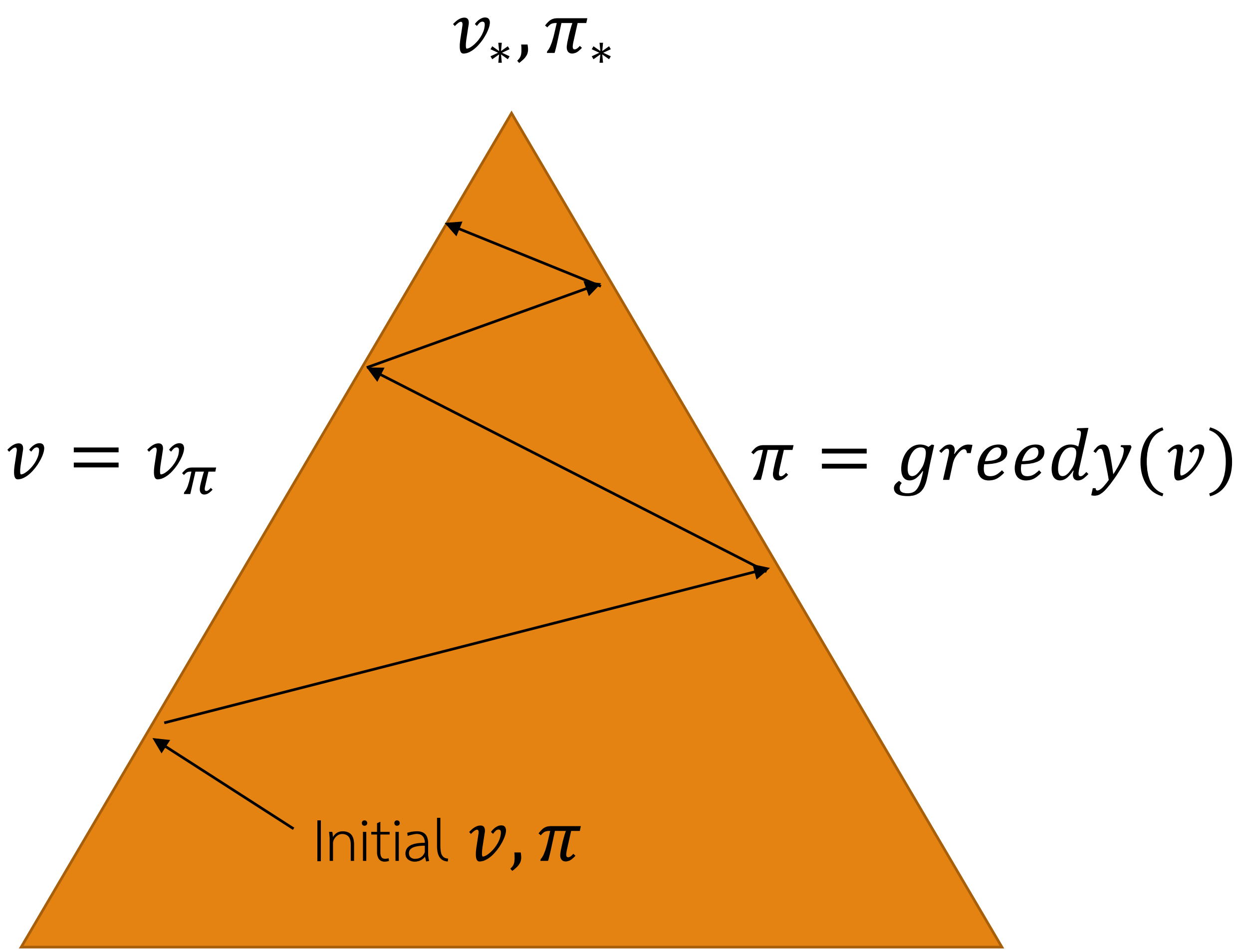
- **Model-free control** to **optimize values** in an **unknown MDP**

Model-free Control

A **control** problem for a **model-free** learning

- We do not have a model
- We want to learn the optimal policy

Policy Iteration Algorithm Revisited



Starting with initial v, π

Loop do:

Perform policy evaluation,

$$v = v_\pi$$

Perform policy improvement,

$$\pi' \geq \pi$$

Monte Carlo Control

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

MC evaluates its target from state S_t using the equation above. So,

$$\mathbb{E}[G_t] = v_\pi$$

If we sample many times and average the return, we will get a true value given a policy.

Monte Carlo Control

Usually, to find the optimal policy, we can "greedify" the value function.

$$\pi'(s) = argmax_a \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) | \, S_t = s, A_t = a]$$

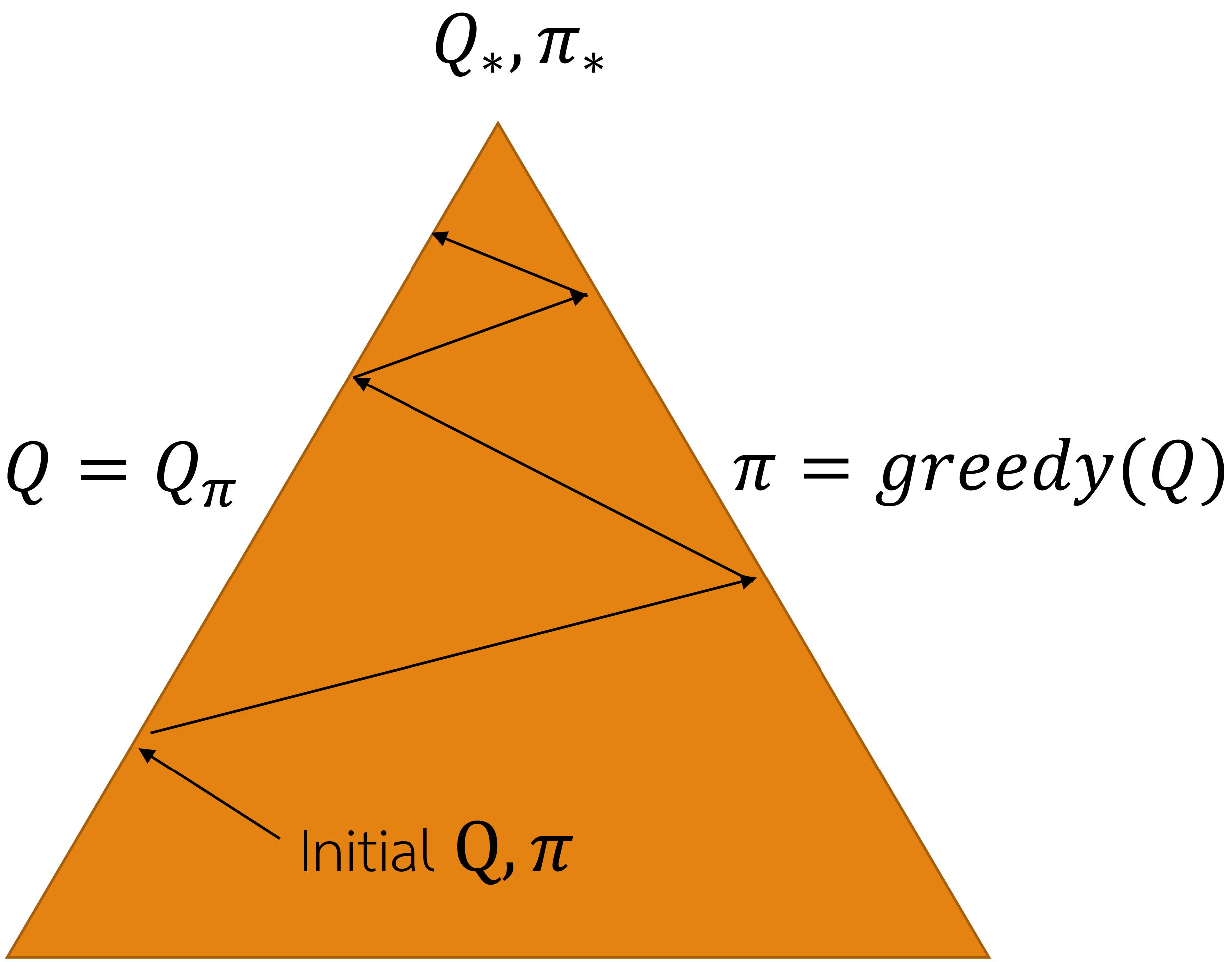
But we cannot use the equation above, because we cannot find the action that gives the maximum expected value. In short, we do not know the model.

Monte Carlo Control

Instead, we can greedify over $q(s, a)$. This one is model-free.

$$\pi'(s) = argmax_a q(s, a)$$

Monte Carlo Control



Perform policy evaluation

- Using Monte Carlo policy evaluation

$$q \approx q_\pi$$

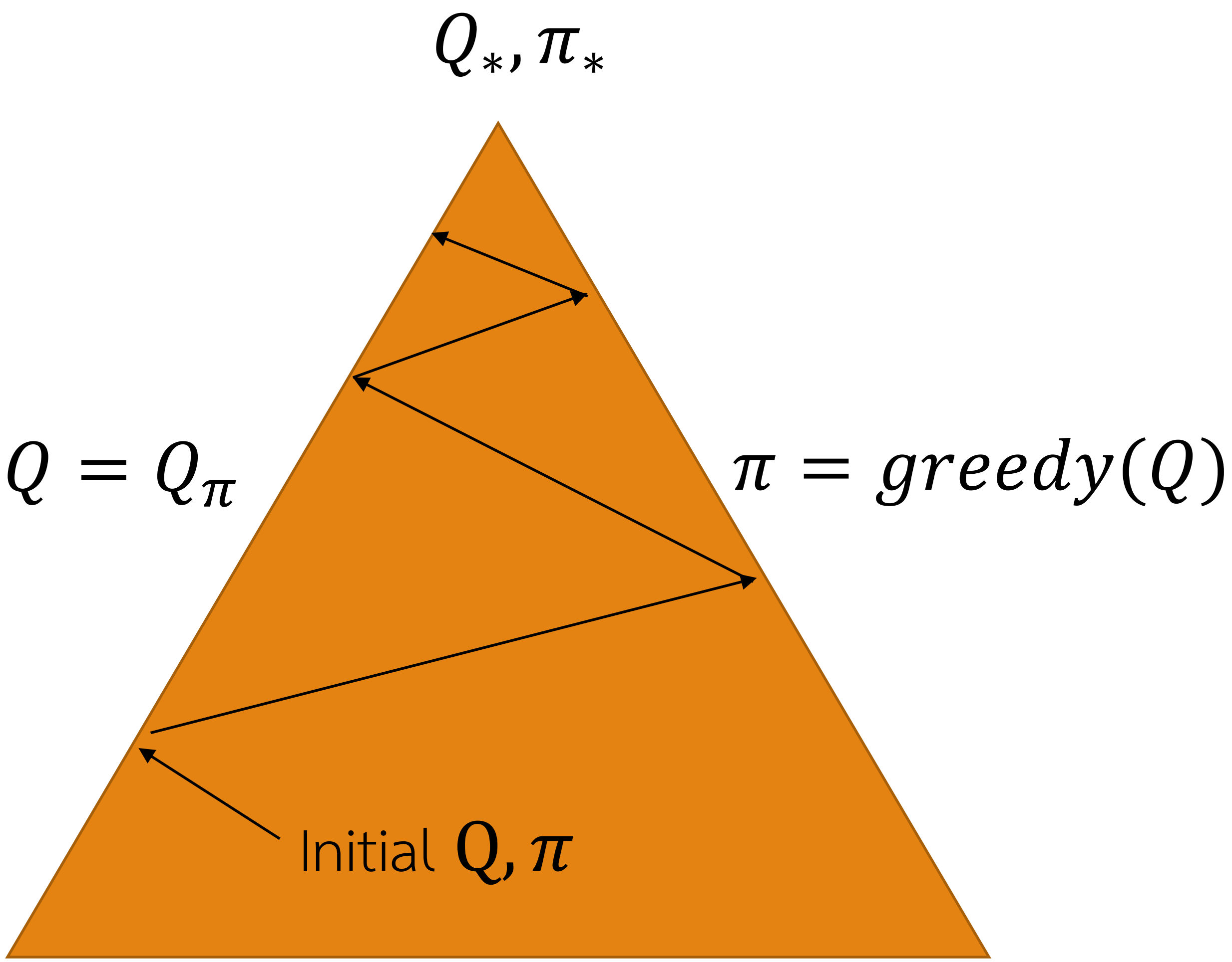
Perform policy improvement

- Using greedy policy improvement

$$\pi' \geq \pi$$

What is the problem here?

Monte Carlo Control



Perform policy evaluation

- Using Monte Carlo policy evaluation

$$q \approx q_\pi$$

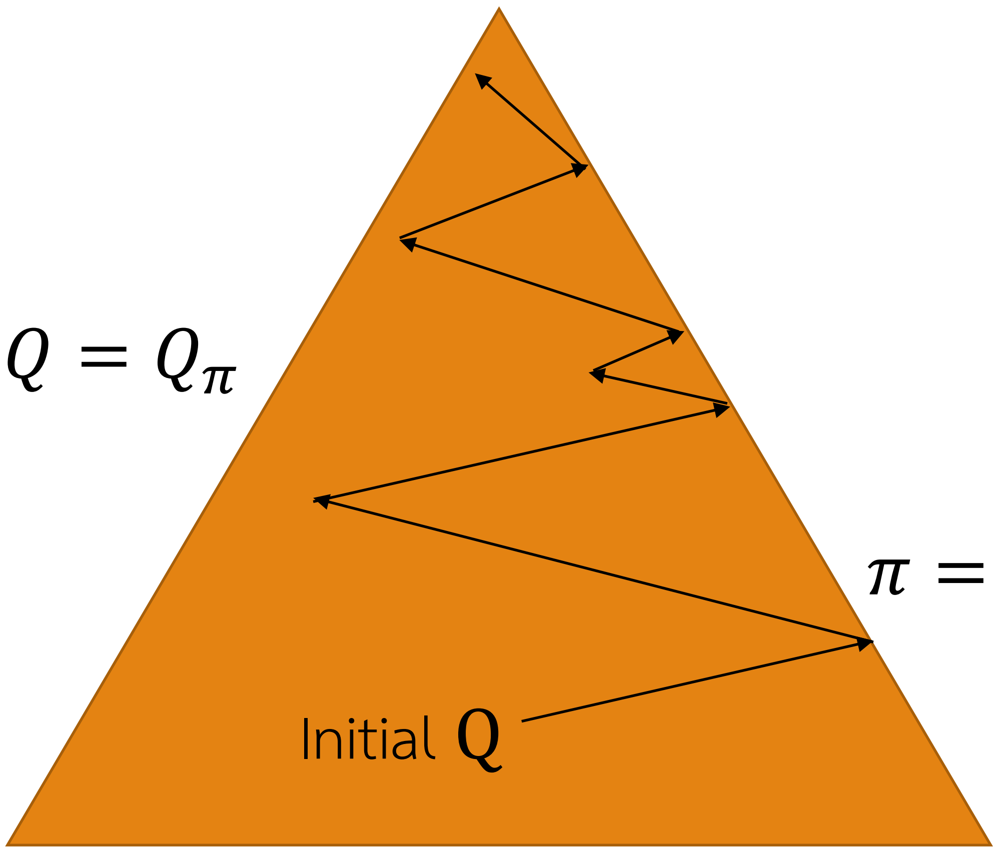
Perform policy improvement

- Using greedy policy improvement

$$\pi' \geq \pi$$

Since we greedify the action value, we have no exploration at all.

Monte Carlo Control



Perform policy evaluation

- Using Monte Carlo policy evaluation

$$q \approx q_\pi$$

Perform policy improvement

- Using **ϵ-greedy** policy improvement

GLIE – Greedy in the Limit with Infinite Exploration

This method can be used to find the optimal policy

- Given infinite exploration, all state-action are explored infinitely many times.

$$\lim_{k \to \infty} N_k(s, a) = \infty$$

- The policy converges to a greedy policy

$$\lim_{k \to \infty} \pi_k(s, a) = I(a = argmax_{a'} q_k(s, a'))$$

Example:

if visit k มาก ค่า $\epsilon \downarrow$

- ϵ-greedy with $\epsilon = \frac{1}{k}$ → จำนวนครั้งของ State action

GLIE Monte Carlo Control

Sample k^{th} episode, e_k, using $\pi = \{S_1, A_1, R_2, \ldots, S_T\} \sim \pi$ → loop

For each S_t and A_t in e_k:

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

ϵ → exploration

$$q(S_t, A_t) \leftarrow q(S_t, A_t) + \frac{1}{N(S_t, A_t)}(G_t - q(S_t, A_t))$$ → ลู่เข้า G_t → ลู่เข้า 0

ยิ่งใกล้กันยิ่งดี

Update the exploration rate and improve policy

$$\epsilon \leftarrow \frac{1}{k}$$

$$\pi \leftarrow \epsilon - greedy(q)$$

Temporal Difference Control

TD learning has many advantages ever MC

- Learn from incomplete sequences
- Lower variance
- Online learning

How about we apply the idea to TD instead of MC?

SARSA → TD ที่คุยกันในรูปแบบ state action

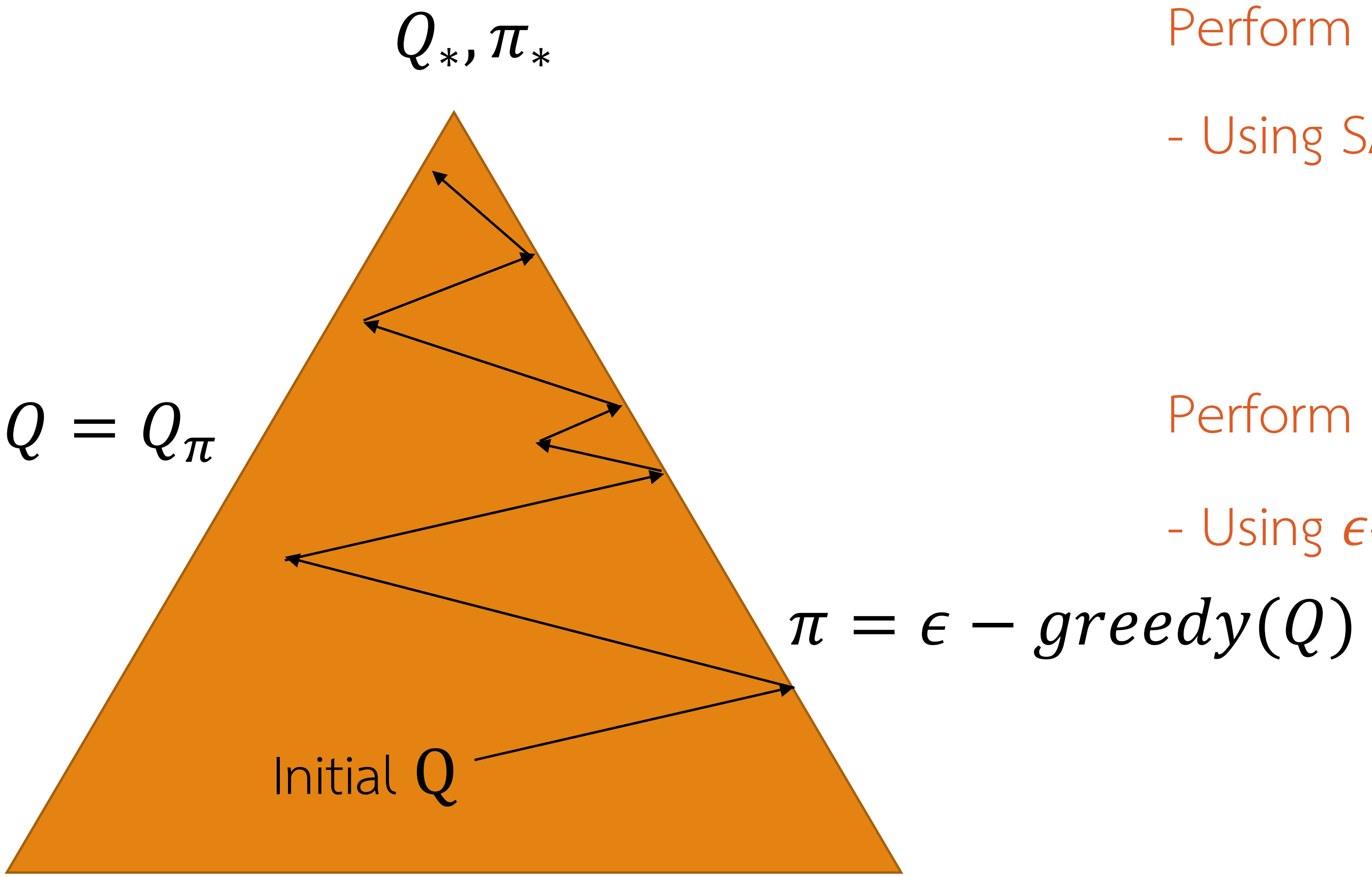
SARSA is a model-free reinforcement learning algorithm that learns from **S**tate, **A**ction, **R**eward, **S**tate, and **A**ction

$$s, a \xrightarrow{\quad r \quad} s', a'$$

$$q(s,a) \leftarrow q(s,a) + \alpha(r + \gamma q(s',a') - q(s,a))$$

รันแค่ next state not G_t anymore

SARSA



Perform policy evaluation

- Using SARSA

$$q \approx q_\pi$$

Perform policy improvement

- Using ϵ-greedy policy improvement implement in homework

Initialize $Q(s, a)$ arbitrarily

Repeat (for each episode): → รันได้หลาย ep.

  Initialize s

  Choose a from s using policy derived from Q (e.g., ε-greedy)

  Repeat (for each step of episode): MC have no this loop

    Take action a, observe r, s'

    Choose a' from s' using policy derived from Q (e.g., ε-greedy)

    update value

    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$ update policy

    $s \leftarrow s'$; $a \leftarrow a'$; มีการแทนที่ s, a ตอนแรก random แล้วหลังจะค่อยๆ update ไปเรื่อยๆ

  until s is terminal

On-Policy and Off-policy Learning

On-policy learning

- Learn from actually performing
- Learn a policy π from samples from π

Policy ภายนอก

ปกติ learn from การลองผิดลองถูก

Off-policy learning

- Learn from other sources
- Learn a policy π from samples from b

Model-based Updates by DP

Policy Evaluation

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

Value Iteration

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

Policy Evaluation

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

Value Iteration

$$q_{k+1}(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_k\,(S_{t+1}, a') \,\middle|\, S_t = s, A_t = a)\right]$$

Model-free Updates

TD

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

SARSA

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

มันดีที่สุดแค่จังหวะนี้

Q-learning

off policy learning

$$q_{t+1}(S_t, A_t) = q_t(S_t, A_t) + \alpha_t(R_{t+1} + \gamma \max_{a'} q_t(S_{t+1}, a') - q_t(S_t, A_t))$$

Off-policy Learning

เพิ่มความหลากหลายในการเรียนรู้

Another way of learning. The agent learns from a behavior policy $b(s, a)$ to learn the target policy $\pi(s, a)$

$$\{S_1, A_1, R_2, \dots, S_T\} \sim b$$

For example,

- Learning from others' behavior (human, robot,...)
- Learning from old policies
- Learning from what-if scenarios

Q-learning

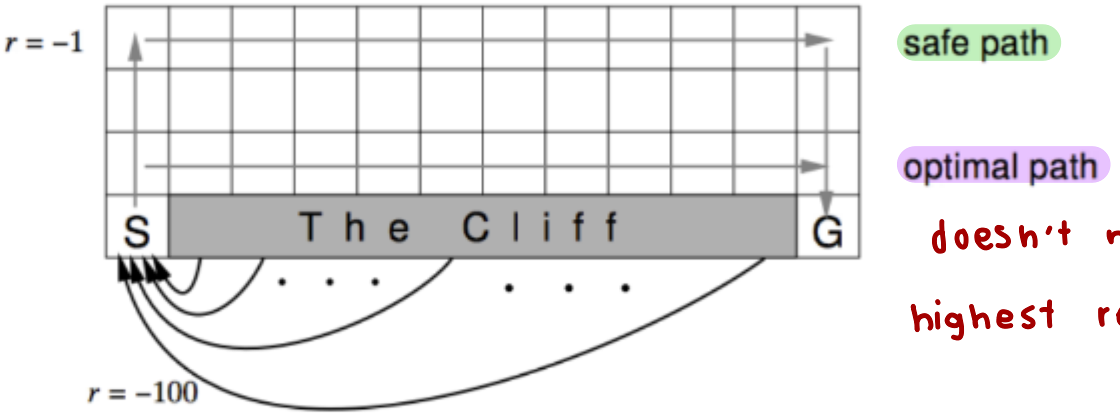
Q-learning is an off-policy, model-free reinforcement learning

$$q_{t+1}(S_t, A_t) = q_t(S_t, A_t) + \alpha_t(R_{t+1} + \gamma \max_{a'} q_t(S_{t+1}, a') - q_t(S_t, A_t))$$

Q-learning learns from the value of the greedy policy, thus an off-policy learning.

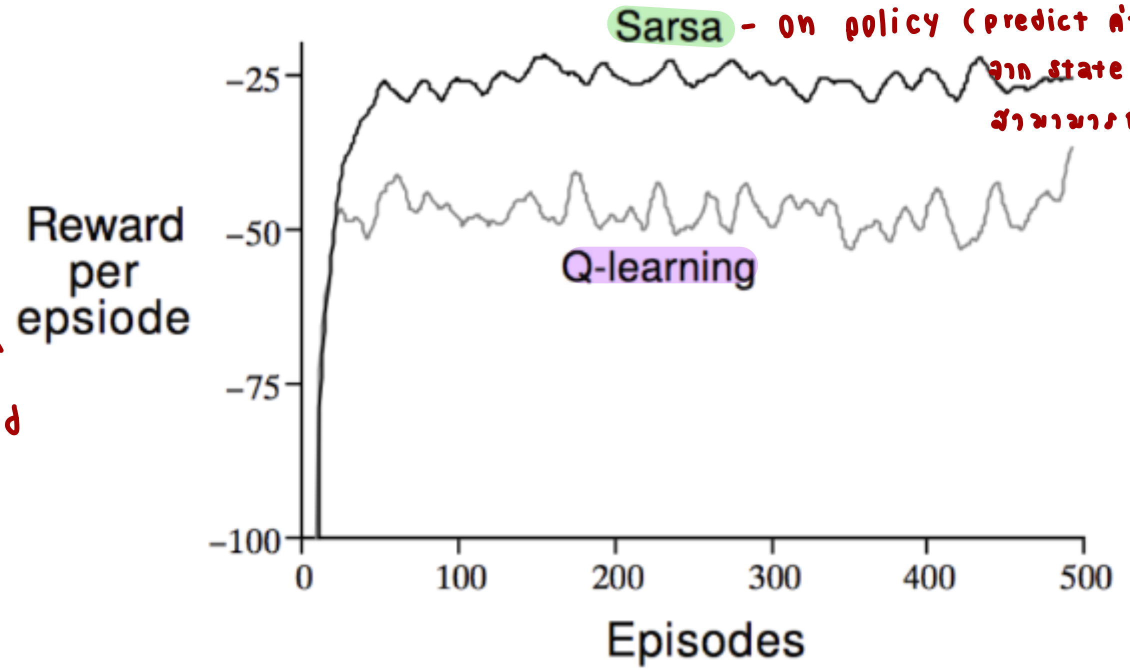
- It converges to the optimal action-value function when we explore all state-action infinitely many times
- No greedy behavior needed

SARSA vs Q-learning



doesn't mean highest reward

Safe path by SARSA, and optimal path by Q-learning



- on policy (predict ค่าปจบ. จาก state ที่ สามามารถไปได้)

Reward from SARSA and Q-learning

Q-learning Issue

ซ้ำซ้อน

Q-learning uses the same value to select actions and evaluate its value.

ตัว update ใน Q learning

$$\max_a q_t(S_{t+1}, a) = q_t(S_{t+1}, argmax_a q_t(S_{t+1}, a))$$

However, these values are approximated.

So it will select overestimated values more and select underestimate values less. → แปลว่าจะอยู่ค่าเดิมๆ ซึ่งอาจจะหลงทาง

Double Q-learning

Since

$$\max_a q_t(S_{t+1}, a) = q_t(S_{t+1}, argmax_a q_t(S_{t+1}, a))$$

Q-learning will use the same values to select action, and evaluate the value

$$R_{t+1} + \gamma \max_{a'} q_t(S_{t+1}, a') = R_{t+1} + \gamma q_t(S_{t+1}, argmax_a q_t(S_{t+1}, a))$$

And this will result in the issue in the last slide. To solve this problem, we can introduce another policy to split action selection from evaluation.

Double Q-learning

Double Q-learning will

- store two q functions: q and q'

- update one q function for each iteration.

q a of q in q'

action

$$R_{t+1} + \gamma q'_t(S_{t+1}, argmax_a q_t(S_{t+1}, a))$$

q a of q' in q

$$R_{t+1} + \gamma q_t(S_{t+1}, argmax_a q'_t(S_{t+1}, a))$$

balance over under estimated

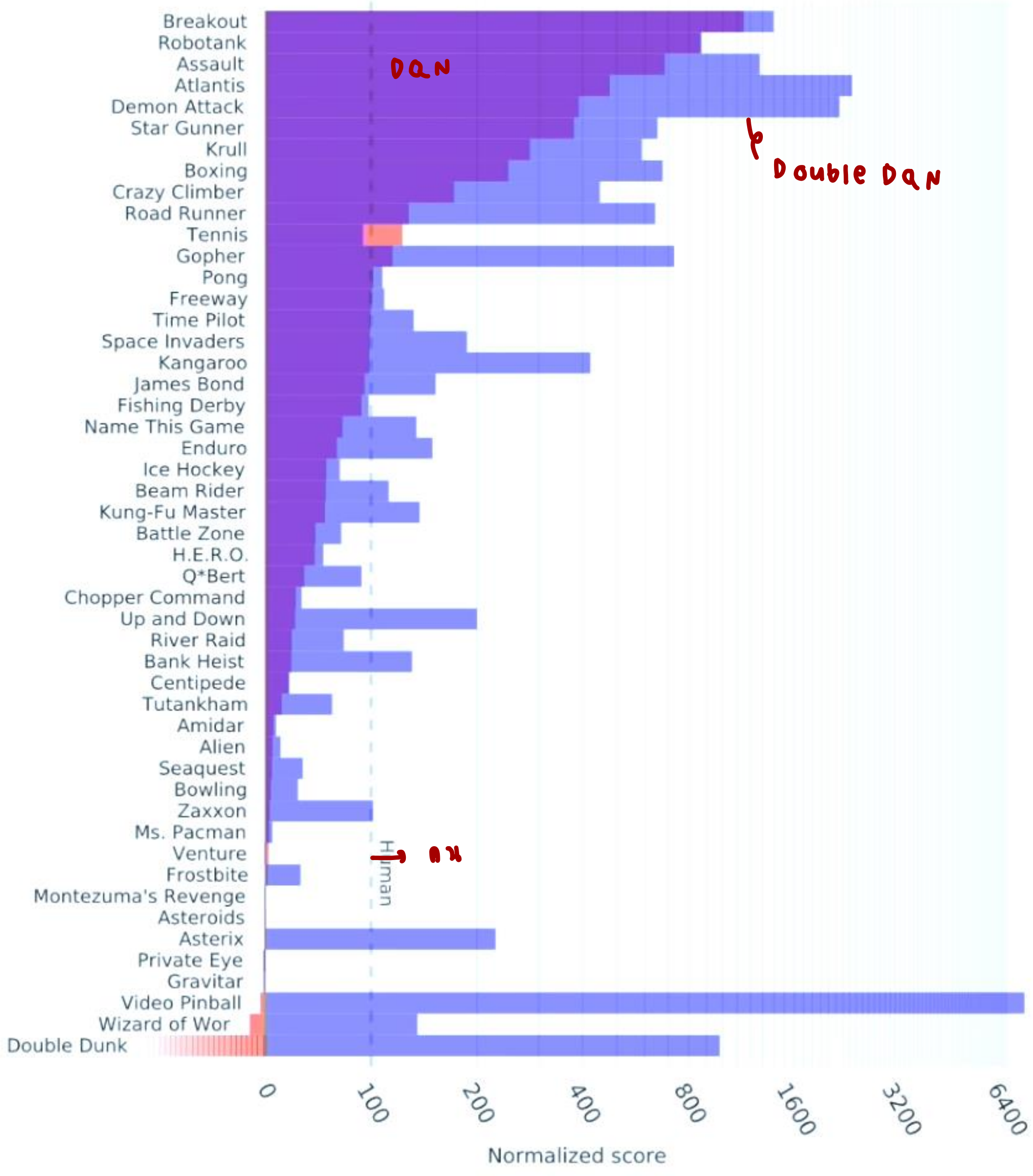
จะสลับกันทำหรือ random ก็ได้

off policy bc not sample policy

q, q' หาจาก initial = 0 เริ่มเทรนเลย

DQN

Double DQN



Generalized Q-learning

We can also learn from a behavior policy $b(s, a)$ while using $\pi(s, a)$

We modify Q-learning to choose an action from behavior policy $A_{t+1} \sim b(S_{t+1}, \cdot)$

try to update for finding policy π

$$q(S_t, A_t) \overset{?}{\leftarrow} q(S_t, A_t) + \alpha_t \left(R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) q(S_{t+1}, a) - q(S_t, A_t) \right)$$

policy ภายนอก

if optimal it will be argmax

else can learn more.

When $b = \pi$, we can call this equation Expected SARSA

Off-policy Q-Learning Control

To control, we want to optimize both policies.

For the target policy π, we greedify the action on $q(s, a)$

$$\pi(S_{t+1}) = argmax_{a'} q(S_{t+1}, a')$$

For the behavior policy b, we can use ϵ-greedy on $q(s, a)$

Thus, Q-Learning target is simplified as:

$$R_{t+1} + \gamma \sum_{a} \pi(a|S_{t+1}) q(S_{t+1}, a) = R_{t+1} + \gamma \max_{\mathrm{a}} q(S_{t+1}, a)$$