Q-Learning

Hung-yi Lee

Outline

Introduction of Q-Learning

Tips of Q-Learning

Q-Learning for Continuous Actions

Critic

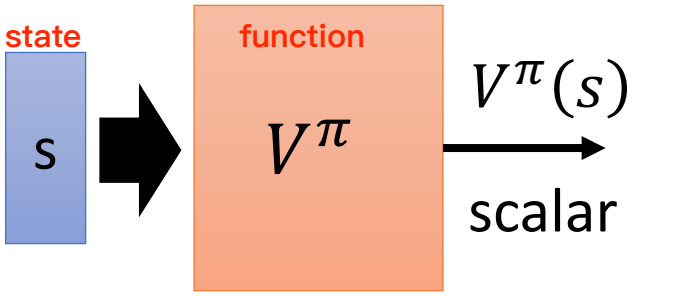
critic只是來評價這個actor好不好

The output values of a critic depend on the actor evaluated.

given一個state s，再配上不同的actor會有不同的 value

- A critic does not directly determine the action.
- Given an actor π, it evaluates how good the actor is
- State value function $V^{\pi}(s)$
  - When using actor π, the *cumulated* reward expects to be obtained after visiting state s

state可能是一個遊戲畫面



output從現在開始到結束所得到可能的reward

$V^{\pi}(s)$ is large

$V^{\pi}(s)$ is smaller

Critic

state　　actor

v以前的阿光(大馬步飛) = bad

v變強的阿光(大馬步飛) = good

左為：critic

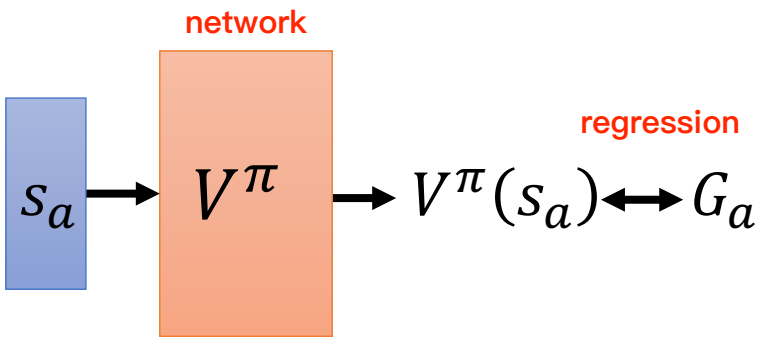
How to estimate $V^{\pi}(s)$

每次都要算cumulative reward，每次要估算就要玩完整場遊戲

- **Monte-Carlo (MC) based approach**
  - The critic watches π playing the game

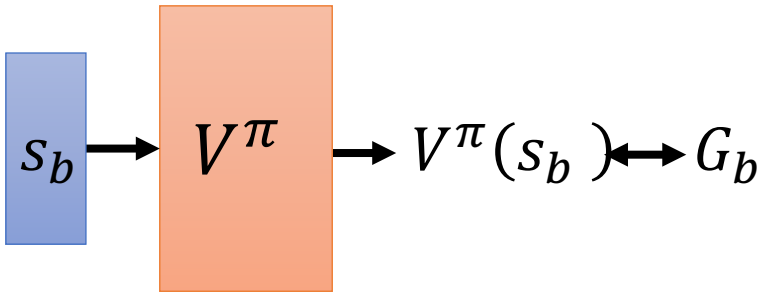
After seeing s_a,

Until the end of the episode, the cumulated reward is G_a



After seeing s_b,

Until the end of the episode, the cumulated reward is G_b



如果今天是使用MC based，則每筆data都要把遊戲玩完一次 這樣太久了！

How to estimate $V^{\pi}(s)$

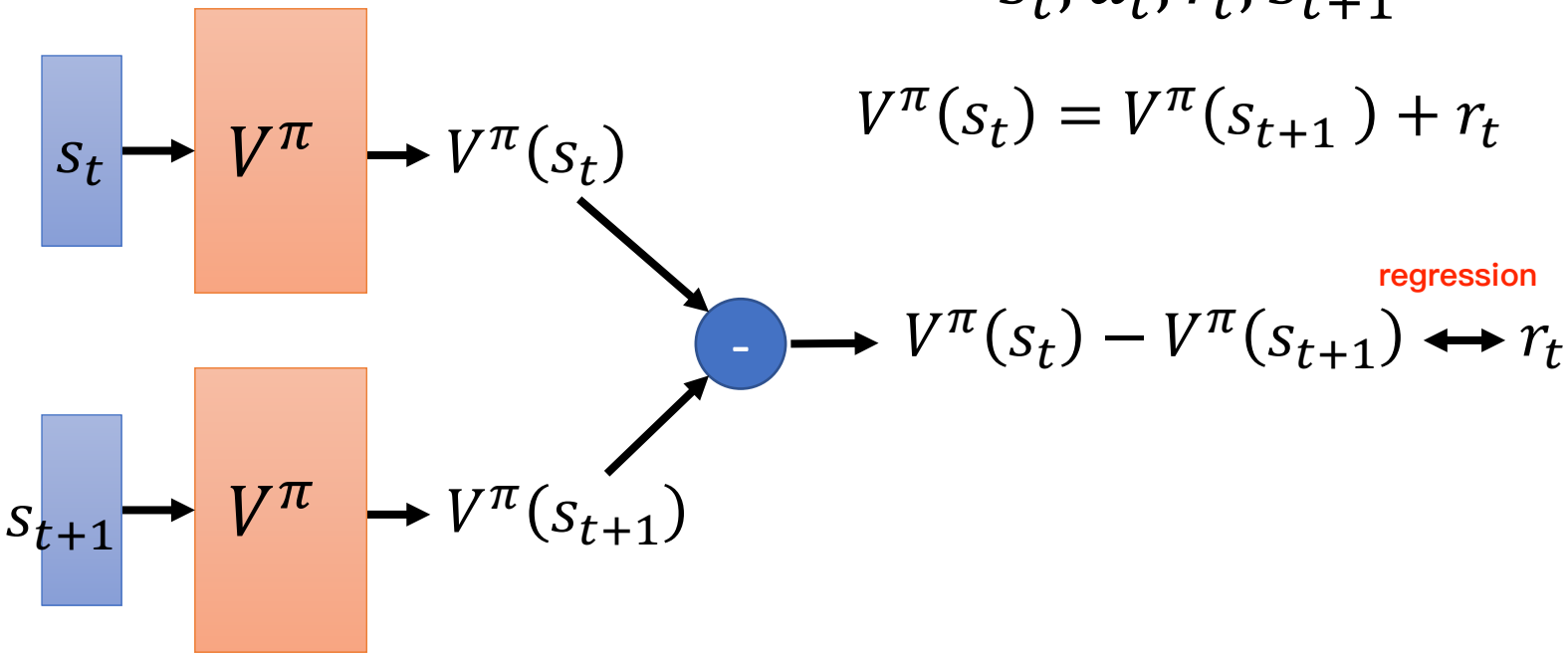
不需要把遊戲玩到底就可以update model

- **Temporal-difference (TD) approach**

因為V是output cumulative scalar

$$\cdots s_t, a_t, r_t, s_{t+1} \cdots$$

$$V^{\pi}(s_t) = V^{\pi}(s_{t+1}) + r_t$$

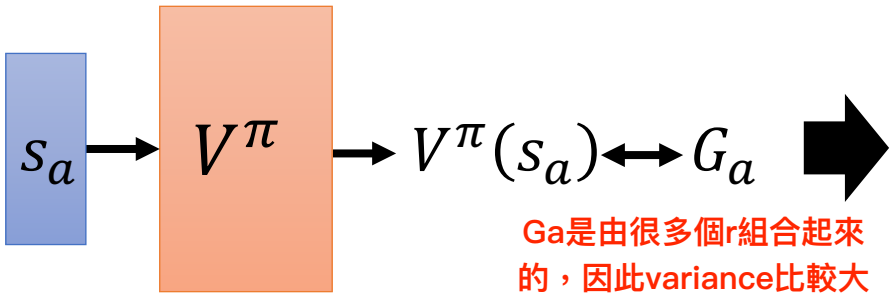


Some applications have very long episodes, so that delaying all learning until an episode's end is too slow.

$$Var[kX] = k^2 Var[X]$$

TD比較常用

MC v.s. TD

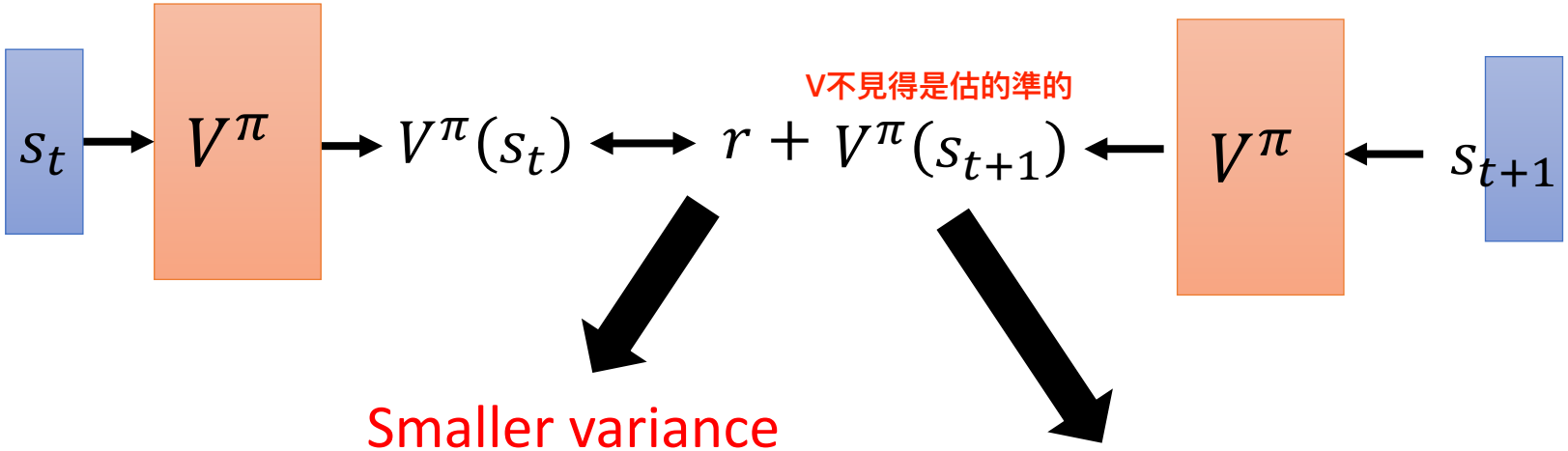


遊戲本身有隨機性的，因此這裡的Ga可能是一個random variable

Larger variance

G_a is the summation of many steps

Ga是由很多個r組合起來的，因此variance比較大



V不見得是估的準的

Smaller variance

May be inaccurate

因為sample太少所以有可能是不準確的

MC v.s. TD

[Sutton, v2, Example 6.4]

- The critic has the following 8 episodes
  - $s_a, r = 0, s_b, r = 0$, END
  - $s_b, r = 1$, END
  - $s_b, r = 1$, END
  - $s_b, r = 1$, END
  - $s_b, r = 1$, END
  - $s_b, r = 1$, END
  - $s_b, r = 1$, END
  - $s_b, r = 0$, END

$$V^{\pi}(s_b) = 3/4$$

$$V^{\pi}(s_a) = ?$$ 0? (MC) 3/4? (TD)

Sa影響到Sb

Monte-Carlo: $V^{\pi}(s_a) = 0$

Sa不會影響到Sb

Temporal-difference:

Sa–>Sb，因此算出來的reward要按照下面式子

$$V^{\pi}(s_a) = V^{\pi}(s_b) + r$$

3/4 3/4 0

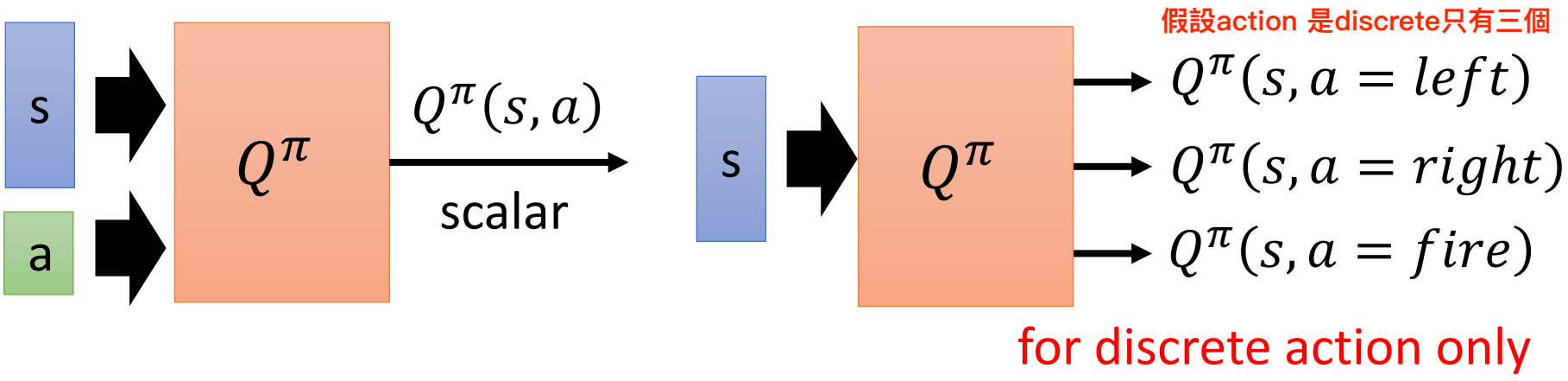
(The actions are ignored here.)

Sa沒有影響到Sb，r=0的trajectory只是巧合

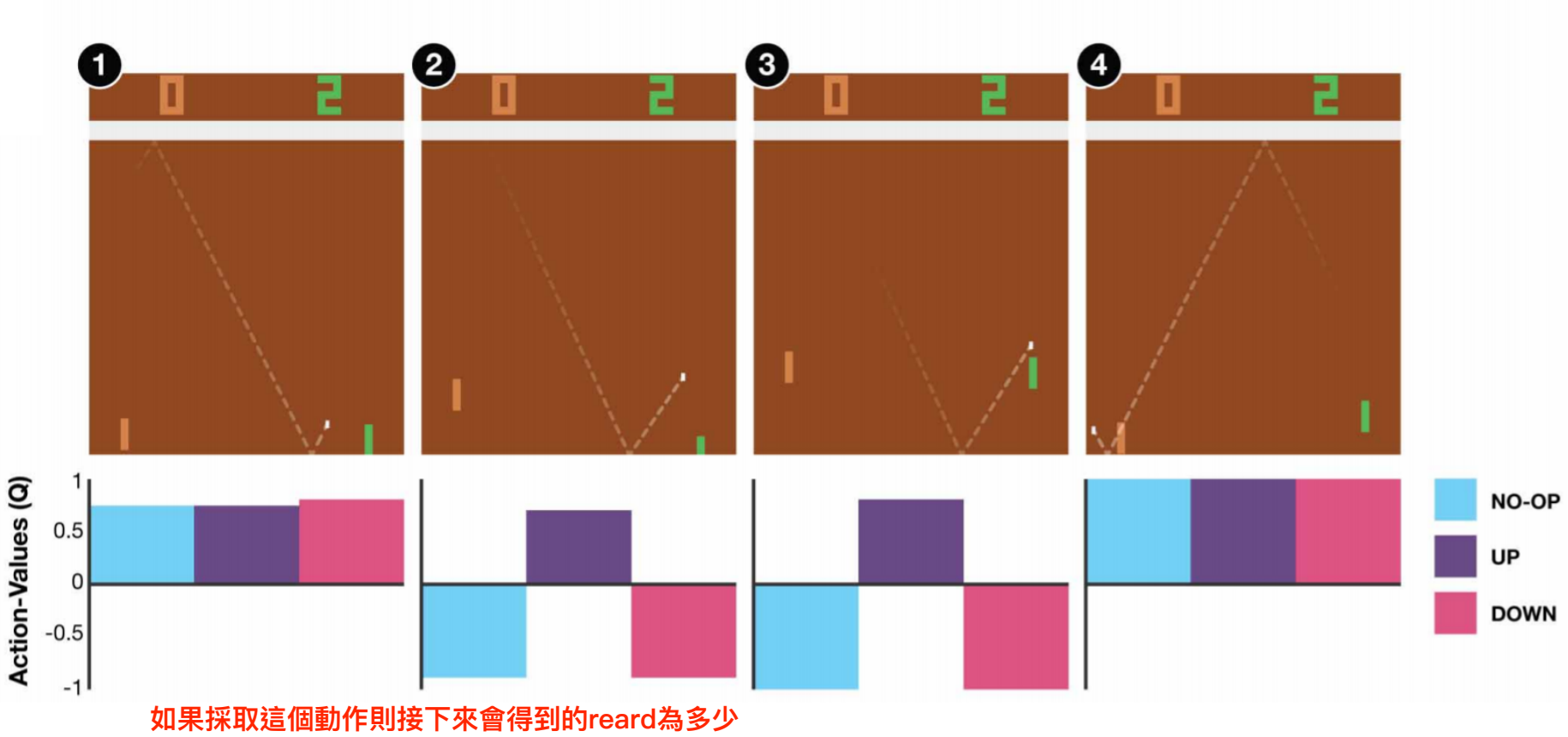
Another Critic

Q：假設在state s 強制採取action a，接下來都用actor pi繼續玩下去

- State-action value function $Q^{\pi}(s,a)$
  - When using actor π, the *cumulated* reward expects to be obtained after taking a at state s



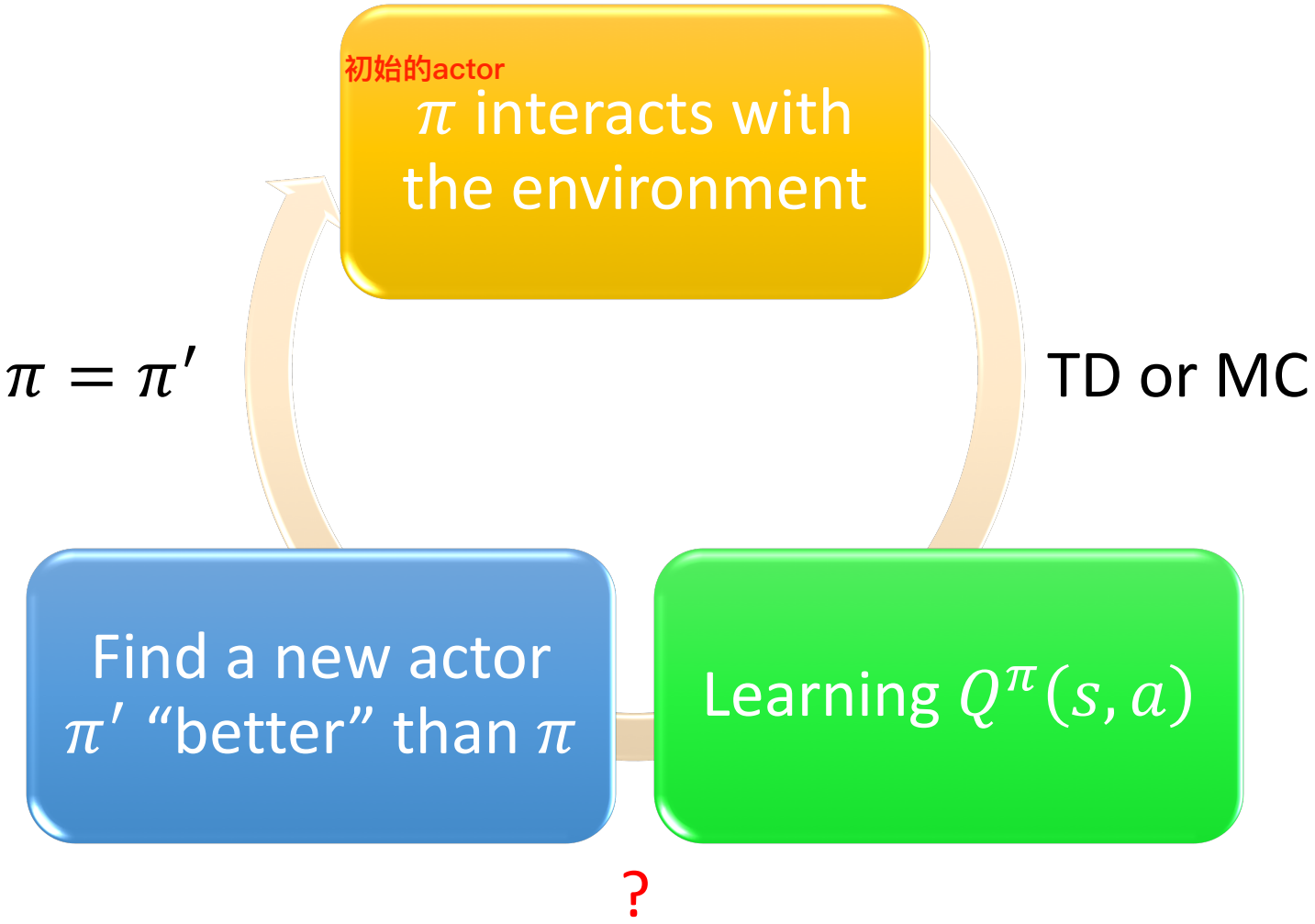
State-action value function



如果採取這個動作則接下來會得到的reard為多少

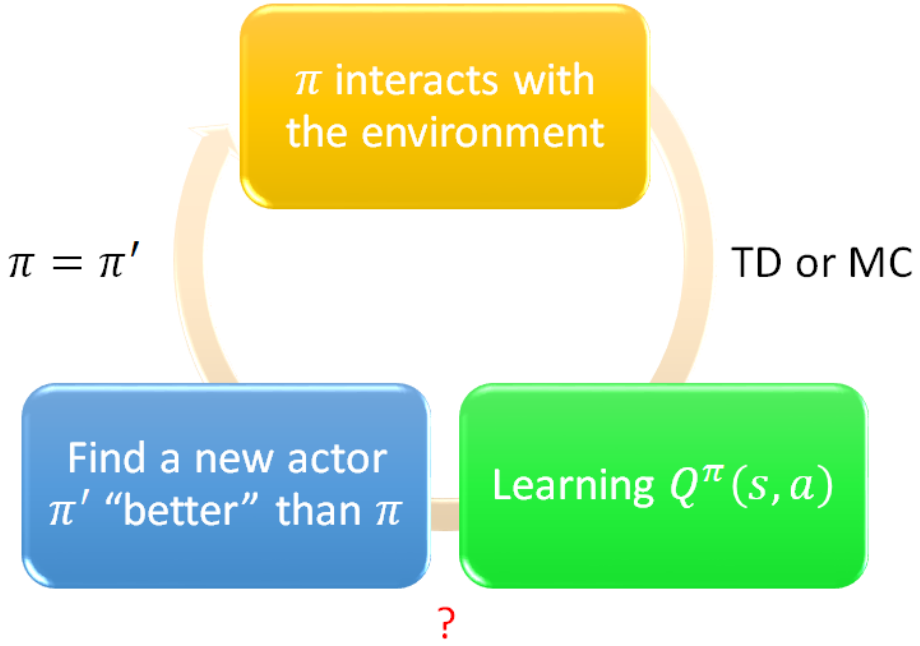
https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf

Another Way to use Critic: Q-Learning



只要擁有一個policy，可以找出Q function，找出Q function後一定可以找到新的一個policy

Q-Learning



- Given $Q^{\pi}(s,a)$, find a new actor π' "better" than π
  - "Better": $V^{\pi'}(s) \geq V^{\pi}(s)$, for all state s

對所有可能的state s 而言，pi的value function一定會小於pi'

$$\pi'(s) = arg\max_{a} Q^{\pi}(s,a)$$

假設已經learn出pi的Q function，則可以透過Q計算哪個action對這個state s可以有最高的 value

- ➢ π' does not have extra parameters. It depends on Q
- ➢ Not suitable for continuous action a (solve it later)

如果a是continuous的話，在解argmax的時候會有問題

但如果是discrete的話則沒有問題

Q-Learning

$$\pi'(s) = arg\max_a Q^{\pi}(s,a)$$

actor pi在state s
所採取的動作

想要證明的objective function

$$V^{\pi'}(s) \geq V^{\pi}(s), \text{ for all state s}$$

$$V^{\pi}(s) = Q^{\pi}\big(s,\pi(s)\big)$$

$$\leq \max_a Q^{\pi}(s,a) = Q^{\pi}\big(s,\pi'(s)\big)$$

$$V^{\pi}(s) \leq \boxed{Q^{\pi}\big(s,\pi'(s)\big)}$$

只有一步之差

$$= E[r_{t+1} + V^{\pi}(s_{t+1}) | s_t = s, a_t = \pi'(s_t)]$$

這邊取期望值，因為遊戲隨機性，某個state採取actor不見得一定會跳到下一個state

$$\leq E[r_{t+1} + Q^{\pi}\big(s_{t+1},\pi'(s_{t+1})\big) | s_t = s, a_t = \pi'(s_t)]$$

$$= E[r_{t+1} + r_{t+2} + V^{\pi}(s_{t+2}) | \dots]$$

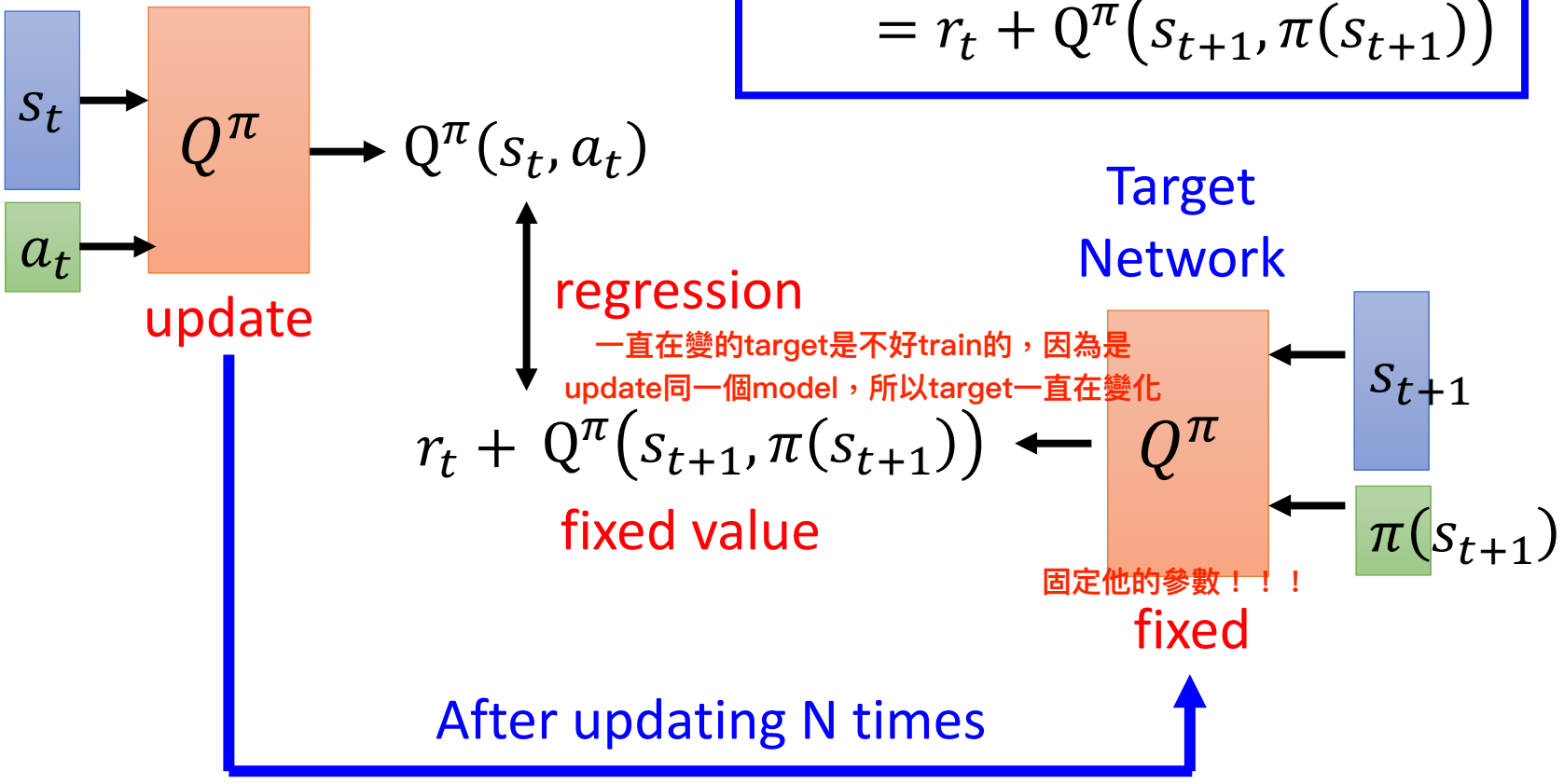
$$\leq E[r_{t+1} + r_{t+2} + Q^{\pi}\big(s_{t+2},\pi'(s_{t+2})\big) | \dots] \dots \leq V^{\pi'}(s)$$

Target Network

$$\cdots s_t, a_t, r_t, s_{t+1} \cdots$$

$$Q^{\pi}(s_t, a_t) = r_t + Q^{\pi}\left(s_{t+1}, \pi(s_{t+1})\right)$$

TD based概念

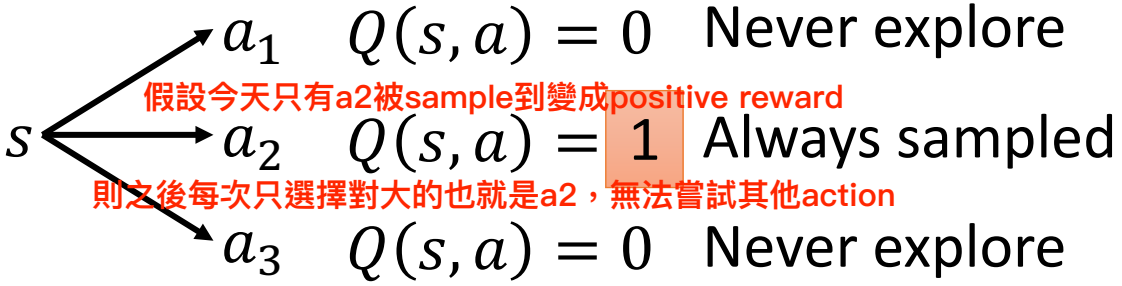


理論上是train了好幾次後再更新一次target network

policy depends on Q function

policy gradient output是stochastic的，是根據action的distribution做sample

Exploration



- The policy is based on Q-function

然而Q function每次都是output固定的！也就是argmax

$$a = arg\max_a Q(s,a)$$

This is not a good way for data collection.

Epsilon Greedy

ε would decay during learning

$$a = \begin{cases} arg\max_a Q(s,a), & with\ probability\ 1-\varepsilon \\ random, & otherwise \end{cases}$$

Boltzmann Exploration

$$P(a|s) = \frac{exp\big(Q(s,a)\big)}{\sum_a exp\big(Q(s,a)\big)}$$

改成output probability，取exp後做normalize，因為Q value有正有負

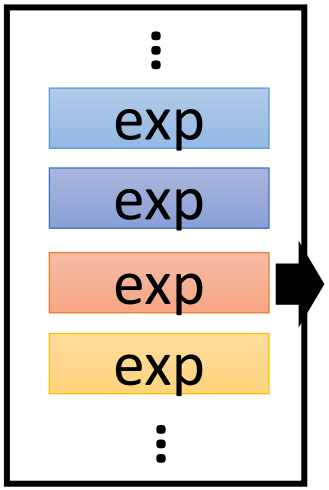
減少跟環境互動所花費的時間，因為RL最長花時間在搜集資料了，現在改成從這個buffer sample pi

把所有蒐集到的data丟到bufer裡面

Replay Buffer

但這個buffer裡面有可能存在不同個pi的action

Put the experience into buffer.



trajectory

s_t, a_t, r_t, s_{t+1}

可能互動10000次就更新pi了，但buffer可能存了50000筆

π interacts with the environment

The experience in the buffer comes from different policies.

Drop the old experience if the buffer is full.

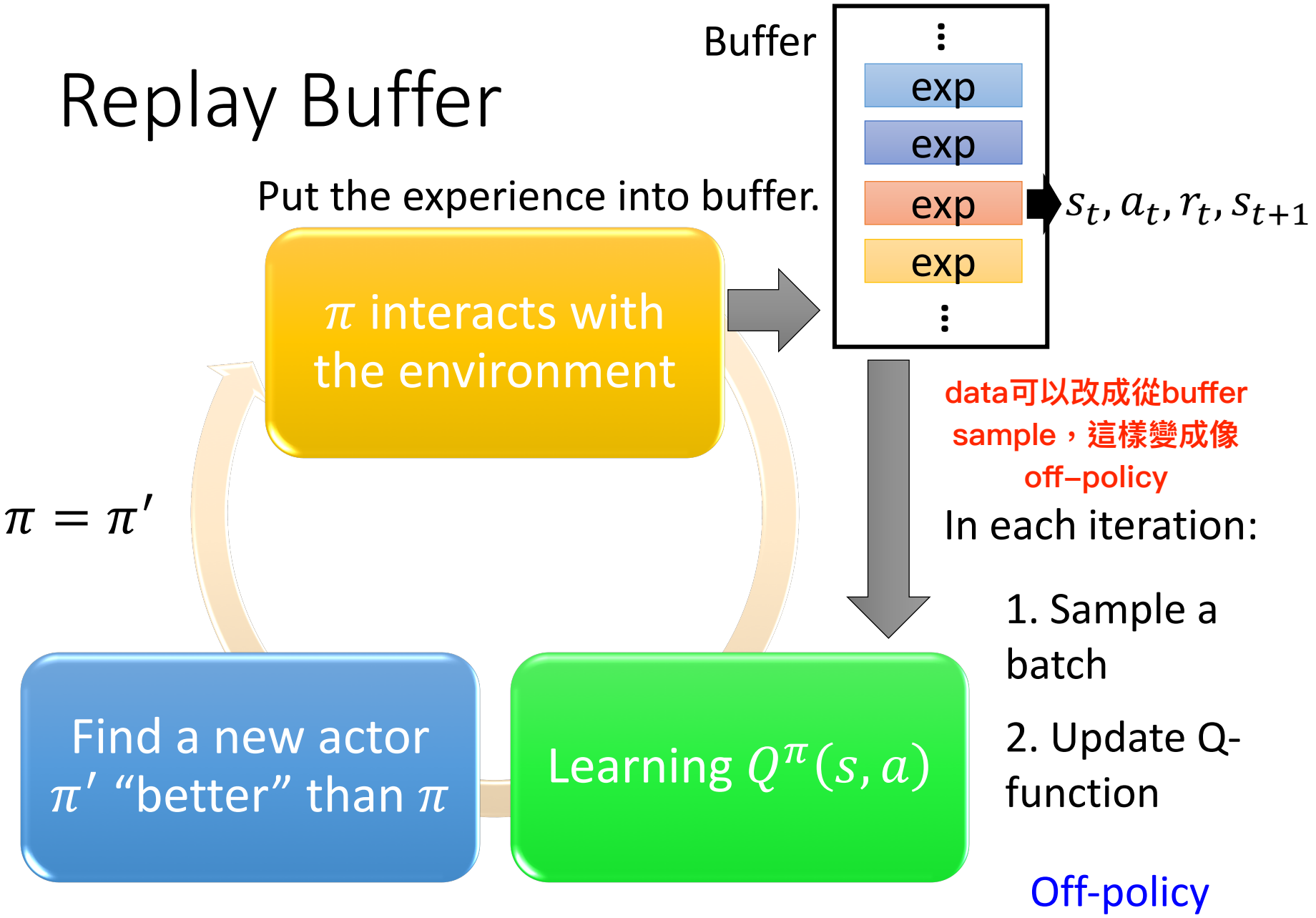
$\pi = \pi'$

Learning $Q^{\pi}(s, a)$

Find a new actor π' "better" than π

Q：buffer裡面存了是不同pi的experience，這樣其實對結果是沒有影響的

Replay Buffer



Typical Q-Learning Algorithm

- Initialize Q-function Q, target Q-function $\hat{Q} = Q$
- In each episode
  - For each time step t 每次互動的過程中
    - Given state s_t, take action a_t based on Q (epsilon greedy) 或是boltzman exploration
    - Obtain reward r_t, and reach new state s_{t+1}
    - Store (s_t, a_t, r_t, s_{t+1}) into butter 如果buffer滿了就丟掉一些
    - Sample (s_i, a_i, r_i, s_{i+1}) from butter (usually a batch)
    - Target $y = r_i + \max_a \hat{Q}(s_{i+1}, a)$ 讓Q value最大的a
    - Update the parameters of Q to make $Q(s_i, a_i)$ close to y (regression)
    - Every C steps reset $\hat{Q} = Q$ update幾次後就更新一次參數

Outline

Introduction of Q-Learning

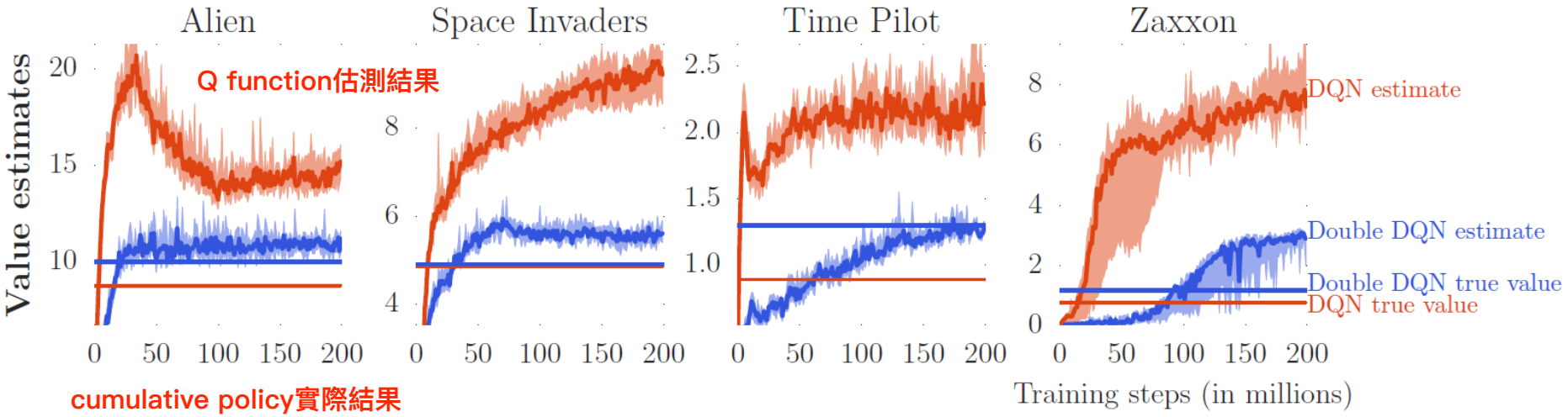
Tips of Q-Learning

Q-Learning for Continuous Actions

Double DQN

Q value往往是被高估的

- Q value is usually over-estimated



Double DQN

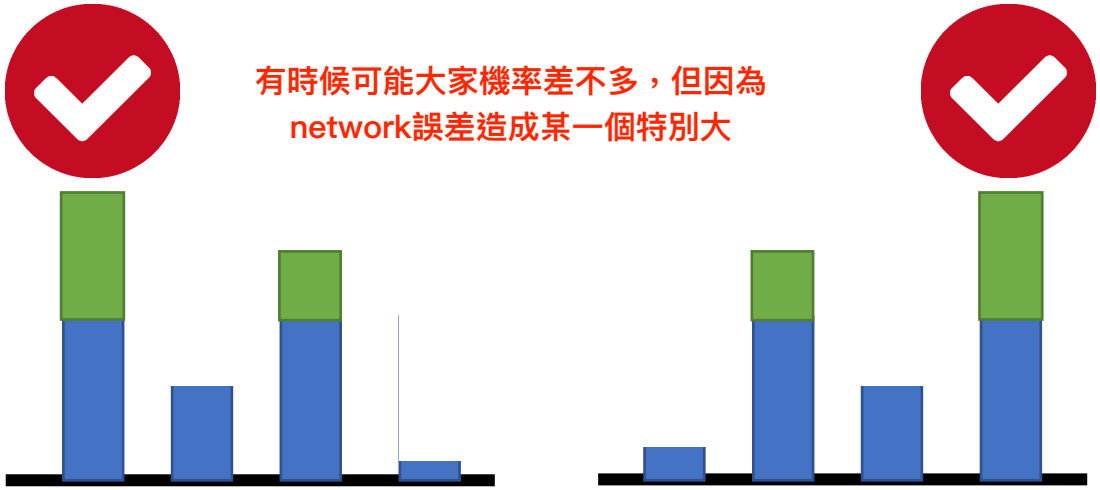
- Q value is usually over estimate

每次都選高估的那個機率

$$Q(s_t, a_t) \longleftrightarrow r_t + \max_a Q(s_{t+1}, a)$$

Tend to select the action that is over-estimated

$Q(s_{t+1}, a)$



Double DQN

- Q value is usually over estimate

Q不只挑選最大的action，還直接計算Q value

$$Q(s_t, a_t) \longleftrightarrow r_t + \max_a Q(s_{t+1}, a)$$

哪來兩個Q呢？有！！在train的時候有一個target network!!

- Double DQN: two functions Q and Q' Target Network

$$Q(s_t, a_t) \longleftrightarrow r_t + Q'\left(s_{t+1}, arg\max_a Q(s_{t+1}, a)\right)$$

Q負責挑選最大的action，但計算value的是Q`

If Q over-estimate a, so it is selected. Q' would give it proper value.

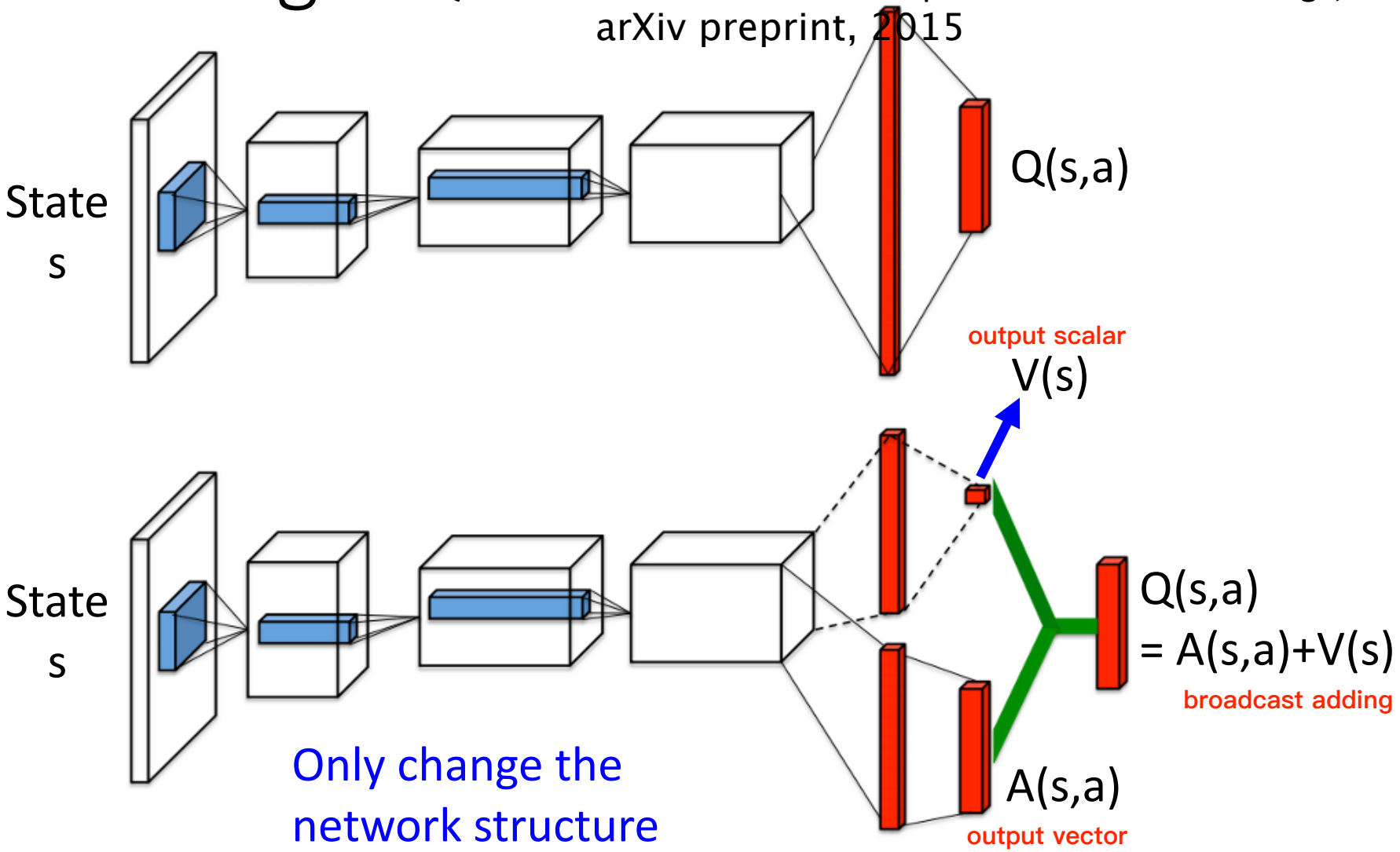
How about Q' overestimate? The action will not be selected by Q.

Hado V. Hasselt, "Double Q-learning", NIPS 2010
Hado van Hasselt, Arthur Guez, David Silver, "Deep Reinforcement Learning with Double Q-learning", AAAI 2016

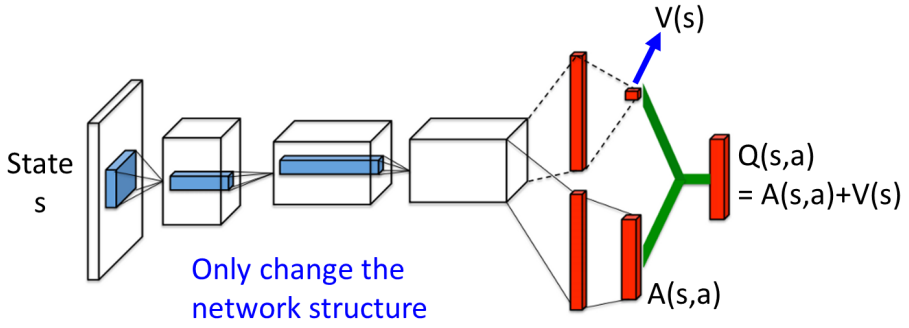
改network架構

Dueling DQN

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, Nando de Freitas, "Dueling Network Architectures for Deep Reinforcement Learning", arXiv preprint, 2015



Dueling DQN



藍色是target

Q(s,a) = V(s) + A(s,a)

action	state			
	3	~~3~~ 4	3	1
	1	~~-1~~ 0	6	1
	2	~~-2~~ -1	3	1
V(s)	2	~~0~~ 1	4	1
A(s,a)	1	3	-1	0
	-1	-1	2	0
	0	-2	-1	0

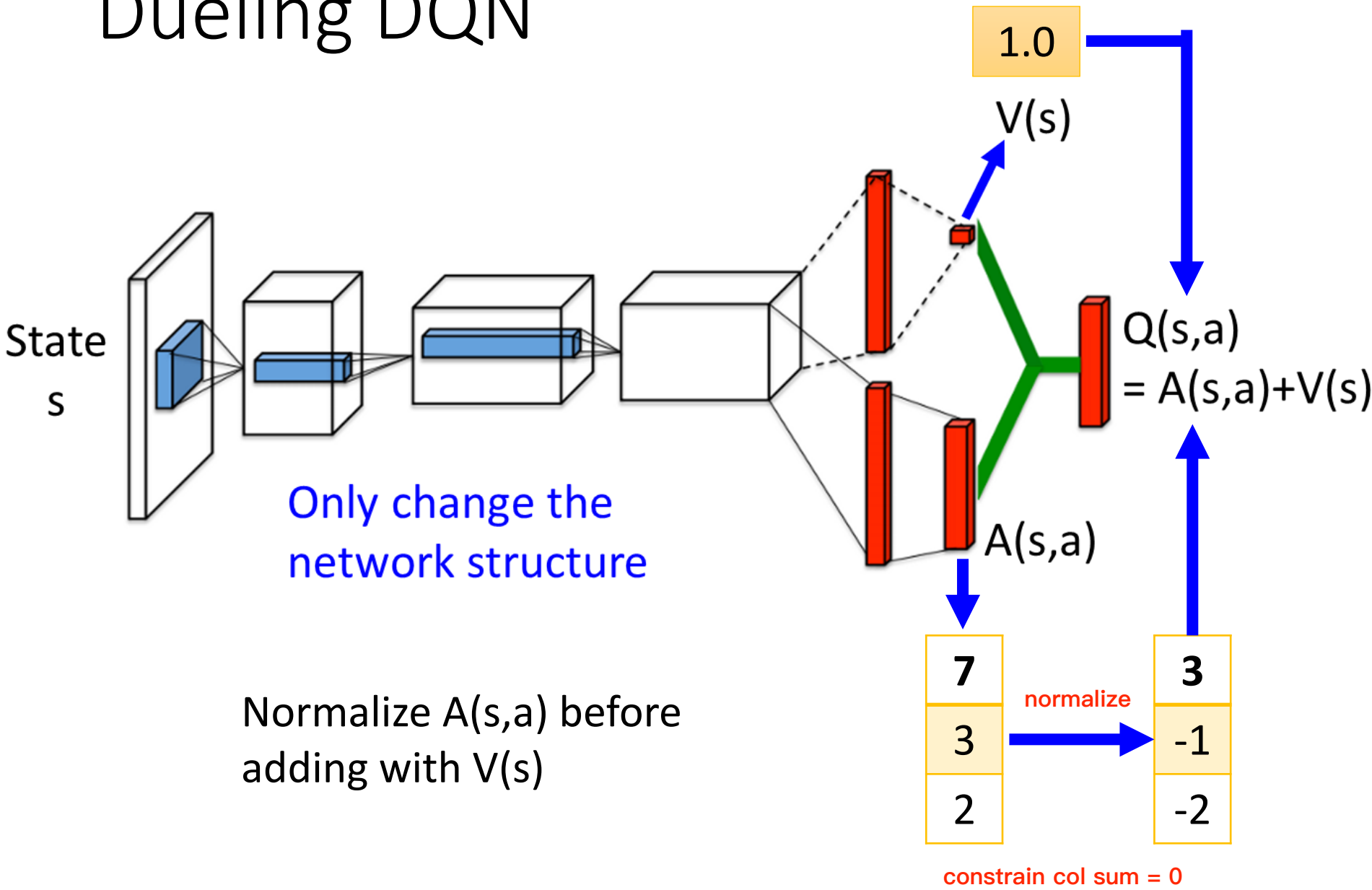
假設只有sample到3, –1，希望能把它改成4, 0
這時候有可能machine去更動V(s)，這樣即使2沒有被sample倒也能夠被estimate

Average of column

為了避免machine只updateA9s,a)

sum of column = 0

Dueling DQN



Normalize A(s,a) before adding with V(s)

https://arxiv.org/abs/1511.05952?context=cs

然而更改sample機率的話，在update 參數的時候也需要做不同weight的調整, more details from paper

Prioritized Reply

The data with larger TD error in previous training has higher probability to be sampled.

(s_t, a_t, r_t, s_{t+1})

Experience Buffer

從buffer sample出來的data每筆重要性不同
因此多考慮這個

假設今天發現某比data的TD error很大，表示這筆data我們train不好，因此我們可以多sample幾次（調高sample到的機率）多訓練

Parameter update procedure is also modified.

s_t, a_t → Q → $Q(s_t, a_t)$

TD error

$r_t + \hat{Q}(s_{t+1}, a_{t+1})$

$a_{t+1} = arg\max_a \hat{Q}(s_{t+1}, a)$

s_{t+1}, a_{t+1} → $\hat{Q}$

平衡MC based and TD based方法

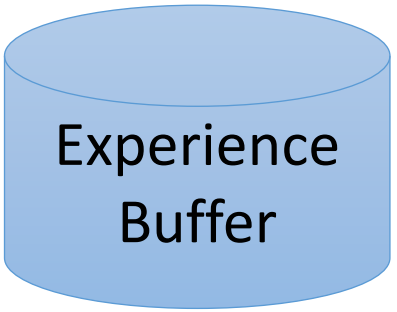
Multi-step

Balance between MC and TD

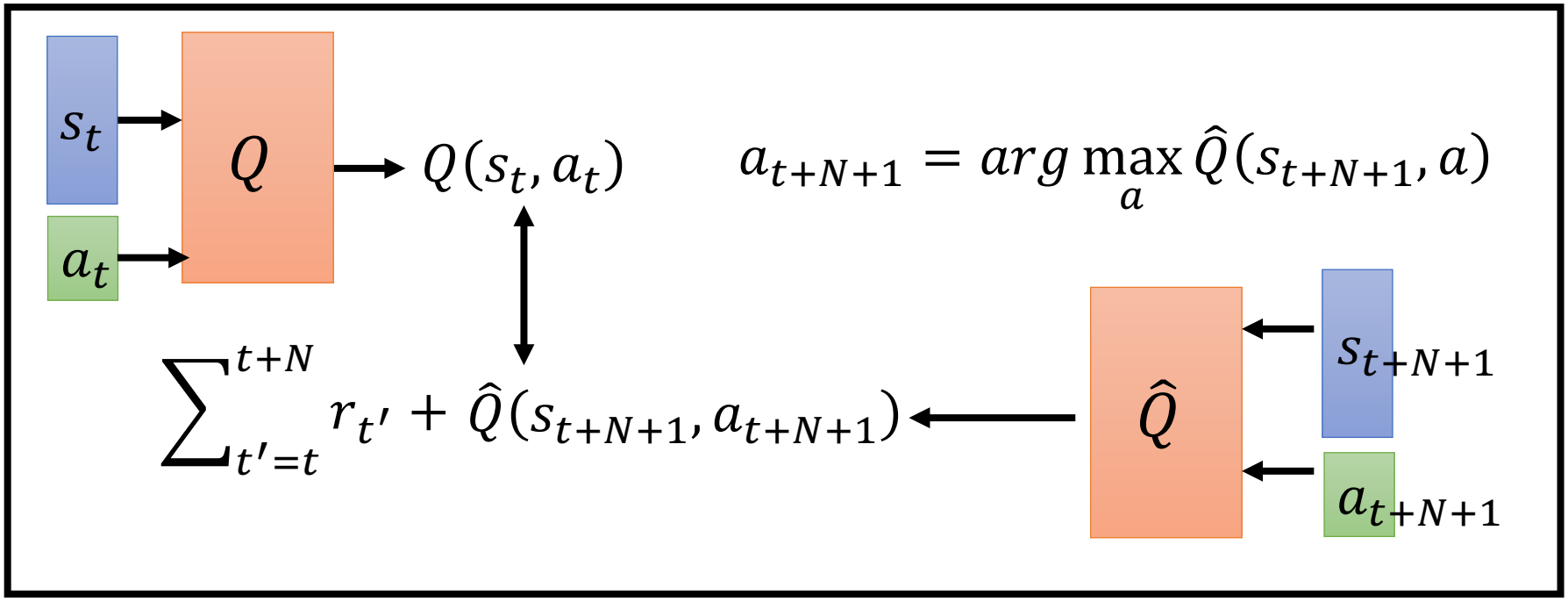
以往是只sample一個state，現在改成sample N 個state

$$(s_t, a_t, r_t, \cdots, s_{t+N}, a_{t+N}, r_{t+N}, s_{t+N+1})$$

~~(s_t, a_t, r_t, s_{t+1})~~



N是hyper parameters，需要調整的



Noisy Net

https://arxiv.org/abs/1706.01905
https://arxiv.org/abs/1706.10295

- Noise on Action (Epsilon Greedy)

$$a = \begin{cases} arg\max_a Q(s,a), & with\ probability\ 1-\varepsilon \\ random, & otherwise \end{cases}$$

- Noise on Parameters

$$a = arg\max_a \tilde{Q}(s,a)$$

Inject noise into the parameters of Q-function **at the beginning of each episode**

在network參數上加上 gaussian noise

$Q(s,a)$ → Add noise → $\tilde{Q}(s,a)$

在頑童一場遊戲的時候network一定要是固定！

The noise would **NOT** change in an episode.

在每個episode開始的時候sample noise加入network，接著全程使用這個network去玩遊戲

Noisy Net

每次output action都有可能是不一樣的（因為有noise），這樣不好train
くㄩ

- Noise on Action
  - Given the same state, the agent may takes different actions.
  - No real policy works in this way

隨機亂試

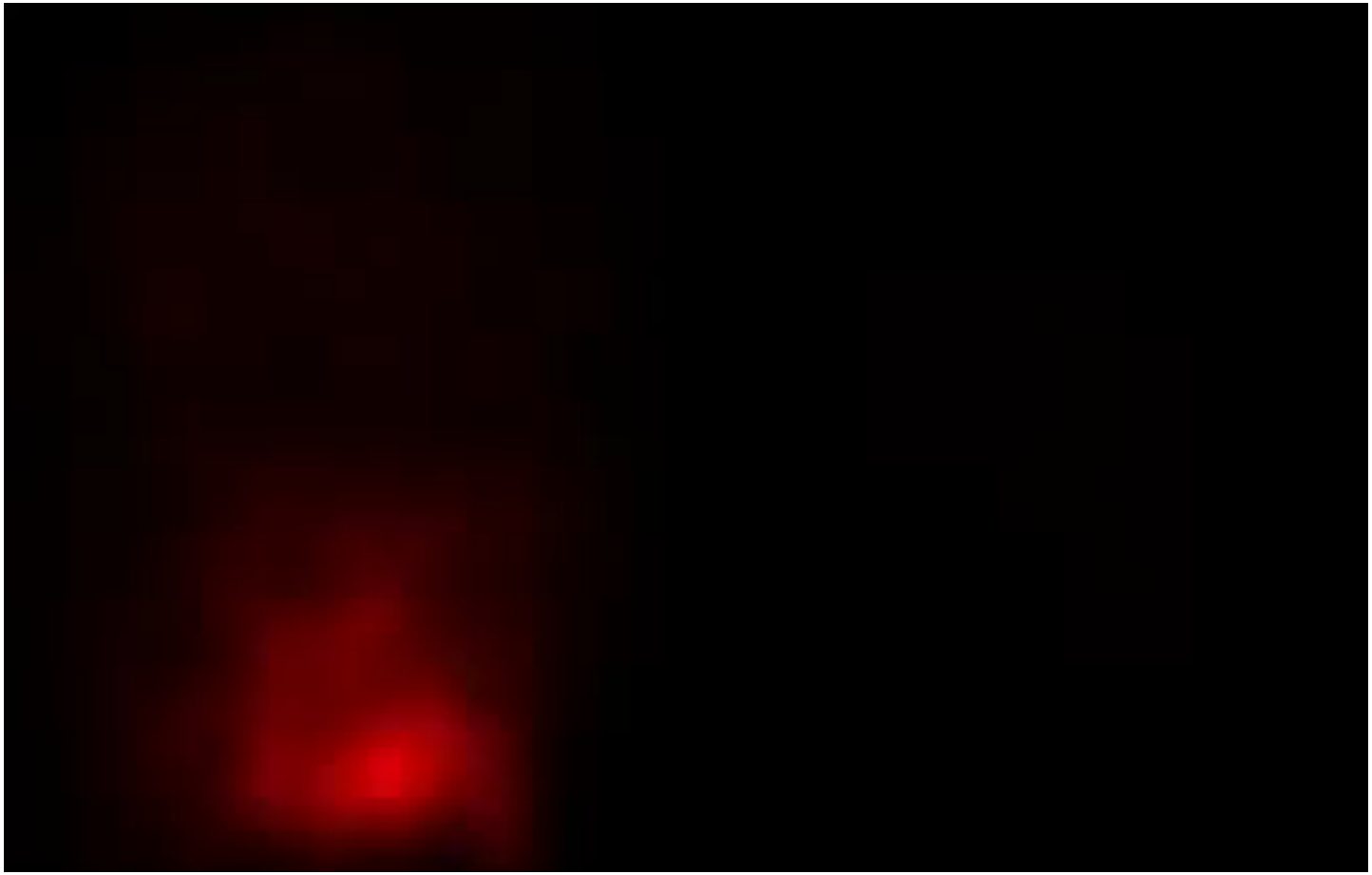
- Noise on Parameters

在整個互動過程中network參數是固定的，導致最後決定action是固定的

  - Given the same (similar) state, the agent takes the same action.
    - ⟶ State-dependent Exploration
  - Explore in a *consistent* way

有系統地試

Dueling DQN - Visualization



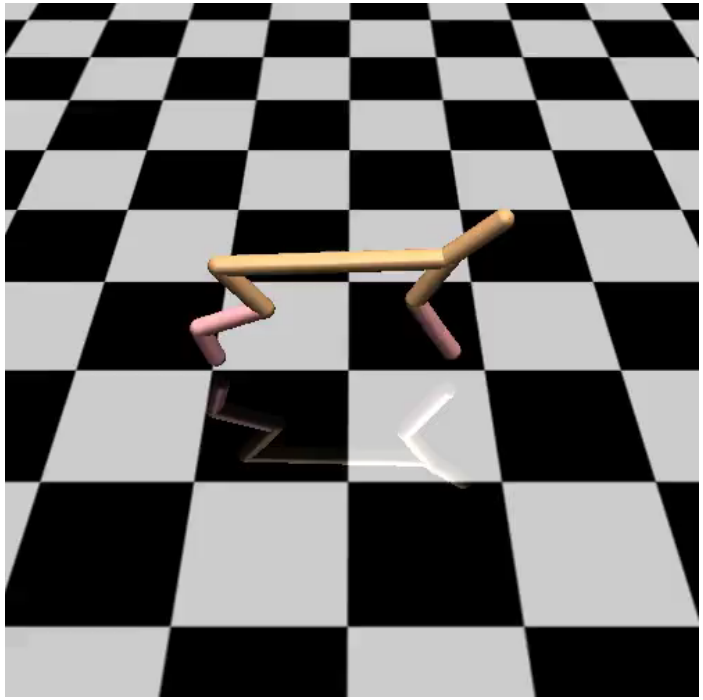
(from the link of the original paper)

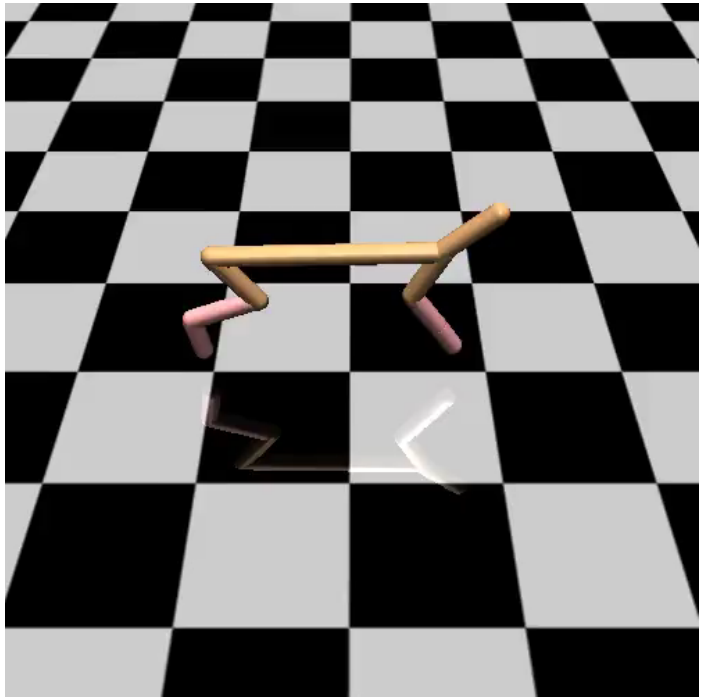
Dueling DQN - Visualization

(from the link of the original paper)

Demo

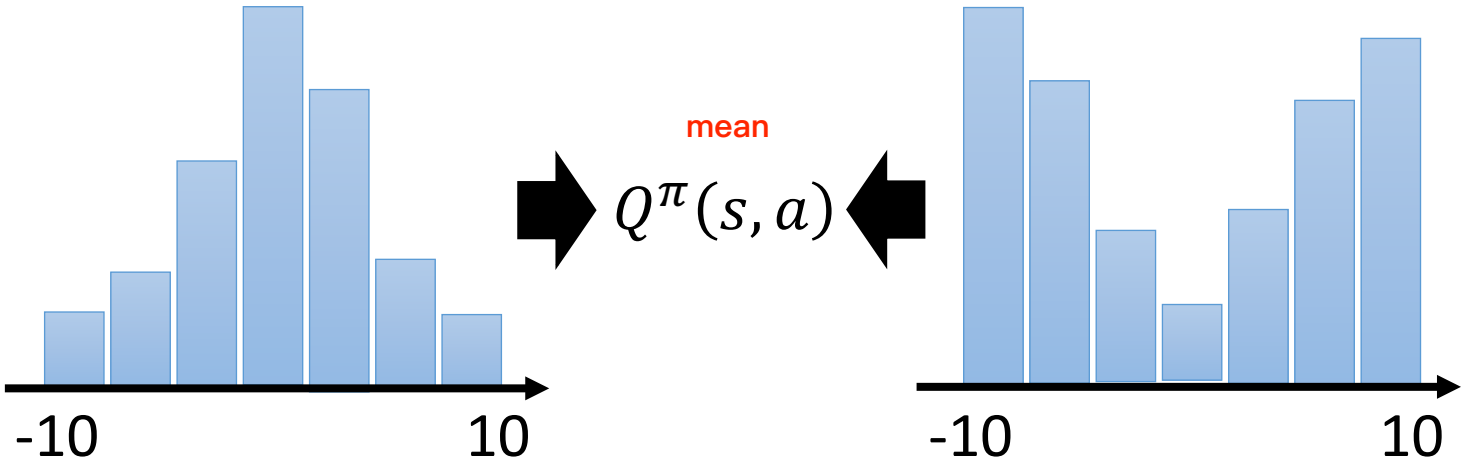
https://blog.openai.com/better-exploration-with-parameter-noise/





Distributional Q-function

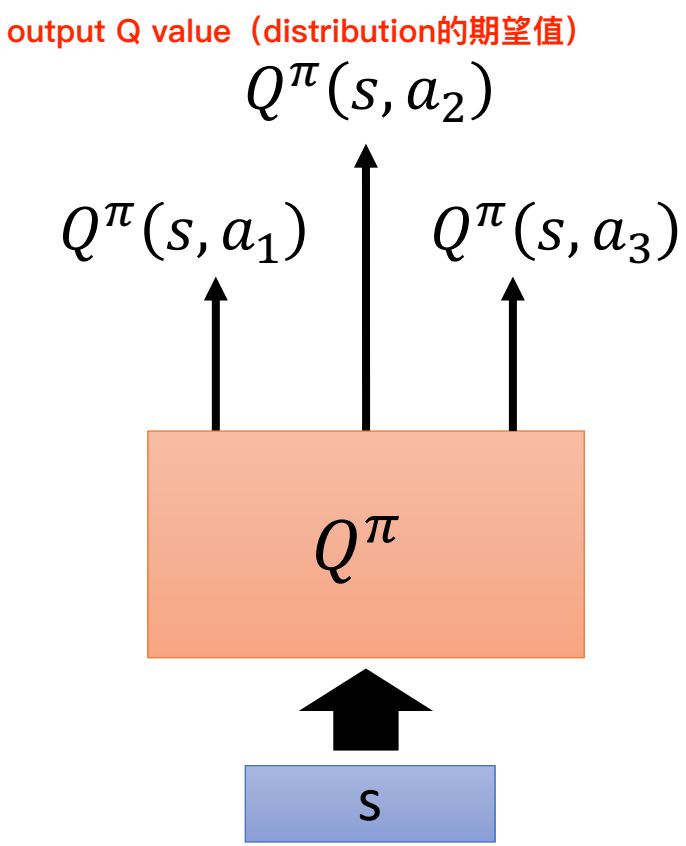
- State-action value function $Q^{\pi}(s,a)$
  - When using actor π, the *cumulated* reward expects to be obtained after seeing observation s and taking a



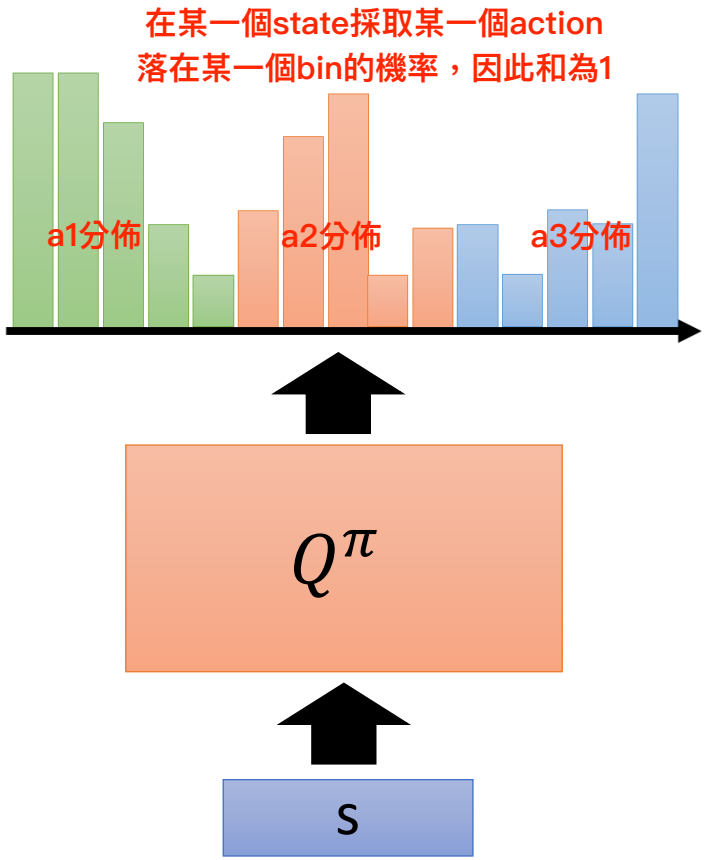
沒有辦法model整個distribution
不能只用mean來代表

Different distributions can have the same values.

Distributional Q-function

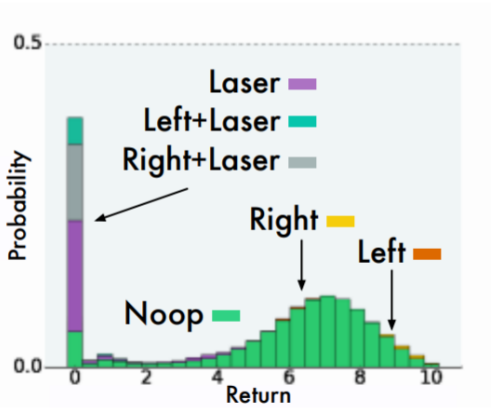


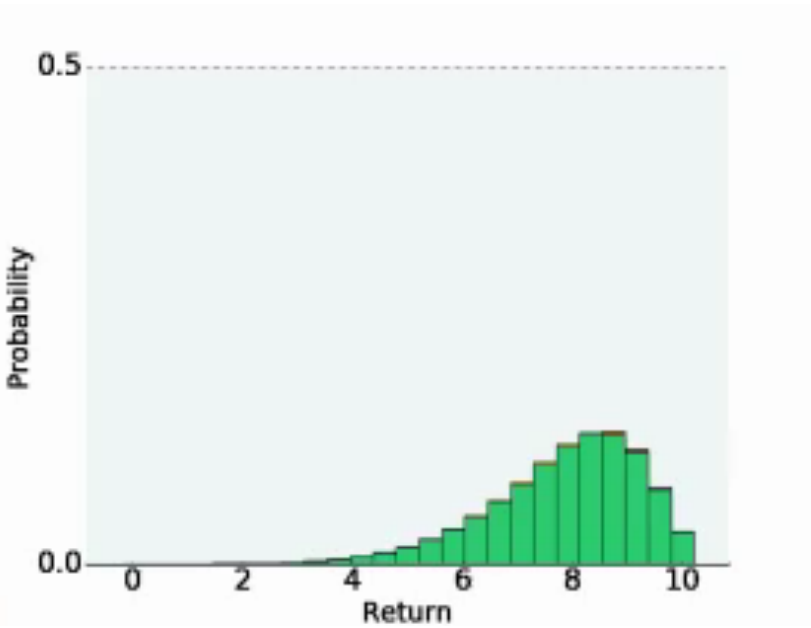
A network with 3 outputs



A network with 15 outputs
(each action has 5 bins)

Demo

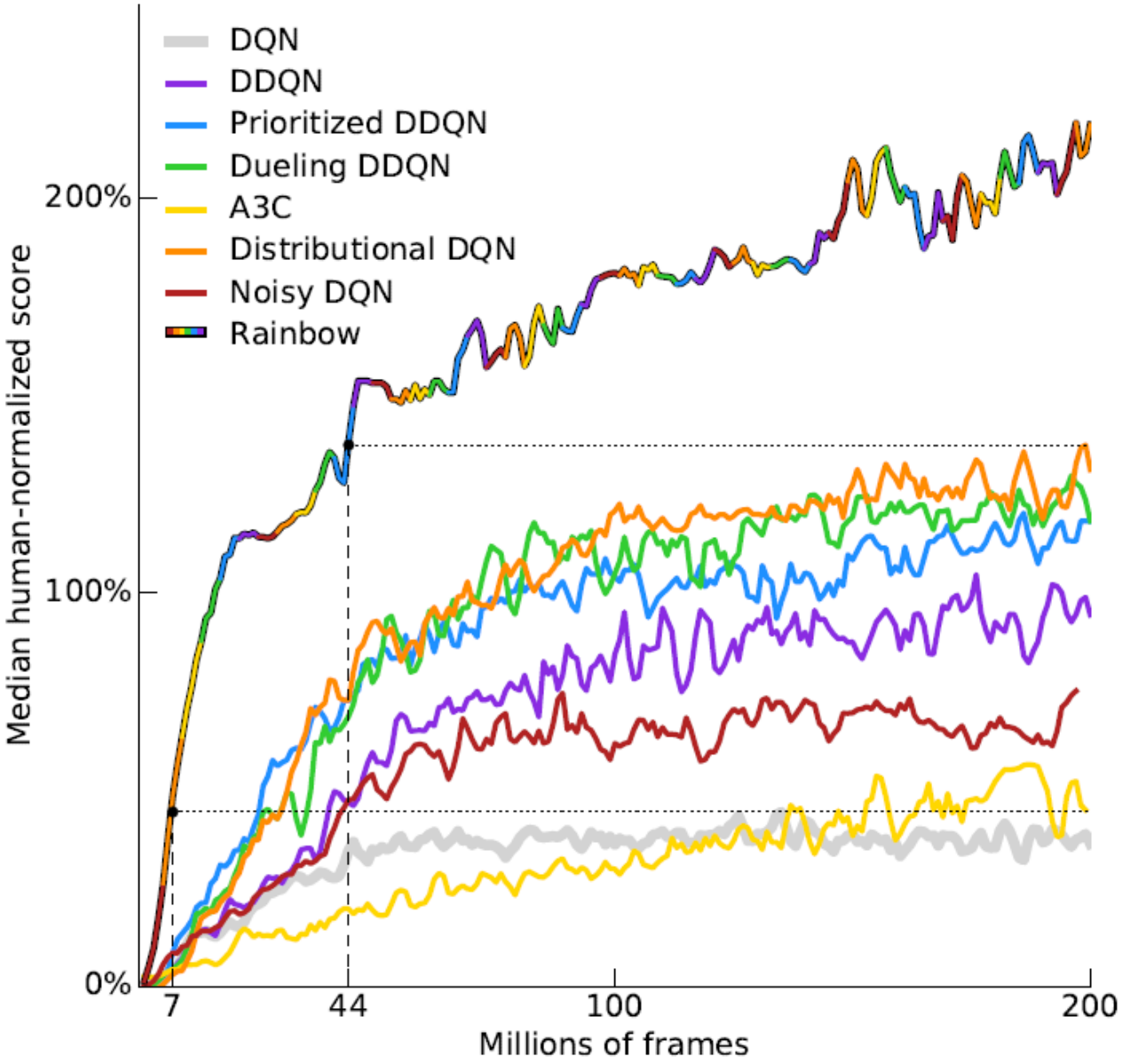




https://youtu.be/yFBwyPuO2Vg

Rainbow

A3C已經包含了multi-step
(TD/MC mix)

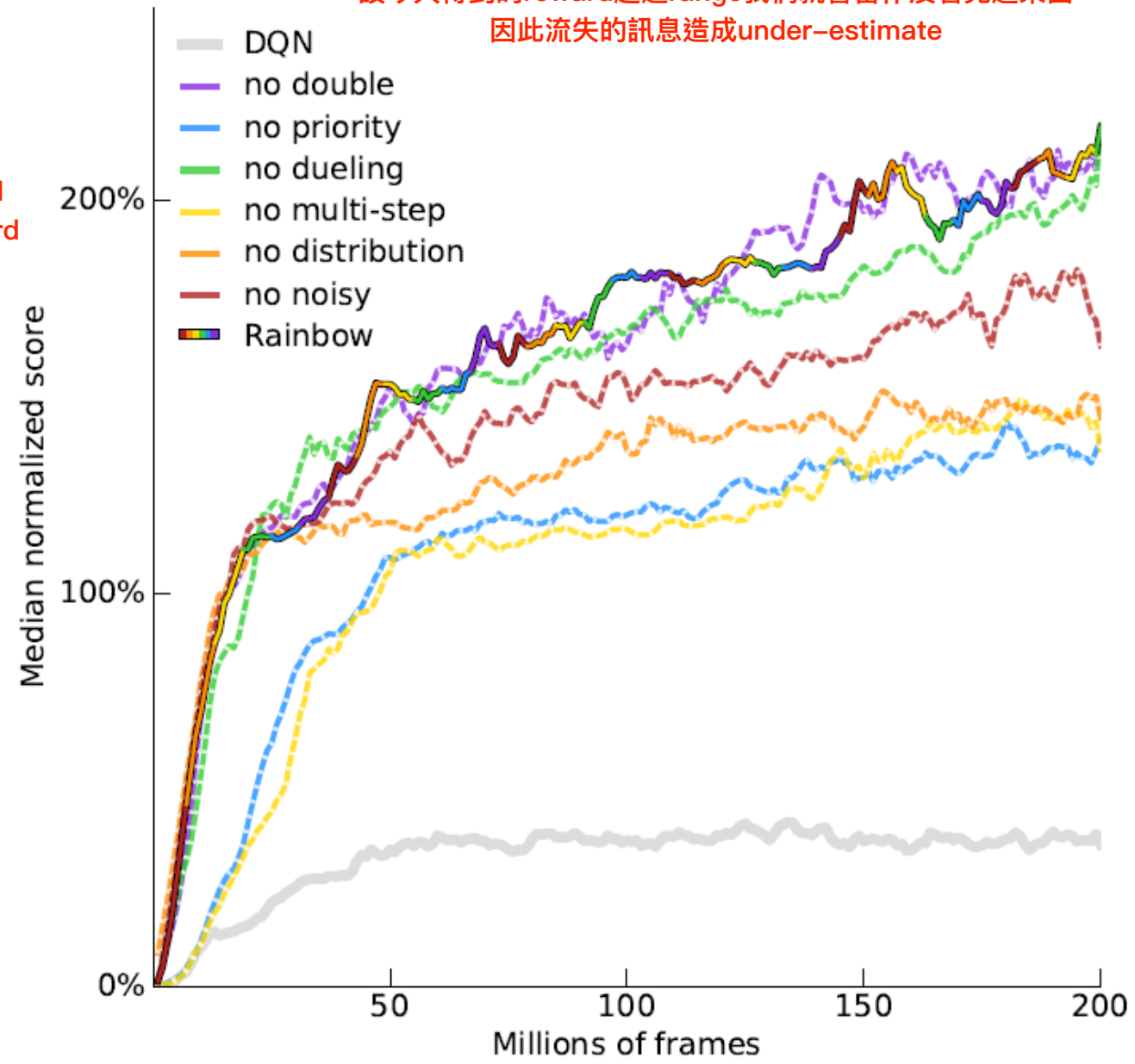


https://arxiv.org/abs/1710.02298

Rainbow

沒有double沒差，因為有distributional DQN就已經不會有over-estimate reward

因為distributional DQN output是一個distribution range，假設今天得到的reward超過range我們就會當作沒看見這東西因此流失的訊息造成under-estimate



https://arxiv.org/abs/1710.02298

Outline

Introduction of Q-Learning

Tips of Q-Learning

Q-Learning for Continuous Actions

policy gradient相對Q learning是比較難train的

因為Q learning保證可以找到Q`來improve model

Continuous Actions

小遊戲可能只有上下左右，就是discrete

假如是自駕車，則是continuous，像是左轉幾度右轉幾度

- Action a is a *continuous vector*

$$a = arg\max_{a} Q(s, a)$$

假如是discrete，則a通常是有限的

但如果是continuous則有無限多組，找argmax很麻煩

因此又要改成sampling

Solution 1 用sample找max

Sample a set of actions: $\{a_1, a_2, \cdots, a_N\}$

See which action can obtain the largest Q value

Solution 2 用gradient ascent找max

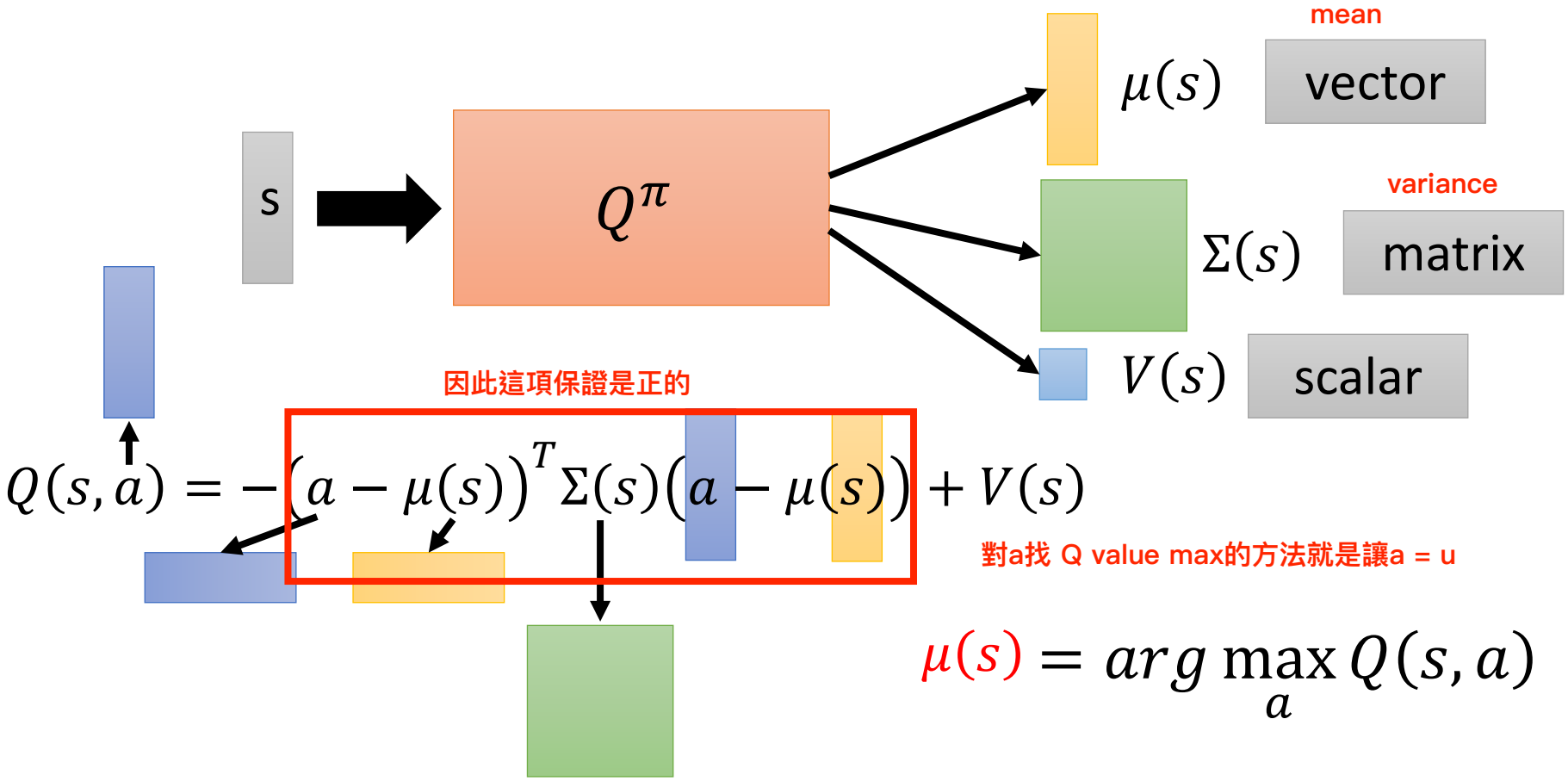
Using gradient ascent to solve the optimization problem.

特別設計一個Q function找出argmax

Continuous Actions

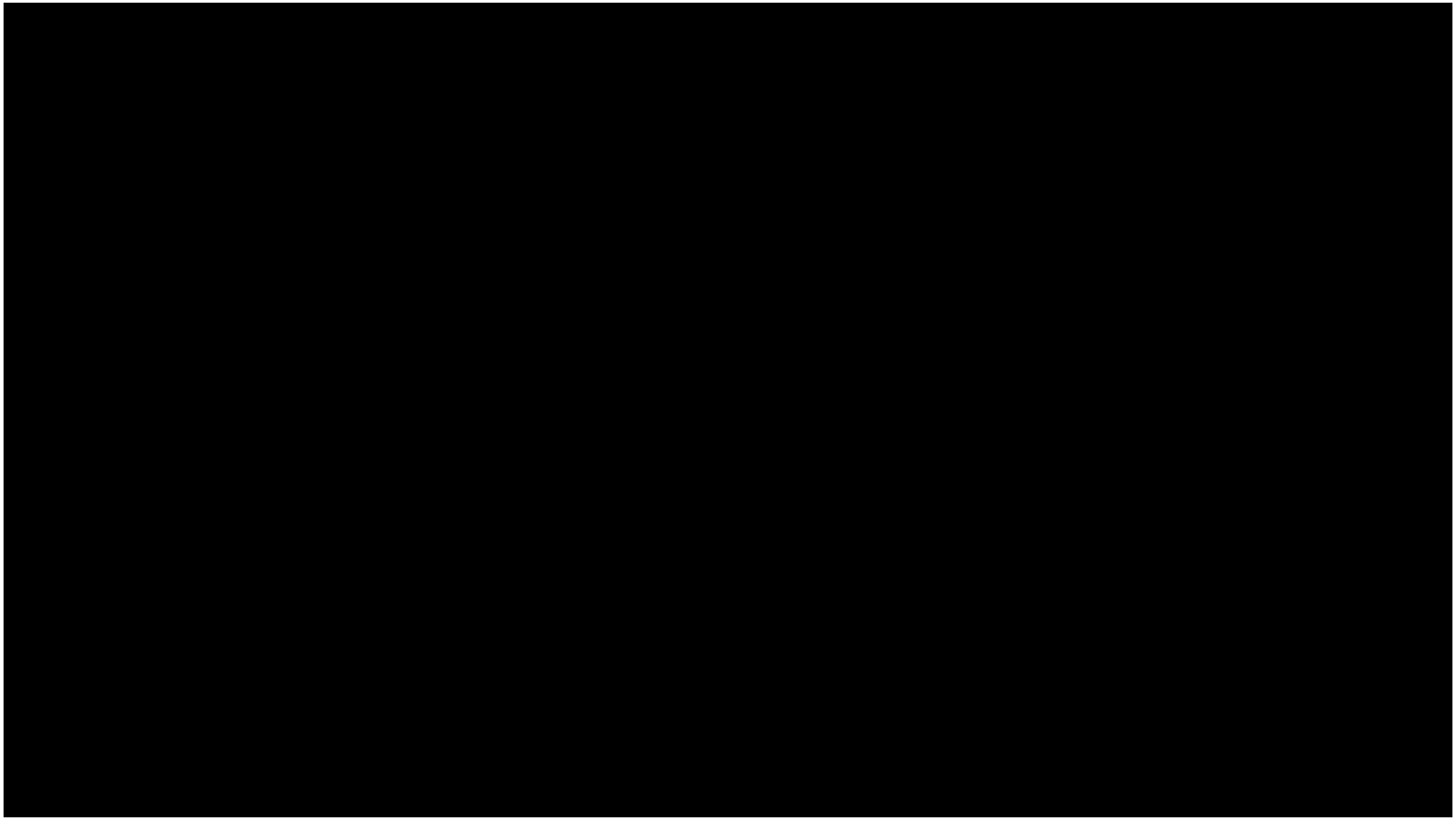
這個matrix是positive definite
在paper中這個matrix(A)會先做處理，最後output的是A的transpose*A
因此保證半正定

Solution 3 Design a network to make the optimization easy.



對a找 Q value max的方法就是讓a = u

$$\mu(s) = arg\max_a Q(s,a)$$



https://www.youtube.com/watch?v=ZhsEKTo7V04

Continuous Actions

Solution 4 Don't use Q-learning