

# Q-Learning

Hung-yi Lee

# Outline

Introduction of Q-Learning

Tips of Q-Learning

Q-Learning for Continuous Actions

# Critic

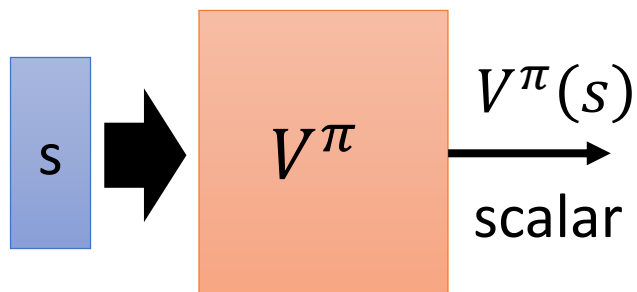
EX: 大/小馬步飛

EX: 佐為

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

EX: 以前/以後的阿光的圍棋實力

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



$V^\pi(s)$  is large



$V^\pi(s)$  is smaller

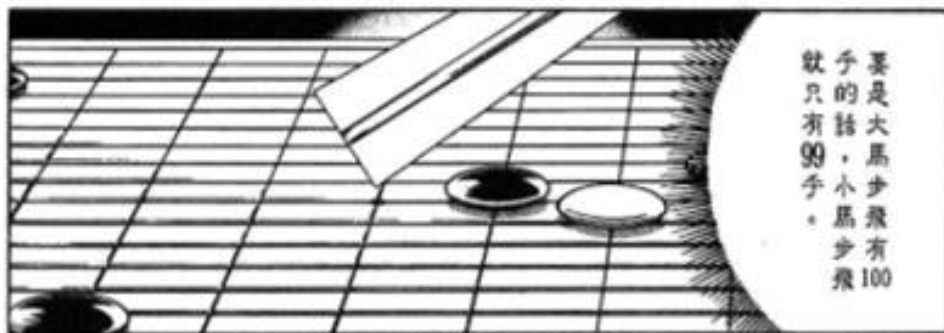
# Critic

v以前的阿光(大馬步飛) = bad

v變強的阿光(大馬步飛) = good



※ 小馬步飛：盤內棋一樣，將棋子放在同一格；大馬步飛則是放在斜對角格。

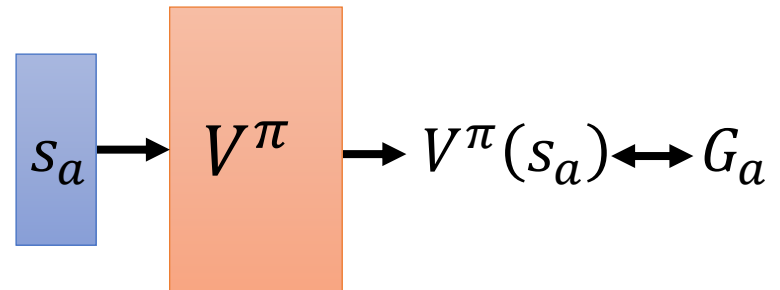


# How to estimate $V^\pi(s)$

- **Monte-Carlo (MC) based approach**
  - The critic watches  $\pi$  playing the game

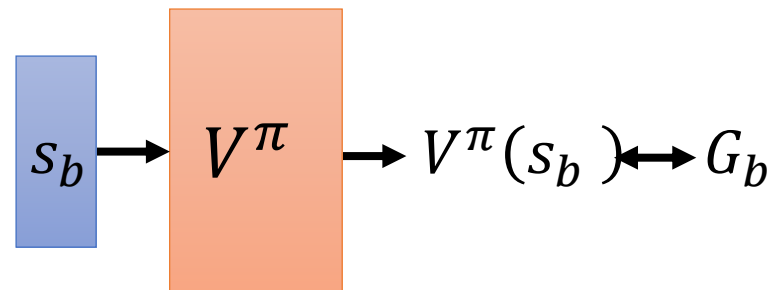
After seeing  $s_a$ ,

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



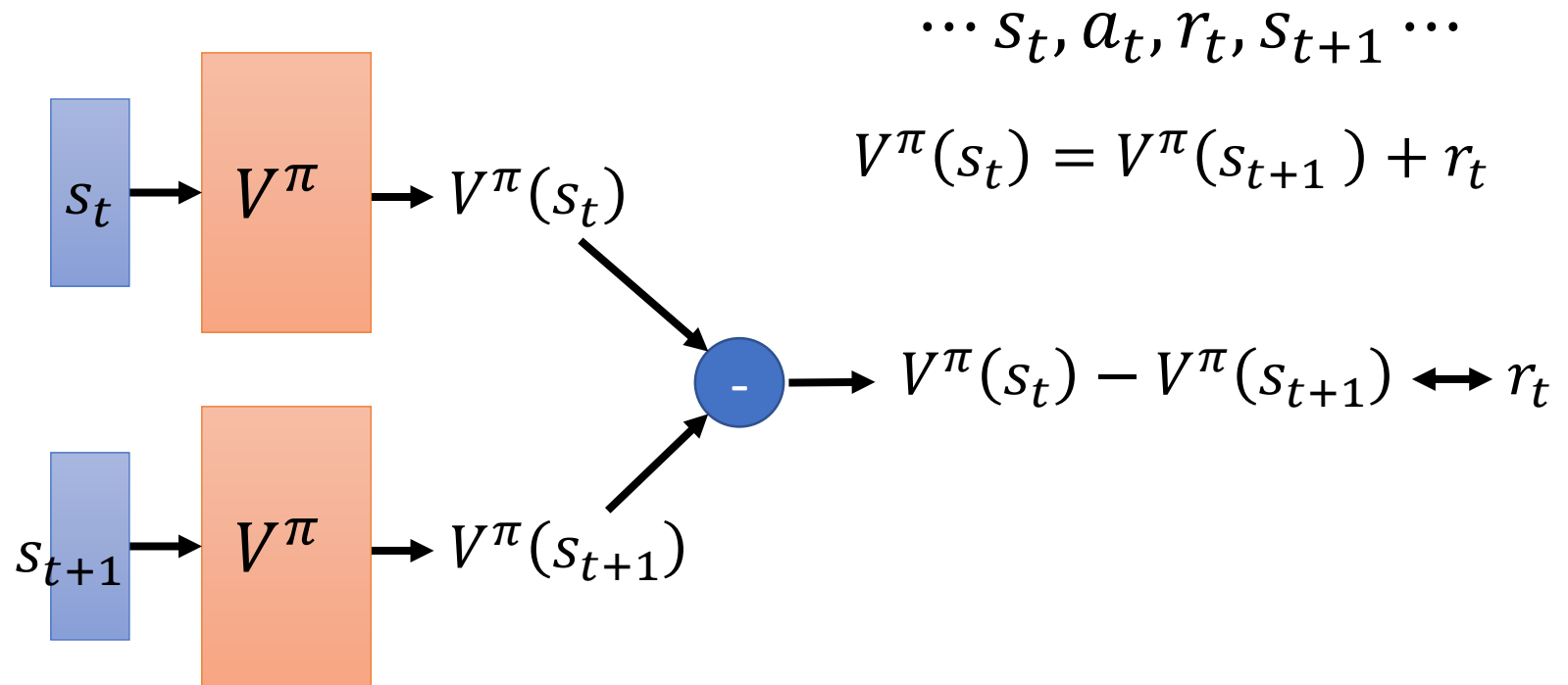
After seeing  $s_b$ ,

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



# How to estimate $V^\pi(s)$

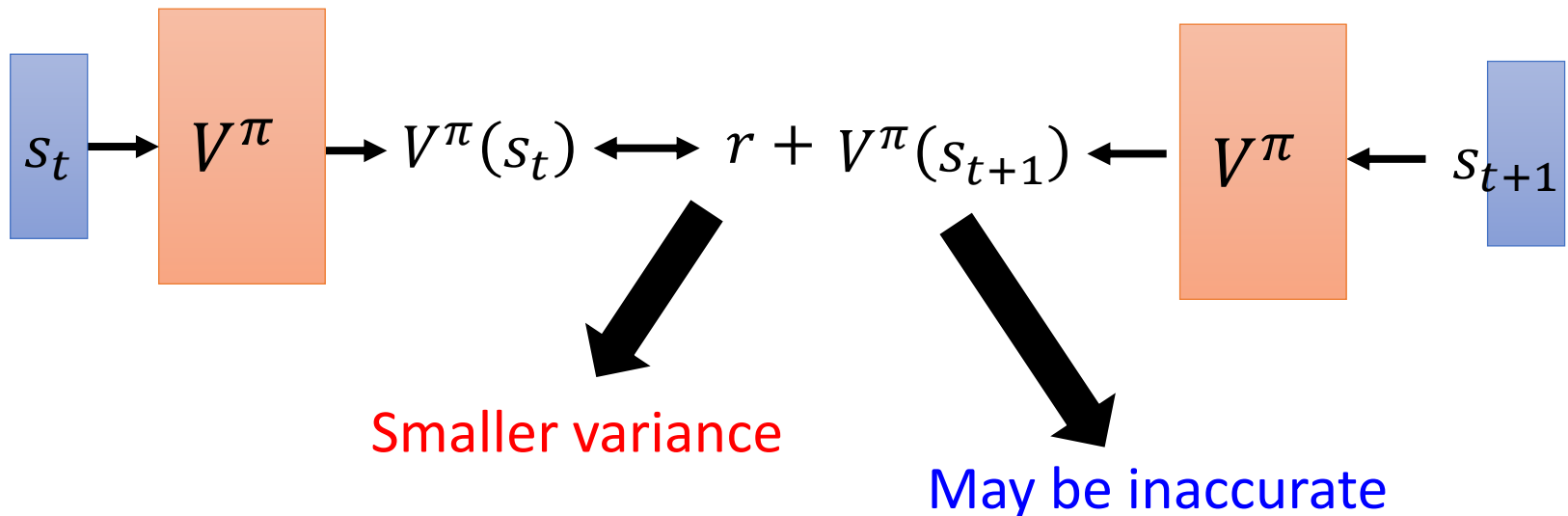
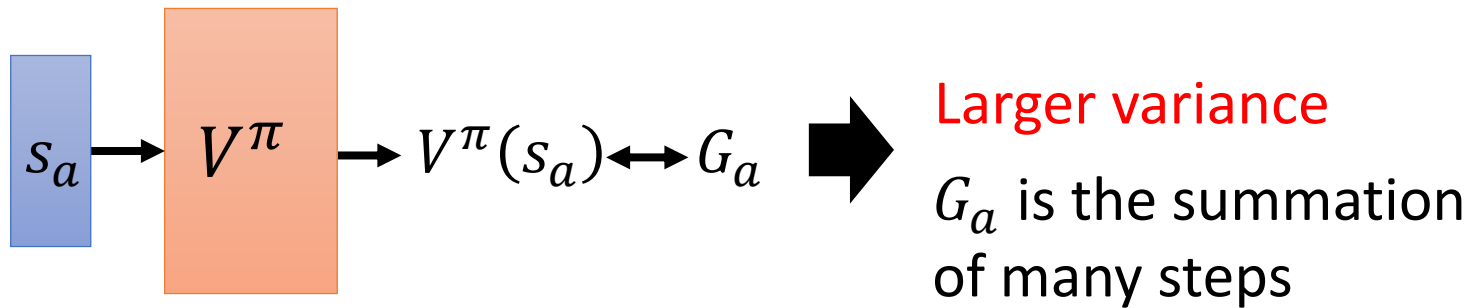
- **Temporal-difference (TD) approach**



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

$$\text{Var}[kX] = k^2 \text{Var}[X]$$

# MC v.s. TD



# MC v.s. TD

[Sutton, v2,  
Example 6.4]

- The critic has the following 8 episodes

- $s_a, r = 0, s_b, r = 0, \text{END}$

- $s_b, r = 1, \text{END}$

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

- $s_b, r = 1, \text{END}$

- $s_b, r = 1, \text{END}$

$$V^\pi(s_a) = ? \quad 0? \quad 3/4?$$

- $s_b, r = 1, \text{END}$

- $s_b, r = 1, \text{END}$

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

- $s_b, r = 1, \text{END}$

Temporal-difference:

- $s_b, r = 0, \text{END}$

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

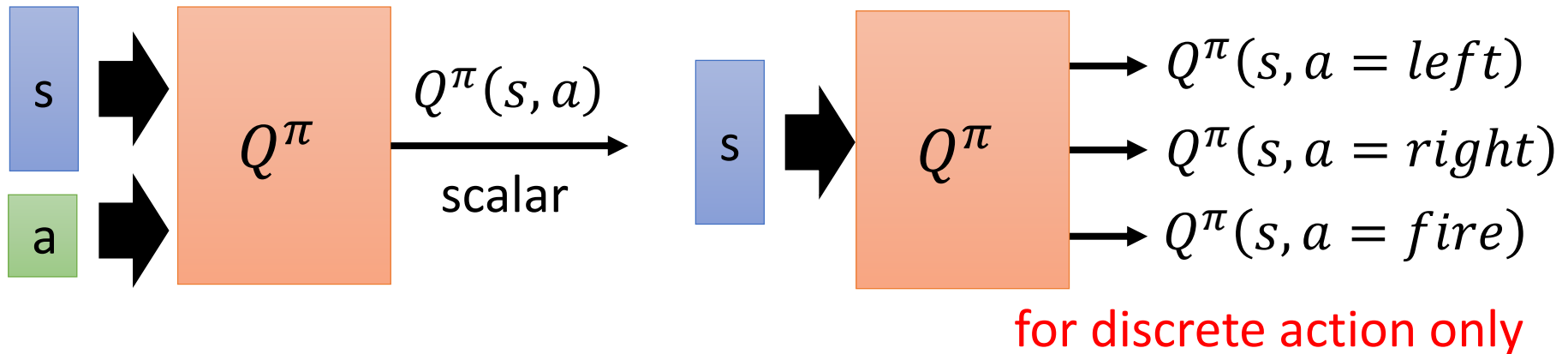
$3/4 \qquad 3/4 \qquad 0$

(The actions are ignored here.)

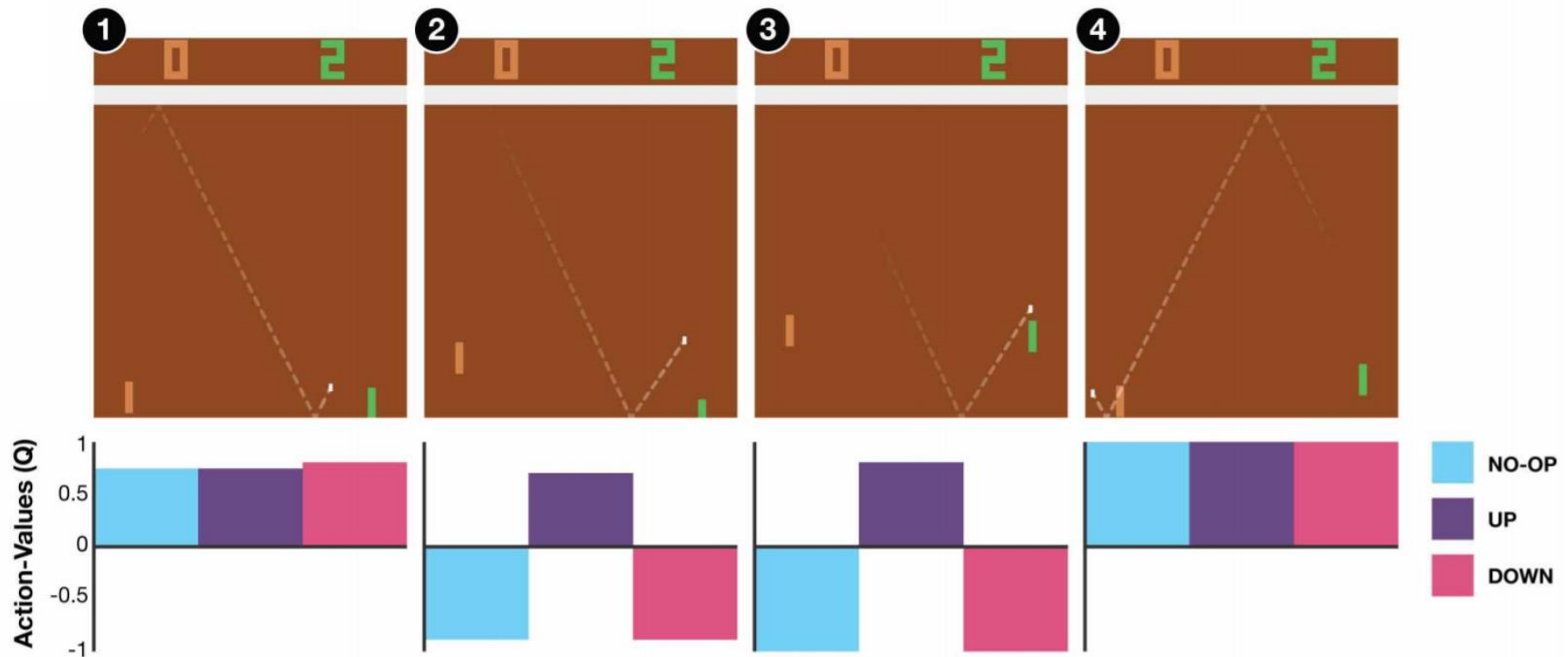


# Another Critic

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



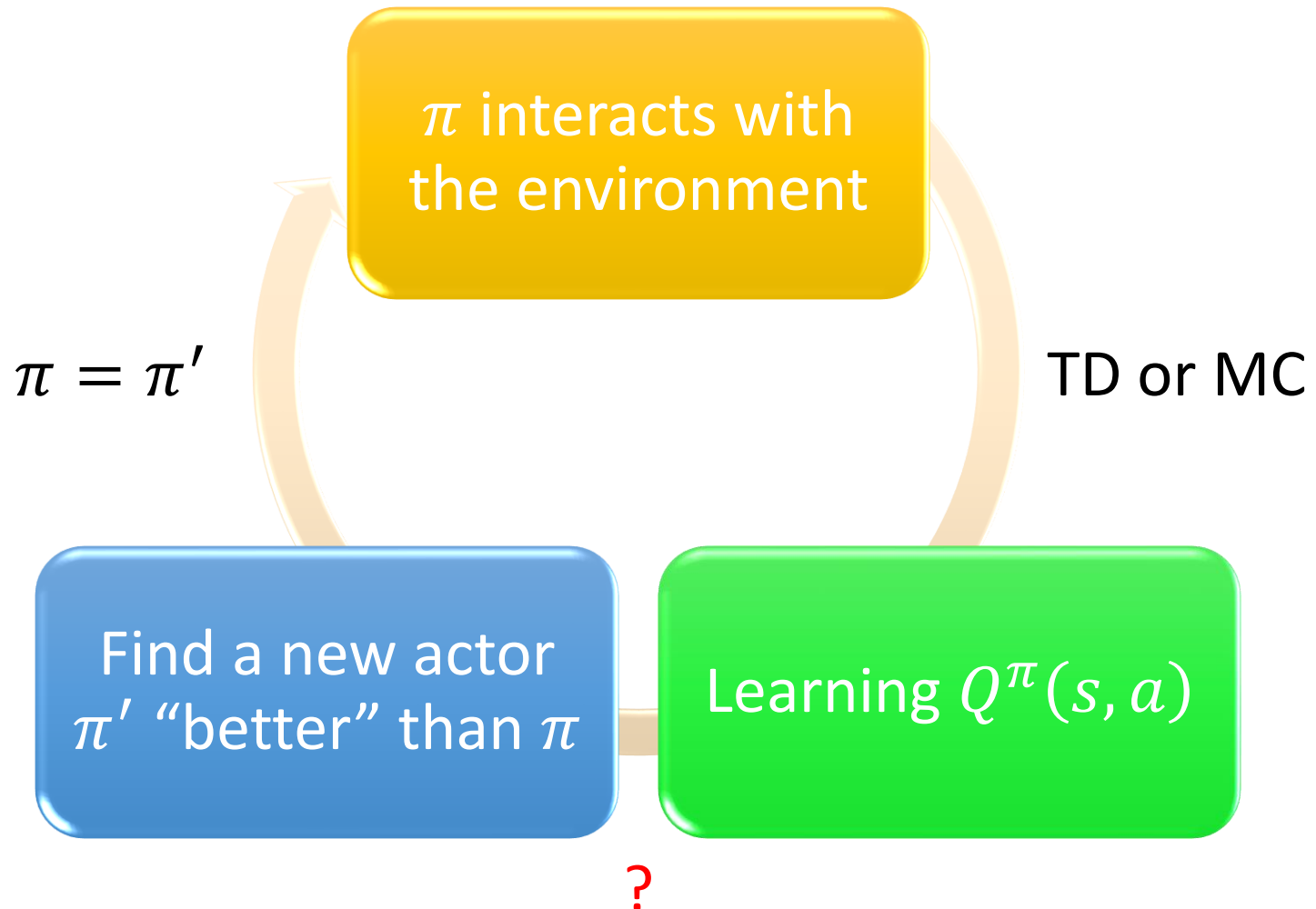
# State-action value function



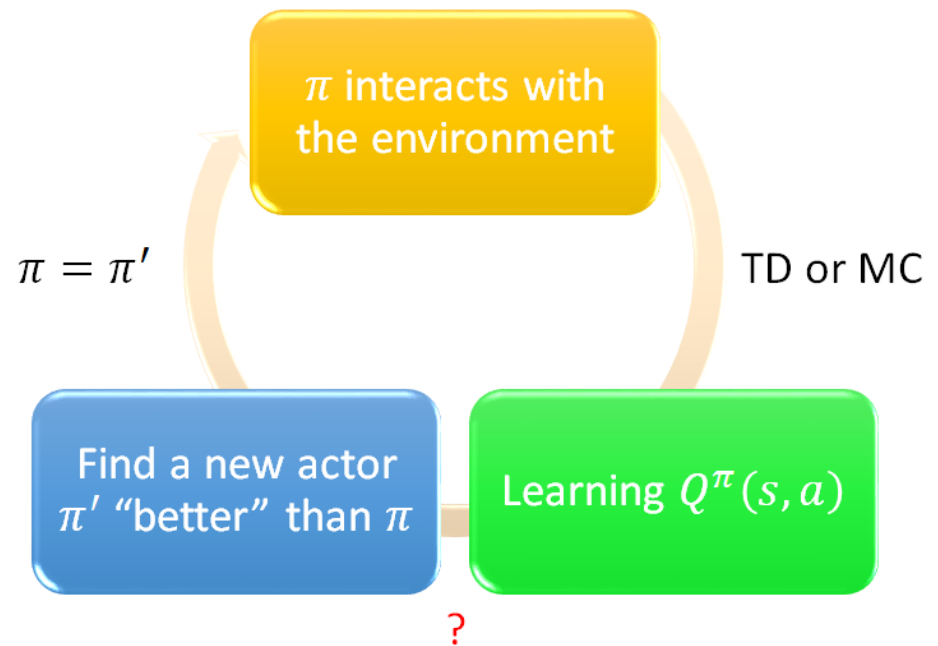
<https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf>

# Another Way to use Critic: Q-Learning

可以用Q function找出一個比較好的actor



# Q-Learning



- Given  $Q^\pi(s, a)$ , find a new actor  $\pi'$  “better” than  $\pi$ 
  - “Better”:  $V^{\pi'}(s) \geq V^\pi(s)$ , for all state  $s$  比較accumulated reward

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

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

## Q-Learning

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

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

$$V^\pi(s) = Q^\pi(s, \pi(s))$$

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

$$V^\pi(s) \leq Q^\pi(s, \pi'(s))$$

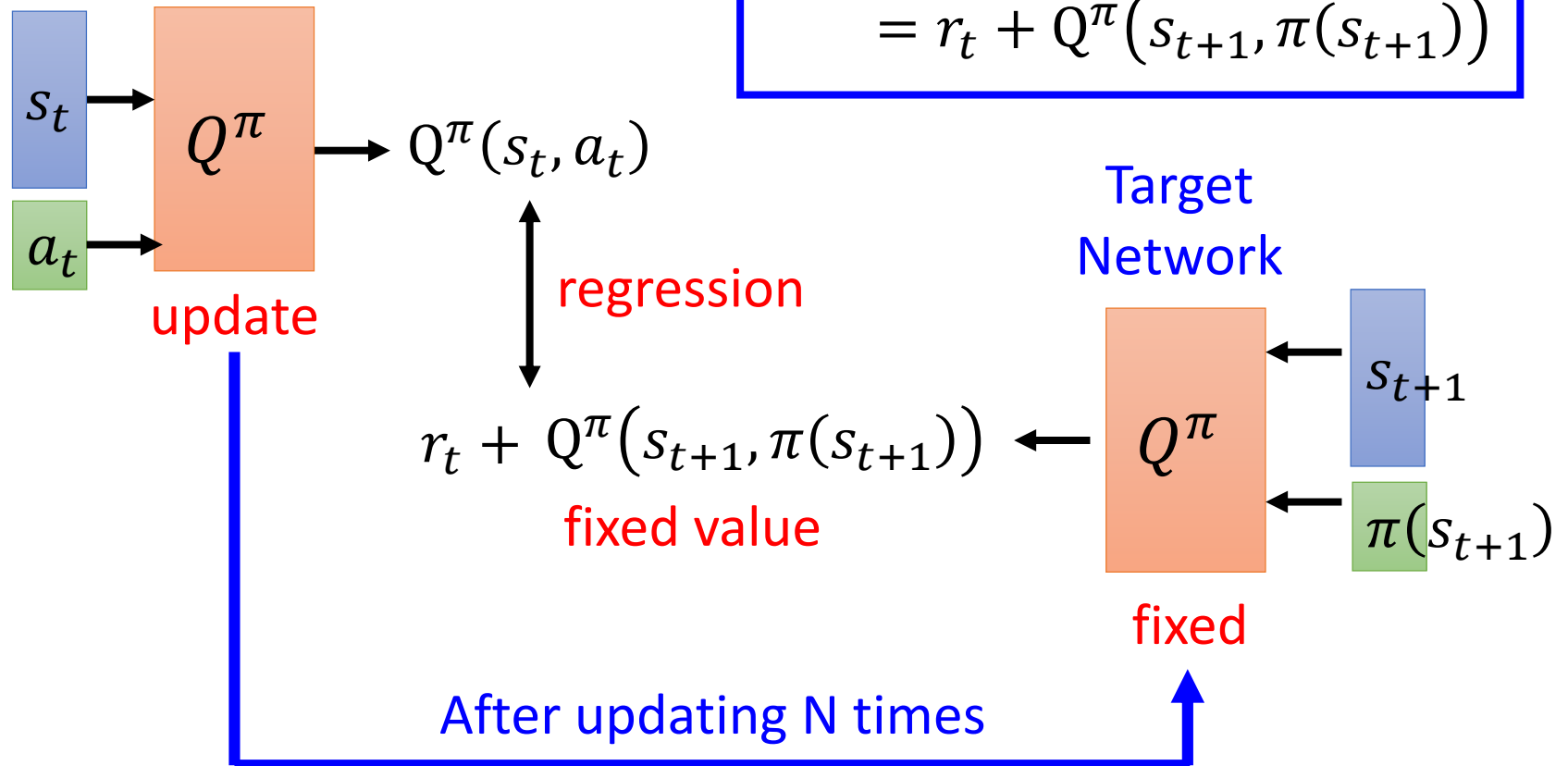
$$= E[r_{t+1} + V^\pi(s_{t+1}) | s_t = s, a_t = \pi'(s_t)]$$

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

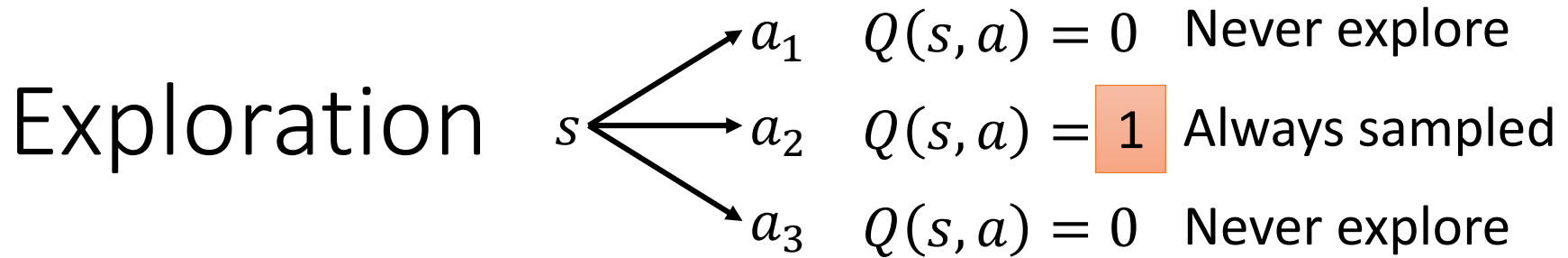
$$= E[r_{t+1} + r_{t+2} + V^\pi(s_{t+2}) | \dots]$$

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

# Target Network



$$\begin{aligned} & \cdots s_t, a_t, r_t, s_{t+1} \cdots \\ & Q^\pi(s_t, a_t) \\ & = r_t + Q^\pi(s_{t+1}, \pi(s_{t+1})) \end{aligned}$$



- The policy is based on Q-function

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

This is not a good way for data collection.

**Epsilon Greedy**  $\varepsilon$  would decay during learning

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

**Boltzmann Exploration**

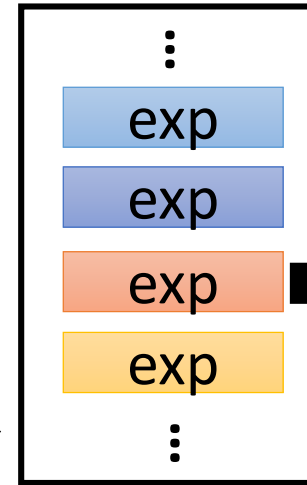
$$P(a|s) = \frac{\exp(Q(s, a))}{\sum_a \exp(Q(s, a))}$$

# Replay Buffer

Put the experience into buffer.

$\pi$  interacts with  
the environment

Buffer



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

The experience in the  
buffer comes from  
different policies.

Drop the old experience  
if the buffer is full.

$\pi = \pi'$

Find a new actor  
 $\pi'$  "better" than  $\pi$

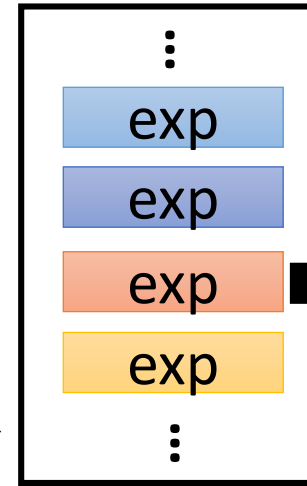
Learning  $Q^\pi(s, a)$



# Replay Buffer

Put the experience into buffer.

Buffer



$\pi$  interacts with  
the environment

$\pi = \pi'$

Find a new actor  
 $\pi'$  "better" than  $\pi$

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

In each iteration:

1. Sample a batch
2. Update Q-function

Off-policy

# Typical Q-Learning Algorithm

- Initialize Q-function  $Q$ , target Q-function  $\hat{Q} = Q$
- In each episode
  - For each time step  $t$ 
    - Given state  $s_t$ , take action  $a_t$  based on  $Q$  (epsilon greedy)
    - Obtain reward  $r_t$ , and reach new state  $s_{t+1}$
    - Store  $(s_t, a_t, r_t, s_{t+1})$  into buffer
    - Sample  $(s_i, a_i, r_i, s_{i+1})$  from buffer (usually a batch)
    - Target  $y = r_i + \max_a \hat{Q}(s_{i+1}, a)$
    - Update the parameters of  $Q$  to make  $Q(s_i, a_i)$  close to  $y$  (regression)
    - Every  $C$  steps reset  $\hat{Q} = Q$

# Outline

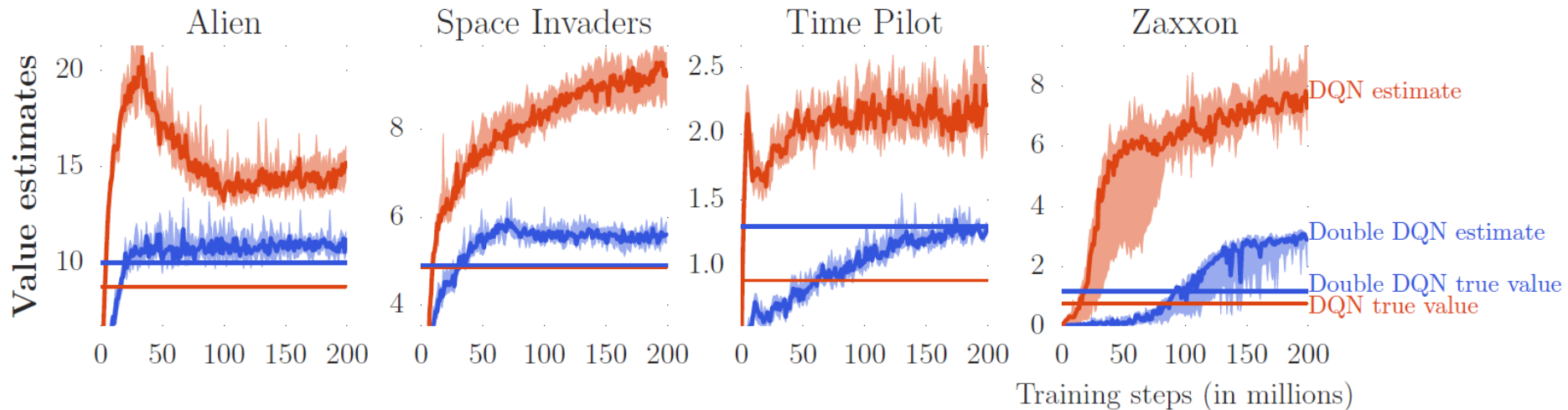
Introduction of Q-Learning

Tips of Q-Learning

Q-Learning for Continuous Actions

# Double DQN

- Q value is usually over-estimated

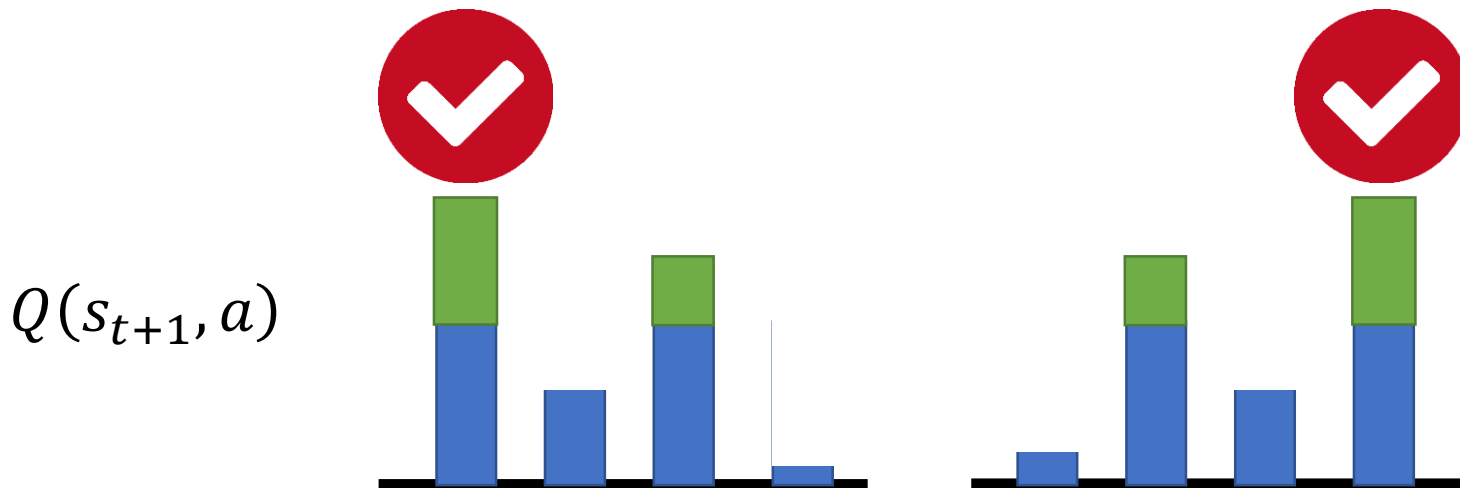


# Double DQN

- Q value is usually over estimate

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

Tend to select the action that is over-estimated



# Double DQN

- Q value is usually over estimate

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

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

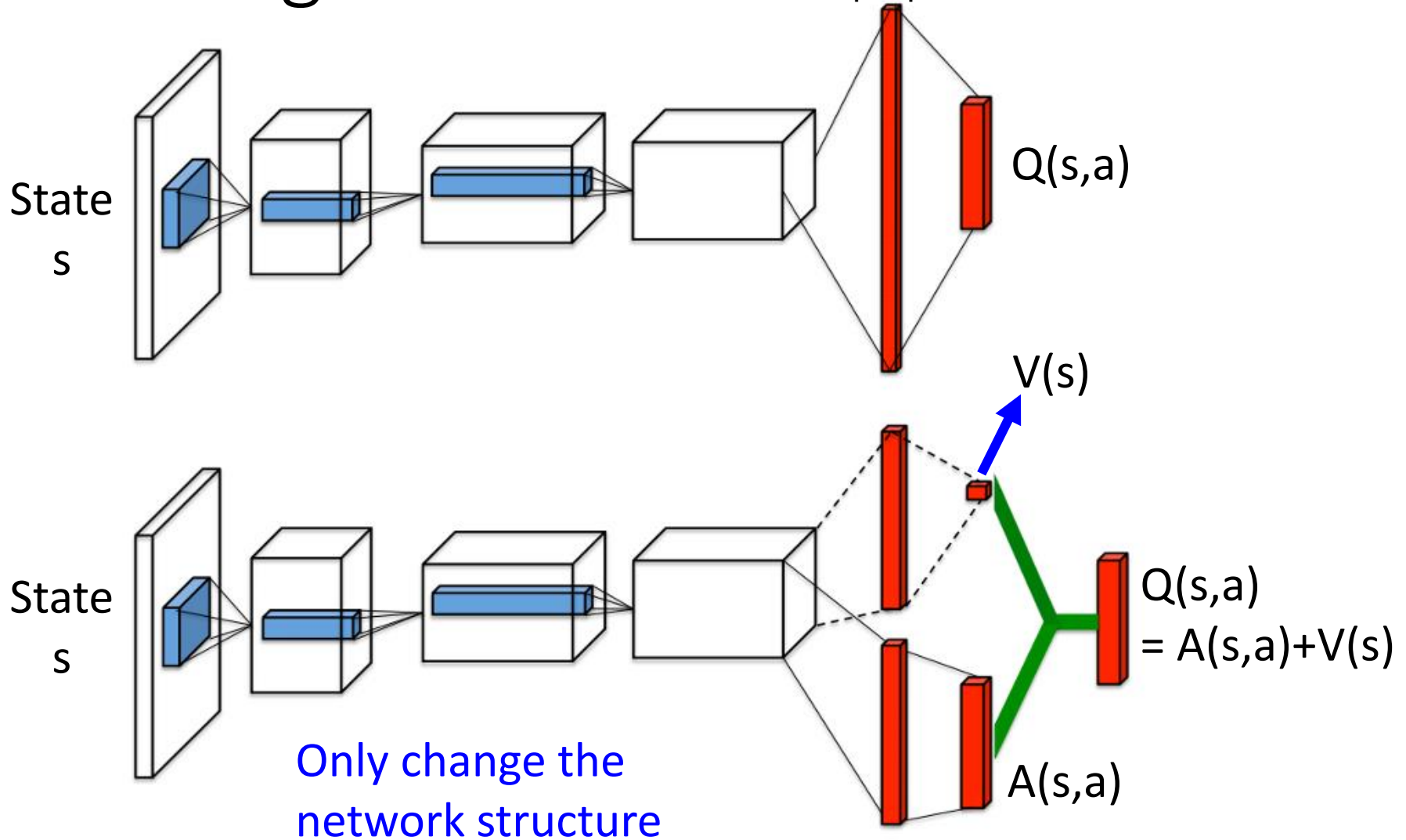
$$Q(s_t, a_t) \longleftrightarrow r_t + Q' \left( s_{t+1}, \arg \max_a Q(s_{t+1}, a) \right)$$

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.

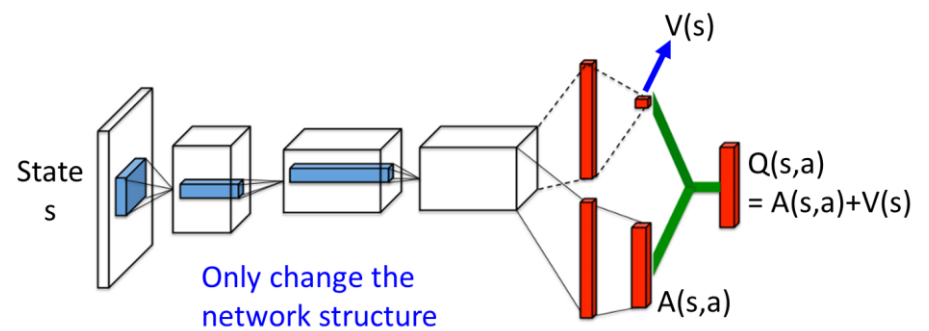
Hado V. Hasselt, "Double Q-learning", NIPS 2010

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

# Dueling DQN



# Dueling DQN



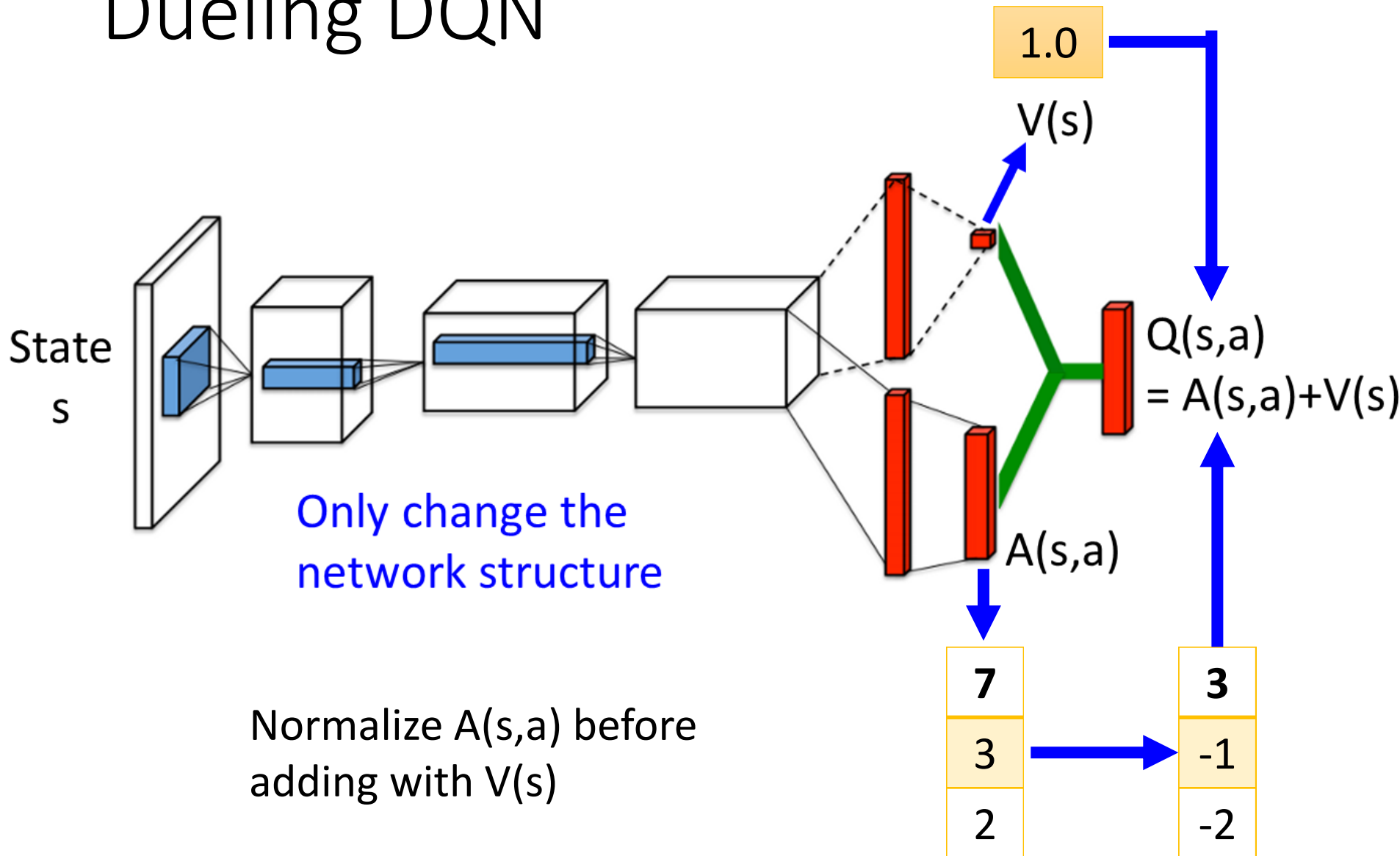
		state			
Q(s,a)	action	3	<del>3</del> 4	3	1
		1	<del>-1</del> 0	6	1
		2	<del>-2</del> -1	3	1

V(s)	Average of column	2	<del>0</del> 1	4	1
		+			

A(s,a)	sum of column = 0	1	3	-1	0
		-1	-1	2	0
		0	-2	-1	0



# Dueling DQN



# Dueling DQN - Visualization



(from the link of the original paper)

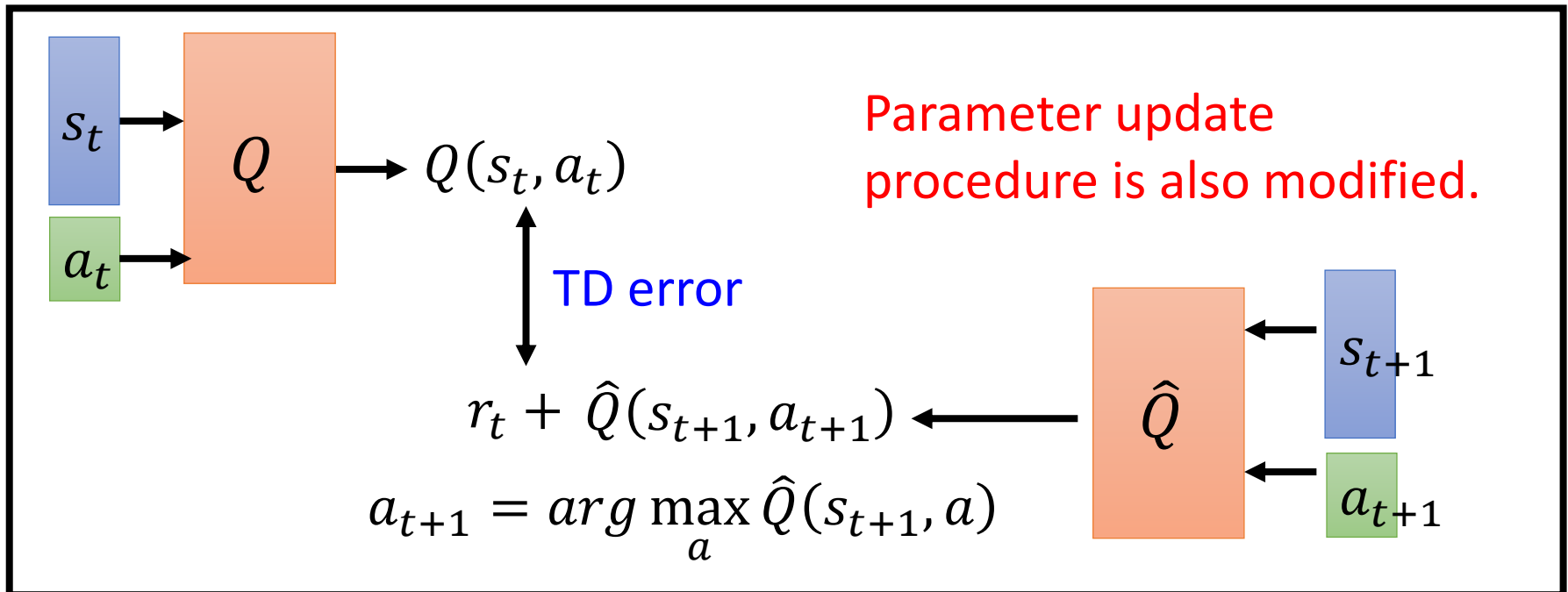
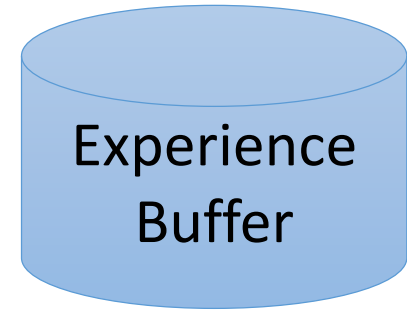
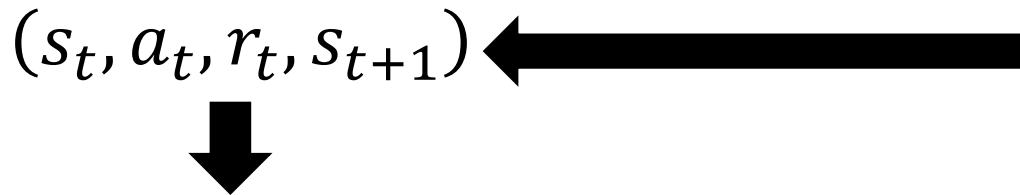
# Dueling DQN - Visualization



(from the link of the original paper)

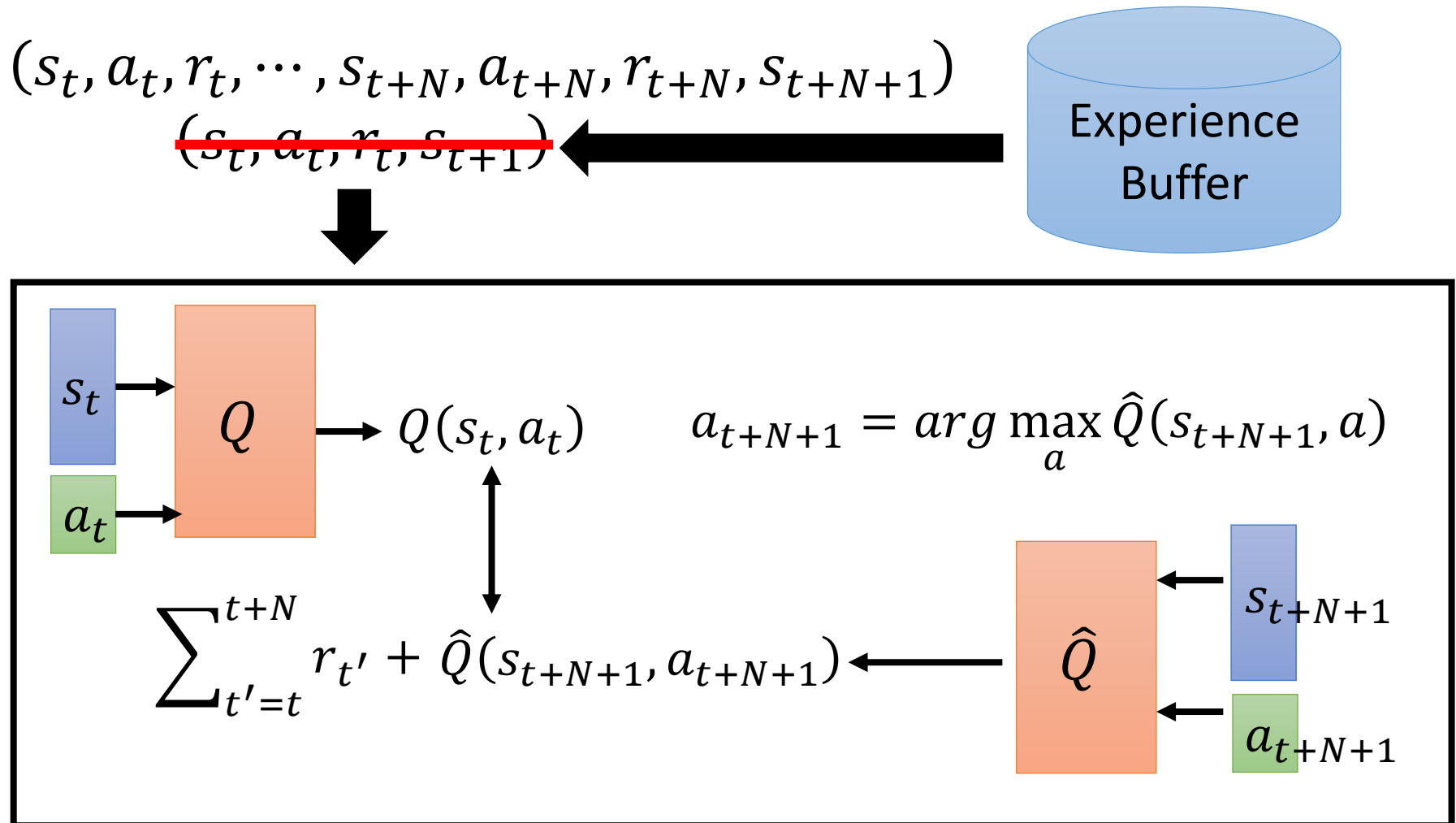
# Prioritized Reply

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



# Multi-step

Balance between MC and TD



# Noisy Net

<https://arxiv.org/abs/1706.01905>

<https://arxiv.org/abs/1706.10295>

- Noise on Action (Epsilon Greedy)

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

- Noise on Parameters

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

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

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

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

# Noisy Net

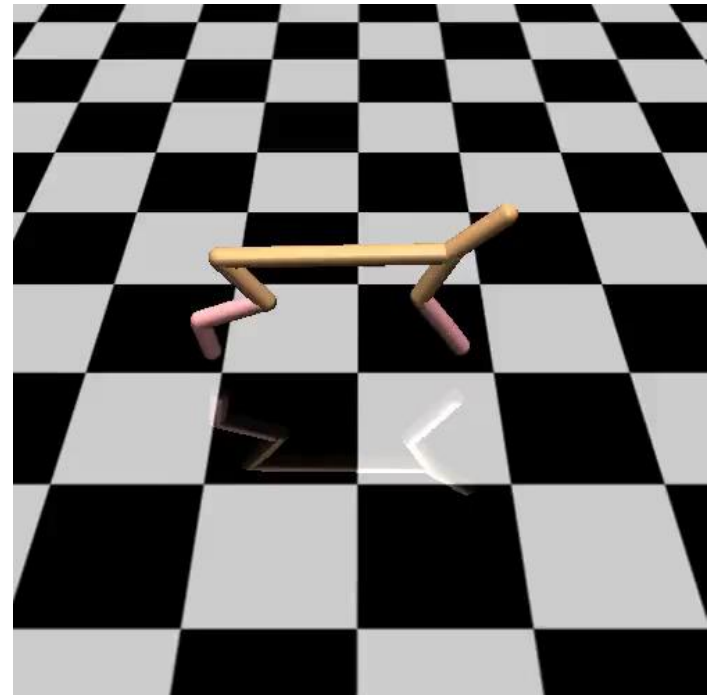
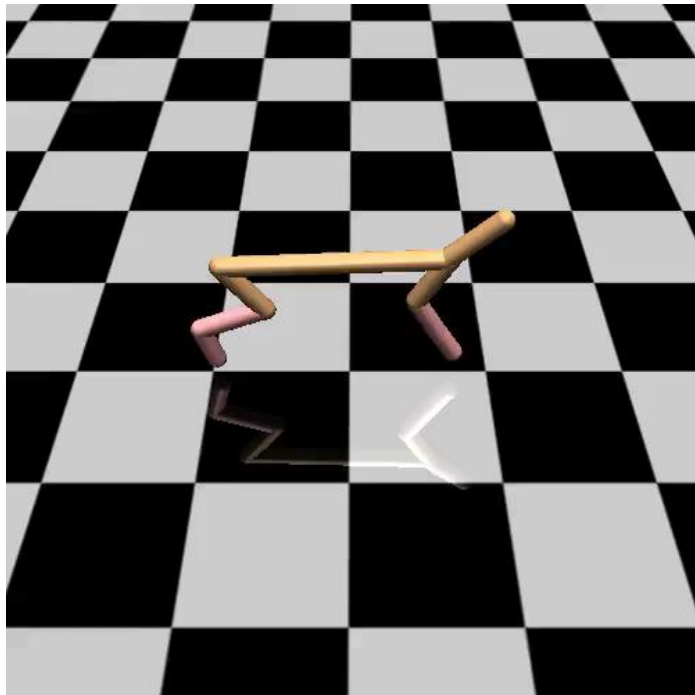
- Noise on Action
  - Given the same state, the agent may takes different actions.
  - No real policy works in this way
- Noise on Parameters
  - Given the same (similar) state, the agent takes the same action.
    - → State-dependent Exploration
  - Explore in a *consistent* way

隨機亂試

有系統地試

# Demo

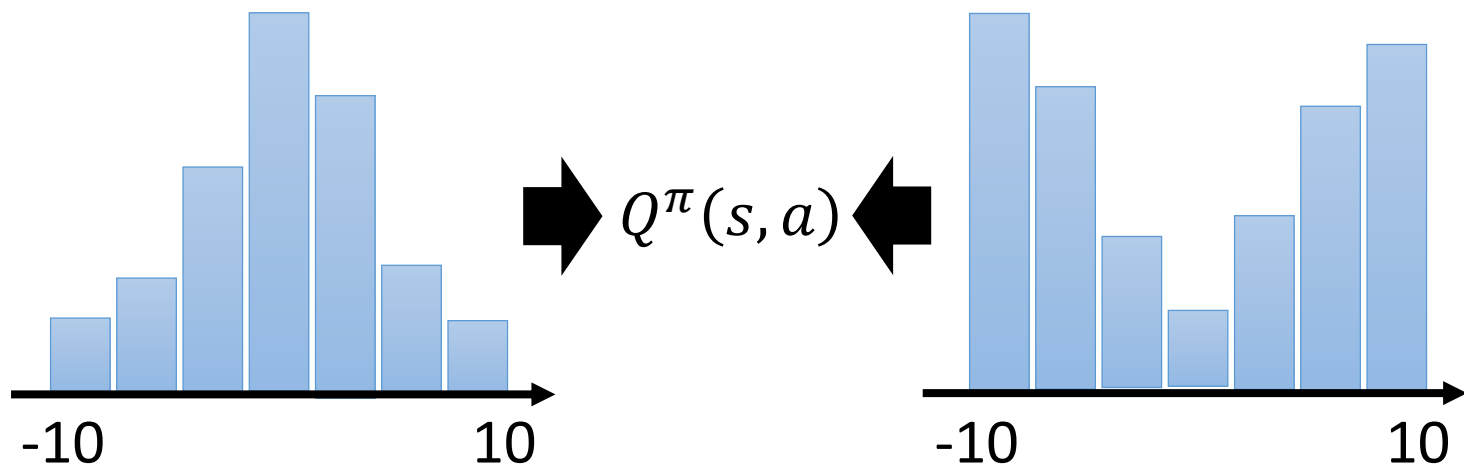
<https://blog.openai.com/better-exploration-with-parameter-noise/>





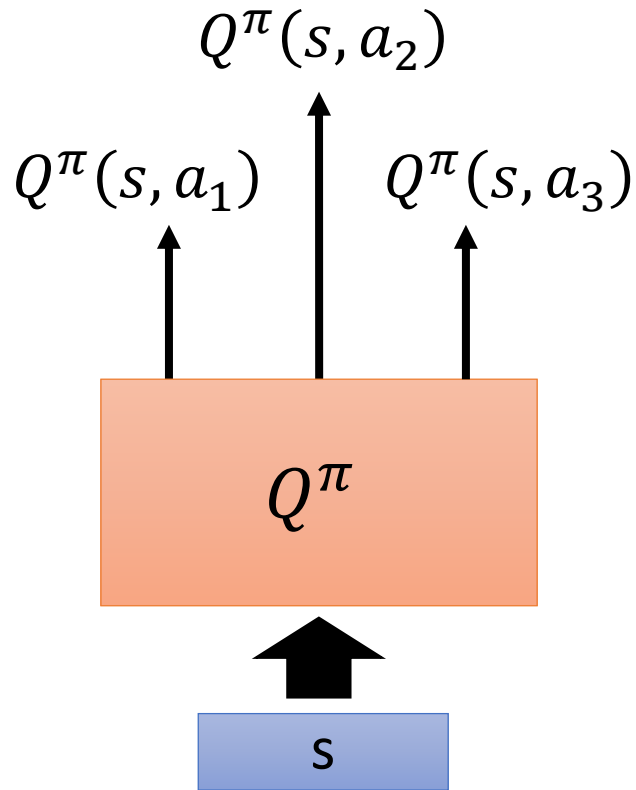
# Distributional Q-function

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

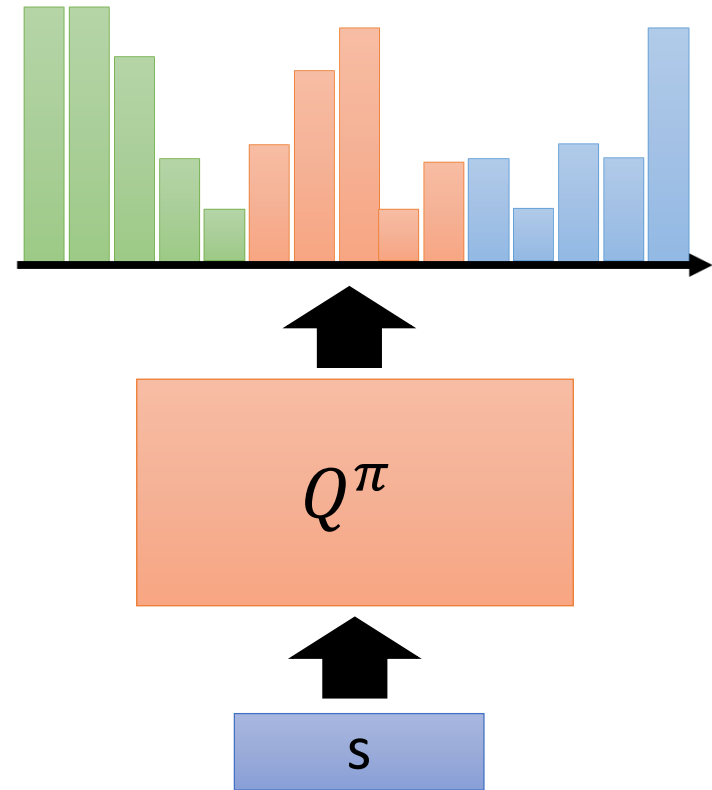


Different distributions can have the same values.

# Distributional Q-function

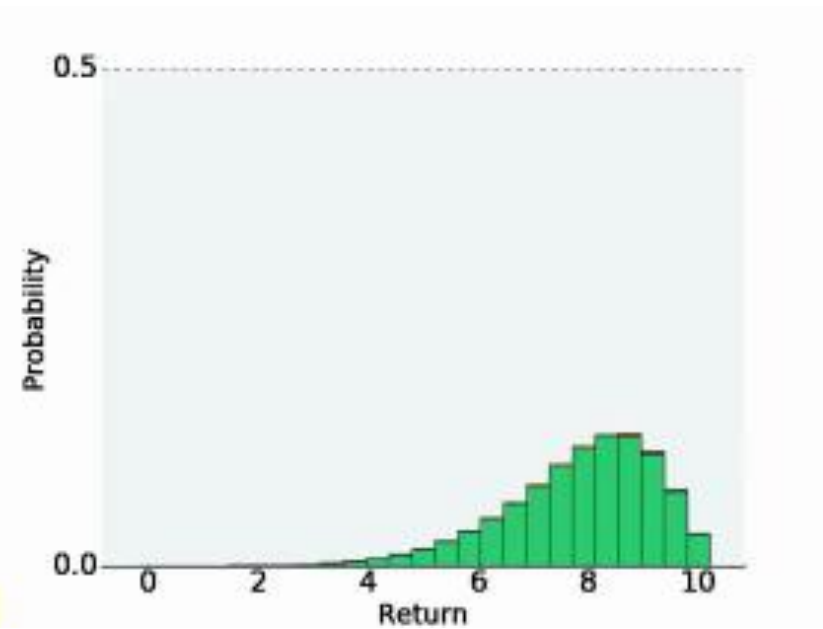
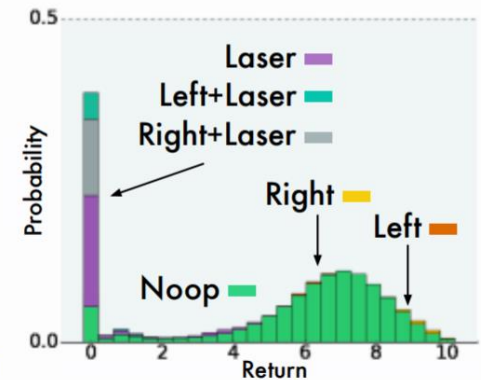


A network with 3 outputs



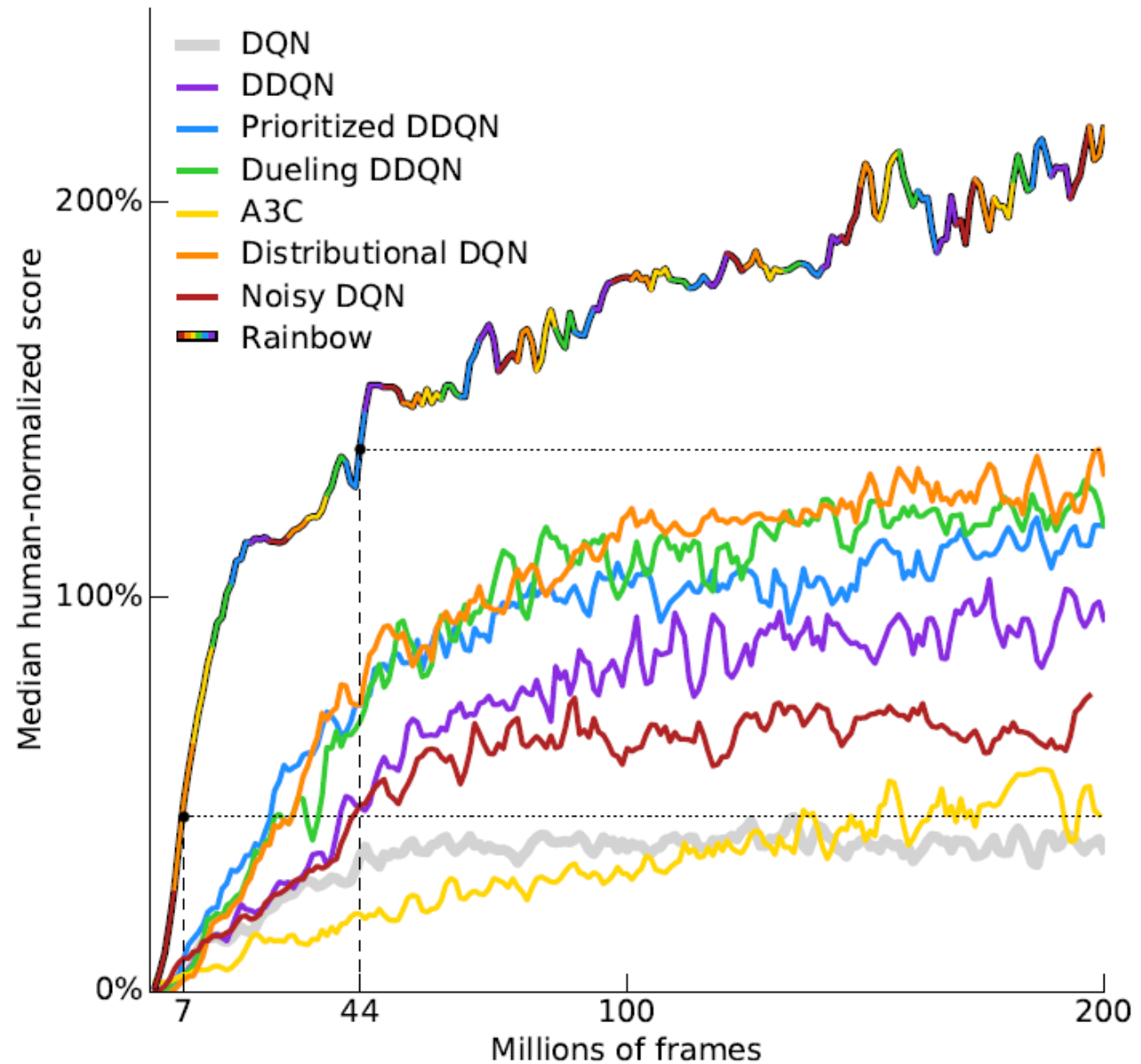
A network with 15 outputs  
(each action has 5 bins)

# Demo

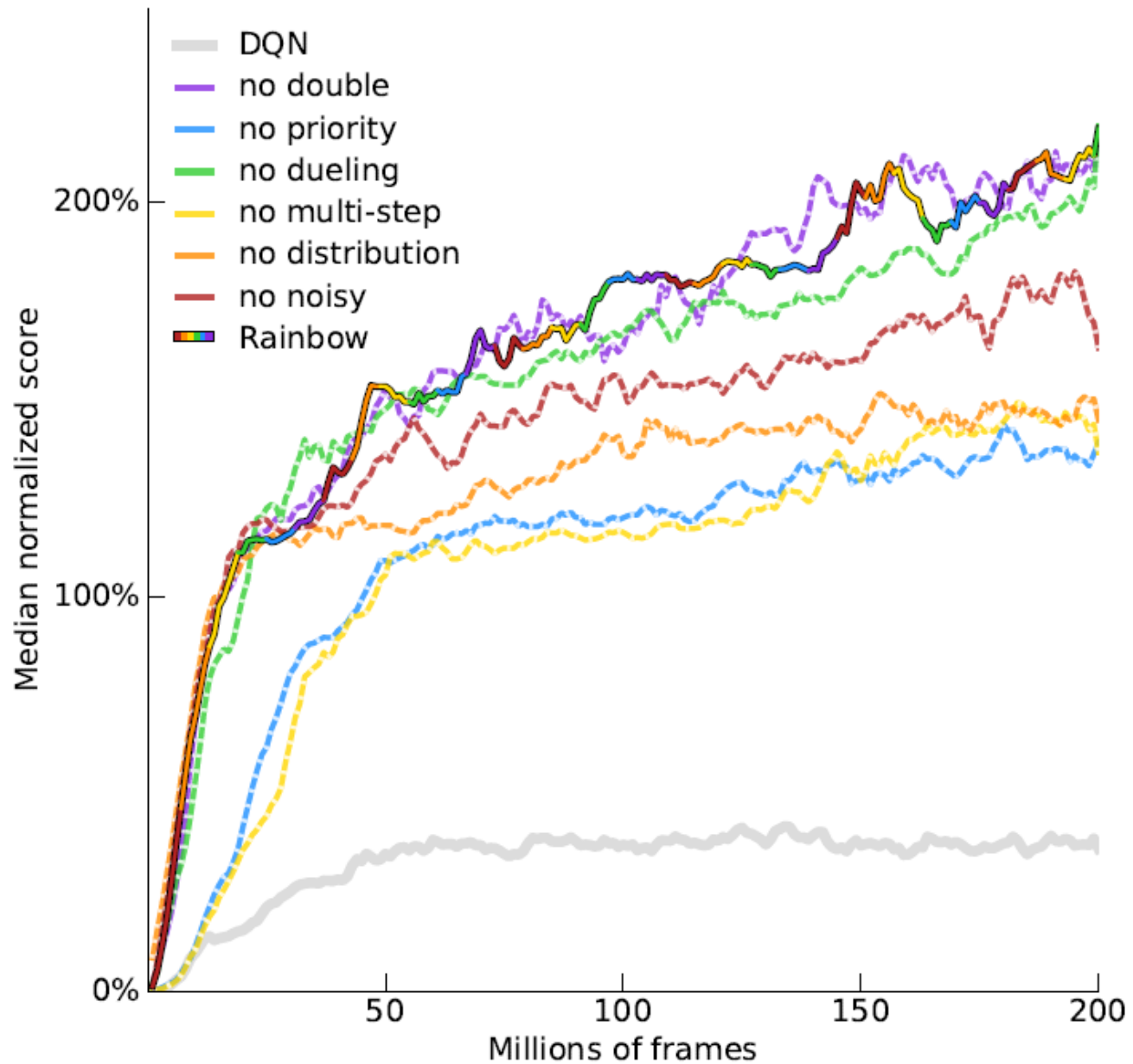


<https://youtu.be/yFBwyPuO2Vg>

# Rainbow



# Rainbow



# Outline

Introduction of Q-Learning

Tips of Q-Learning

Q-Learning for Continuous Actions

# Continuous Actions

- Action  $a$  is a *continuous vector*

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

## **Solution 1**

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

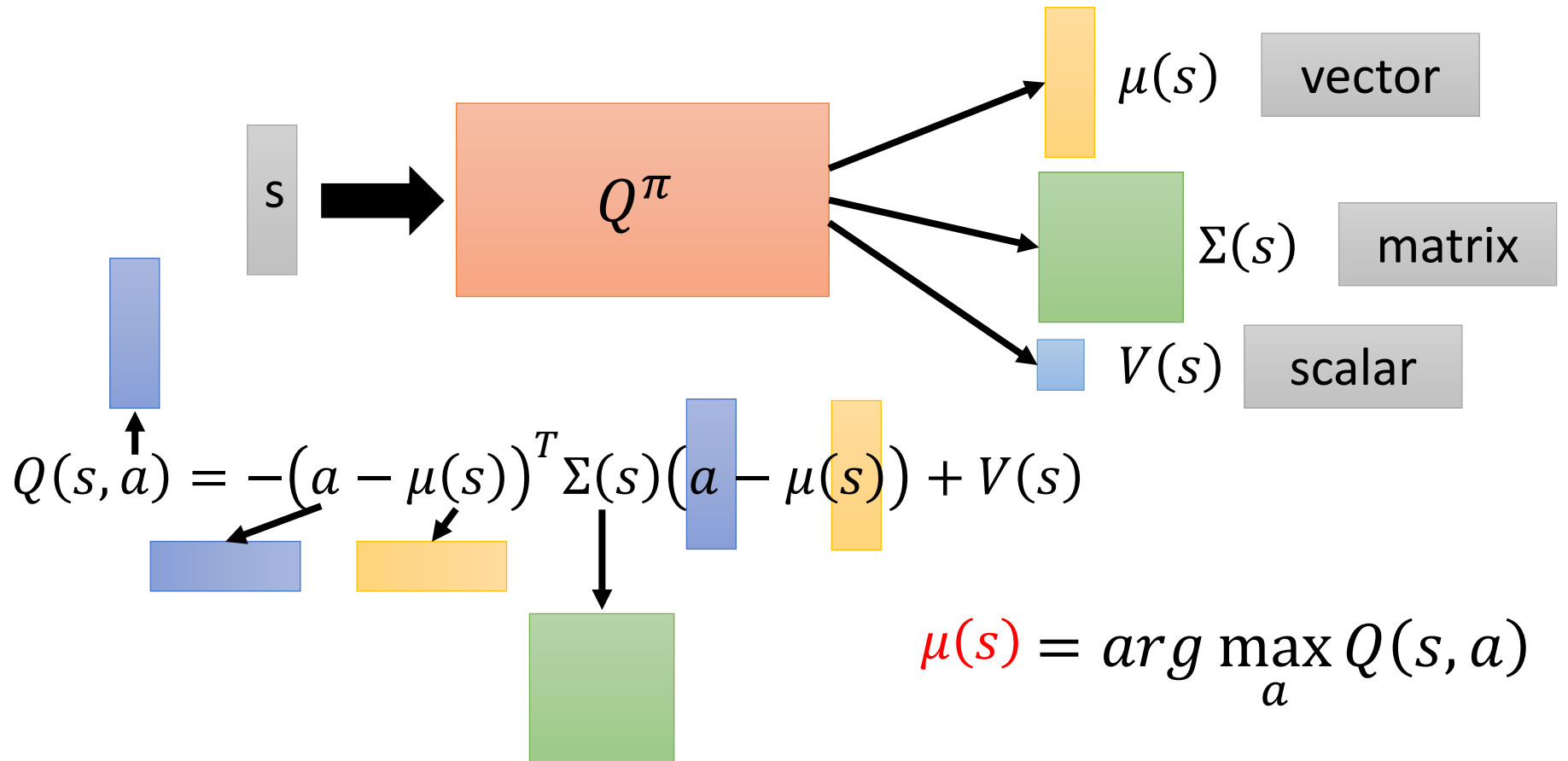
See which action can obtain the largest Q value

## **Solution 2**

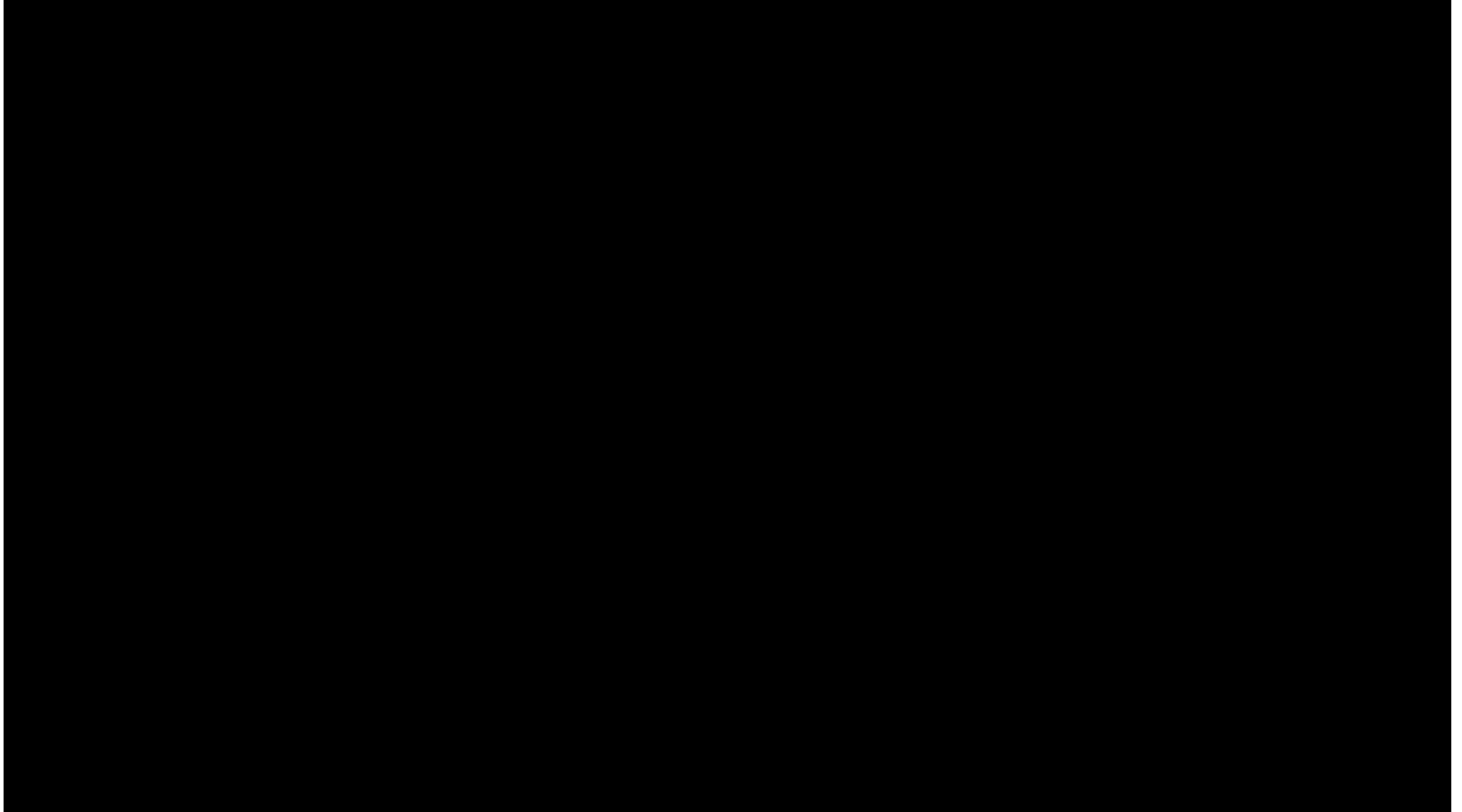
Using gradient ascent to solve the optimization problem.

# Continuous Actions

**Solution 3** Design a network to make the optimization easy.



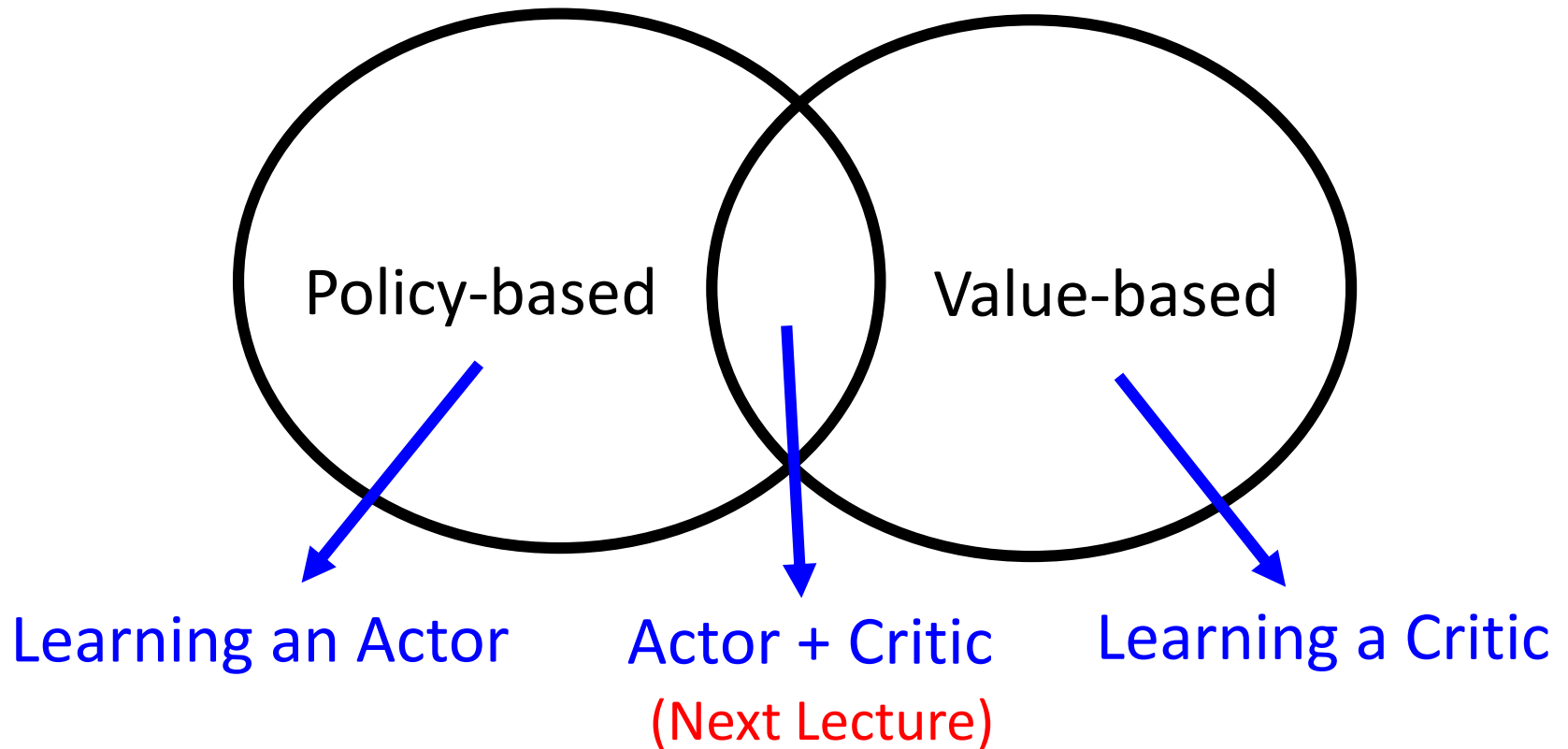




<https://www.youtube.com/watch?v=ZhsEKTo7V04>

# Continuous Actions

**Solution 4** Don't use Q-learning



# Acknowledgement

- 感謝林雨新同學發現投影片上的錯字