



RLChina 2020

# Introduction to Reinforcement Learning and Value-based Methods

Zongqing Lu  
Peking University

2020/7/27

# Content



RLChina 2020

## ❖ Introduction to Reinforcement Learning

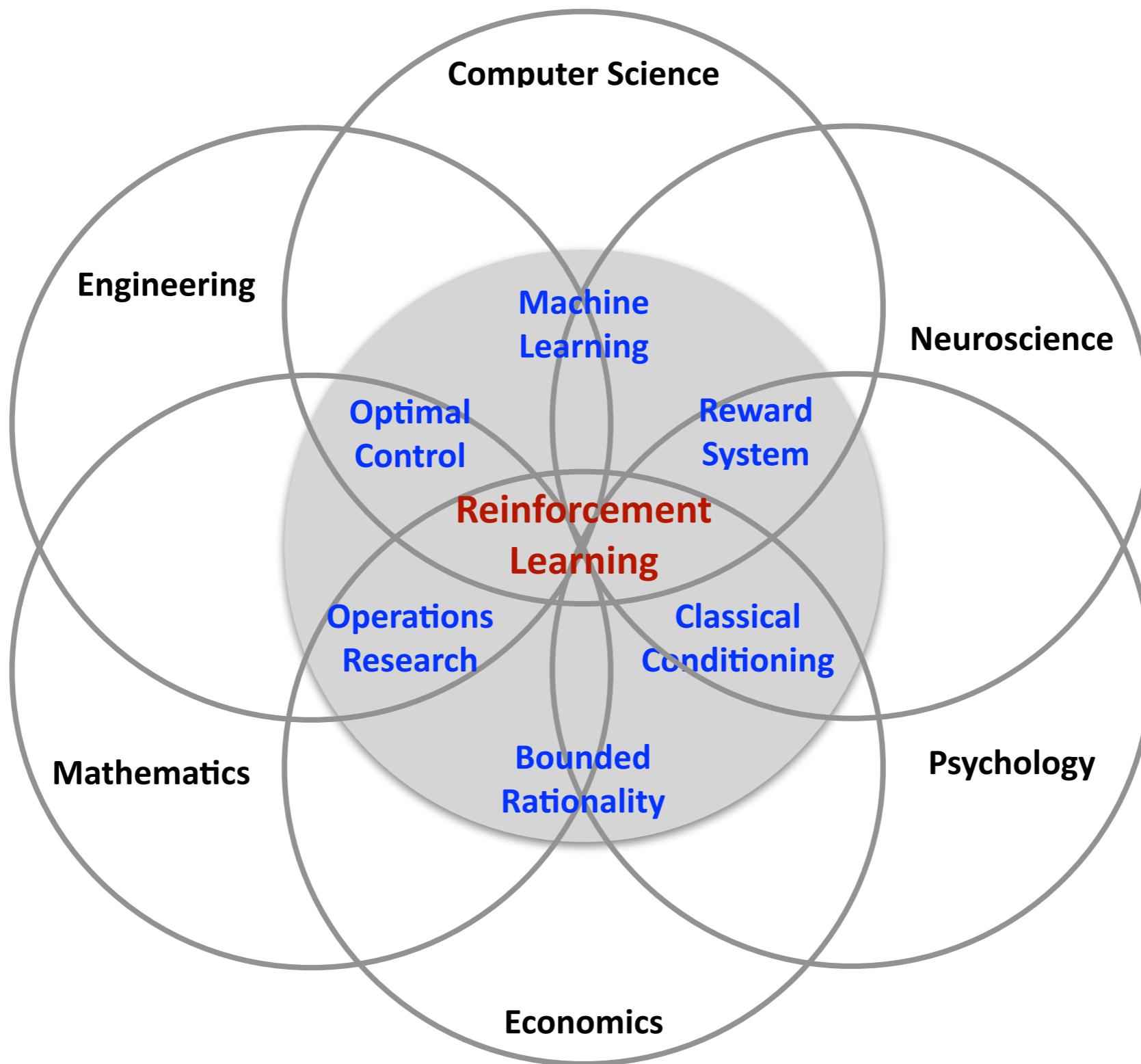
- About RL
- RL problem
- Markov Decision Processes

## ❖ Value-based Methods

- Dynamic Programming
- Monte Carlo
- TD Learning
- Off-policy Learning
- DQN and its variants

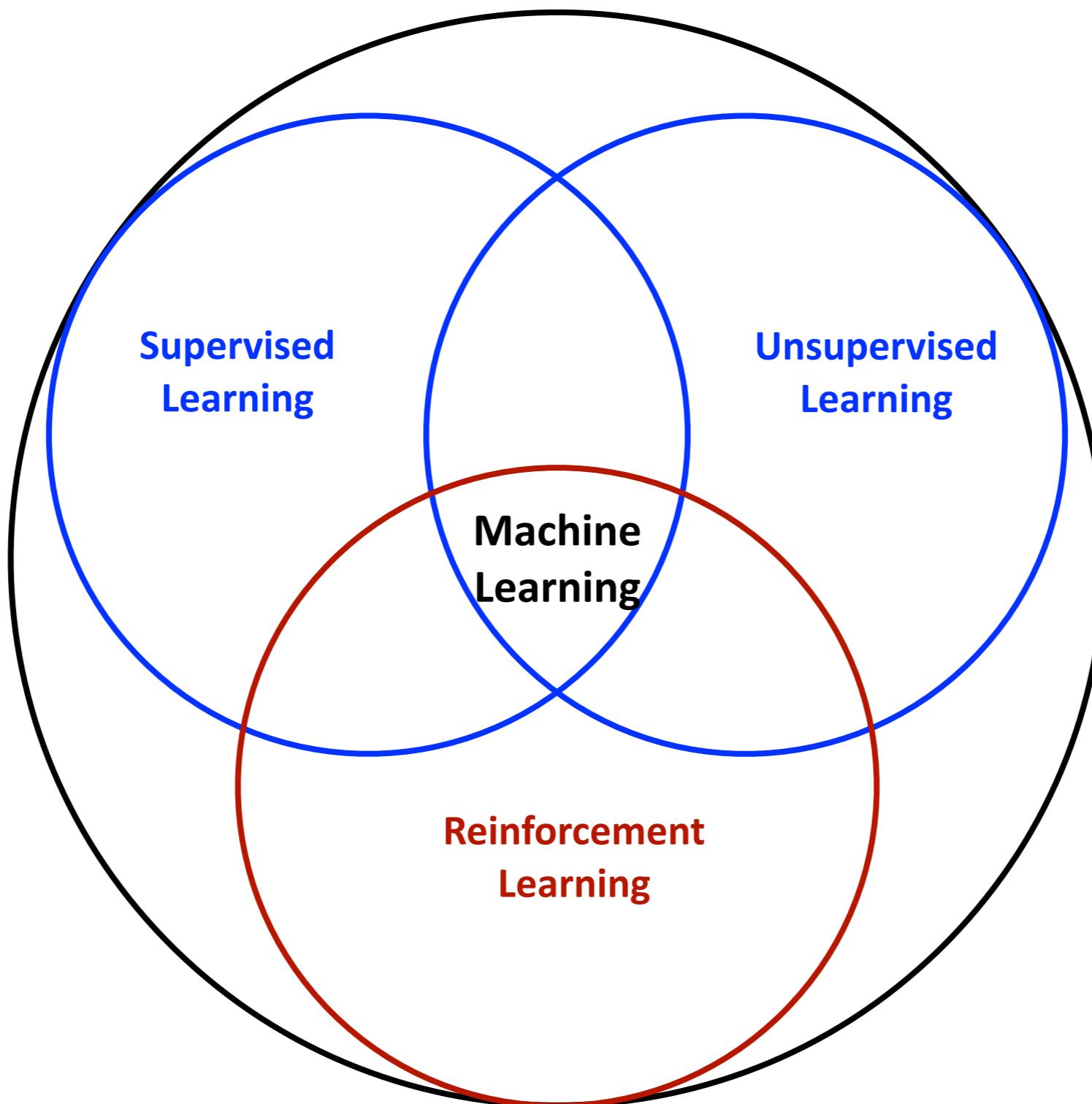


# Many Faces of Reinforcement Learning





# Branches of Machine Learning



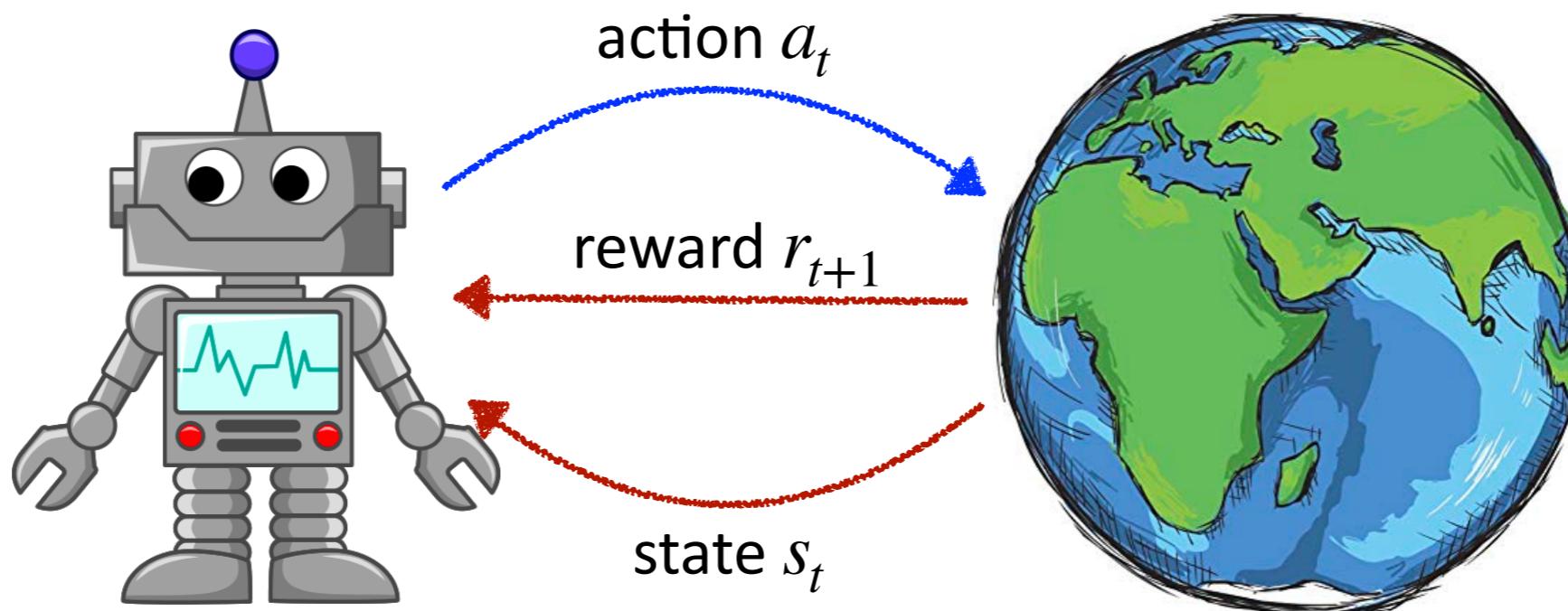


# Characteristics of Reinforcement Learning

- ❖ What makes reinforcement learning different from other machine learning paradigms?
  - There is no supervisor, only a reward signal, trial-and-error
  - Feedback is delay, not instantaneous
  - Time really matters (sequential, non-iid data)
  - Agent's action affects the subsequent data it receives



# The RL Problem



**The agent is to maximize the cumulative reward**

All goals and purposes can be described by the maximization of the expected cumulative reward

# Inside An RL Agent



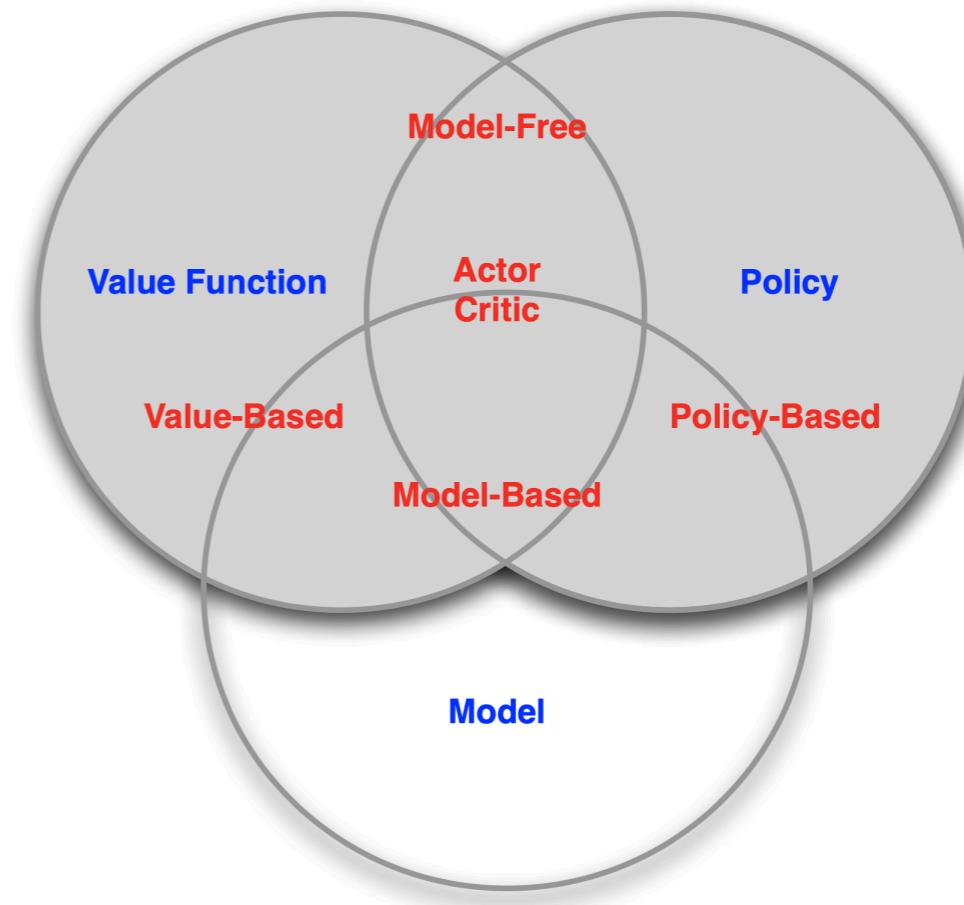
RLChina 2020

- ❖ An RL agent may include one or more of these components
  - **Policy**: agent's behavior function
  - **Value function**: how good is a state and/or action
  - **Model**: agent's representation of the environment



# Categorizing RL Agents

- ❖ Value based
  - No policy
  - Value function
- ❖ Policy based
  - Policy
  - No value function
- ❖ Actor Critic
  - Policy
  - Value function
- ❖ Model free
  - Policy and/or value function
  - No model
- ❖ Model based
  - Policy and/or value function
  - Model





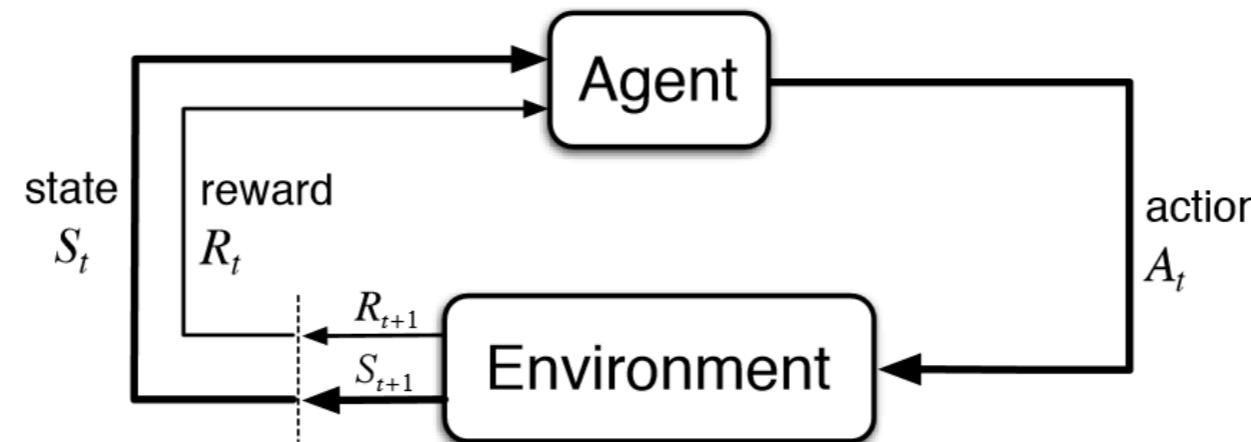
# Markov Decision Processes

- ❖ Markov decision processes formally describe an environment for reinforcement learning
  - The environment is fully observable
- ❖ Almost all RL problems can be formalized as MDPs
  - Optimal control primarily deals with continuous MDPs
  - Partially observable problems can be converted into MDPs
  - Bandits are MDPs with one state
- ❖ All states in MDP has “**Markov**” property
  - $\mathbb{P}[S_{t+1} | S_t] = \mathbb{P}[S_{t+1} | S_1, \dots, S_t]$
  - Current state encapsulates all the statistics we need to decide the future

# MDP



RLChina 2020



$$\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$$

- ❖  $\mathcal{S}$  - a set of states
- ❖  $\mathcal{A}$  - a set of actions
- ❖  $\mathcal{P}$  - transition probability function,
  - $\mathcal{P}_{ss'}^a = \mathbb{P} [S_{t+1} = s' | S_t = s, A_t = a]$
- ❖  $\mathcal{R}$  - reward function,
  - $\mathcal{R}_s^a = \mathbb{E} [R_{t+1} | S_t = s, A_t = a]$
- ❖  $\gamma$  - discounting factor for future reward,  $\gamma \in [0,1]$

# Policy



RLChina 2020

- ❖ A policy  $\pi$  is a distribution over actions given states
  - $\pi(a | s) = \mathbb{P}[A_t = a | S_t = s]$
- ❖ Given a MDP  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$  and a policy  $\pi$ 
  - The state sequence  $S_1, S_2, \dots$  is a Markov process  $\langle \mathcal{S}, \mathcal{P}^\pi \rangle$
  - The state and reward sequence  $S_1, R_2, S_2, \dots$  is a Markov reward process  $\langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma \rangle$ , where

$$\mathcal{P}_{s,s'}^\pi = \sum_{a \in \mathcal{A}} \pi(a | s) \mathcal{P}_{ss'}^a$$

$$\mathcal{R}_s^\pi = \sum_{a \in \mathcal{A}} \pi(a | s) \mathcal{R}_s^a$$



# Value Function

- ❖ The state-value function  $v_\pi(s)$  of an MDP is the expected return starting from state  $s$ , and then following policy  $\pi$

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi [G_t | S_t = s] \\ &= \mathbb{E}_\pi [R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s] \\ &= \mathbb{E}_\pi [R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \dots) | S_t = s] \\ &= \mathbb{E}_\pi [R_{t+1} + \gamma G_{t+1} | S_t = s] \\ &= \mathbb{E}_\pi [R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \end{aligned}$$

- ❖ The action-value function  $q_\pi(s, a)$  of an MDP is the expected return starting from state  $s$ , taking action  $a$  and then following policy  $\pi$

$$\begin{aligned} Q_\pi(s, a) &= \mathbb{E}_\pi [R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a] \\ &= \mathbb{E}_\pi [R_{t+1} + \gamma \mathbb{E}_{a' \sim \pi} Q(S_{t+1}, a') | S_t = s, A_t = a] \end{aligned}$$



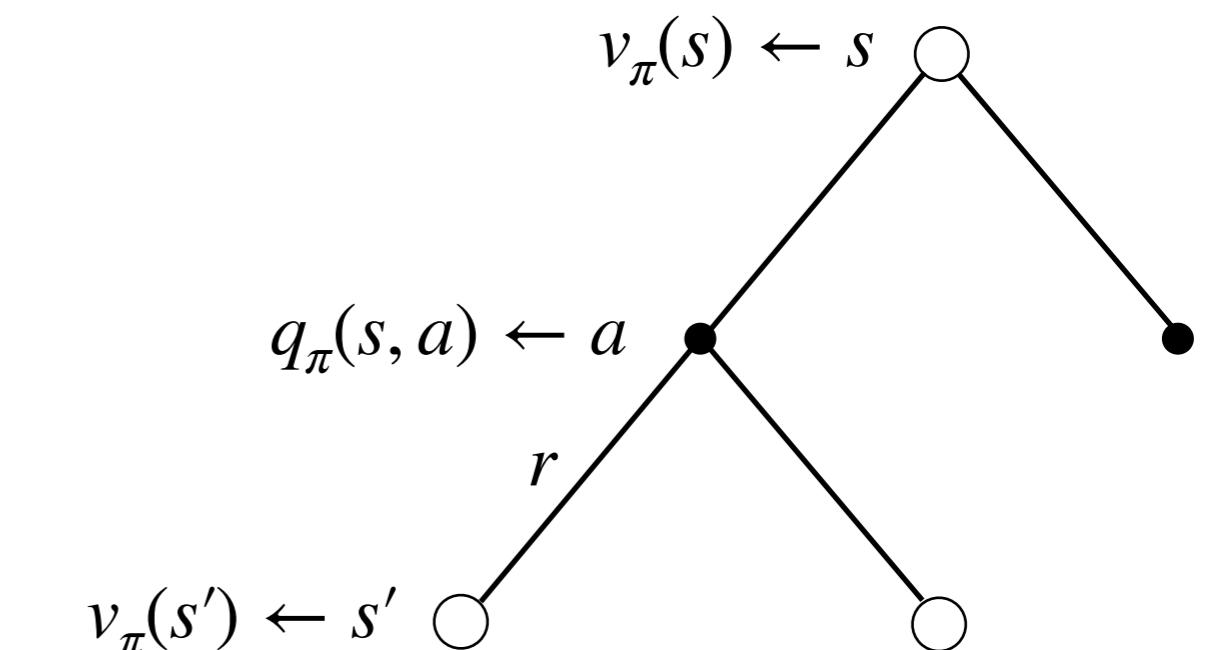
# Bellman Expectation Equations

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a | s) q_\pi(s, a)$$

$$q_\pi(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a | s) \left( \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right)$$

$$q_\pi(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a' | s') q_\pi(s', a')$$



# Optimal Value Function



- ❖ The optimal state-value function  $v_*(s)$  is the maximum value function over all policies

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

- ❖ The optimal action-value function  $q_*(s, a)$  is the maximum value function over all policies

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

# Optimal Policy



RLChina 2020

- ❖ Define  $\pi \geq \pi'$  if  $v_\pi(s) \geq v_{\pi'}(s), \forall s$ 
  - There exists an optimal policy  $\pi_*$  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,  
 $q_{\pi_*}(s, a) = q_*(s, a)$
  - If we know  $q_*(s, a)$ , we immediately have the optimal policy

# Bellman Optimality Equations

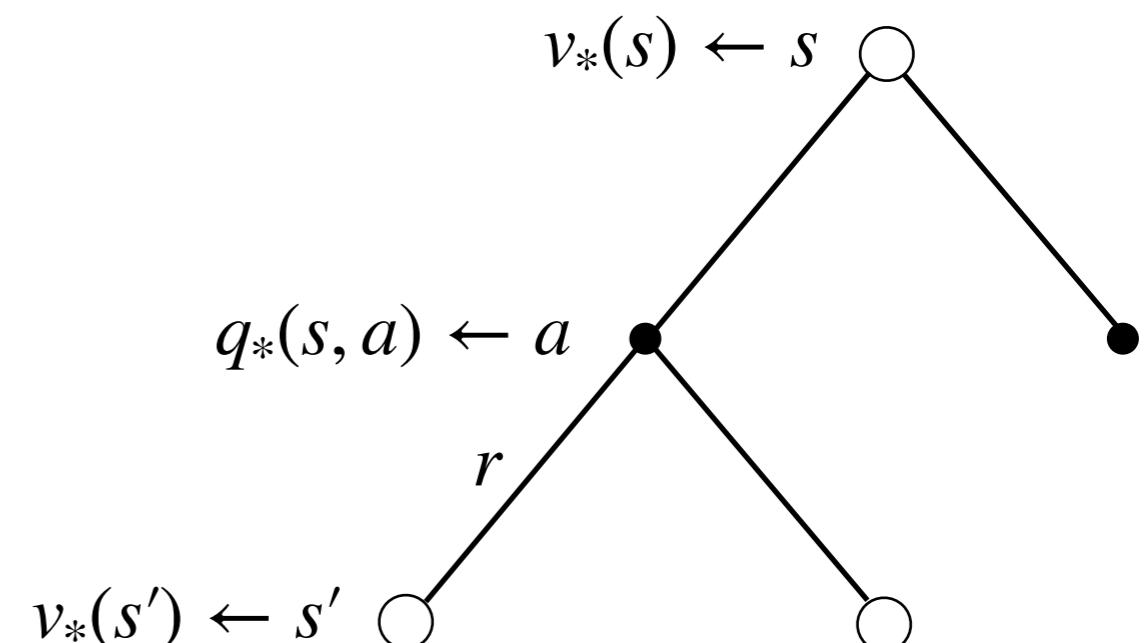


$$v_*(s) = \max_a q_*(s, a)$$

$$q_*(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

$$v_*(s) = \max_a \left( \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s') \right)$$

$$q_*(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q_*(s', a')$$



# Solving the Bellman Optimality Equation



- ❖ If we have complete information of the environment, this turns into a planning problem, solvable by Dynamic Programming
- ❖ Unfortunately, in most scenarios, we do not know  $\mathcal{P}_{ss'}^a$ , or  $\mathcal{R}_s^a$ , we cannot solve MDPs by directly applying Bellman equation

# Outline



RLChina 2020

## ❖ Introduction to Reinforcement Learning

- About RL
- RL problem
- Markov Decision Processes

## ❖ Value-based Methods

- Dynamic Programming
- Monte Carlo
- TD Learning
- Off-policy Learning
- DQN and its variants

# Dynamic Programming



- ❖ DP assumes full knowledge of the MDP
- ❖ Planning in an MDP
- ❖ For prediction:
  - Input: MDP  $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$  and policy  $\pi$
  - Output: value function  $v_\pi$
- ❖ For control:
  - Input: MDP  $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
  - Output: optimal value function  $v_*$  and  $\pi_*$

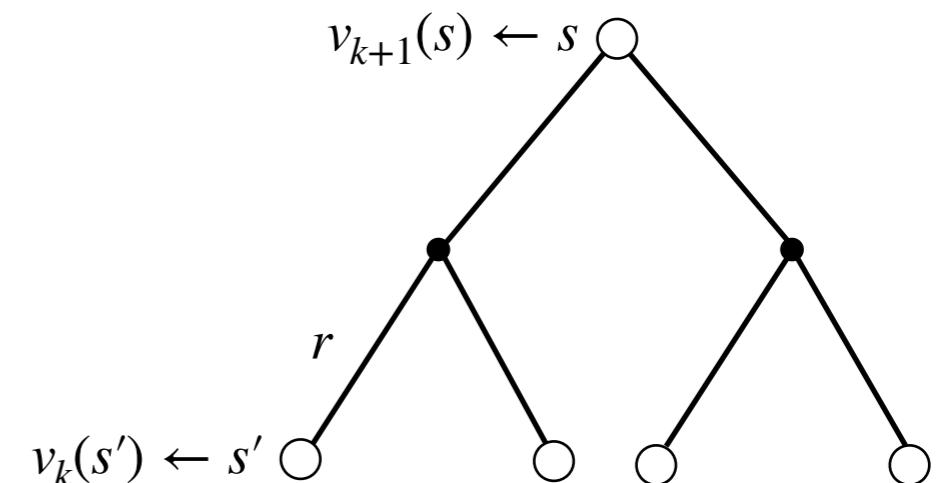


# Policy Evaluation

- ❖ Problem: compute the value function for a given policy  $\pi$
- ❖ Solution: iteration of Bellman Expectation backup
- ❖  $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_\pi$ 
  - At each iteration  $k + 1$
  - Synchronous update  $v_{k+1}(s)$  from  $v_k(s')$ ,  $\forall s \in \mathcal{S}$ , where  $s'$  is a successor state of  $s$

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a | s) \left( \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

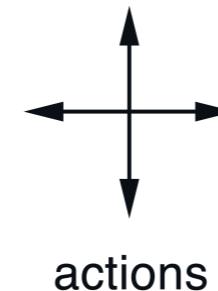
$$v^{k+1} = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v^k$$





# Evaluating a Random Policy

- ❖ Undiscounted episodic MDP
- ❖ Actions leading out of the grid make state unchanged
- ❖ Two terminal states
- ❖ A uniform random policy



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

$R_t = -1$   
on all transitions

$$\pi(\text{up} \mid \cdot) = \pi(\text{down} \mid \cdot) = \pi(\text{right} \mid \cdot) = \pi(\text{left} \mid \cdot) = 0.25$$



# Policy Evaluation in Gridworld

$v_k$  for the random policy

$k = 0$

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

$k = 3$

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

$k = 1$

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

$k = 10$

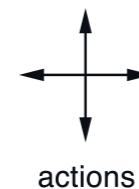
0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = 2$

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

$k = \infty$

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



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

$R_t = -1$   
on all transitions



# Policy Improvement

- ❖ Consider a deterministic policy,  $a = \pi(s)$
- ❖ Improve the policy by acting greedily

$$\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}} q_\pi(s, a)$$

- ❖ This improves the value from any state  $s$  over one step

$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$$

- ❖ It therefore improves the value function,  $v_{\pi'}(s) \geq v_\pi(s)$

$$\begin{aligned} v_\pi(s) &\leq q_\pi(s, \pi'(s)) = \mathbb{E}_{\pi'} [R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \\ &\leq \mathbb{E}_{\pi'} [R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s] \\ &\leq \mathbb{E}_{\pi'} [R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2}, \pi'(S_{t+2})) | S_t = s] \\ &\leq \mathbb{E}_{\pi'} [R_{t+1} + \gamma R_{t+2} + \dots | S_t = s] = v_{\pi'}(s) \end{aligned}$$



# Policy Improvement (cont'd)

- ❖ If improvement stops,

$$q_{\pi}(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_{\pi}(s, a) = q_{\pi}(s, \pi(s)) = v_{\pi}(s)$$

- ❖ Then the Bellman optimality equation has been satisfied

$$v_{\pi}(s) = \max_{a \in \mathcal{A}} q_{\pi}(s, a)$$

- ❖ Therefore  $v_{\pi}(s) = v_*(s), \forall s \in \mathcal{S}$
- ❖  $\pi$  is an optimal policy

# Policy Improvement in Gridworld

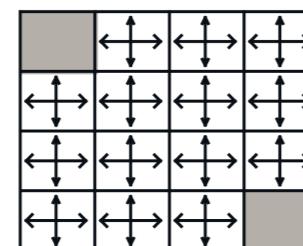


RLChina 2020

$V_k$  for the  
random policy

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

greedy policy  
w.r.t.  $V_k$

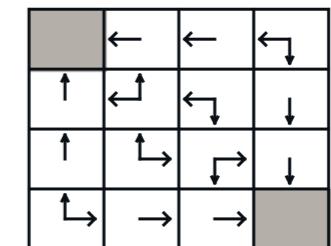


— random policy

$V_k$  for the  
random policy

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

greedy policy  
w.r.t.  $V_k$



$$k = 0$$

$$k = 1$$

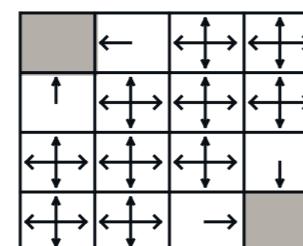
k = 2

k = 3

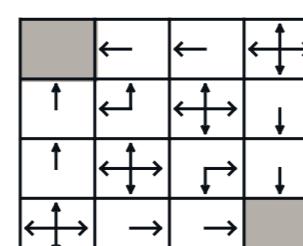
$k = 10$

$$k \equiv \alpha$$

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

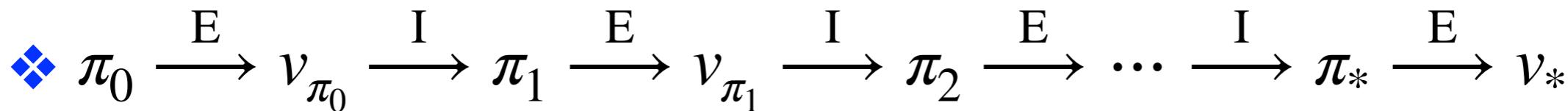


0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

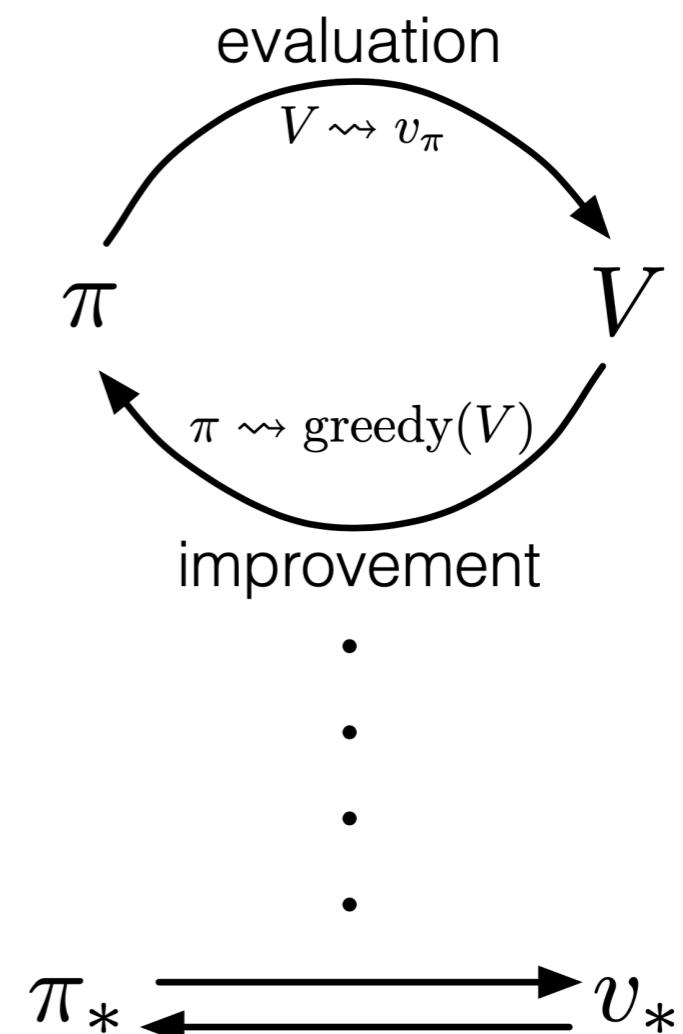
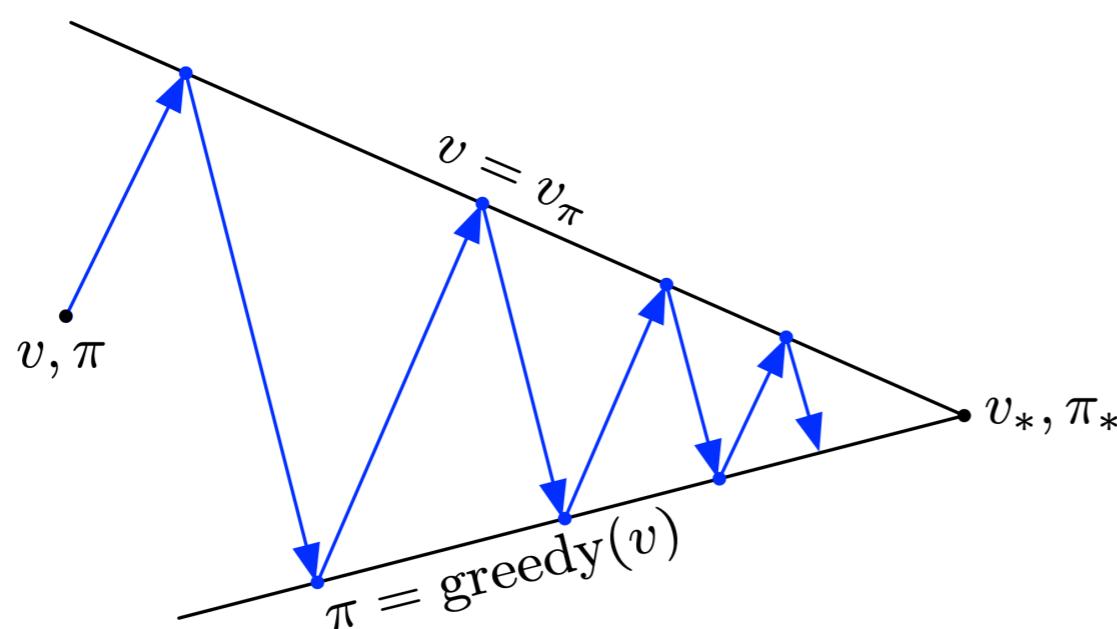




# Policy Iteration



- ❖ Policy evaluation: estimate  $v_\pi$ 
  - Iterative policy evaluation
- ❖ Policy improvement: generate  $\pi' \geq \pi$ 
  - Greedy policy improvement





# Policy Iteration (cont'd)

- ❖ Does policy evaluation need to converge to  $v_\pi$ ?
- ❖ Stop after  $k$  iterations of policy evaluation
  - $k = 3$  is sufficient for the gridworld example
- ❖ Why not update policy every iteration? i.e.,  $k = 1$ 
  - This is *value iteration*



# Value Iteration

❖ Any optimal policy can be subdivided into two components:

- An optimal first action  $A_*$
- Followed by an optimal policy from successor state  $s'$

❖ Deterministic Value Iteration

- If we know the solution to subproblems  $v_*(s')$
- Then solution  $v_*(s)$  can be found by one-step lookahead

$$v_*(s) \leftarrow \max_{a \in \mathcal{A}} \left( \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s') \right)$$

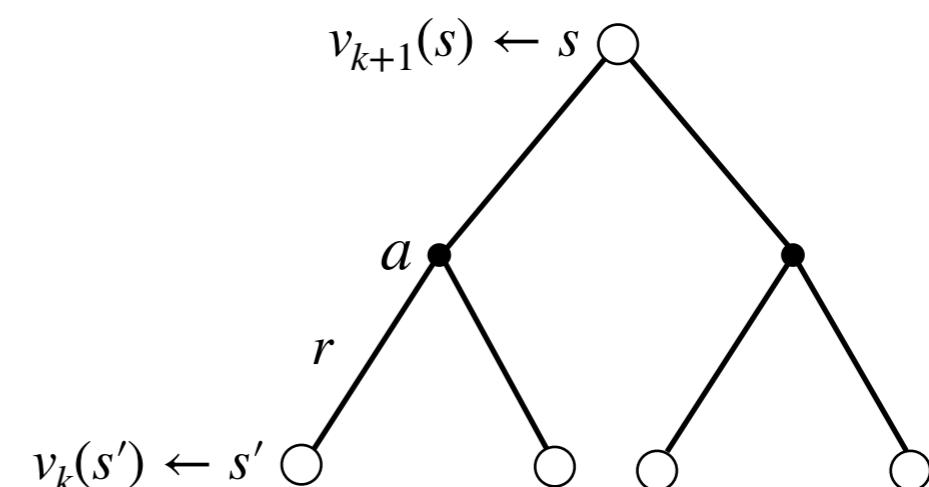
- Value iteration is applying these updates iteratively
- Intuition: start with final rewards and work backwards



# Value Iteration

- ❖ Problem: find optimal policy  $\pi$
- ❖ Solution: iteration of Bellman optimality backup
- ❖  $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_*$
- ❖ Each iteration  $k + 1, \forall s \in \mathcal{S}$ , update  $v_{k+1}(s)$  from  $v_k(s')$
- ❖ Convergence to  $v_*$
- ❖ Unlike policy iteration, there is no explicit policy

$$v_{k+1}(s) = \max_{a \in \mathcal{A}} \left( \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$



# Short Path Example



RLChina 2020

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

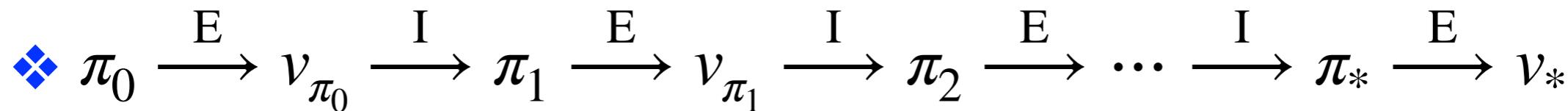
 $V_6$ 

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

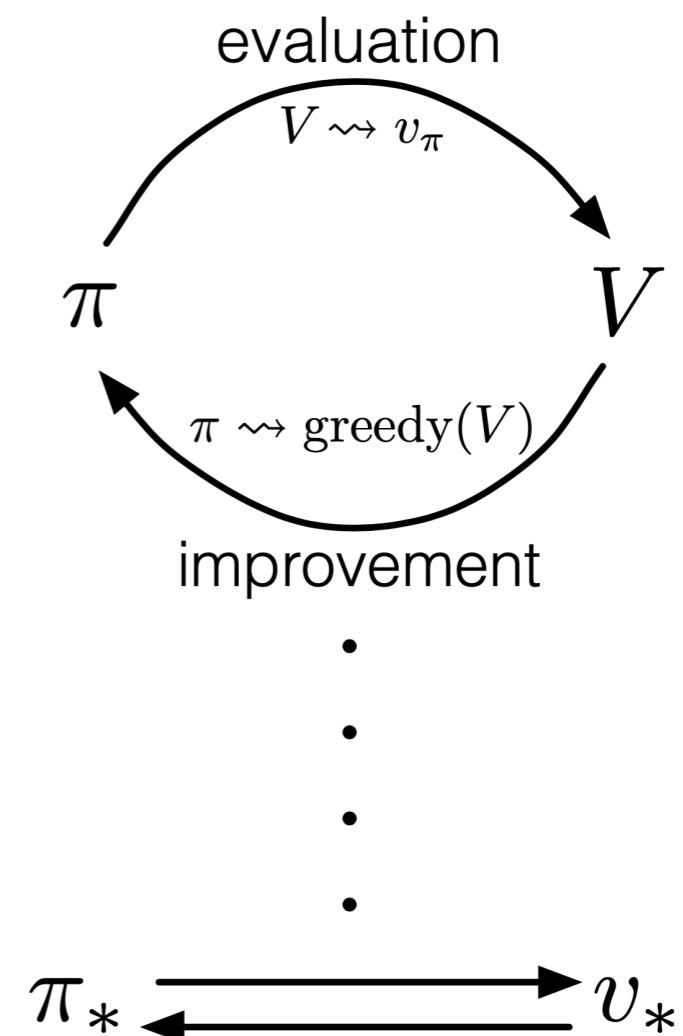
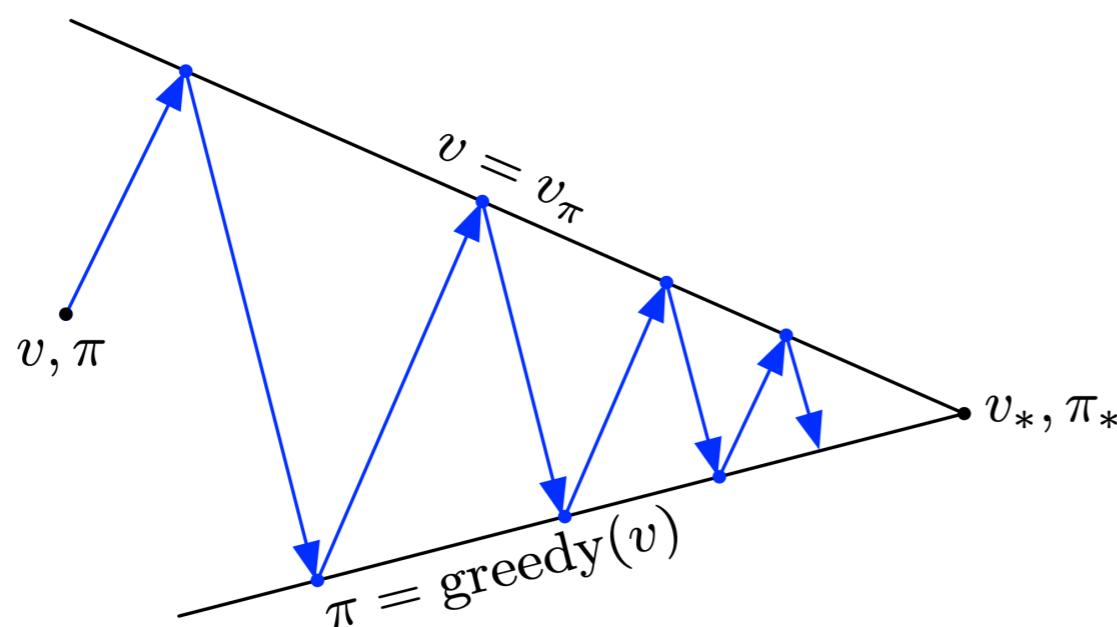
 $V_7$



# Generalized Policy Iteration



- ❖ Policy evaluation: estimate  $v_\pi$ 
  - Any policy evaluation algorithm
- ❖ Policy improvement: generate  $\pi' \geq \pi$ 
  - Any policy improvement algorithm



# Questions



RLChina 2020

- ❖ Does iterative policy evaluation converges to  $v_\pi$ ?
- ❖ Does policy iteration converges to  $v_*$ ?
- ❖ Does value iteration converges to  $v_*$ ?
- ❖ Is the solution unique?
- ❖ How fast do these algorithms converge?

**Yes, can be proved based on contraction mapping theory !**



# Contraction Mapping

## ❖ Value function space

- Consider a vector space  $\mathcal{V}$  with  $|\mathcal{S}|$  dimensions over value functions
- Each point in this space fully specifies a state-value function  $v$

## ❖ Value function $\infty$ -norm

- The distance between state-value functions  $u$  and  $v$  is measured by  $\infty$ -norm

$$\|u - v\|_\infty = \max_{s \in \mathcal{S}} |u(s) - v(s)|$$

## ❖ Define the Bellman expectation backup operator $\mathcal{T}^\pi$

$$\mathcal{T}^\pi(v) = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v$$

## ❖ The operator is a $\gamma$ -contraction, i.e., it makes value functions closer by at least $\gamma$

$$\begin{aligned} \|\mathcal{T}^\pi(u) - \mathcal{T}^\pi(v)\|_\infty &= \|(\mathcal{R}^\pi + \gamma \mathcal{P}^\pi u) - (\mathcal{R}^\pi + \gamma \mathcal{P}^\pi v)\|_\infty \\ &= \|\gamma \mathcal{P}^\pi(u - v)\|_\infty \\ &\leq \|\gamma \mathcal{P}^\pi\| \|u - v\|_\infty \\ &\leq \gamma \|u - v\|_\infty \end{aligned}$$



# Contraction Mapping Theorem

- ❖ For any metric space  $\mathcal{V}$  that is complete under an operator  $\mathcal{T}(v)$ , where  $\mathcal{T}$  is a  $\gamma$ -contraction
  - $\mathcal{T}$  converges to a unique fixed point
  - At a linear convergence rate of  $\gamma$

# Convergence of Policy Evaluation and Policy Iteration



RLChina 2020

- ❖ The Bellman expectation operator  $\mathcal{T}^\pi$  has a unique fixed point
- ❖  $v_\pi$  is a fixed point of  $\mathcal{T}^\pi$  (by Bellman expectation equation)
- ❖ Iterative policy evaluation converges on  $v_\pi$
- ❖ Policy iteration converges on  $v_*$



# Convergence of Value Iteration

- ❖ Define the Bellman optimality backup operator  $\mathcal{T}^*$

$$\mathcal{T}^*(v) = \max_{a \in \mathcal{A}} \mathcal{R}^a + \gamma \mathcal{P}^a v$$

- ❖ The operator is a  $\gamma$ -contraction (similar to previous proof)

$$\| \mathcal{T}^*(u) - \mathcal{T}^*(v) \|_\infty \leq \gamma \| u - v \|_\infty$$

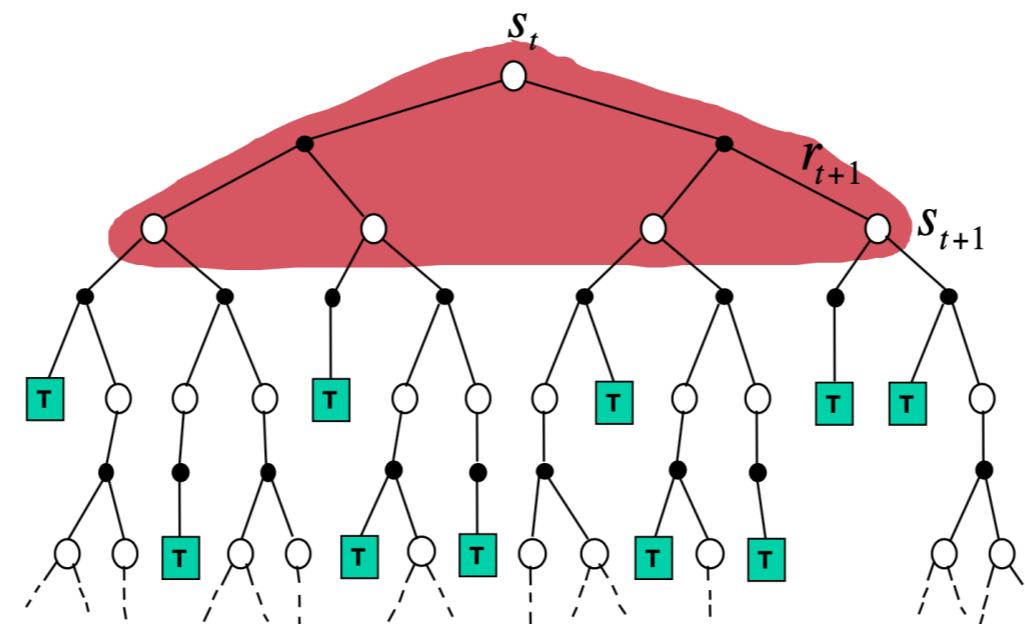
- ❖ The Bellman optimality operator  $\mathcal{T}^*$  has a unique fixed point
- ❖  $v_*$  is a fixed point of  $\mathcal{T}^*$  (by Bellman optimality equation)
- ❖ Value iteration converges on  $v_*$

# DP Summary



RLChina 2020

- ❖ Algorithms are based on state-value function  $v_\pi(s)$  or  $v_*(\pi)$
- ❖ Complexity  $O(mn^2)$  per iteration, for  $m$  actions and  $n$  states
- ❖ DP uses full-width backups, and for each backup
  - Every successor state and action is considered
  - Using knowledge of the MDP transitions and reward function
  - For large problem, **DP suffers from the curse of dimensionality**



# Outline



RLChina 2020

## ❖ Introduction to Reinforcement Learning

- About RL
- RL problem
- Inside an RL agent
- Markov Decision Processes

## ❖ Value-based Methods

- Dynamic Programming
- Monte Carlo
- TD Learning
- Off-policy Learning
- DQN and its variants

# Monte-Carlo Methods



RLChina 2020

- ❖ MC methods learn directly from episodes of experience
- ❖ MC is model-free: no knowledge of MDP transitions/rewards
- ❖ MC learns from complete episodes: no bootstrapping
- ❖ MC uses the simplest possible idea: value = mean return
- ❖ Can only apply MC to episodic MDPs
  - All episodes must terminate



# Monte-Carlo Policy Evaluation

- ❖ Goal: learn  $v_\pi$  from episodes of experience under policy  $\pi$

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

- ❖ Recall that the return is the total discounted rewards:

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

- ❖ Recall that the value function is the expected return from  $s$

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

- ❖ Monte-Carlo policy evaluation uses empirical mean return instead of expected return



# Monte-Carlo Policy Evaluation (cont'd)

- ❖ To evaluate state  $s$
- ❖ The **first/every** timestep  $t$  that state  $s$  is visited in an episode
- ❖ Increment counter  $N(s) \leftarrow N(s) + 1$
- ❖ Increment total return  $S(s) \leftarrow S(s) + G_t$
- ❖ Value is estimated by mean return  $V(s) \leftarrow S(s)/N(s)$
- ❖ By law of large numbers,  $V(s) \rightarrow v_\pi(s)$  as  $N(s) \rightarrow \infty$



# Incremental Monte-Carlo Updates

- ❖ Incremental mean

$$\mu_k = \frac{1}{k} \sum_{j=1}^k x_j = \frac{1}{k} (x_k + (k-1)\mu_{k-1}) = \mu_{k-1} + \frac{1}{k} (x_k - \mu_{k-1})$$

- ❖ Update  $V(s)$  incrementally after an episode
- ❖ For each state  $s_t$  with return  $G_t$

$$N(s_t) \leftarrow N(s_t) + 1$$

$$V(s_t) \leftarrow V(s_t) + \frac{1}{N(s_t)} (G_t - V(s_t))$$

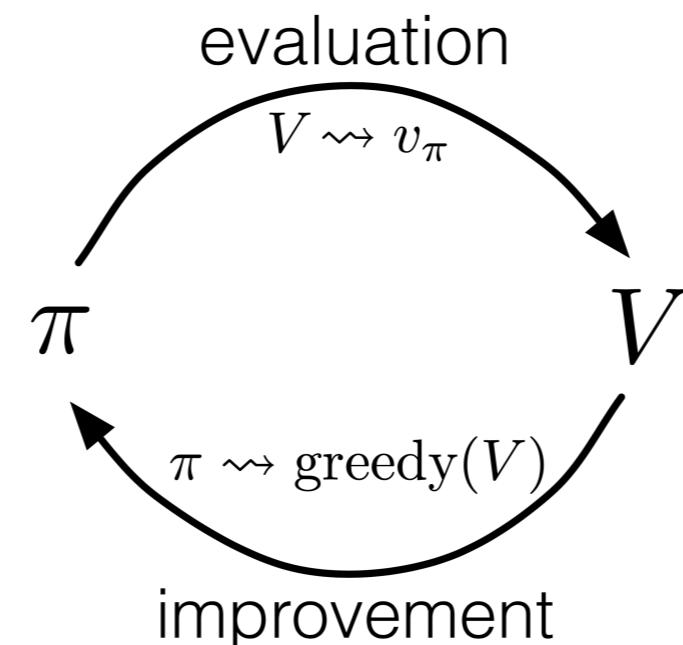
- ❖ In non-stationary problems, it can be used to track a running mean, i.e., forget old episodes

$$V(s_t) \leftarrow V(s_t) + \alpha (G_t - V(s_t))$$



# Monte-Carlo Policy Iteration

- ❖ Generalized Policy Iteration via Monte-Carlo Evaluation



- **Policy evaluation:** Monte-Carlo policy evaluation,  $V = v_\pi$ ?
- **Policy improvement:** Greedy policy improvement?

# Policy Iteration Using Action-Value Function



- ❖ Greedy policy improvement over  $V(s)$  requires model of MDPs

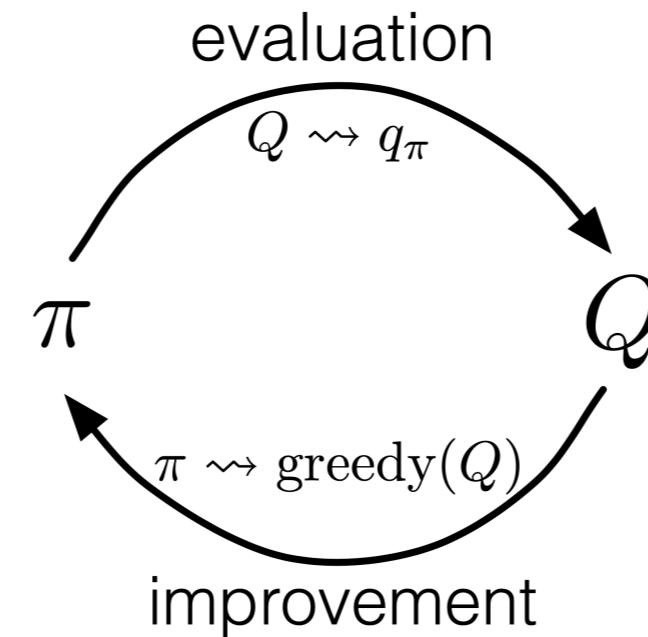
$$\pi'(s) = \arg \max_{a \in \mathcal{A}} (\mathcal{R}_s^a + \mathcal{P}_{ss'}^a V(s'))$$

- ❖ Greedy policy improvement over  $Q(s, a)$  is **model-free**

$$\pi'(s) = \arg \max_{a \in \mathcal{A}} Q(s, a)$$



# Monte-Carlo Policy Iteration



- Policy evaluation: Monte-Carlo policy evaluation,  $Q = q_\pi$
- Policy improvement: Greedy policy improvement?



# $\epsilon$ -greedy Exploration

- ❖ Simplest idea for ensuring continual exploration
- ❖ All  $|\mathcal{A}|$  actions are tried with non-zero probability
- ❖ With probability  $1 - \epsilon$  choose the greedy action
- ❖ With probability  $\epsilon$  choose a random action

$$\pi(a | s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \arg \max_{a \in \mathcal{A}} Q(s, a) \\ \epsilon/m & \text{otherwise} \end{cases}$$



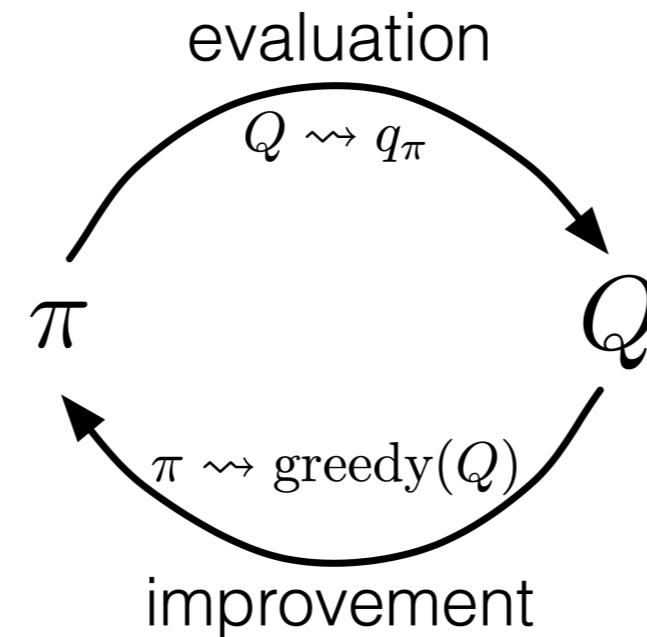
# $\epsilon$ -greedy Policy Improvement

- ❖ For any  $\epsilon$ -greedy policy  $\pi$ , the  $\epsilon$ -greedy policy  $\pi'$  with respect to  $q_\pi$  is an improvement,  $v_{\pi'} \geq v_\pi$

$$\begin{aligned} q_\pi(s, \pi'(s)) &= \sum_{a \in \mathcal{A}} \pi'(a | s) q_\pi(s, a) \\ &= \epsilon/m \sum_{a \in \mathcal{A}} q_\pi(s, a) + (1 - \epsilon) \max_{a \in \mathcal{A}} q_\pi(s, a) \\ &\geq \epsilon/m \sum_{a \in \mathcal{A}} q_\pi(s, a) + (1 - \epsilon) \sum_{a \in \mathcal{A}} \frac{\pi(a | s) - \epsilon/m}{1 - \epsilon} q_\pi(s, a) \\ &= \sum_{a \in \mathcal{A}} \pi(a | s) q_\pi(s, a) \\ &= v_\pi(s) \end{aligned}$$

- ❖ From policy improvement theorem,  $v_{\pi'} \geq v_\pi$

# Monte-Carlo Control



## ❖ Every episode:

- Policy evaluation: Monte-Carlo policy evaluation,  $Q = q_\pi$
- Policy improvement:  $\epsilon$ -greedy policy improvement



# Greedy in the limit with Infinite Exploration

## ❖ GLIE

- All state-action pairs are explored infinitely often

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

- The policy converges on a greedy policy

$$\lim_{k \rightarrow \infty} \pi_k(a | s) = \mathbf{1}(a = \arg \max_{a' \in \mathcal{A}} Q_k(s, a'))$$

❖  $\epsilon$ -greedy is GLIE if  $\epsilon$  reduces to zero at  $\epsilon_k = \frac{1}{k}$

# GLIE Monte-Carlo Control



- ❖ Sample  $k$ th episode using  $\pi$ :  $\{S_1, A_1, R_2, \dots, S_T\} \sim \pi$
- ❖ For each state-action pair in the episode

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

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} (G_t - Q(S_t, A_t))$$

- ❖ Improve the policy based on new action-value function

$$\epsilon \leftarrow 1/k$$

$$\pi \leftarrow \epsilon - \text{greedy}(Q)$$

- ❖ GLIE Monte-Carlo control converges to the optimal action-value function,  $Q(s, a) \rightarrow q_*(s, a)$

# Outline



RLChina 2020

## ❖ Introduction to Reinforcement Learning

- About RL
- RL problem
- Markov Decision Processes

## ❖ Value-based Methods

- Dynamic Programming
- Monte Carlo
- TD Learning
- Off-policy Learning
- DQN and its variants

# Temporal-Difference Learning



- ❖ TD methods learn directly from episodes of experience
- ❖ TD is model-free: no knowledge of MDP transitions/rewards
- ❖ TD learns from incomplete episodes: by bootstrapping
- ❖ TD updates a guess towards a guess



# MC and TD Policy Evaluation

❖ Goal: learn  $v_\pi$  online from experience under policy  $\pi$

❖ Incremental every-visit MC

- $V(S_t)$  is updated toward actual return  $G_t$

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

❖ Simplest TD learning

- $V(S_t)$  is updated toward estimated return  $R_{t+1} + \gamma V(S_{t+1})$

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

- $R_{t+1} + \gamma V(S_{t+1})$  is called the *TD target*
- $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$  is called the *TD error*



# Advantages and Disadvantages of MC and TD

- ❖ TD can learn *before* knowing the final outcome
  - TD can learn online after every step
  - MC must wait until end of episode after return is known
- ❖ TD can learn *without* the final outcome
  - TD works in continuing (non-terminating) environments
  - MC only works for episodic (terminating) environments

# Bias/Variance Tradeoff

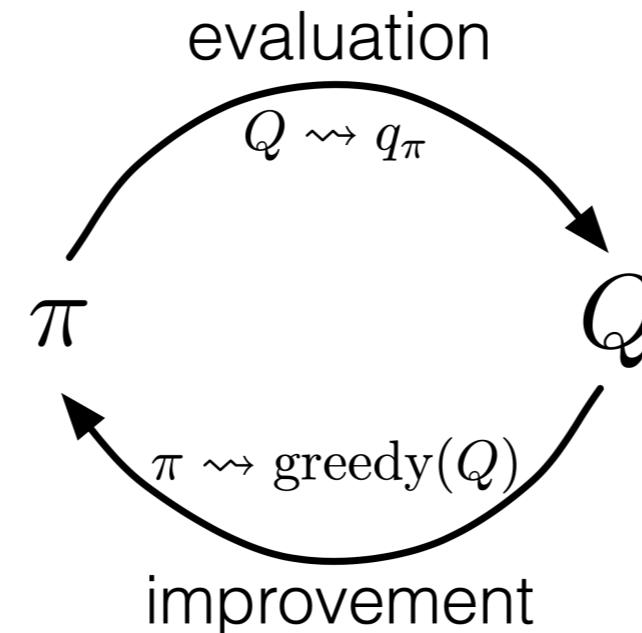


- ❖ Return  $G_t = R_{t+1} + \gamma R_{t+2} + \dots, \gamma^{T-1} R_T$  is *unbiased* estimate of  $v_\pi(S_t)$
- ❖ True TD target  $R_{t+1} + \gamma v_\pi(S_{t+1})$  is *unbiased* estimate of  $v_\pi(S_t)$
- ❖ TD target  $R_{t+1} + \gamma V(S_{t+1})$  is *biased* estimate of  $v_\pi(S_t)$
- ❖ TD target is much lower variance than the return
  - Return depends on **many** actions, transitions, rewards
  - TD target depends on **only one** action, transition, reward

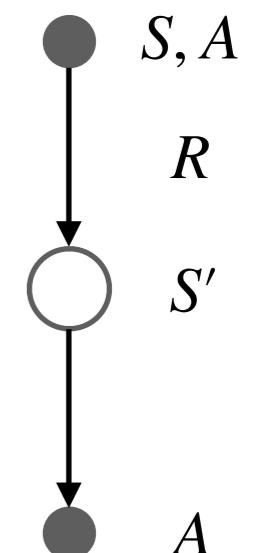


# TD Control: SARSA

- ❖ Every timestep:
  - Policy evaluation:  $Q \approx q_\pi$
  - Policy improvement:  $\epsilon$ -greedy policy improvement



❖ 
$$Q(S, A) \leftarrow Q(S, A) + \alpha \left( R + \gamma Q(S', A') - Q(S, A) \right)$$



# SARSA Algorithm



RLChina 2020

Sarsa (on-policy TD control) for estimating  $Q \approx q_*$

Algorithm parameters: step size  $\alpha \in (0, 1]$ , small  $\varepsilon > 0$

Initialize  $Q(s, a)$ , for all  $s \in \mathcal{S}^+$ ,  $a \in \mathcal{A}(s)$ , arbitrarily except that  $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

    Initialize  $S$

    Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)

    Loop for each step of episode:

        Take action  $A$ , observe  $R, S'$

        Choose  $A'$  from  $S'$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$$

$S \leftarrow S'; A \leftarrow A'$ ;

    until  $S$  is terminal

# Outline



RLChina 2020

## ❖ Introduction to Reinforcement Learning

- About RL
- RL problem
- Inside an RL agent
- Markov Decision Processes

## ❖ Value-based Methods

- Dynamic Programming
- Monte Carlo
- TD Learning
- Off-policy Learning
- DQN and its variants

# Off-Policy Learning



RLChina 2020

## ❖ Off-policy learning

- Evaluate *target policy*  $\pi(a | s)$  to compute  $v_\pi(s)$  or  $q_\pi(s, a)$ , while following *behavior policy*  $\mu(a | s)$

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

## ❖ Why is this important?

- Learn from observing humans or other agents
- Reuse experience generated from old policies  $\pi_1, \pi_2, \dots, \pi_{t-1}$
- Learn about *optimal policy* while following *exploratory policy*
- Learn about *multiple policies* while following *one policy*



# Importance Sampling

- ❖ Estimate the expectation of a difference distribution

$$\begin{aligned}\mathbb{E}_{X \sim P}[f(X)] &= \sum P(X)f(X) \\ &= \sum Q(X)\frac{P(X)}{Q(X)}f(X) \\ &= \mathbb{E}_{X \sim Q}\left[\frac{P(X)}{Q(X)}f(X)\right]\end{aligned}$$

# Importance Sampling for Off-Policy MC

- ❖ Use returns generated from  $\mu$  to evaluate  $\pi$
- ❖ Weight return  $G_t$  according to similarity between policies
- ❖ Multiply importance sampling corrections along whole episode

$$G_t^{\pi/\mu} = \frac{\pi(A_t | S_t)}{\mu(A_t | S_t)} \frac{\pi(A_{t+1} | S_{t+1})}{\mu(A_{t+1} | S_{t+1})} \dots \frac{\pi(A_T | S_T)}{\mu(A_T | S_T)} G_t$$

- ❖ Update values towards corrected return

$$V(S_t) \leftarrow V(S_t) + \alpha \left( G_t^{\pi/\mu} - V(S_t) \right)$$

- ❖ Importance sampling can dramatically increase variance

# Importance Sampling for Off-Policy TD

- ❖ Use TD targets generated from  $\mu$  to evaluate  $\pi$
- ❖ Weighted TD target  $R + \gamma V(S')$  by importance sampling
- ❖ Only need a single importance correction

$$V(S_t) \leftarrow V(S_t) + \alpha \left( \frac{\pi(A_t | S_t)}{\mu(A_t | S_t)} (R_{t+1} + \gamma V(S_{t+1})) - V(S_t) \right)$$

- ❖ Much lower variance than Monte-Carlo importance sampling

# Q Learning



RLChina 2020

- ❖ Now considering off-policy learning of action-values  $Q(s, a)$
- ❖ **No** importance sampling is required
- ❖ Action is chosen using behavior policy  $A_t \sim \mu(\cdot | S_t)$
- ❖ But we consider alternative successor action  $A' \sim \pi(\cdot | S_{t+1})$
- ❖ Update  $Q(S_t, A_t)$  towards value of alternative action

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t))$$



# Off-Policy Control with Q-Learning

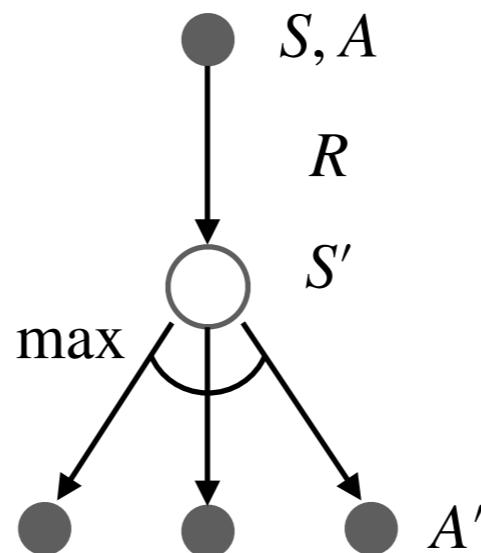
- ❖ The target policy  $\pi$  is **greedy** w.r.t.  $Q(s, a)$

$$\pi(S_{t+1}) = \arg \max_{a'} Q(S_{t+1}, a')$$

- ❖ The behavior policy  $\mu$  is e.g.  **$\epsilon$ -greedy** w.r.t  $Q(s, a)$
- ❖ The Q-learning target then simplifies:

$$\begin{aligned} & R_{t+1} + \gamma Q(S_{t+1}, A') \\ &= R_{t+1} + \gamma Q\left(S_{t+1}, \underset{a'}{\operatorname{argmax}} Q(S_{t+1}, a')\right) \\ &= R_{t+1} + \underset{a'}{\max} \gamma Q(S_{t+1}, a') \end{aligned}$$

# Off-Policy Control with Q-Learning (cont'd)



- ❖ Q-learning converges to the optimal action-value function

$$Q(s, a) \rightarrow q_*(s, a)$$

# Off-Policy Control with Q-Learning (cont'd)



RLChina 2020

Q-learning (off-policy TD control) for estimating  $\pi \approx \pi_*$

Algorithm parameters: step size  $\alpha \in (0, 1]$ , small  $\varepsilon > 0$

Initialize  $Q(s, a)$ , for all  $s \in \mathcal{S}^+$ ,  $a \in \mathcal{A}(s)$ , arbitrarily except that  $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

    Initialize  $S$

    Loop for each step of episode:

        Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)

        Take action  $A$ , observe  $R, S'$

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

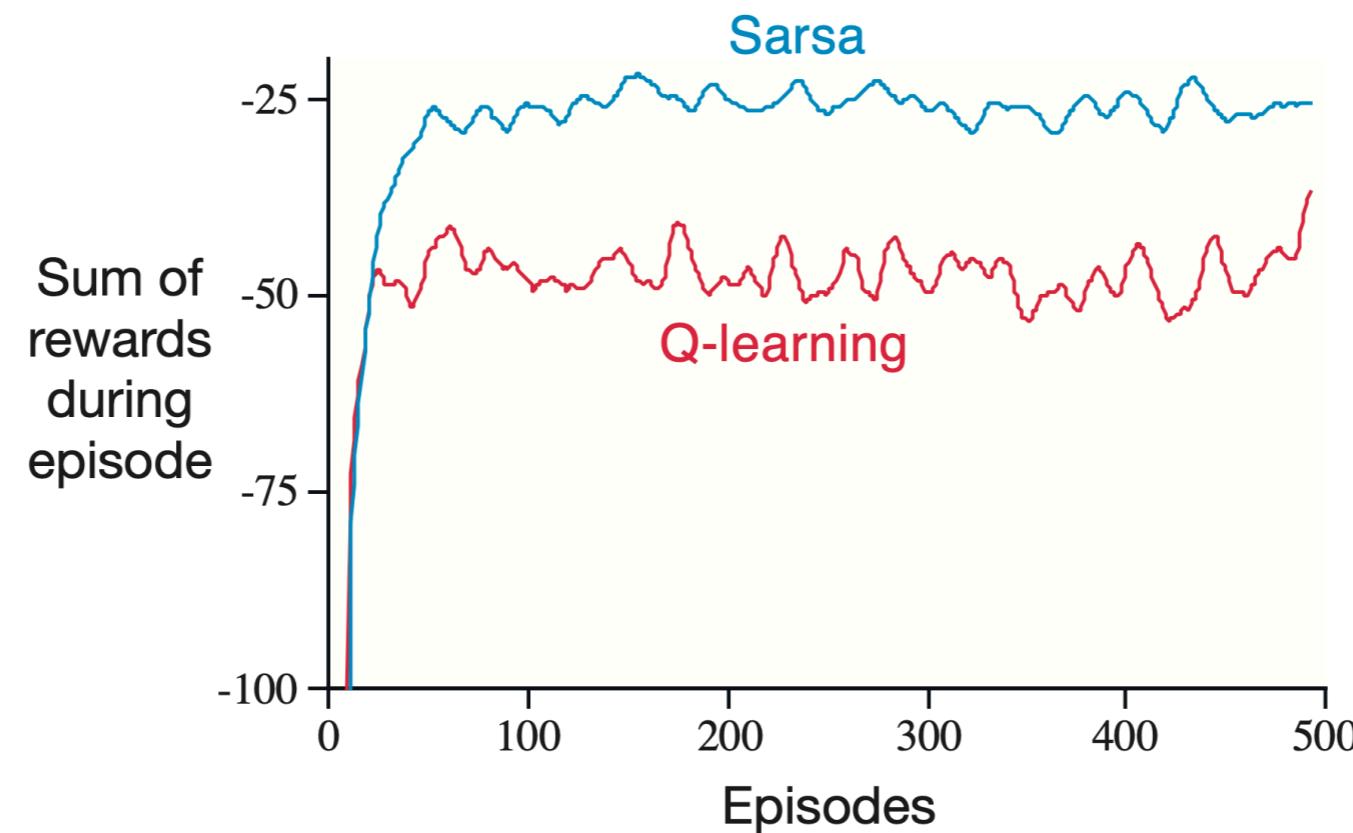
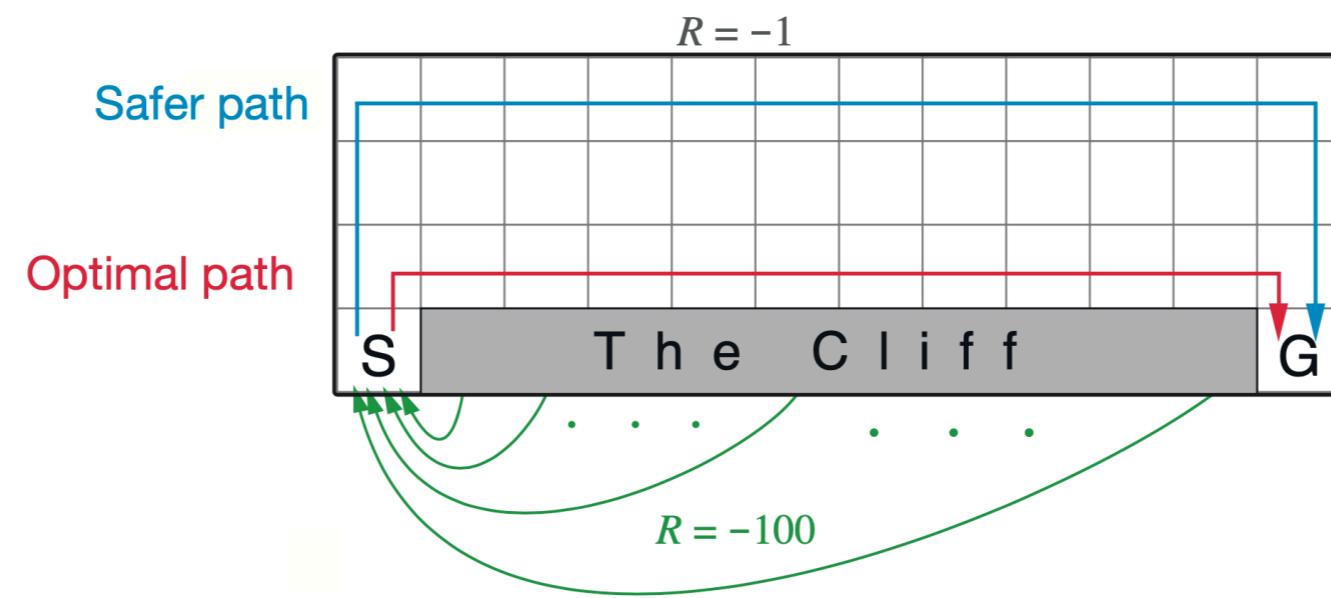
$S \leftarrow S'$

    until  $S$  is terminal

# Cliff Walking Example

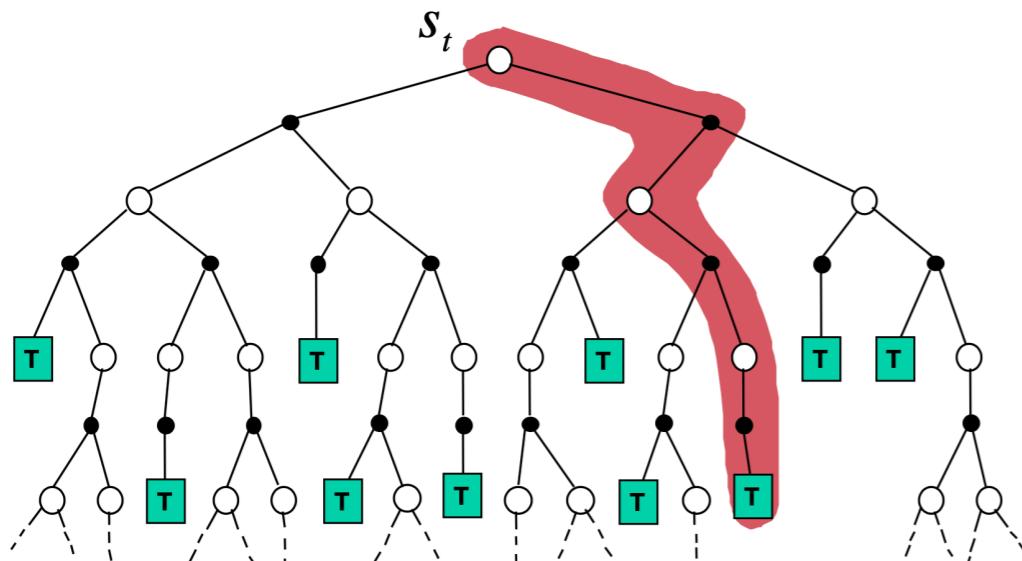


RLChina 2020

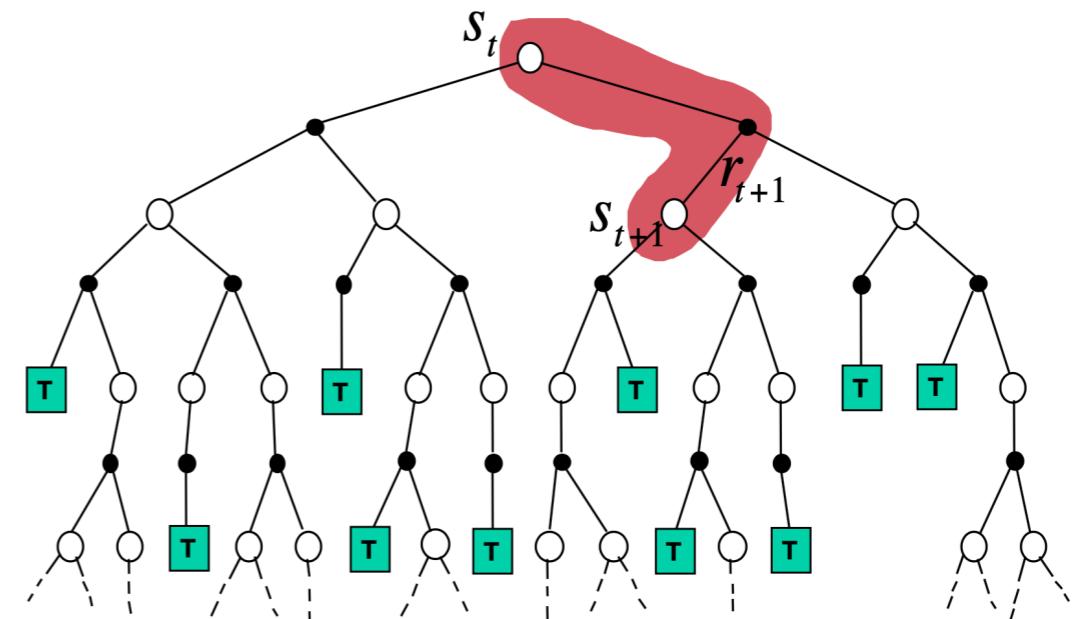




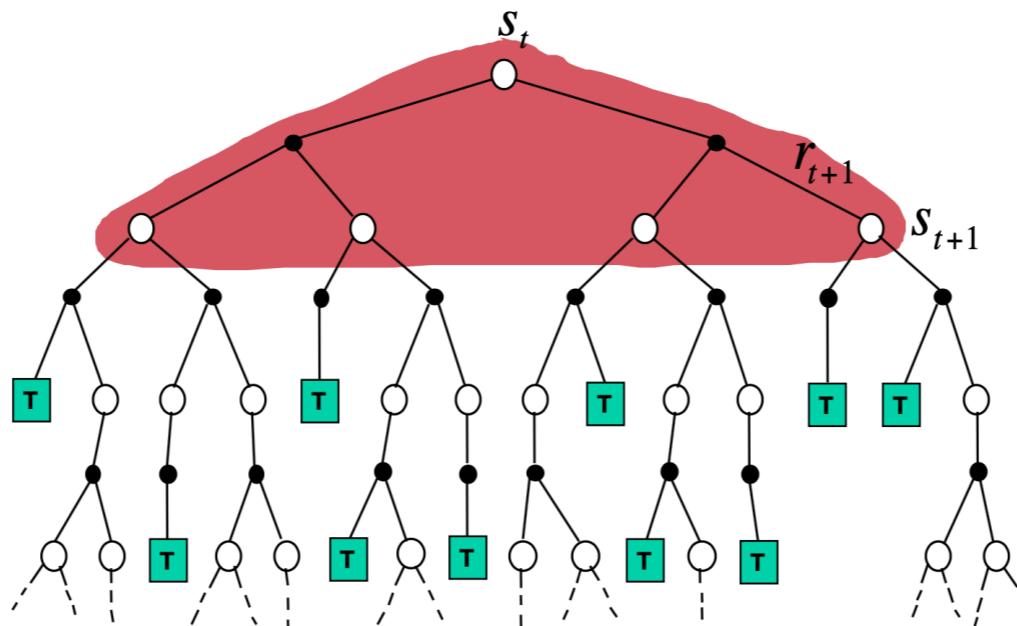
# Summary on DP, MC, and TD



MC backup



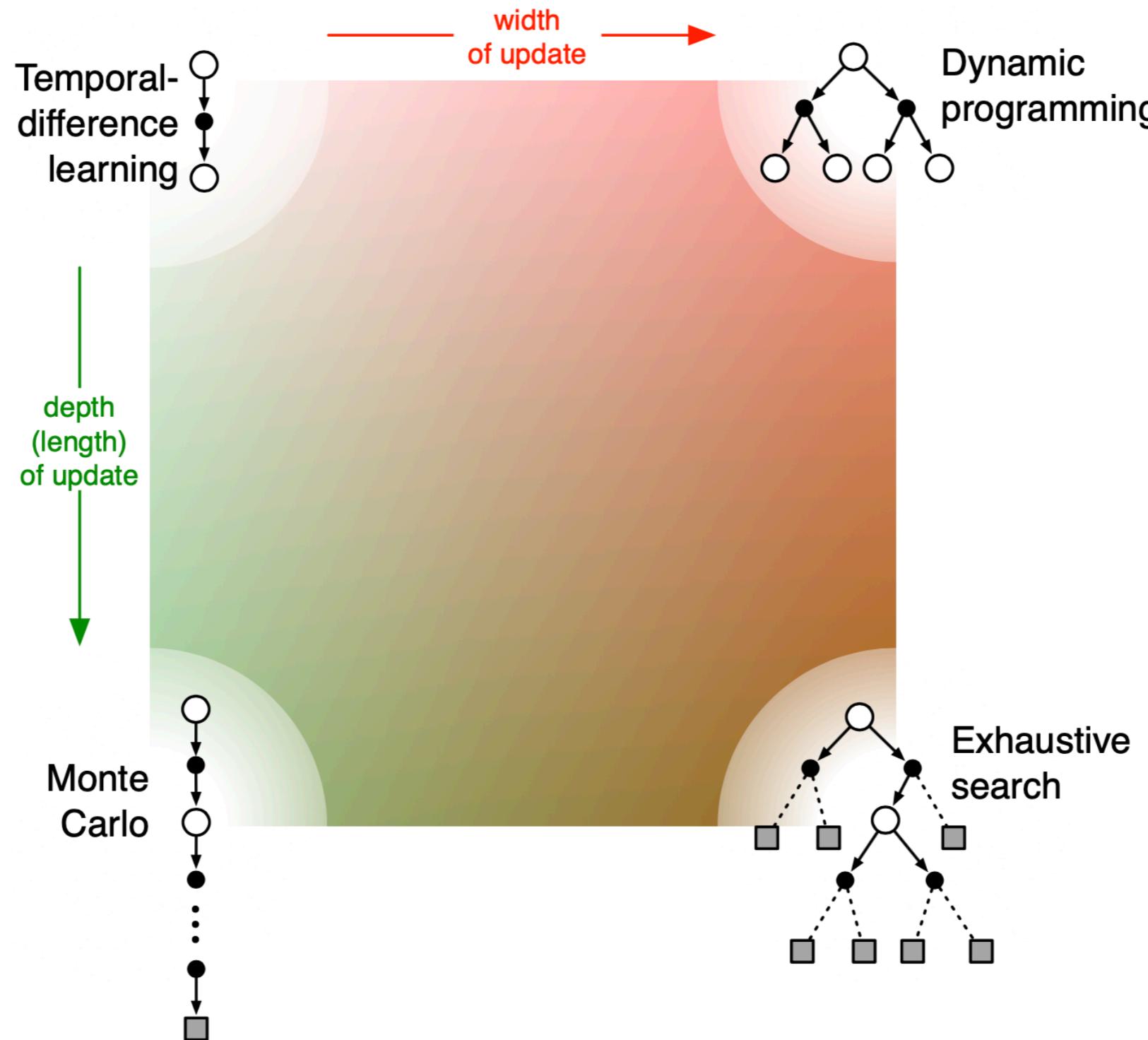
TD backup



DP backup



# Unified View of Reinforcement Learning



# Outline



## ❖ Introduction to Reinforcement Learning

- About RL
- RL problem
- Inside an RL agent
- Markov Decision Processes

## ❖ Value-based Methods

- Dynamic Programming
- Monte Carlo
- TD Learning
- Off-policy Learning
- DQN and its variants

# Reinforcement Learning in Practical



- ❖ Reinforcement learning can solve large problems, e.g.,
  - Backgammon:  $10^{20}$  states
  - Computer Go:  $10^{170}$  states
  - Robots: continuous state space
- ❖ How can we scale up the model-free methods?

# Value Function Approximation



RLChina 2020

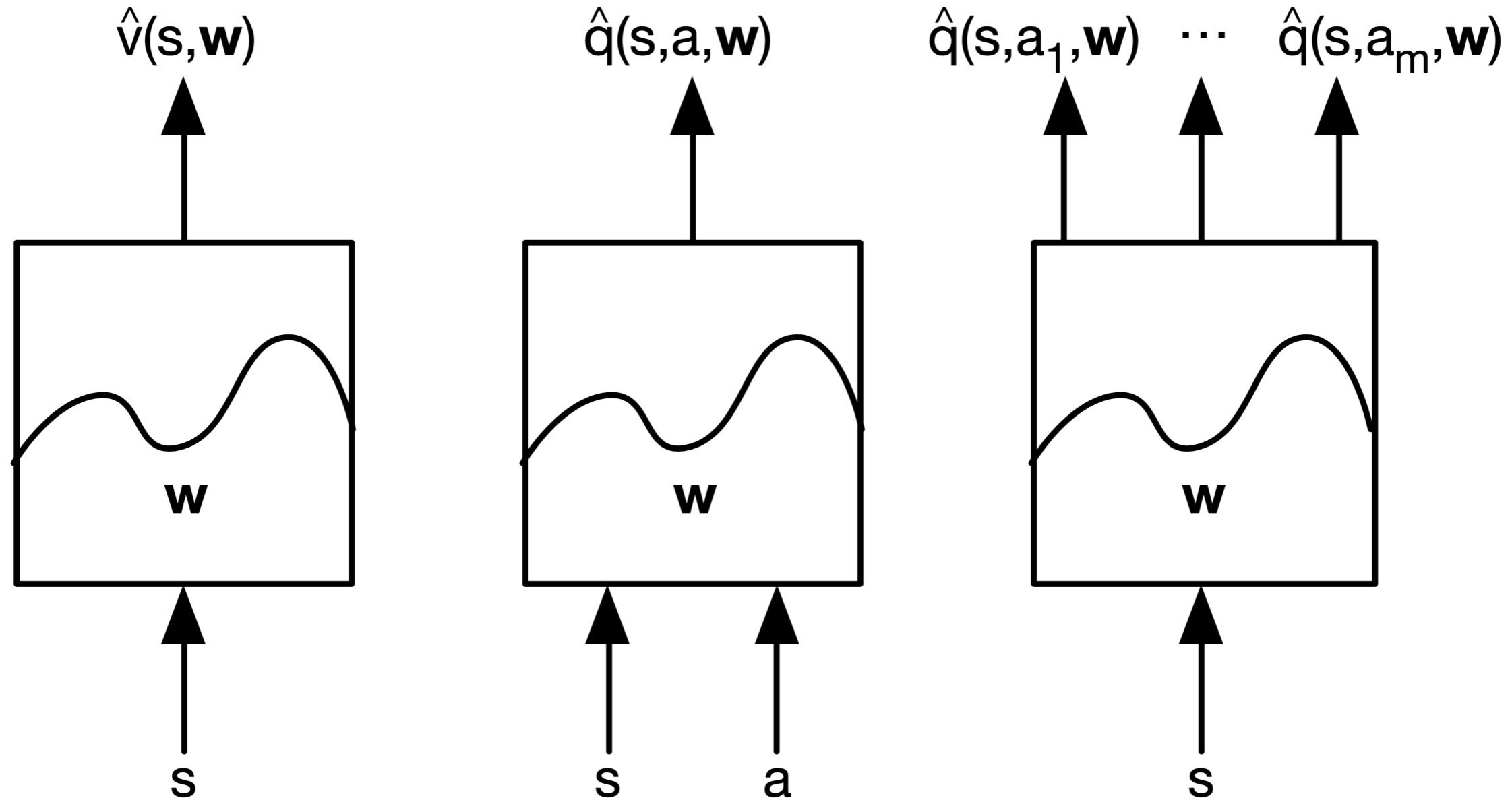
- ❖ We have represented value function by a lookup table
  - Every state  $s$  has an entry  $V(s)$
  - Every state-action pair  $s, a$  has an entry  $Q(s, a)$
- ❖ Problem with large MDPs
  - There are too many states and actions to store in memory
  - It is too slow to learn the value of each state individually
- ❖ Solution for large MDPs
  - Estimate value function with function approximation

$$\hat{v}(s, w) \approx v_\pi(s)$$

$$\hat{q}(s, a, w) \approx q_\pi(s, a)$$

- Generalize from seen states to unseen states
- Update parameter  $w$  using MC or TD learning

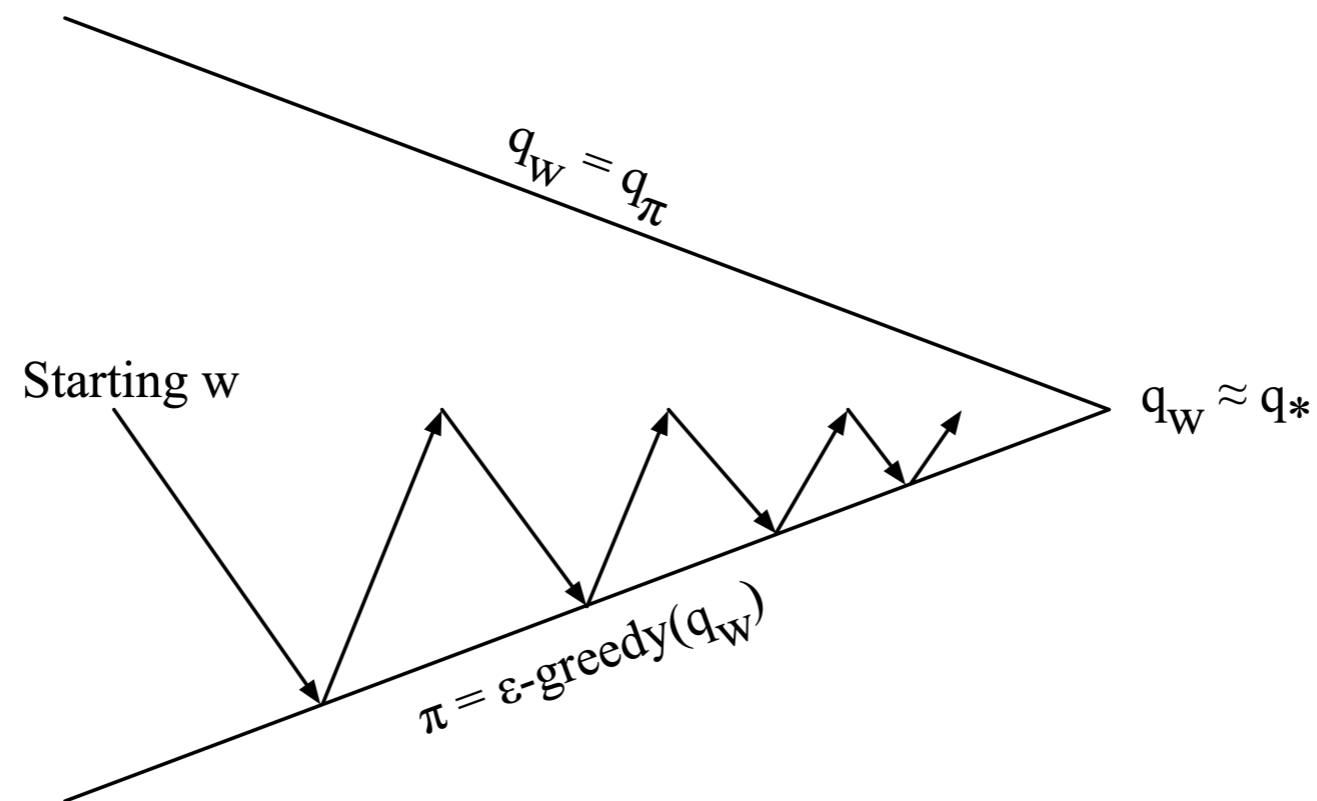
# Types of Value Function Approximation





# Control with Value Function Approximation

- ❖ Policy evaluation: *approximate* policy evaluation,  $\hat{q}(\cdot, \cdot, w) \approx q_\pi$
- ❖ Policy improvement:  $\epsilon$ -greedy policy improvement



# Action-Value Function Approximation



- ❖ Approximate the action-value function

$$\hat{q}(S, A, \mathbf{w}) \approx q_{\pi}(S, A)$$

- ❖ Minimize mean-squared error between approximate action-value function  $\hat{q}(S, A, \mathbf{w})$  and true function  $q_{\pi}(S, A)$

$$J(\mathbf{w}) = \mathbb{E}_{\pi}[(q_{\pi}(S, A) - \hat{q}(S, A, \mathbf{w}))^2]$$

- ❖ Using stochastic gradient descent to find a local minimum

$$\Delta \mathbf{w} = \alpha(q_{\pi}(S, A) - \hat{q}(S, A, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w})$$

# Action-Value Function Approximation (cont'd)



- ❖ We do not know  $q_\pi(S, A)$
- ❖ We substitute a target for  $q_\pi(S, A)$ 
  - For MC, the target is the return  $G_t$

$$\Delta \mathbf{w} = \alpha(G_t - \hat{q}(S_t, A_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w})$$

- For TD, the target is the TD target  $R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w})$

$$\Delta \mathbf{w} = \alpha(R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w})$$

# Online Q Learning



RLChina 2020

1. Take action  $a$  according to  $\epsilon$ -greedy policy and observe  
 $(s, a, r, s')$
  2.  $\Delta \mathbf{w} = \alpha(r + \gamma \max_{a'} \hat{q}(s', a', \mathbf{w}) - \hat{q}(s, a, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(s, a, \mathbf{w})$
- **Sequential states are strongly correlated!**
  - **Target Value is always changing!**

# Deep Q Network



- ❖ DQN uses **experience replay** and **target network**
  - Take  $\epsilon$ -greedy action w.r.t.  $Q(s, a; \mathbf{w})$
  - Store transition  $(s_t, a_t, r_{t+1}, s_{t+1})$  in replay memory  $\mathcal{D}$
  - Sample random mini-batch of transitions  $(s, a, r, s')$  from  $\mathcal{D}$
  - Optimize MSE between Q-network and Q-learning targets

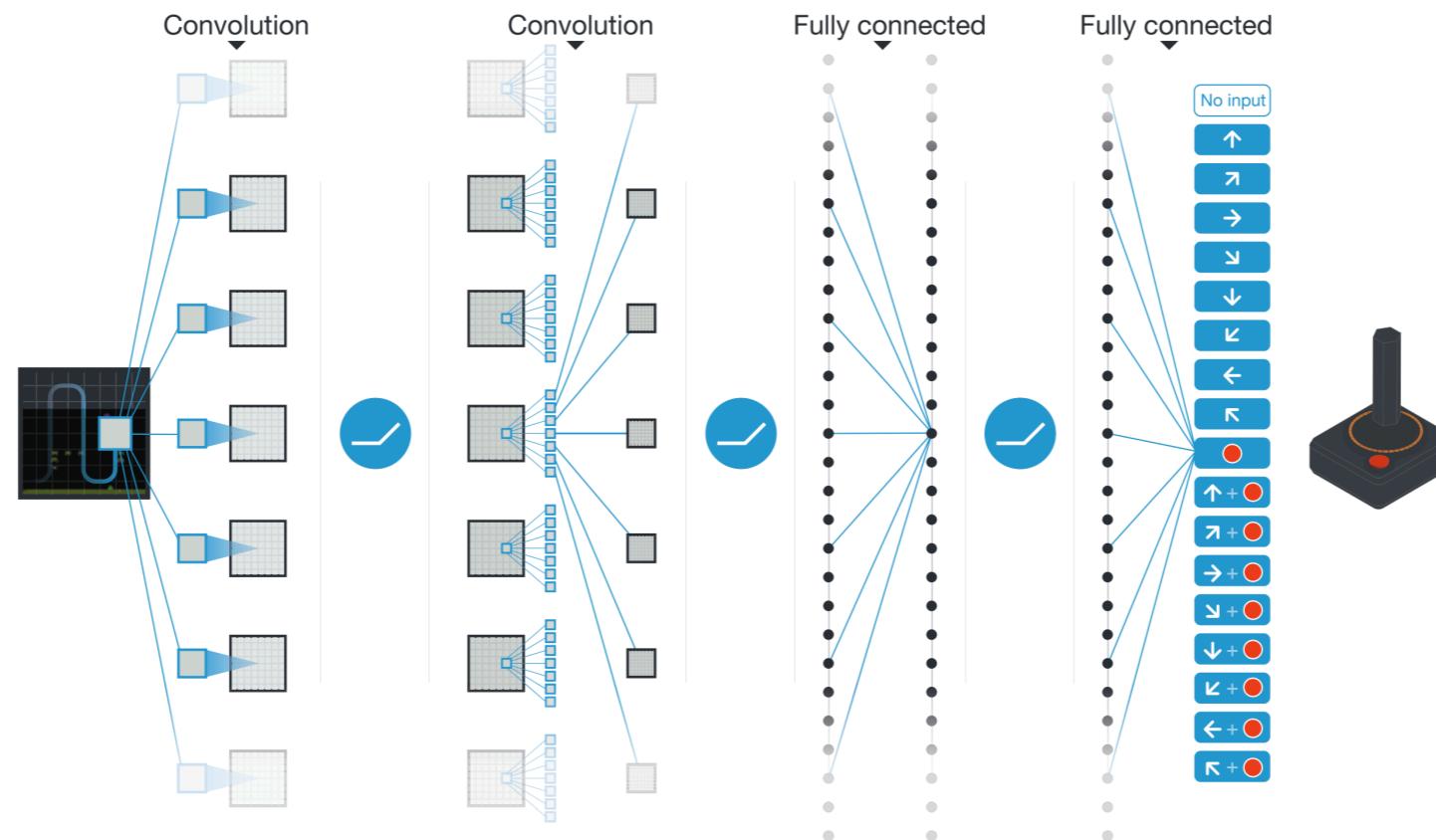
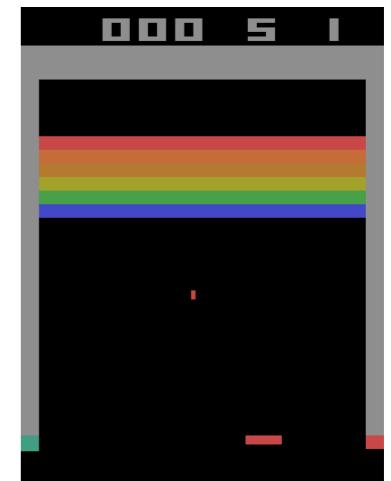
$$\mathcal{L}(\mathbf{w}) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[ \left( r + \gamma \max_{a'} Q(s', a'; \mathbf{w}^-) - Q(s, a; \mathbf{w}) \right)^2 \right]$$

- Using variant of SGD
- Every  $N$  timesteps,  $\mathbf{w}^- = \mathbf{w}$

# DQN in Atari

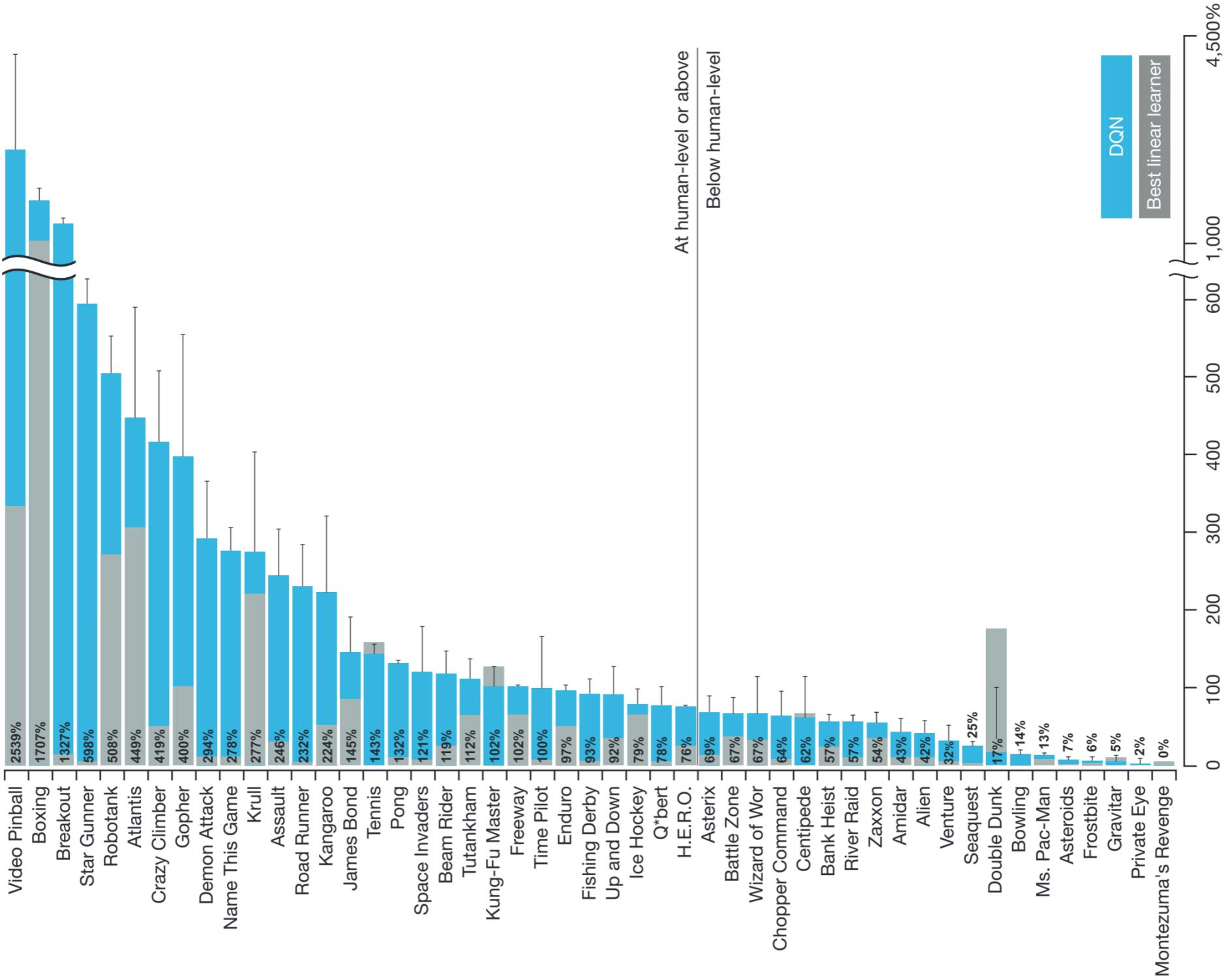


- ❖ End-to-end learning of  $Q(s, a)$  from pixels  $s$
- ❖ Input state  $s$  is stack of raw pixels from last 4 frames
- ❖ Output is  $Q(s, a)$  for 18 joystick/button positions
- ❖ Reward is change in score for that step





# DQN Results in Atari



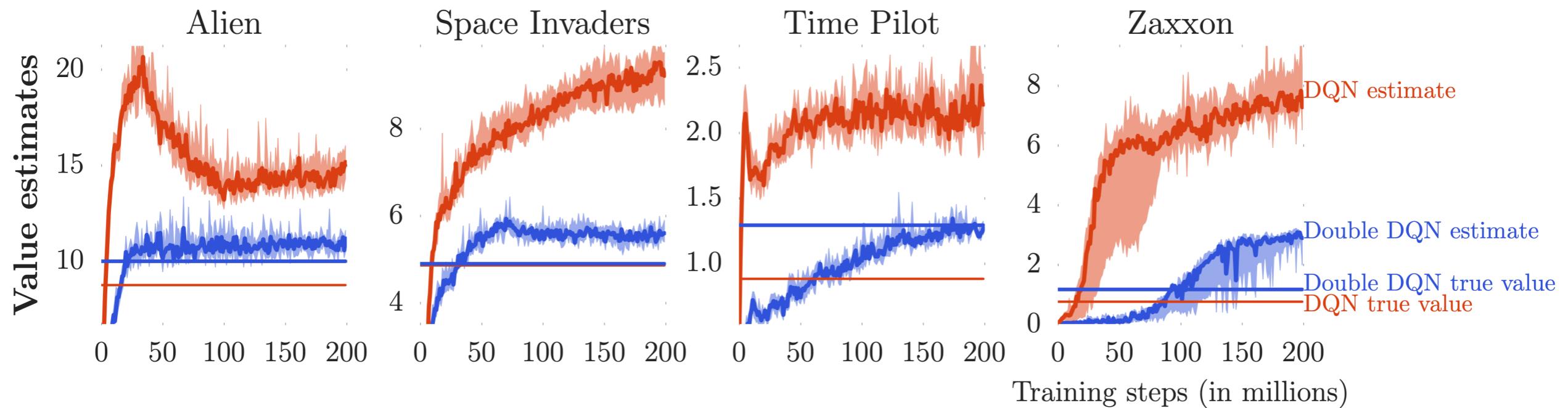


# How much does DQN help?

	Replay Fixed-Q	Replay Q-learning	No replay Fixed-Q	No replay Q-learning
Breakout	316.81	240.73	10.16	3.17
Enduro	1006.3	831.25	141.89	29.1
River Raid	7446.62	4102.81	2867.66	1453.02
Seaquest	2894.4	822.55	1003	275.81
Space Invaders	1088.94	826.33	373.22	301.99



# How accurate are the Q-values?



## ❖ Overestimation

- Target value:  $r + \gamma \max_{a'} Q(s', a'; \mathbf{w}^-)$
- This is the problem!**
- $\mathbb{E}[\max(X_1, X_2)] \geq \max(\mathbb{E}[X_1], \mathbb{E}[X_2])$
  - For bootstrapping (learning estimates from estimates), such overestimation can be problematic.

# Double DQN



RLChina 2020

- ❖  $\max_{a'} Q(s', a'; \mathbf{w}^-) = Q(s', \arg \max_{a'} Q(s', a'; \mathbf{w}^-); \mathbf{w}^-)$ 
  - Action selected according to  $Q(\cdot, \cdot; \mathbf{w}^-)$
  - Value also from  $Q(\cdot, \cdot; \mathbf{w}^-)$
- ❖ Use different networks to choose action and evaluate value
  - Current network to evaluate action
  - Target network to evaluate value

$$r + \gamma Q(s', \arg \max_{a'} Q(s', a'; \mathbf{w}); \mathbf{w}^-)$$

# Double DQN



❖ DQN uses **experience replay** and **target networks**

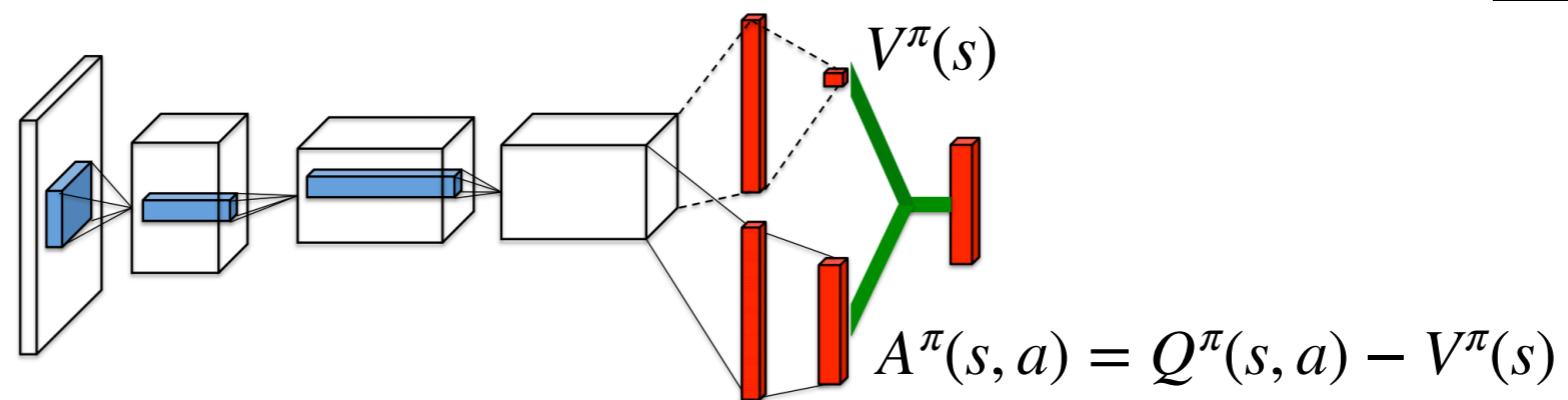
- Take action  $a_t$  according to  $\epsilon$ -greedy policy
- Store transition  $(s_t, a_t, r_{t+1}, s_{t+1})$  in replay memory  $\mathcal{D}$
- Sample random mini-batch of transitions  $(s, a, r, s')$  from  $\mathcal{D}$
- Optimize MSE between Q-network and Q-learning targets

$$\mathcal{L}(\mathbf{w}) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[ \left( r + \gamma \underbrace{\max_{a'} Q(s', a'; \mathbf{w}); \mathbf{w}^-}_{\text{Q-learning target}} - Q(s, a; \mathbf{w}) \right)^2 \right]$$

- Using variant of SGD
- $\mathbf{w}^- = (1 - \tau)\mathbf{w} + \tau\mathbf{w}^-$

# Dueling DQN

- ❖ **Motivation:** it is unnecessary to know the exact value of each action at every timestep
- ❖ Dueling DQN learns state value without having to learn the effect of each action for each state



- ❖  $Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha)$  “unidentifiable”
- ❖  $Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + (A(s, a; \theta, \alpha) - \max_{a' \in \mathcal{A}} A(s, a'; \theta, \alpha))$
- ❖  $Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + (A(s, a; \theta, \alpha) - \frac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} A(s, a'; \theta, \alpha))$

# *n*-Step DQN



RLChina 2020

$$\diamond R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n \max_a Q(S_{t+n}, a; \theta)$$

- Less biased target values when Q values are inaccurate
- Faster learning when early on
- But, only actually correct when learning on-policy

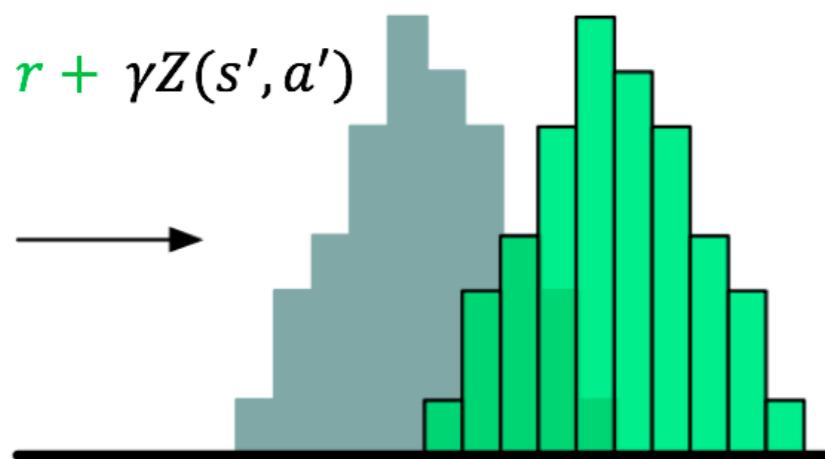
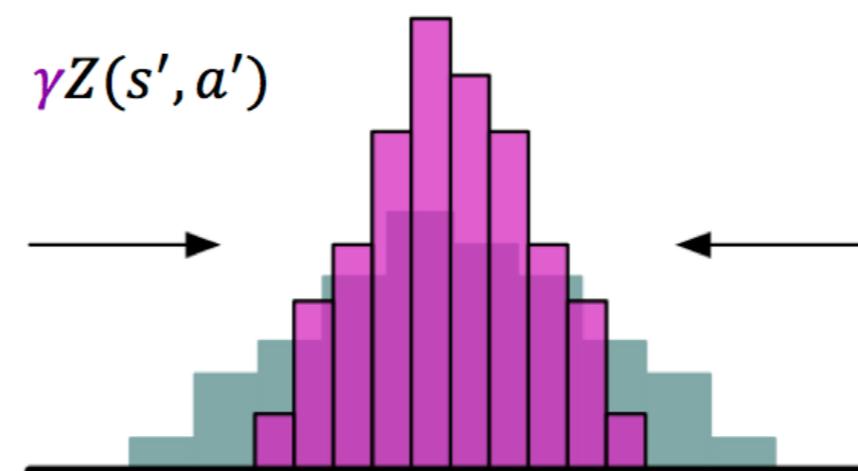
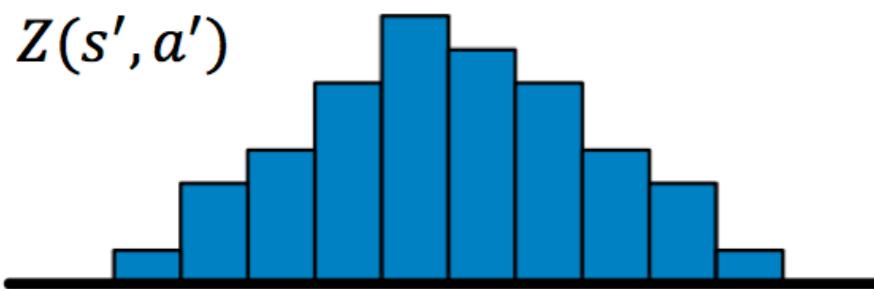
## ❖ How to deal with it?

- Ignore the problem  
Often works in practice
- Dynamically choose N to get only on-policy data  
Work well when data is on-policy and action space is small
- Importance sampling

# Distributional DQN



- ❖  $Q^\pi(s, a) = \mathbb{E}[G_t]$
- ❖  $Q^*(s, a) = r + \gamma \max_{a'} Q^*(s', a')$
- ❖  $Z(s, a) = r + \gamma Z(s', a')$





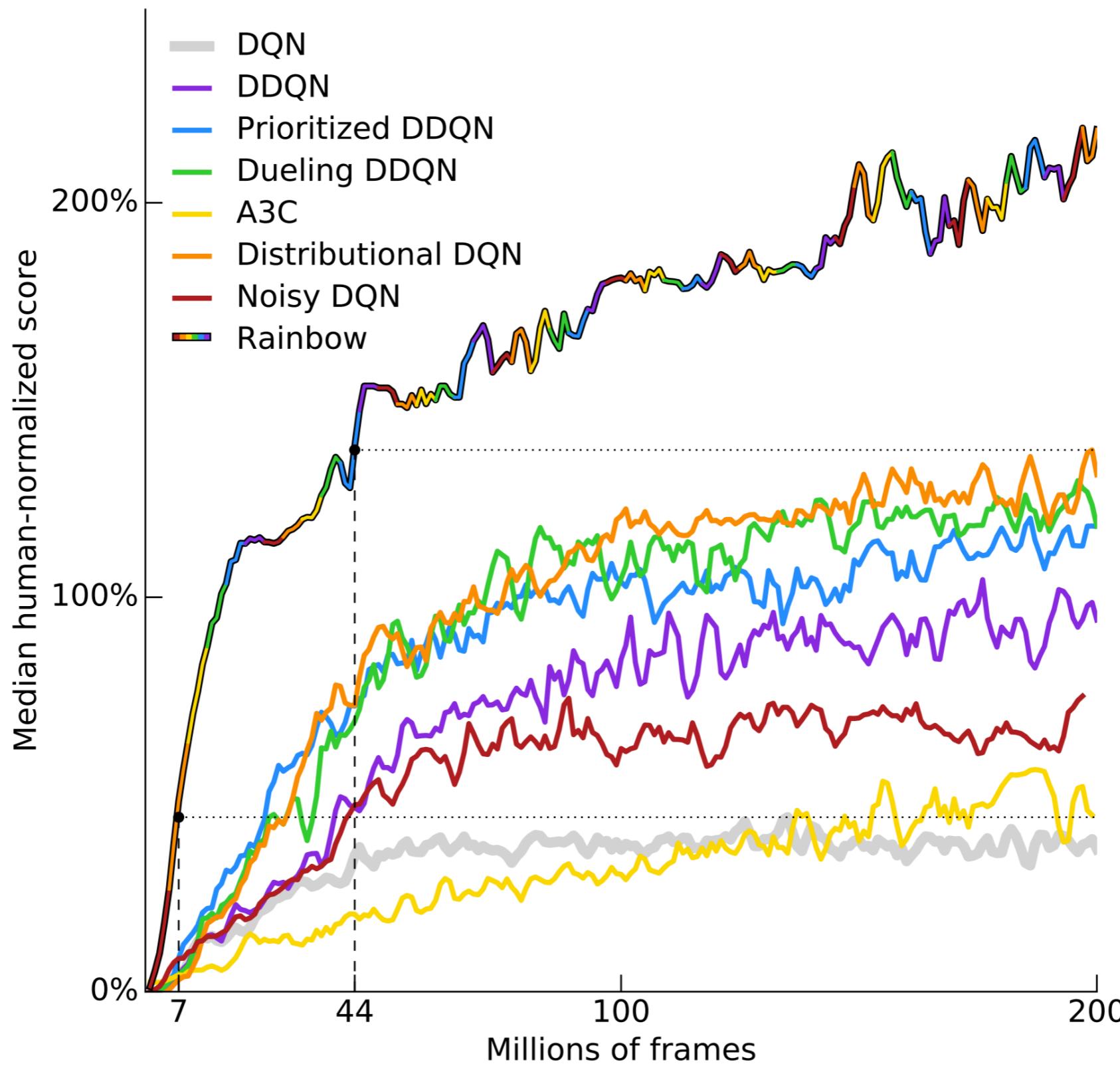
# Prioritized Experience Replay

- ❖ Is uniform sampling from reply buffer an efficient way to learn? Does each sample contribute to the learning equally?
- ❖ The higher the TD error, the greater the degree of update there is to the neural network weights
- ❖ Sampling experiences according to TD error
  - $P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$ , (1)  $p_i = 1/\text{rank}(\delta_i)$ ; (2)  $p_i = |\delta_i| + \epsilon$
  - Biased sampling
    - $w_i = \left( \frac{1}{N} \cdot \frac{1}{P(i)} \right)^\beta$
    - Better than DQN in 41 of 49 Atari 2600 games

# Rainbow



RLChina 2020



# Can DQN work with Continuous Actions?



- ❖ DQN uses experience replay and target network

- Take action  $a_t = \arg \max_{a_t} Q(s, a; \mathbf{w})$  or act randomly
- Store transition  $(s_t, a_t, r_{t+1}, s_{t+1})$  in replay memory  $\mathcal{D}$
- Sample random mini-batch of transitions  $(s, a, r, s')$  from  $\mathcal{D}$
- Optimize MSE between Q-network and Q-learning targets

$$\mathcal{L}(\mathbf{w}) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[ \left( r + \gamma \max_{a'} Q(s', a'; \mathbf{w}^-) - Q(s, a; \mathbf{w}) \right)^2 \right]$$

## How to perform the max?

- Using variant of SGD
- Every  $N$  timesteps,  $\mathbf{w}^- = \mathbf{w}$

# DDPG



RLChina 2020

## ❖ Learning an approximate maximizer

- $\max_a Q(s, a; \phi) = Q(s, \arg \max_a Q(s, a); \phi)$
- Train another network  $\mu(s; \theta)$  such that  $\mu(s; \theta) \approx \arg \max_a Q(s, a; \phi)$
- Solve  $\theta = \arg \max_\theta Q(s, \mu(s; \theta); \phi)$
- By chain rule:  $\frac{dQ_\phi}{d\theta} = \frac{dQ_\phi}{da} \frac{da}{d\theta}$
- New target:  $r + \gamma Q(s', \mu(s'; \theta^-); \phi^-)$

# DDPG



RLChina 2020

❖ DDPG uses experience replay and target networks

- Take action  $a_t = \mu(s, \theta) + \mathcal{N}_t$  (exploration)
- Store transition  $(s_t, a_t, r_{t+1}, s_{t+1})$  in replay memory  $\mathcal{D}$
- Sample random mini-batch of transitions  $(s, a, r, s')$  from  $\mathcal{D}$
- Compute  $\delta = r + \gamma Q(s', \mu(s'; \theta^-); \phi^-) - Q(s, a; \phi)$
- $\Delta\phi \leftarrow \alpha\delta \frac{dQ(s, a; \phi)}{d\phi}$
- $\Delta\theta \leftarrow \beta \frac{dQ(s, a; \phi)}{da} \frac{d\mu(s; \theta)}{d\theta}$
- Soft update  $\phi^-$  and  $\theta^-$



# TD3: Twin Delayed DDPG

- ❖ Clipped double Q learning for overestimation
  - Two Q functions use a single target
  - TD target:  $y = r + \gamma \min_{i=1,2} Q_{\theta_i'}(s', \mu_{\phi'}(s'))$
- ❖ Delayed policy update
  - Value and policy updates are deeply coupled
  - Policy updates less frequently than Q (stable target)
- ❖ Target policy smoothing
  - Deterministic policy can overfit to narrow peaks developed by Q function approximator
  - $a_{\text{clip}}(s') = \mu_{\phi'}(s') + \text{clip}(\mathcal{N}(0, \sigma), -c, +c)$

$$y = r + \gamma \min_{i=1,2} Q_{\theta_i'}(s', a_{\text{clip}})$$



# Practical Tips for Deep Q Networks

- ❖ DQN takes cares to stabilize
  - Test on easy tasks first, make sure your implementation is correct
- ❖ Large replay buffers help improve stability
- ❖ It takes time, be patient - might be no better than random for a while
- ❖ Start with high exploration ( $\epsilon$ ) and gradually reduce, also for learning rate
- ❖ DDQN helps a lot in practice, simple and no downsides
- ❖ Run multiple random seeds, it is very inconsistent between runs



# Convergence of Control Algorithms

Algorithm	Table Lookup	Linear	Non-linear
MC Control	✓	(✓)	✗
SARSA	✓	(✓)	✗
Q-learning	✓	✗	✗

(✓) = chatters around near-optimal value function



# References

- Richard Sutton and Andrew Barto. Reinforcement learning: An introduction. MIT press, 2017.
- David Silver, Reinforcement Learning Course, UCL
- Sergey Levin, Deep Reinforcement Learning Course, UC Berkeley
- Mnih et al., Human-level control through deep reinforcement learning. Nature, 2015, 518(7540): 529-533.
- Van et al., Deep reinforcement learning with double q-learning, AAAI 2016.
- Wang et l., Dueling network architectures for deep reinforcement learning, ICML 2016
- Bellemare et al., A distributional perspective on reinforcement learning, ICML 2017
- Schaul et al., Prioritized experience replay, arXiv:1511.05952, 2015
- Hessel et al., Rainbow: Combining Improvements in Deep Reinforcement Learning, AAAI 2018
- Lillicrap et al., Continuous control with deep reinforcement learning, ICLR 2016
- Fujimoto et al., Addressing function approximation error in actor-critic methods, ICML 2018.