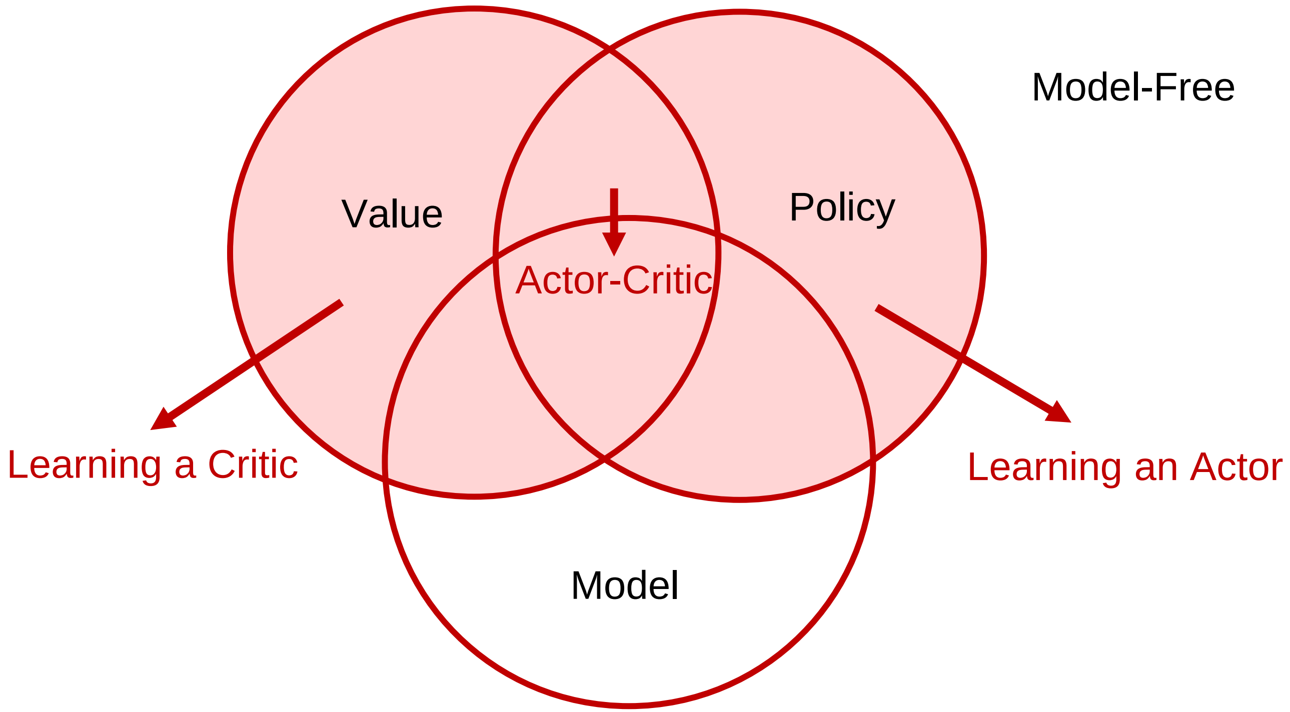
Applied Deep Learning

Value-Based Reinforcement Learning

May 10th, 2021 http://adl.miulab.tw

National Taiwan University
國立臺灣大學

RL Agent Taxonomy



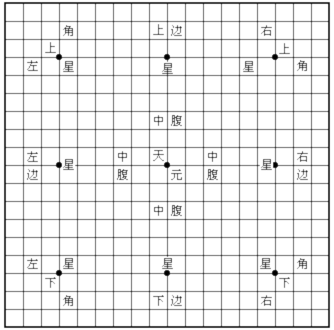
3

Value-Based Approach

Learning a Critic

Value Function

- A value function is a prediction of future reward (with action *a* in state *s*)
- Q-value function gives expected total reward
  - from state s and action a
  - under policy π
  - with discount factor γ



$$Q^\pi(s,a) = \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s, a]$$

- Value functions decompose into a Bellman equation

$$Q^\pi(s,a) = \mathbb{E}_{s',a'}[r + \gamma Q^\pi(s',a') \mid s, a]$$

Optimal Value Function

- An optimal value function is the maximum achievable value

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

- The optimal value function allows us act optimally

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

- The optimal value informally maximizes over all decisions

$$\begin{aligned} Q^*(s,a) &= r_{t+1} + \gamma \max_{a_{t+1}} r_{t+2} + \gamma^2 \max_{a_{t+2}} r_{t+3} + \ldots \\ &= r_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \end{aligned}$$

- Optimal values decompose into a Bellman equation

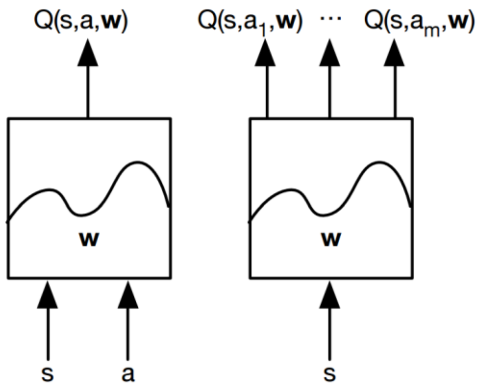
$$Q^*(s,a) = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q^*(s',a') \mid s,a]$$

Value Function Approximation

- Value functions are represented by a *lookup table*

$$Q(s, a) \ \ \forall s, a$$

  - too many states and/or actions to store
  - too slow to learn the value of each entry individually
- Values can be estimated with *function approximation*

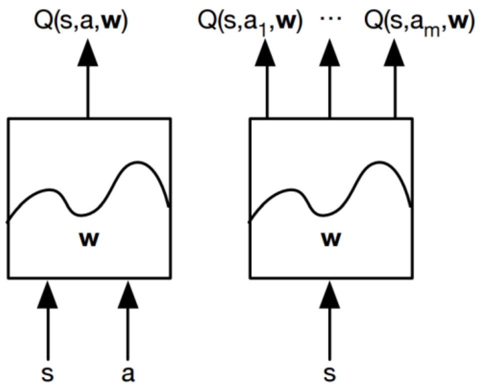


Q-Networks

- Q-networks represent value functions with weights w

$$Q(s, a, w) \approx Q^*(s, a)$$

  - generalize from seen states to unseen states
  - update parameter w for function approximation



Q-Learning

- Goal: estimate optimal Q-values
  - Optimal Q-values obey a Bellman equation

$$Q^*(s,a) = \mathbb{E}_{s'}\boxed{r + \gamma \max_{a'} Q^*(s',a')} \mid s,a]$$

learning target

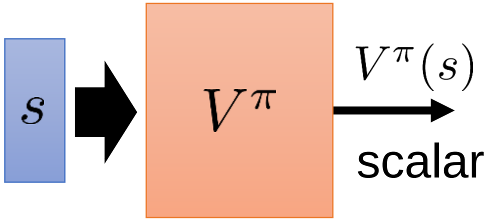
  - *Value iteration* algorithms solve the Bellman equation

$$Q_{i+1}(s,a) = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q_i(s',a') \mid s,a]$$

Critic = Value Function

- Idea: how good the actor is
- State value function: when using actor π, the *expected total reward* after seeing observation (state) *s*

$$V^\pi(s)\ \forall s \quad = \mathbb{E}[G_t \mid s_t = s]$$



larger $V^\pi(s)$

smaller $V^\pi(s)$

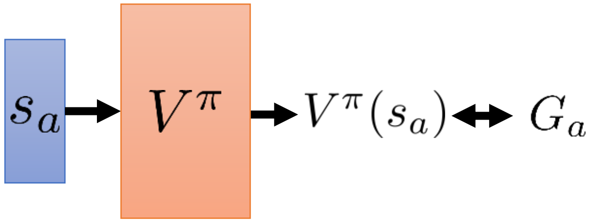
A critic does not determine the action
An actor can be found from a critic

Monte-Carlo for Estimating $V^{\pi}(s)$

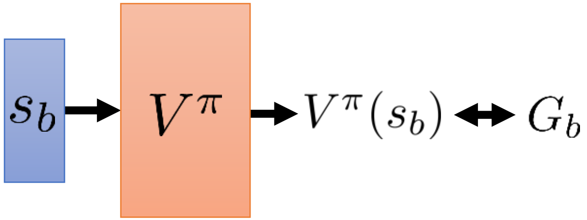
- Monte-Carlo (MC)
  - The critic watches π playing the game
  - MC learns directly from *complete* episodes: no bootstrapping

Idea: value = *empirical mean* return

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

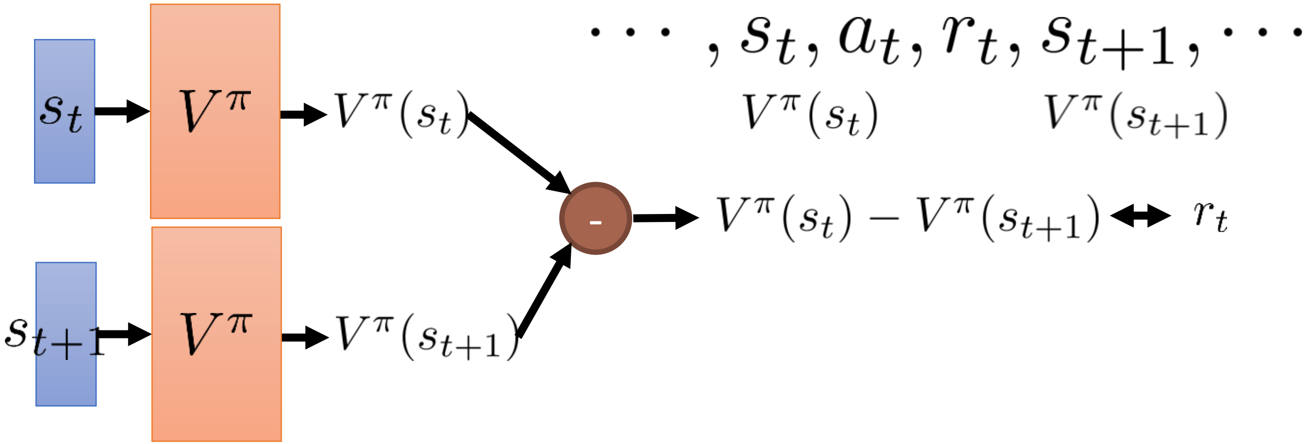


Issue: long episodes delay learning

Temporal-Difference for Estimating $V^\pi(s)$

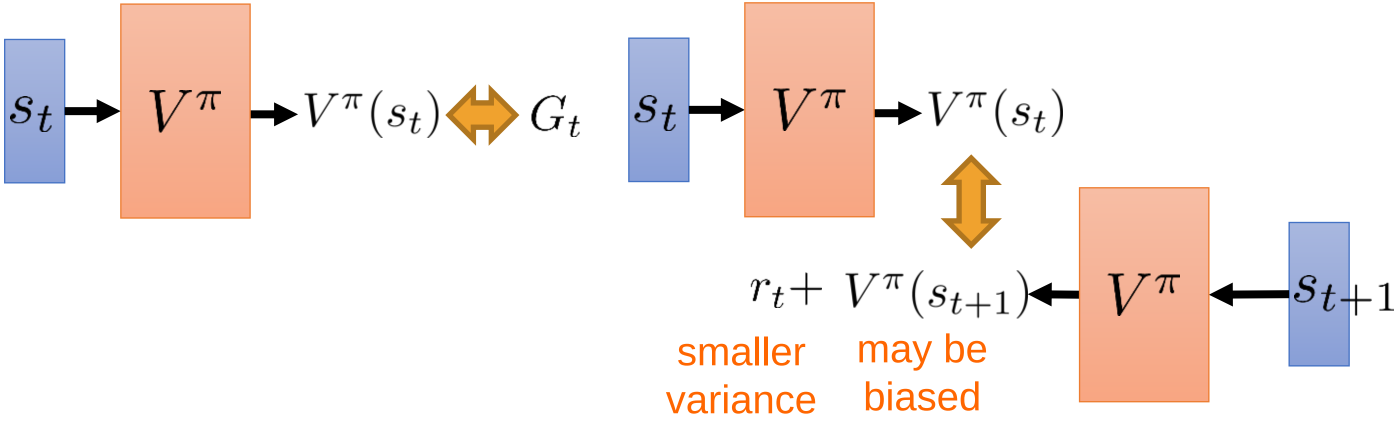
- Temporal-difference (TD)
  - The critic watches π playing the game
  - TD learns directly from *incomplete* episodes by *bootstrapping*
  - TD updates a guess towards a guess

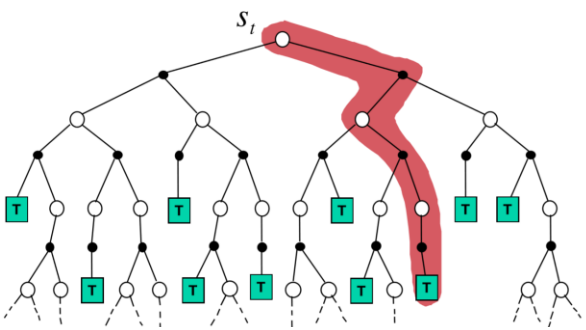
Idea: update value toward *estimated* return



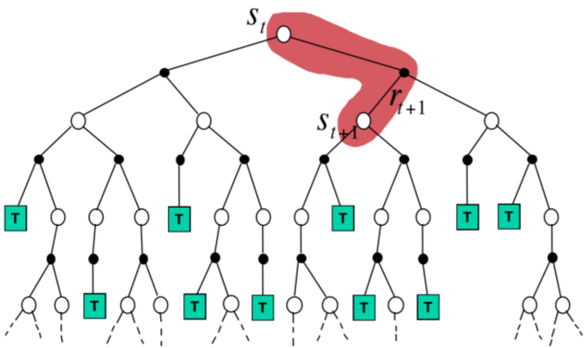
MC v.s. TD

- Monte-Carlo (MC)
  - Large variance
  - Unbiased
  - No Markov property
- Temporal-Difference (TD)
  - Small variance
  - Biased
  - Markov property





$$V'^{\pi}(s_t)$$
$$= V^{\pi}(s_t) + \alpha(G_t - V^{\pi}(s_t))$$



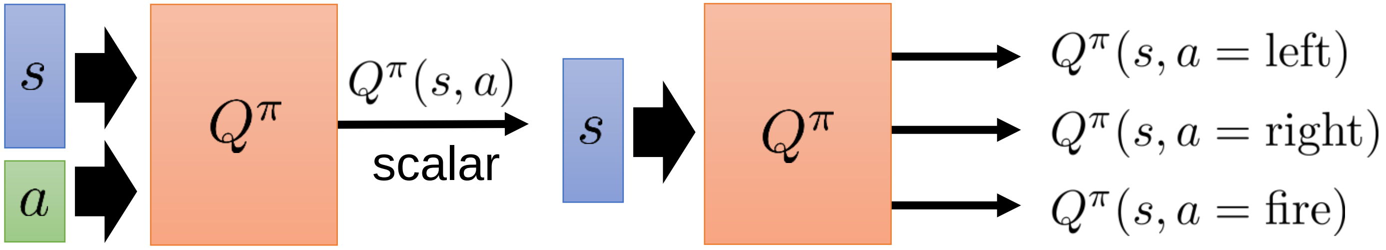
$$V'^{\pi}(s_t)$$
$$= V^{\pi}(s_t) + \alpha(r_{t+1} + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t))$$

MC v.s. TD

Critic = Value Function

- State-action value function: when using actor π, the *expected total reward* after seeing observation (state) s and taking action a

$$Q^{\pi}(s,a)\ \forall s,a = \mathbb{E}[G_t \mid s_t = s, a_t = a]$$



for discrete action only

Q-Learning

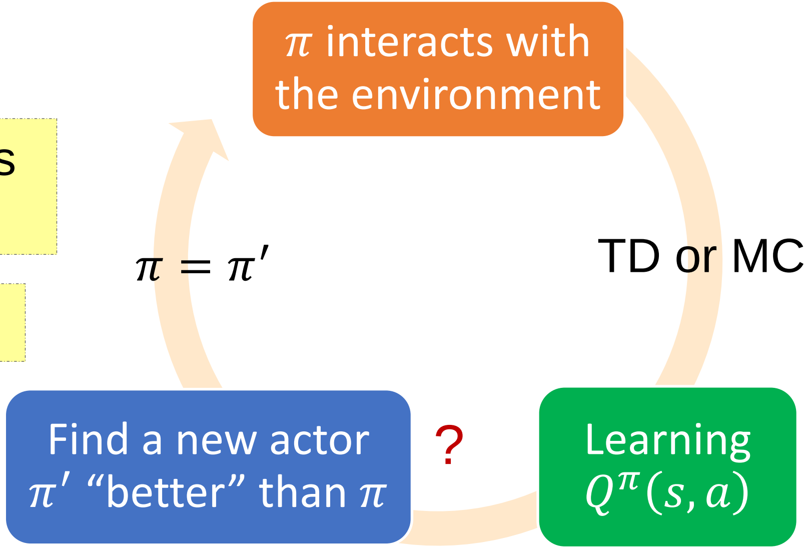
- Given $Q^{\pi}(s,a)$, find a new actor π' "better" than π

$$V^{\pi'}(s) \geq V^{\pi}(s) \ \forall s$$

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

π' does not have extra parameters (depending on value function)

not suitable for continuous action



Q-Learning

- Goal: estimate optimal Q-values
  - Optimal Q-values obey a Bellman equation

$$Q^*(s,a) = \mathbb{E}_{s'}\left[\boxed{r + \gamma \max_{a'} Q^*(s',a')} \mid s,a\right]$$

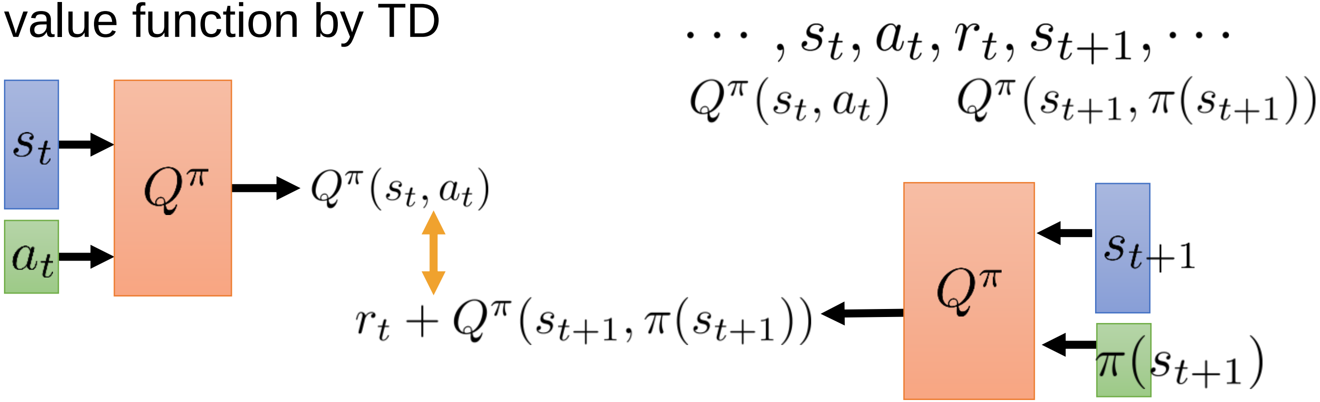
learning target

  - *Value iteration* algorithms solve the Bellman equation

$$Q_{i+1}(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q_i(s',a') \mid s,a\right]$$

Deep Q-Networks (DQN)

- Estimate value function by TD



- Represent value function by deep Q-network with weights w

$$Q(s, a, w) \approx Q^*(s, a)$$

- Objective is to minimize MSE loss by SGD

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

Deep Q-Networks (DQN)

- Objective is to minimize MSE loss by SGD

$$\mathcal{L}(w) = \mathbb{E}\Big[\Big(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w) \Big)^2 \Big]$$

- Leading to the following Q-learning gradient

$$\frac{\partial \mathcal{L}(w)}{\partial w} = \mathbb{E}\Big[\Big(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w) \Big) \frac{\partial Q(s, a, w)}{\partial w} \Big]$$

Issue: naïve Q-learning oscillates or diverges using NN due to:
1) correlations between samples 2) non-stationary targets

Stability Issues with Deep RL

- Naive Q-learning oscillates or diverges with neural nets
  1. Data is sequential
     - Successive samples are correlated, non-iid (independent and identically distributed)
  2. Policy changes rapidly with slight changes to Q-values
     - Policy may oscillate
     - Distribution of data can swing from one extreme to another
  3. Scale of rewards and Q-values is unknown
     - Naive Q-learning gradients can be unstable when backpropagated

Stable Solutions for DQN

- DQN provides a stable solutions to deep value-based RL
  1. Use experience replay
     - Break correlations in data, bring us back to iid setting
     - Learn from all past policies
  2. Freeze target Q-network
     - Avoid oscillation
     - Break correlations between Q-network and target
  3. Clip rewards or normalize network adaptively to sensible range
     - Robust gradients

Stable Solution 1: Experience Replay

- To remove correlations, build a dataset from agent's experience
  - Take action at according to ϵ-greedy policy small prob for exploration
  - Store transition (s_t, a_t, r_t, s_{t+1}) in replay memory *D*
  - Sample random mini-batch of transitions (s, a, r, s') from *D*

s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3 $\rightarrow$ s, a, r, s'
s_3, a_3, r_4, s_4
$\ldots$
$s_t, a_t, r_{t+1}, s_{t+1}$ $\rightarrow$ $s_t, a_t, r_{t+1}, s_{t+1}$

  - Optimize MSE between Q-network and Q-learning targets

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

Exploration

$$s \rightarrow a_1 \quad Q(s, a_1) = 0 \quad \text{never explored}$$

$$s \rightarrow a_2 \quad Q(s, a_2) = \boxed{1} \quad \text{always sampled}$$

$$s \rightarrow a_3 \quad Q(s, a_3) = 0 \quad \text{never explored}$$

- The policy is based on Q-function

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

not good for data collection → inefficient learning

- Exploration algorithms
  - Epsilon greedy

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

ε would decay during learning

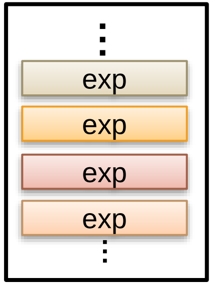
  - Boltzmann sampling

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

Replay Buffer

put the experience into buffer

π interacts with the environment



the experience in the buffer comes from different π

drop the old one if full

$\pi = \pi'$

Find a new actor π' "better" than π

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

Replay Buffer

put the experience into buffer

π interacts with the environment

exp
exp
exp
exp

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

$\pi = \pi'$

Find a new actor π' "better" than π

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

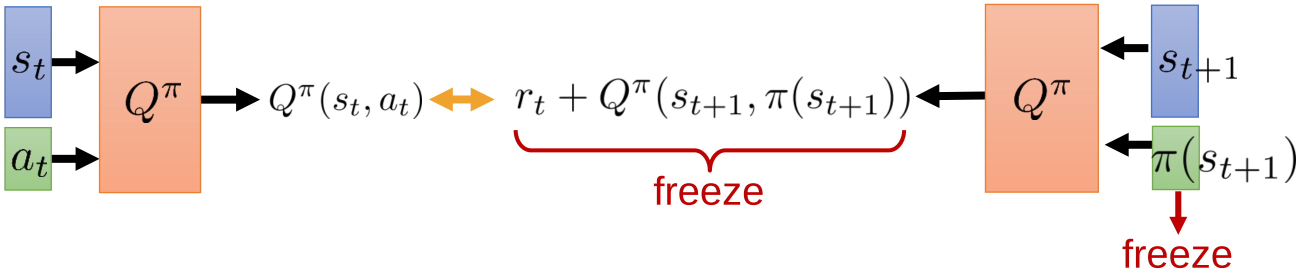
In each iteration:

1. Sample a batch
2. Update Q-function

Off-policy

Stable Solution 2: Fixed Target Q-Network

- To avoid oscillations, fix parameters used in Q-learning target



  - Compute Q-learning targets w.r.t. old, fixed parameters w^-
$$r + \gamma \max_{a'} \hat{Q}(s', a', w^-)$$
  - Optimize MSE between Q-network and Q-learning targets
$$\mathcal{L}(w) = \mathbb{E}_{s,a,r,s' \sim D}\left[\left(r + \gamma \max_{a'} \hat{Q}(s', a', w^-) - Q(s, a, w)\right)^2\right]$$
  - Periodically update fixed parameters $w^- \leftarrow w$

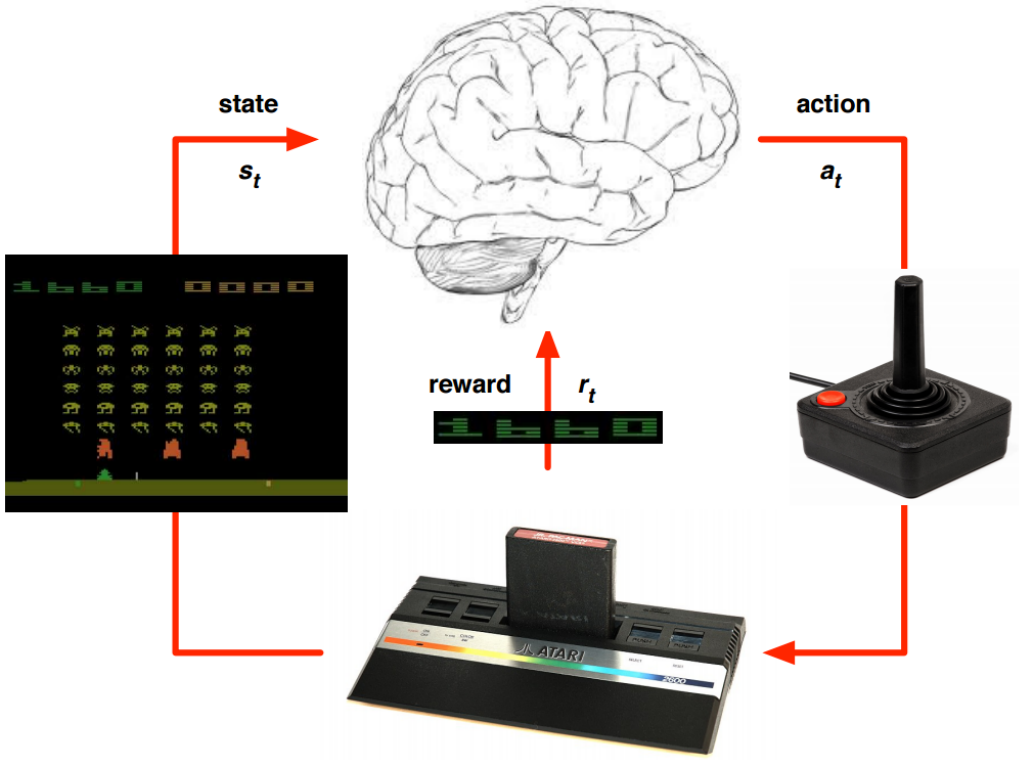
Stable Solution 3: Reward / Value Range

- To avoid oscillations, control the reward / value range
  - DQN clips the rewards to $[-1, +1]$
    - Prevents too large Q-values
    - Ensures gradients are well-conditioned

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)
    - Update the parameters of Q to make $Q(s_i, a_i) \approx r_i + \max_a \hat{Q}(s_{i+1}, a)$
    - Every C steps reset $\hat{Q} = Q$

Deep RL in Atari Games

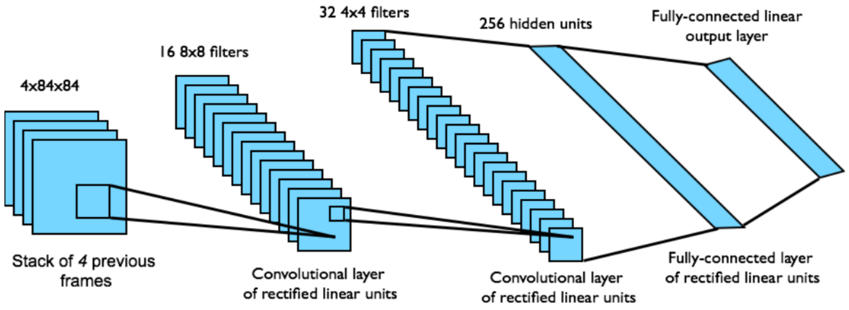


DQN in Atari

- Goal: end-to-end learning of values *Q*(*s*, *a*) from pixels

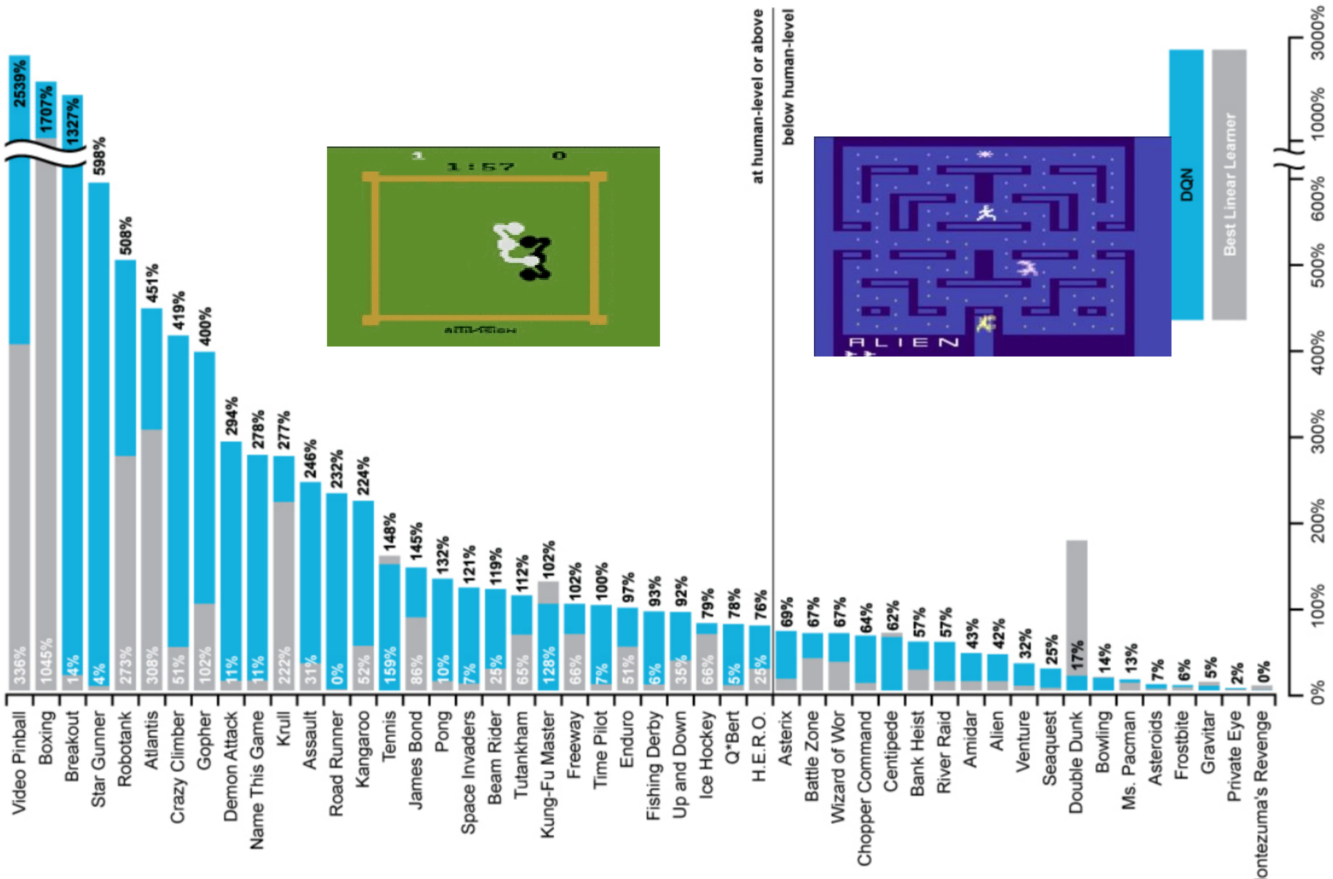
$$\mathcal{L}(w) = \mathbb{E}_{s,a,r,s'\sim D}\left[\left(r + \gamma \max_{a'} Q(s', a', w^-) - Q(s, a, w)\right)^2\right]$$

  - Input: state is stack of raw pixels from last 4 frames
  - Output: *Q*(*s*, *a*) for all joystick/button positions *a*
  - Reward is the score change for that step



DQN Nature Paper [link] [code]

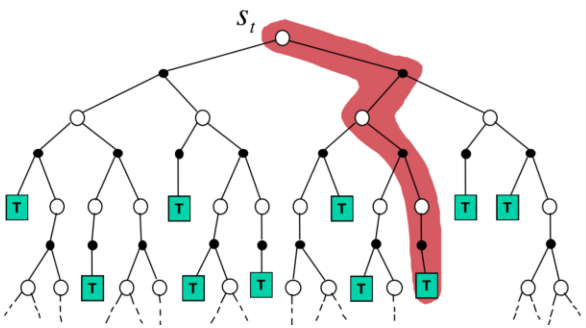
DQN in Atari

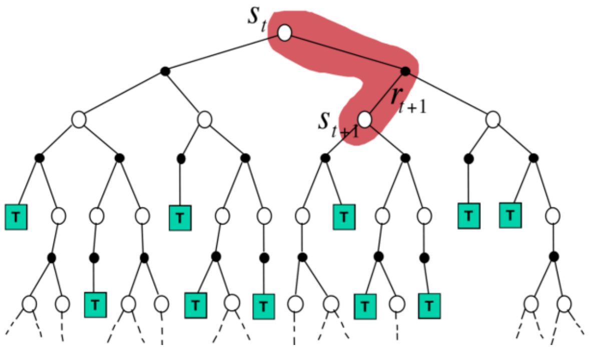


DQN Nature Paper [link] [code]

Concluding Remarks

- RL is a general purpose framework for **decision making** under interactions between agent and environment
- A value-based RL measures how good each state and/or action is via a value function
  - Monte-Carlo (MC) v.s. Temporal-Difference (TD)





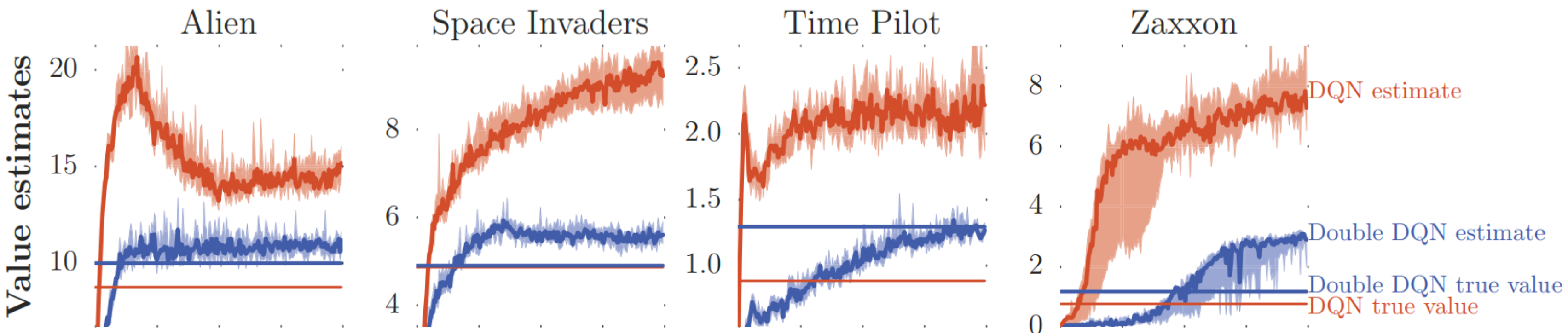
32

Advanced DQN

DQN 進階模型

Double DQN

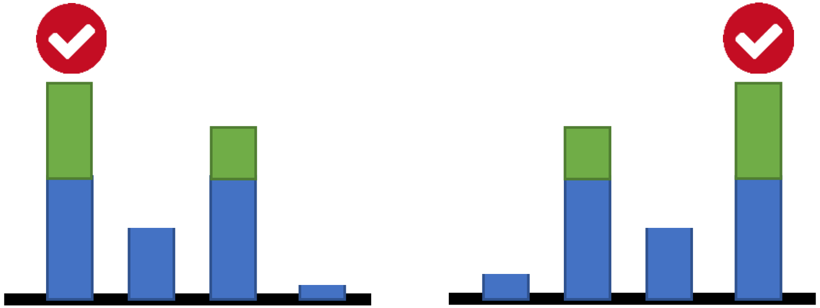
- Q value is usually over-estimated



Double DQN

- Nature DQN

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



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

Issue: tend to select the action that is over-estimated

Double DQN

- Nature DQN

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

- Double DQN: remove upward bias caused by $\max_{a} Q(s,a,w)$

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

  - Current Q-network w is used to select actions
  - Older Q-network w^- is used to evaluate actions

If Q over-estimate a, so it is selected. $\hat{Q}$ would give it proper value.
How about $\hat{Q}$ overestimate? The action will not be selected by Q.

Dueling DQN

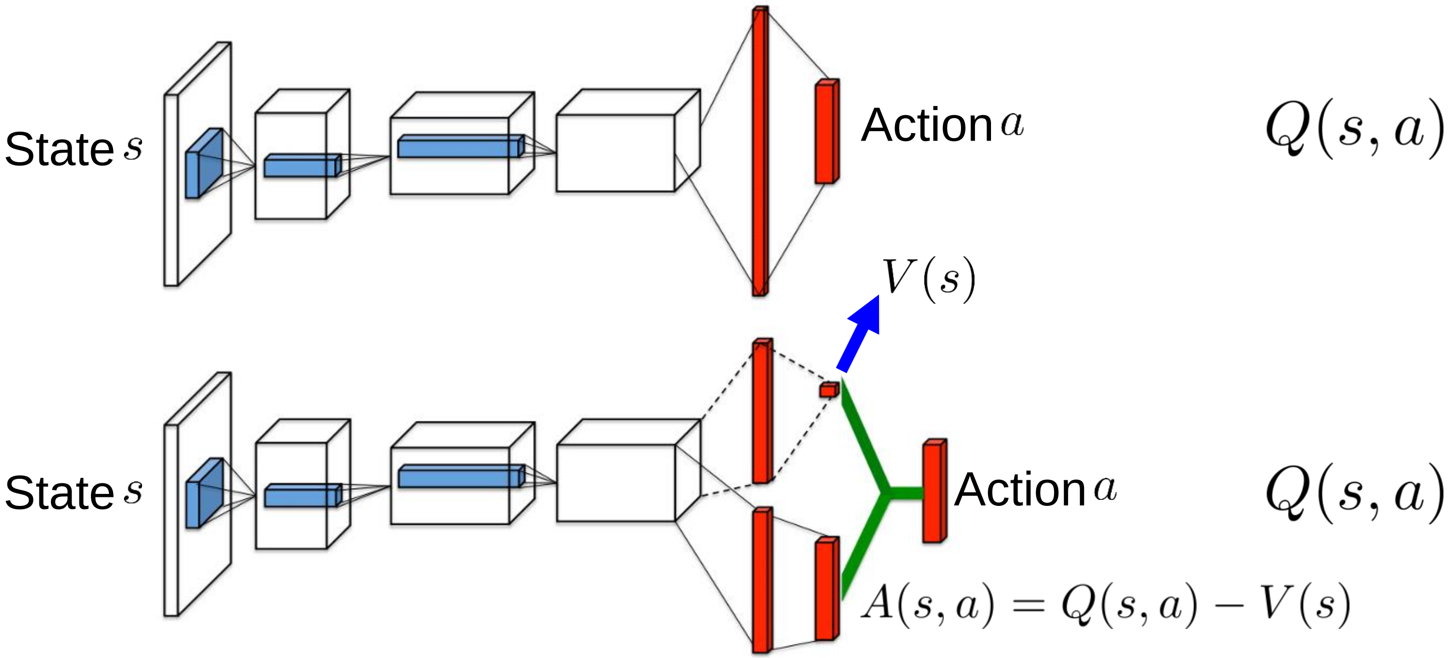
- Dueling Network: split Q-network into two channels

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

  - Action-independent value function
    - Value function estimates how good the state is $V(s)$
  - Action-dependent advantage function
    - Advantage function estimates the additional benefit $A(s,a)$

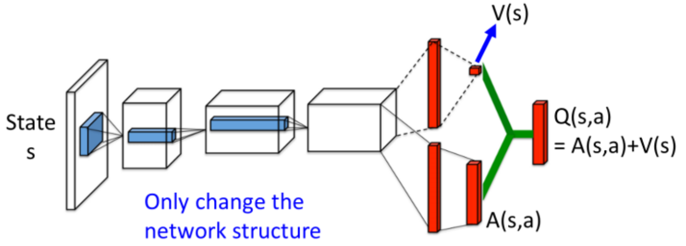
Wang et al., “Dueling Network Architectures for Deep Reinforcement Learning”, arXiv preprint, 2015.

Dueling DQN



Wang et al., "Dueling Network Architectures for Deep Reinforcement Learning", arXiv preprint, 2015.

Dueling DQN



$$Q(s,a)$$

state

action

3	~~3~~ **4**	3	1
1	~~-1~~ **0**	6	1
2	~~-2~~ **-1**	3	1

=

$$V(s)$$ average of column

2	~~0~~ **1**	4	1

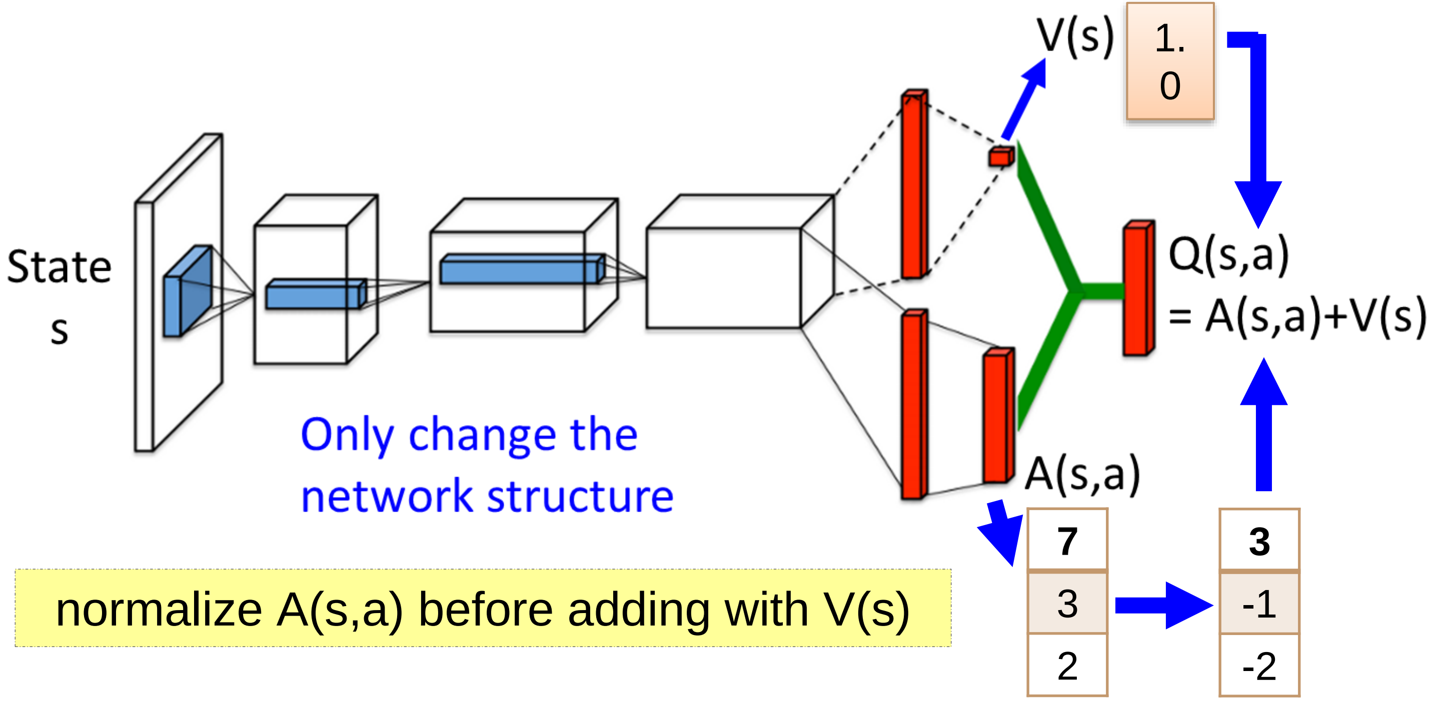
+

$$A(s,a)$$ sum of column = 0

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

Wang et al., “Dueling Network Architectures for Deep Reinforcement Learning”, arXiv preprint, 2015.

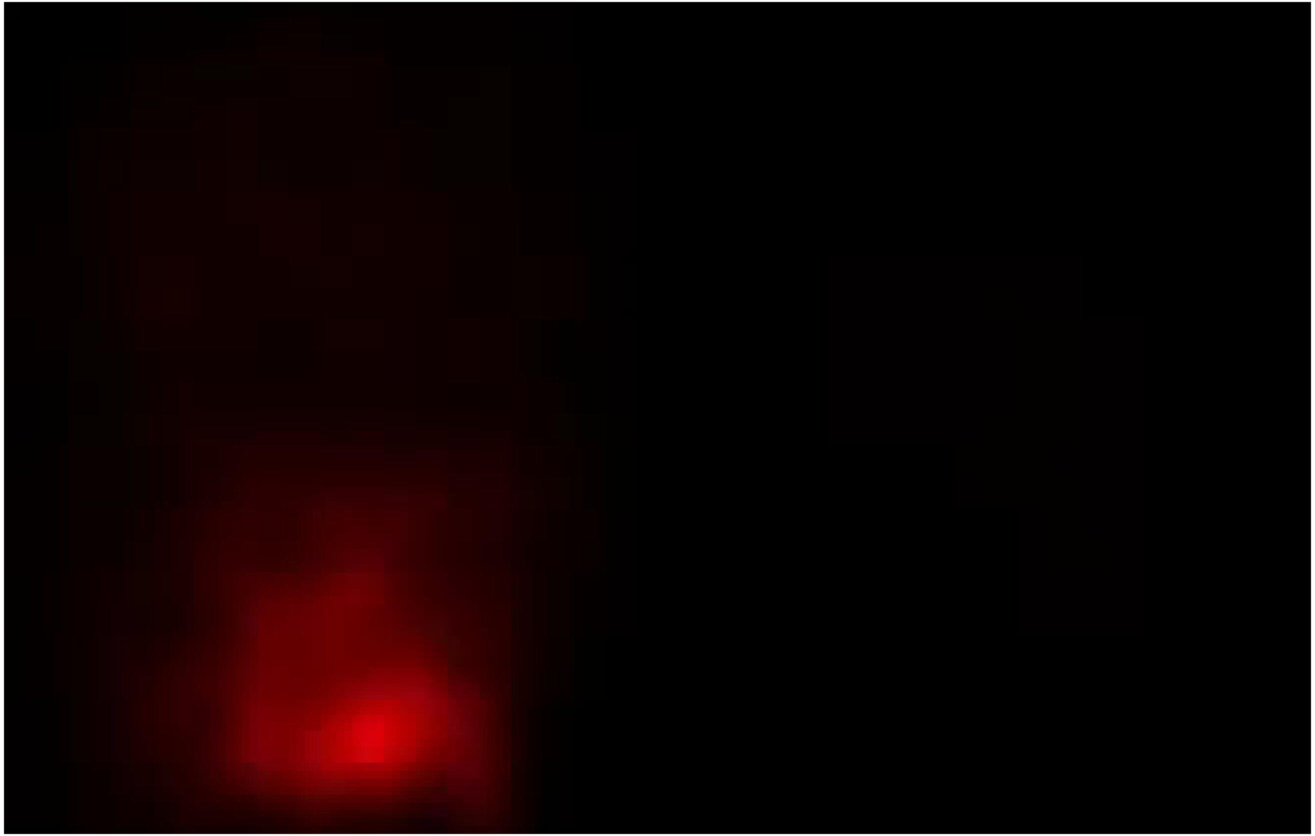
Dueling DQN



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

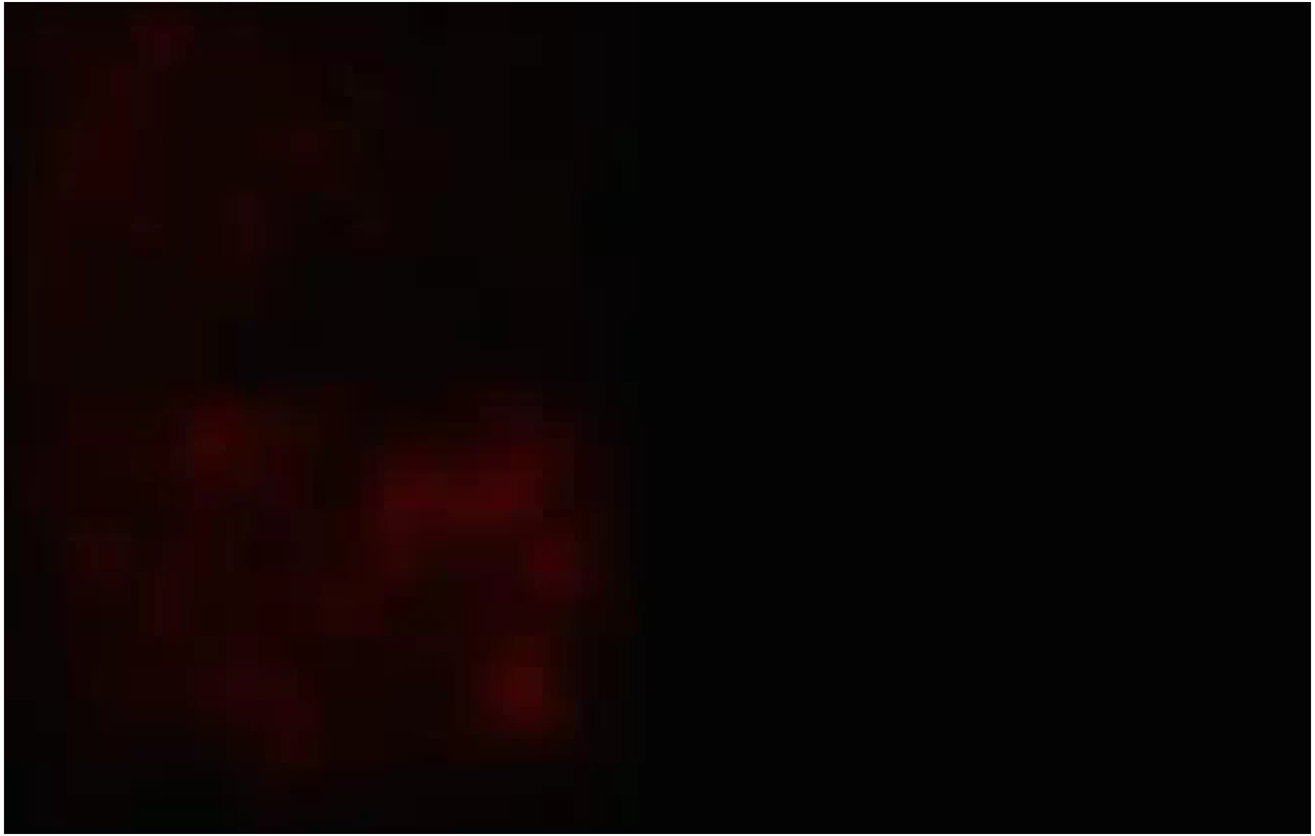
Wang et al., "Dueling Network Architectures for Deep Reinforcement Learning", arXiv preprint, 2015.

Dueling DQN - Visualization



Wang et al., “Dueling Network Architectures for Deep Reinforcement Learning”, arXiv preprint, 2015.

Dueling DQN - Visualization

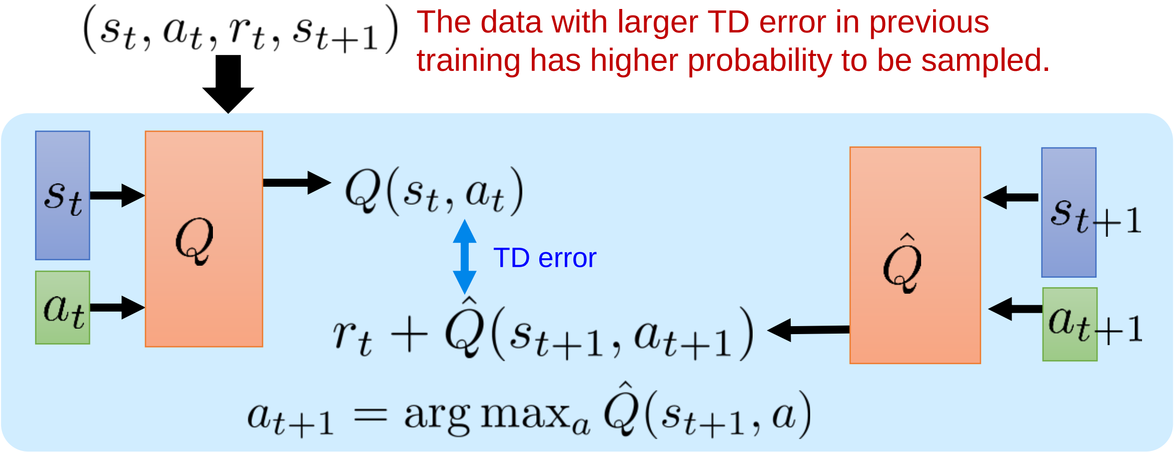


Wang et al., “Dueling Network Architectures for Deep Reinforcement Learning”, arXiv preprint, 2015.

Prioritized Replay

- Prioritized Replay: weight experience based on *surprise*
  - Store experience in priority queue according to the error

$$\left| r + \gamma \max_{a'} Q(s', a', w^-) - Q(s, a, w) \right|$$



Parameter update procedure is also modified.

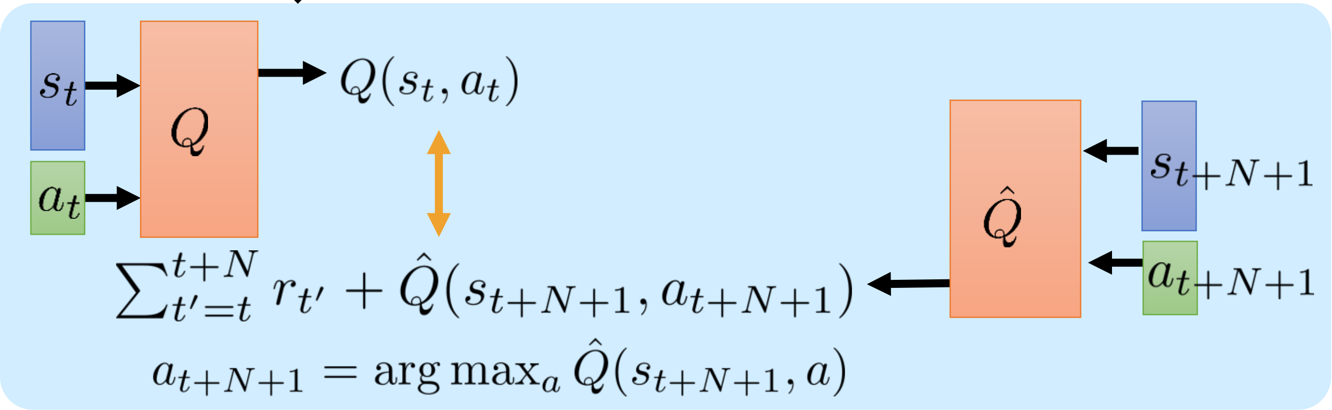
Schaul et al., “Prioritized Experience Replay”, arXiv preprint, 2015.

Multi-Step

- Idea: balance between MC and TD

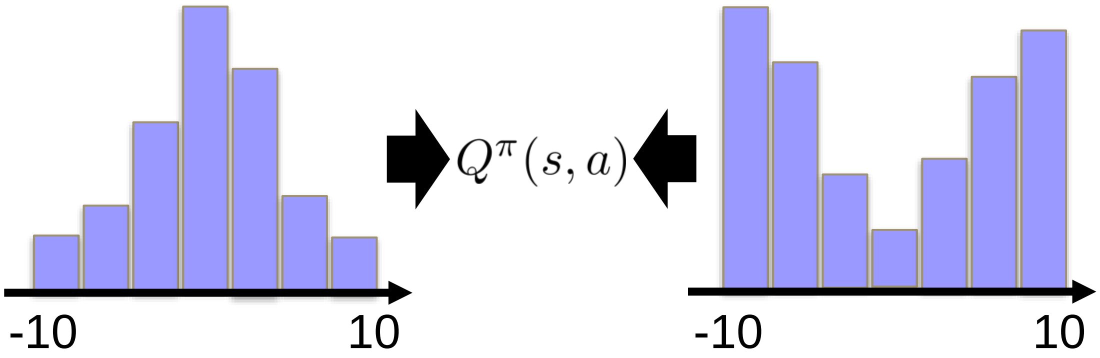
$$(s_t, a_t, r_t, \cdots, s_{t+N}, a_{t+N}, r_{t+N}, s_{t+N+1})$$

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



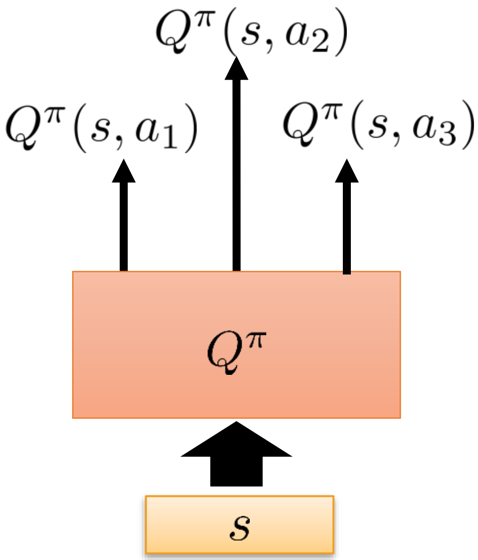
Distributional Q-function

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

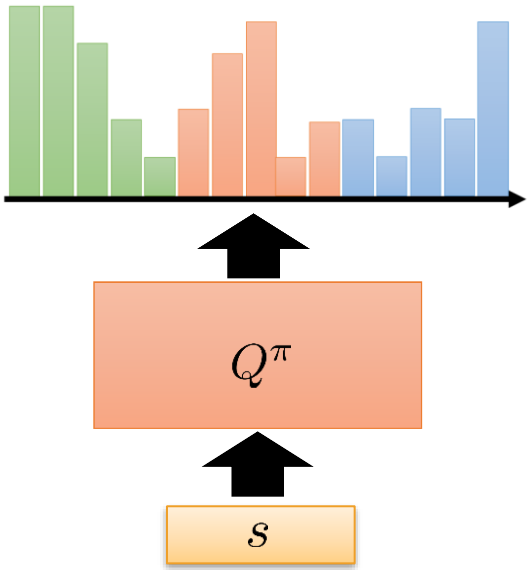


Different distributions can have the same values.

Distributional Q-function

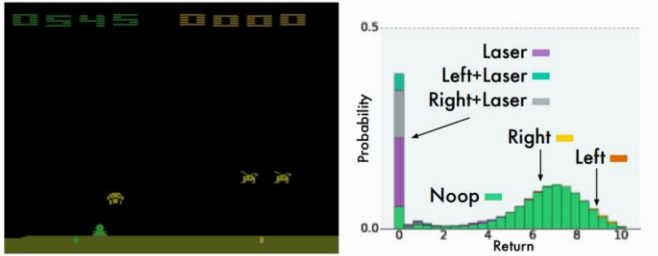


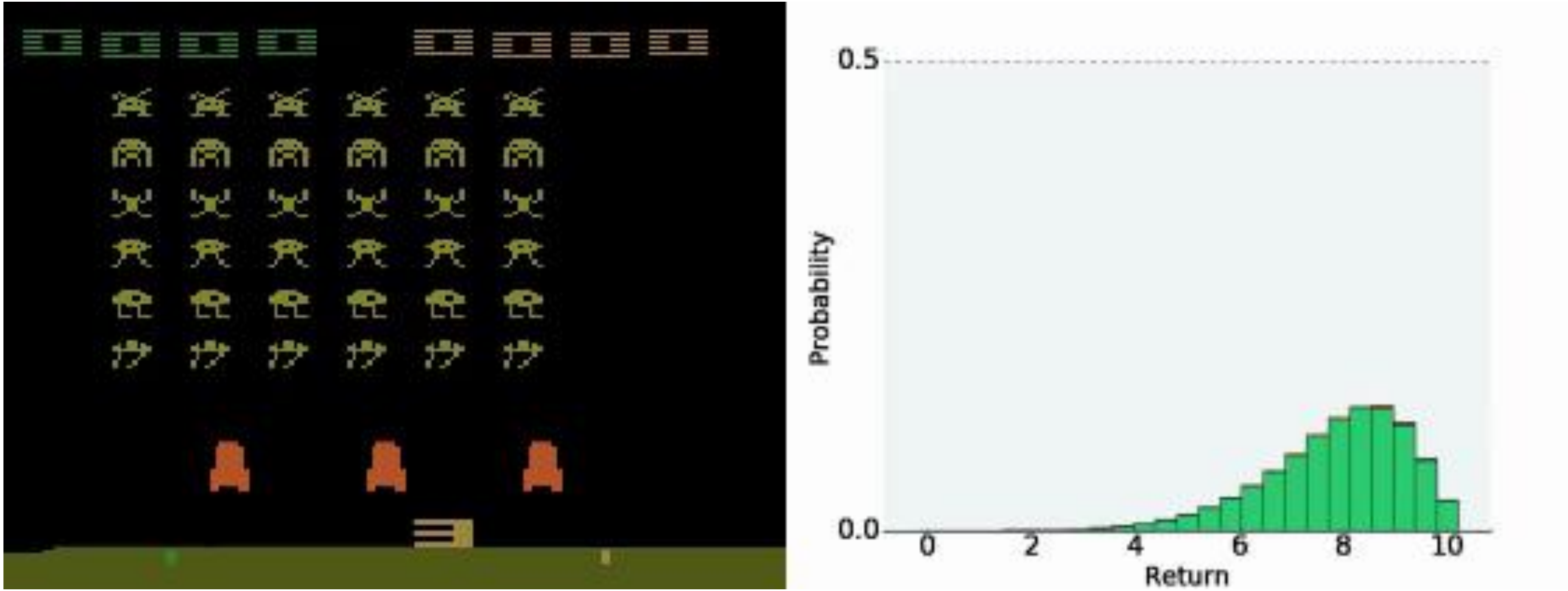
A network with 3 outputs



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

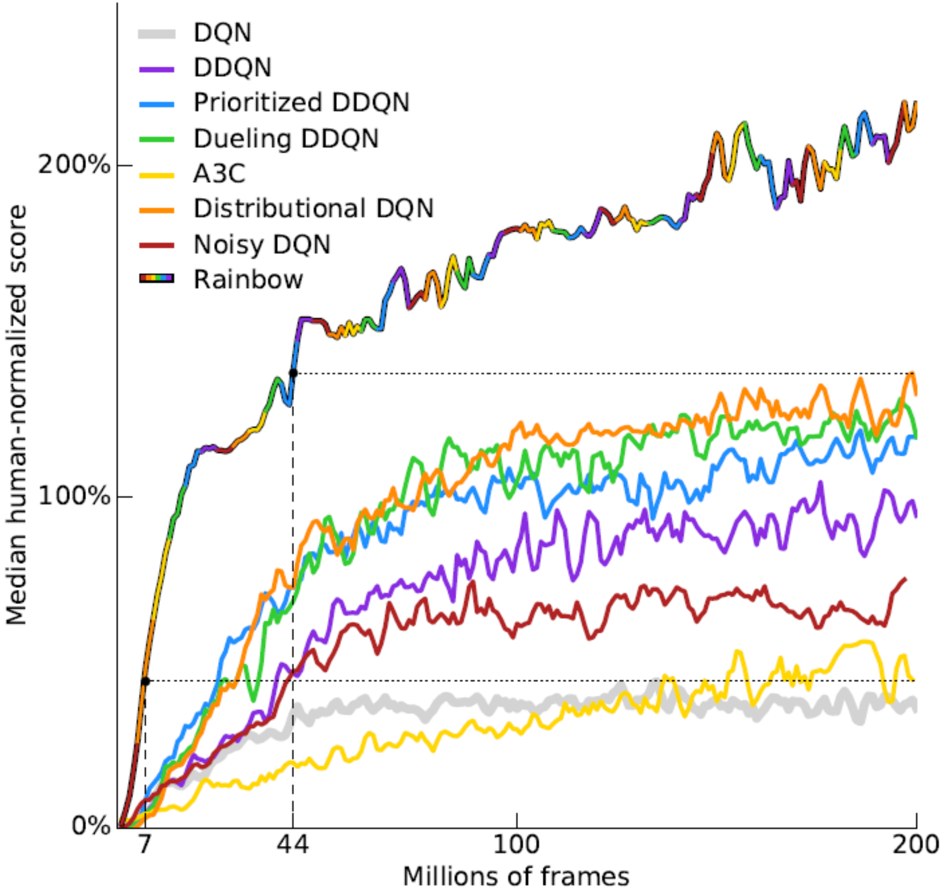
Demo





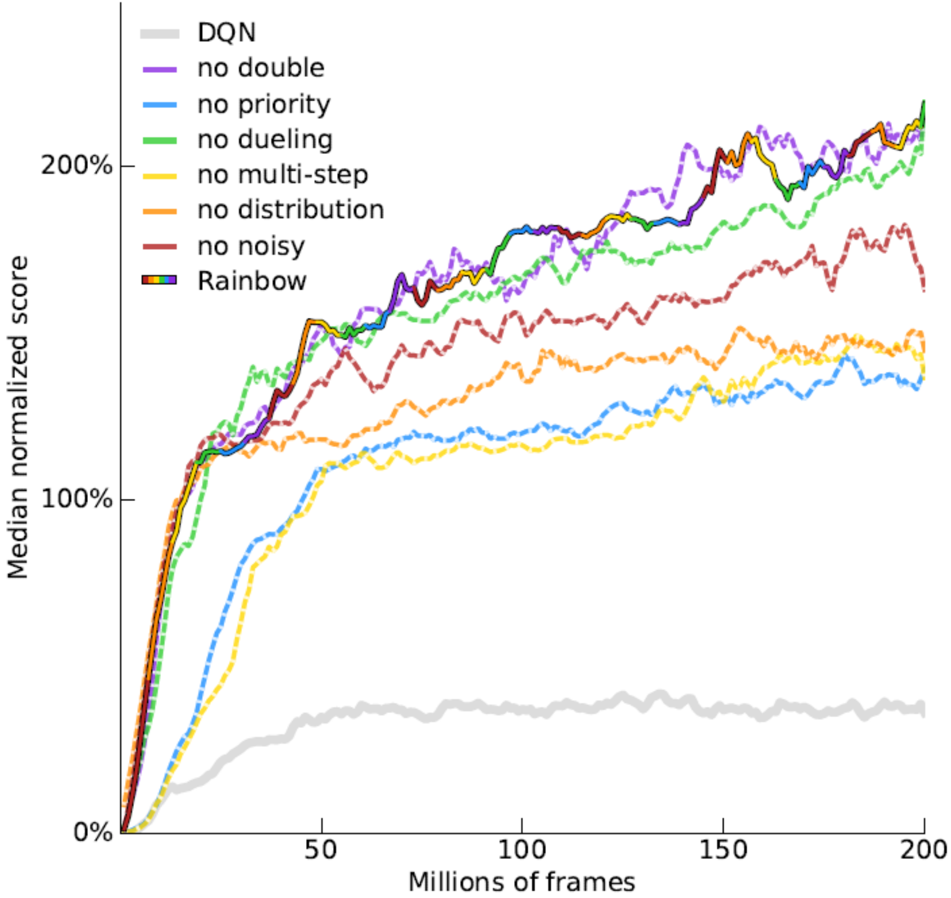
https://youtu.be/yFBwyPuO2Vg

Rainbow



Hessel et al., “Rainbow: Combining Improvements in Deep Reinforcement Learning”, arXiv preprint, 2017.

Rainbow



Hessel et al., "Rainbow: Combining Improvements in Deep Reinforcement Learning", arXiv preprint, 2017.

Concluding Remarks

- DQN training tips
  - Double DQN
  - Dueling DQN
  - Prioritized replay
  - Multi-step
  - Distributional DQN