

# L16: Advanced Off-Policy RL

**Hao Su**

(slides prepared with the help from Zhan Ling)

**Spring, 2021**

*Contents are based on website*

<b>H</b>	Show this help
<b>Left &amp; Right</b>	Previous & Next step
<b>P</b>	Presenter console
<b>F5 / ESC</b>	Fullscreen: Enter / Exit

# Agenda

- Key Ideas of Off-policy RL
- Tricks to Overcome Value Estimation
- Tricks to Address Rare Beneficial Samples
- Tricks to Accelerate Reward Propagation
- Tricks in Value Network Architecture Design
- Tricks by Considering Uncertainty of Value Estimation
- Tricks in Leveraging State-conditioned Exploration Noise
- Off-Policy RL Frameworks in Practice

click to jump to the section.

<b>H</b>	Show this help
<b>Left &amp; Right</b>	Previous & Next step
<b>P</b>	Presenter console
<b>F5 / ESC</b>	Fullscreen: Enter / Exit

## **Key Ideas of Off-policy RL**

# Off-Policy RL

Key ideas:

- Use a replay buffer to store samples that might be collected from long before.
- Build a value network approximator  $Q_\theta(s, a)$  and learn  $\theta$  by minimizing the TD loss with samples from the replay buffer.
- Had  $Q_\theta$  been learned, policy  $\pi(s)$  can be considered as solving an optimization problem over the value network:

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

- For discrete action space, the optimum  $a$  can be computed by enumerating all possible actions (recall the DQN algorithm).

## Recall: Q-Learning for Tabular RL

1. Given transitions  $\{(s, a, s', r)\}$  from some trajectories, how to improve the current Q-function?
  - By Temporal Difference learning, the update target for  $Q(S, A)$  is
    - $R + \gamma \max_a Q(S', a)$
  - Take a small step towards the target
    - $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
2. Given  $Q$ , how to improve policy?
  - Take the greedy policy based on the current  $Q$ 
    - $\pi(s) = \operatorname{argmax}_a Q(s, a)$
3. Given  $\pi$ , how to generate trajectories?
  - $\epsilon$ -greedy policy in the environment.

# Continuous Q-Learning

- The major challenge for continuous Q-learning is in how we compute  $\max_a Q_\theta(s, a)$ .
- For discrete action space, the optimum  $a$  can be computed by enumerating all possible actions (recall the DQN algorithm).
- However, for continuous action space, computing  $\max_a Q_\theta(s, a)$  cannot be achieved by enumeration.
- *Q: How to do Q-learning if the action space is continuous?*

# Continuous Q-Learning

- The most straight-forward idea: Solving an optimization problem to compute  $a$ :

$$\underset{a}{\text{maximize}} \quad Q_{\theta}(s, a)$$

- Limitations:
  - Very slow!
  - May get stuck in local minima.

## Continuous Deterministic Policy Network

- If optimizing  $a$  every time is too slow, let us use a neural network to memorize the decisions in the past!
- We consider a simple policy family deterministic policies ( $Q$ : *Why this is reasonable here, for off-policy RL?*).
- Deterministic policies can be approximated as a network  $\pi_\phi : \mathcal{S} \mapsto \mathbb{R}$ .
- Suppose that we have a learned value network  $Q_\theta(s, a)$ . Then, the optimal policy can be obtained by "fine-tuning" our policy network:

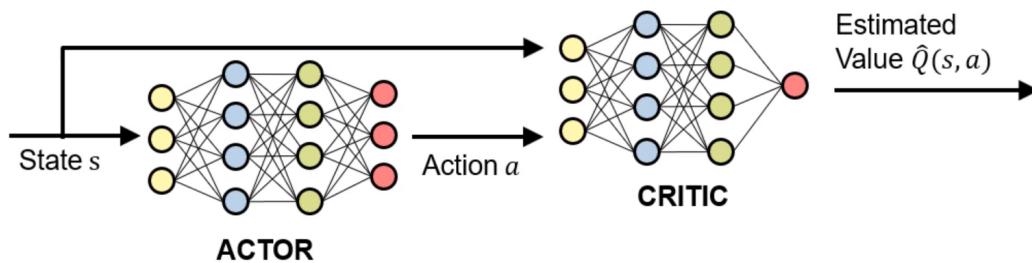
$$\underset{\phi}{\text{maximize}} \quad Q_\theta(s, \pi_\phi(s))$$

- $Q$ : *What are the potential benefit of using a policy network other than being faster?*

## TD-based Q Function Learning

We can still use TD-loss to learn  $Q_\theta$ . Given a transition sample  $(s, a, s', r)$ :

- In tabular Q-learning, the update target for  $Q(s, a)$  is  $r + \gamma \max_a Q(s', a)$ ;
- In continuous deterministic Q-learning, the update target becomes  $r + \gamma Q(s', \pi_\phi(s'))$ .
- In literature, the policy network is also referred as "actors" and the value network referred as "critics"



## Have We Finished? Revisit the Three Questions

1. Given transitions  $\{(s, a, s', r)\}$  from some trajectories, how to improve the current Q-function?
  - We have derived the update target for  $Q(S, A) = r + \gamma Q(s', \pi_\phi(s'))$ .
2. Given  $Q$ , how to improve policy?
  - We introduced a policy network  $\pi_\phi$  and update it by solving

$$\underset{\phi}{\text{maximize}} \quad Q_\theta(s, \pi_\phi(s))$$

3. Given  $\pi$ , how to generate trajectories?
  - We also need exploration in continuous action space!

## Exploration in Continuous Action Space

- For discrete action space, we can use  $\epsilon$ -greedy.
- For continuous action space, we can add small perturbations to the actions output by the policy network to do exploration.
- Example:
  - Gaussian noise:  $\mathcal{N}(0, \sigma^2)$ ;
  - Ornstein-Uhlenbeck random process (temporally correlated random noises).

# Deep Deterministic Policy Gradient (DDPG)

- Initialize replay buffer  $D$ , Q network  $Q_\theta$  and  $\pi_\phi$ .
- For every episode:
  - Sample the initial state  $s_0 \sim P(s_0)$
  - Repeat until the episode is over
    - Let  $s$  be the current state
    - **select  $a = \pi_\phi(s) +$  random noise**
    - Execute  $a$  in the environment, and receive the reward  $r$  and the next state  $s'$
    - Add transitions  $(s, a, s')$  in  $D$
    - Sample a random batch from  $D$  and build the batch TD loss
    - Perform one or a few gradient descent steps on the TD loss
    - **Perform one or a few gradient descent steps on the policy loss**

*Continuous control with deep reinforcement learning*

# Trouble and Tricks in Practice

- Troubles in practice:
  - Value overestimation
  - Rare beneficial samples in replay buffer
  - Slow reward propagation
- Tricks in practice:
  - Dueling Network
  - Distributional action-value network
  - State-conditional exploration noise

## **Tricks to Overcome Value Estimation**

## Issue: Value Overestimation

- Value network approximates the value. Policy networks are optimized over value networks. If  $Q$  of some sub-optimal action  $a'$  is overestimated and larger than the optimal one, policy network tends to choose suboptimal actions and hurts the performance.
- Theoretical intuition:
  - Target of TD update is  $Q_\theta^{target}(s) = r(s, a, s') + \gamma \max_{a'} Q_\theta(s', a')$
  - Assume  $Q_\theta(s, a) = Q_{gt}(s, a) + \epsilon$  with  $\mathbb{E}[\epsilon] = 0$ , where  $Q_{gt}$  is the ground-truth  $Q$  function
  - $\mathbb{E}_\epsilon[\max_{a'} Q_\theta(s', a')] = \mathbb{E}_\epsilon[\max_{a'} Q_{gt}(s, a) + \epsilon] \geq \max_{a'} \mathbb{E}_\epsilon[Q_{gt}(s', a') + \epsilon] = \max_{a'} Q_{gt}(s', a')$
  - In practice,  $\mathbb{E}_\epsilon[\max_{a'} Q_\theta(s', a')] > \max_{a'} Q_{gt}(s', a')$ , which make target of TD update also overestimated.

*Issues in using function approximation for reinforcement learning*

# Double Q-Learning

- The max operator in Q-learning uses the same values both to select and to evaluate an action. This makes it more likely to select overestimated values, resulting in overoptimistic value estimates.
- Solution: Decouple the selection from the evaluation with two value functions  $Q_{\theta_i}(s, a), i = 0, 1$ .
- Discrete action space:
  - Randomly choose one of the  $Q$  functions to collect samples from the environment.
  - The TD loss of  $Q_{\theta_i}$  is

$$TD_{\theta_i}(s, a) = \|Q_{\theta_i}(s, a) - [r(s, a, s') + \gamma Q_{\theta'_{1-i}}(s', \text{argmax}_{a'} Q_{\theta_i}(s', a'))]\|^2.$$

- Continuous action space (DDPG-based): keep  $(Q_{\theta_0}, \pi_{\theta_0})$  and  $(Q_{\theta_1}, \pi_{\theta_1})$ 
  - Randomly choose one of the  $\pi$  functions to collect samples from the environment.
  - The target of  $Q_{\theta_i}$  is randomly chosen from

$$r(s, a, s') + \gamma Q_{\theta'_{1-k}}(s', \pi_{\theta_k}(s')).$$

- In theory and practice, double Q-Learning has advantages over Q-Learning in some cases.

*Double Q-learning  
Deep Reinforcement Learning with Double Q-learning*

## Clipped Double Q-Learning

- Double Q-Learning may still overestimate values. Clipped double Q-Learning (for continuous action case) claims to further suppress value overestimation by

$$TD_{\theta_i}(s, a) = \|Q_{\theta_i}(s, a) - [r(s, a, s') + \gamma \min_j Q_{\theta_j}(s', \pi_\phi(s'))]\|^2$$

- With clipped double Q-learning, the value target is smaller than that with a single value function.
- This update rule may induce an *underestimation bias*. The authors claim that this is far preferable to overestimation bias, as unlike overestimated actions, the value of underestimated actions will not be explicitly propagated through the policy update
- In practice, two value functions are enough. Using more cannot provide extra benefits.
- Note that there is only one policy network. It is a multi-head structure. This architecture has been used by multiple successful continuous Q-learning frameworks.

*Addressing function approximation error in actor-critic methods*

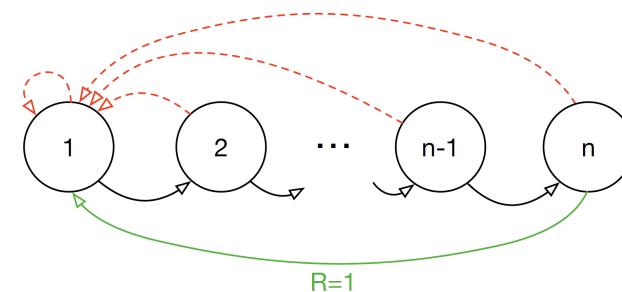
## **Tricks to Address Rare Beneficial Samples**

## Issue: Rare Beneficial Samples in the Replay Buffer

- In environment that suffers from hard exploration, reward signals are rare in replay buffer. It is not efficient to uniformly choice training samples from the buffer.
- Example: Montezuma's revenge, Blind Cliffwalk.

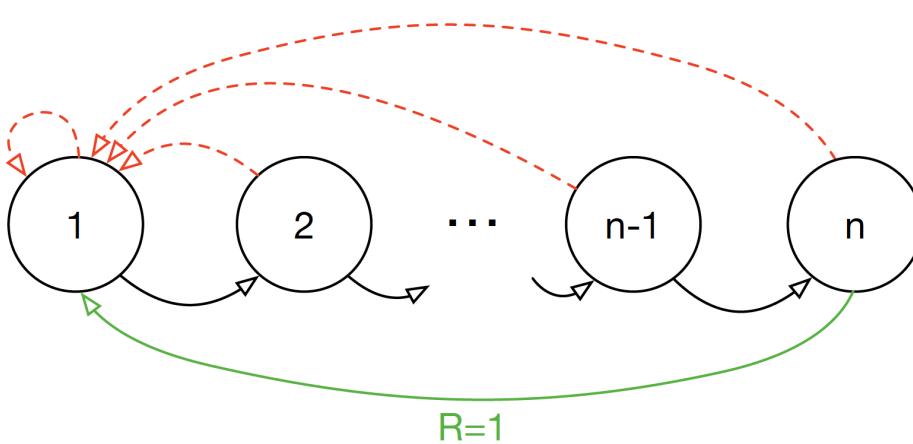


Prioritized experience replay



# Blind Cliffwalk

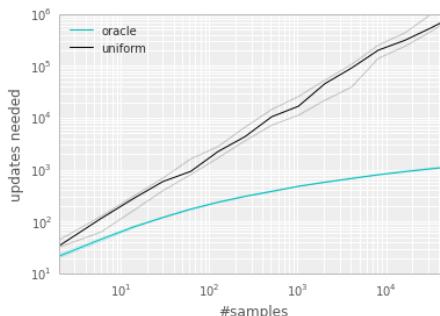
- Two actions at each state: right and **wrong**.
- Episode is terminated whenever the agent takes the **wrong** action.
- Agent will get reward 1 after taking  $n$  right actions.



Prioritized experience replay

# Analysis with Q-Learning

- In expectation, they are  $\mathcal{O}(2^n)$  transitions with reward 0 in replay buffer before the agent get the first reward 1.
- With uniform sampling, the agent learn from successful transitions with probability  $\mathcal{O}(2^{-n})$  in the beginning stage and this will dramatically slow down the reward propagation.
- Compare performance between uniform sampling and oracle sampling. Oracle greedily selects the transition that maximally reduces the global loss in its current state.



Prioritized experience replay

# Prioritized Experience Replay

- Define a priority  $p_i$  of a transition pair  $i = (s, a, s')$  with TD error  $\delta = Q_\theta(s) - (R + \gamma Q_\theta(s', \pi_\phi(s')))$ .
  - **Proportional prioritization:**  $p_i = |\delta| + \epsilon$ , where  $\epsilon$  is a small positive constant.
  - **Rank-based prioritization**  $p_i = \frac{1}{\text{rank}(|\delta_i|)}$ , where  $\text{rank}(|\delta_i|)$  is the rank of absolute value of TD error in the replay buffer.
- Probability of sampling transition  $i$  as  $P_i = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$  where  $p_i > 0$  is the priority of transition  $i$ .
- Comment: Probability of being sampled is monotonic in a transition's priority and guaranteeing a non-zero probability even for the lowest-priority transition.

*Prioritized experience replay*

## Prioritized Experience Replay

- Reinforcement learning process is highly non-stationary, thus the variance of update is typically very high. Only focusing on bad transitions exacerbates the issue. To stabilize training, importance sampling is added to smooth out the distribution to estimate the policy update:
  - $w'_i = (NP_i)^{-\beta}, w_i = \frac{w'_i}{\max_j w'_j}$ .
  - TD loss with importance sampling is  $\sum_i w_i ||\delta_i||^2$ .
  - In practice,  $\beta$  is linearly annealed from its initial value, e.g.,  $\beta_0 = 0.5$ , to 1.

## **Tricks to Accelerate Reward Propagation**

## Slow Reward Propagation

- Q-learning accumulates a single reward and then uses the greedy action at the next step to bootstrap. The propagation speed is very slow when horizon is long and reward signal is sparse.
- Solution:  $TD(n)$  can be used to update value network.
  - $TD_{\theta_i}^n(s_t, a_t) = \|Q_{\theta}(s_t, a_t) - [\sum_{i=0}^{n-1} \gamma^i R(s_{t+i}, a_{t+i}, s'_{t+i}) + \gamma^n Q_{\theta}(s'_{t+n}, \pi_{\phi}(s'_{t+n}))]\|^2$ .
  - $n = 3$  is used in Rainbow for Atari games.

*Rainbow: Combining Improvements in Deep Reinforcement Learning*

## **Tricks in Value Network Architecture Design**

# Dueling Network

- For discrete action space, researchers find that decoupling  $Q(s, a)$  into  $V(s) + A(s, a)$  can help.
- Intuitively,  $V$  only depends on  $s$ , which captures features depending only on states and may generalize to similar states.  $A$  should focus on how action may affect the performance.
- A simple idea is represent  $Q_\theta(s, a)$  with  $V_{\theta_V}(s) + A_{\theta_A}(s, a)$ . However, adding any constant  $C$  to  $V_{\theta_V}(s)$  and subtracting  $C$  from  $A_{\theta_A}(s, a)$  also works. This decomposition is unidentifiable, and  $V_{\theta_V}(s)$  may not recover the value function.
- One possible solution:  $Q_\theta(s, a)$  with  $V_{\theta_V}(s) + (A_{\theta_A}(s, a) - \max_{a' \in \mathcal{A}} A_{\theta_A}(s, a'))$ . Now, for  $a^* = \operatorname{argmax}_{a' \in \mathcal{A}} Q_\theta(s, a') = \operatorname{argmax}_{a' \in \mathcal{A}} A_{\theta_A}(s, a')$ . So  $Q_\theta(s, a^*) = V_{\theta_V}(s)$ , which means  $V_{\theta_V}(s)$  really represents the value function.
- In practice, average is used to replace the max operator in  $V_{\theta_V}(s) + (A_{\theta_A}(s, a) - \max_{a' \in \mathcal{A}} A_{\theta_A}(s, a'))$ , which increases stability.

*Dueling network architectures for deep reinforcement learning*

# **Tricks by Considering Uncertainty of Value**

## **Estimation**

# Stochasticity in the Environments

- In stochastic environment, for example, stepping with an action can provide different results. Then scalar  $Q$  value can only provide the mean and lost the information about the reward distribution.
- Even in deterministic environments like Atari games, stochasticity does occur in a number of guises:
  - from state, for example image, aliasing
  - learning from a non-stationary policy
  - from approximation errors.

*A Distributional Perspective on Reinforcement Learning*

# Value Network with Discrete Distribution

- Instead of one value, value network predicts a discrete distribution on a set of atoms  $\{z_i = V_{MIN} + i\Delta z | 0 \leq i < N\}$ ,  $\Delta z := \frac{V_{MAX} - V_{MIN}}{N-1}$ . The probability of atom  $z_i$  is  $p_i$ .
- $Q_\theta(s, a) = \sum_i p_{\theta,i}(s, a)z_i$ .
- Update rules of  $Q$ ,  $x_t$  is state at time-step  $t$ .

---

**Algorithm 1** Categorical Algorithm

---

```
input A transition  $x_t, a_t, r_t, x_{t+1}, \gamma_t \in [0, 1]$ 
       $Q(x_{t+1}, a) := \sum_i z_i p_i(x_{t+1}, a)$ 
       $a^* \leftarrow \arg \max_a Q(x_{t+1}, a)$ 
       $m_i = 0, \quad i \in 0, \dots, N - 1$ 
      for  $j \in 0, \dots, N - 1$  do
        # Compute the projection of  $\hat{T}z_j$  onto the support  $\{z_i\}$ 
         $\hat{T}z_j \leftarrow [r_t + \gamma_t z_j]_{V_{MIN}}^{V_{MAX}}$ 
         $b_j \leftarrow (\hat{T}z_j - V_{MIN})/\Delta z \quad \# b_j \in [0, N - 1]$ 
         $l \leftarrow \lfloor b_j \rfloor, u \leftarrow \lceil b_j \rceil$ 
        # Distribute probability of  $\hat{T}z_j$ 
         $m_l \leftarrow m_l + p_j(x_{t+1}, a^*)(u - b_j)$ 
         $m_u \leftarrow m_u + p_j(x_{t+1}, a^*)(b_j - l)$ 
      end for
output  $-\sum_i m_i \log p_i(x_t, a_t)$  # Cross-entropy loss
```

---

## **Tricks in Leveraging State-conditioned Exploration Noise**

*Read by Yourself*

## State-conditioned Exploration Noise

- Using the same exploration policy for all the states is not efficient in practice.

*Rainbow: Combining Improvements in Deep Reinforcement Learning*

## Noisy Nets (For Discrete Action Space)

- For discrete action space, we can replace the last linear layer of the value network  $Q_\theta$  with a **noisy linear layer** to form a noisy  $Q$  network and generate exploration actions directly from the noisy  $Q$ .
- Let original linear layer be  $y = wx + b$ , **noisy linear layer** will be  $y = wx + b + (\sigma^w \circ \epsilon^w x + \sigma^b \circ \epsilon^b)$ , where  $\circ$  represents element-wise multiplication,  $\sigma^w, \sigma^b$  are learnable parameters,  $\epsilon$  are noise random variables.
- Greedily selecting on the noisy  $Q$  value has already provided exploration. Other parts are similar to DQN. The only difference is that noisy  $Q$  network is used to replace the classical  $Q$  network.
- Updating  $Q$  network with TD loss.

*Rainbow: Combining Improvements in Deep Reinforcement Learning*

## Parameterized Squashed Gaussian policy (For Continuous Action Space)

- For continuous action space, the policy network can output a parameterized Gaussian distribution.
- $\pi_\phi(s) = \tanh(\mu_\phi(s) + \sigma_\phi(s) \circ \epsilon)$ , where  $\epsilon = (0, I)$ .
- $\mu_\phi(s)$  is the mean value of the agent's policy at state  $s$ , which can be used in evaluation.
- $\sigma_\phi(s)$  is the standard deviation of the exploration noise.
- In practice, policy network predicts  $\log \sigma$  and clip it in to range  $[\log \sigma_{min}, \log \sigma_{max}]$ . In SAC,  $\log \sigma_{min} = -20, \log \sigma_{max} = 2$ .
- To train  $\sigma_\phi(s)$ , we need entropy regularization which will be discussed in SAC in the next lecture.

*Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*

*Read by Yourself*

## Off-Policy RL Frameworks in Practice

# Practical Off-Policy Algorithms

- In practice, a good off-policy algorithm needs to ensemble tricks and solve the exploration problem.
- Example algorithms:
  - Rainbow
  - Soft-Actor-Critic

*Rainbow: Combining Improvements in Deep Reinforcement Learning*

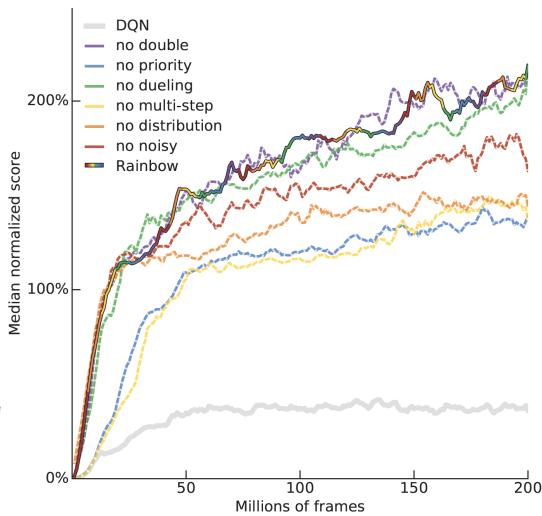
# Rainbow

- For discrete action space, Rainbow is a famous algorithm which ensembles all the tricks to get better performance.
- Tricks used in Rainbow:
  - Prioritized replay
  - Multi-step learning / TD(3) loss
  - Distributional RL
  - Noisy Net
  - Double Q-learning
  - Dueling Q-learning

*Rainbow: Combining Improvements in Deep Reinforcement Learning*

## Ablation study of tricks in Rainbow

- Prioritized replay, multi-step learning, distributional RL are the most important tricks in Rainbow.



*Rainbow: Combining Improvements in Deep Reinforcement Learning*

# Soft-Actor-Critic (SAC)

- For continuous action space, Soft-Actor-Critic is a famous algorithm which combines some tricks with an entropy-based exploration method.
- Tricks used in SAC:
  - Clipped Double Q-Learning
  - Parameterized Squashed Gaussian policy

*Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*