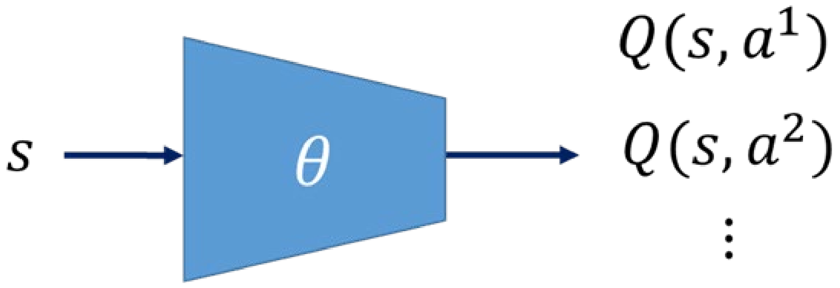
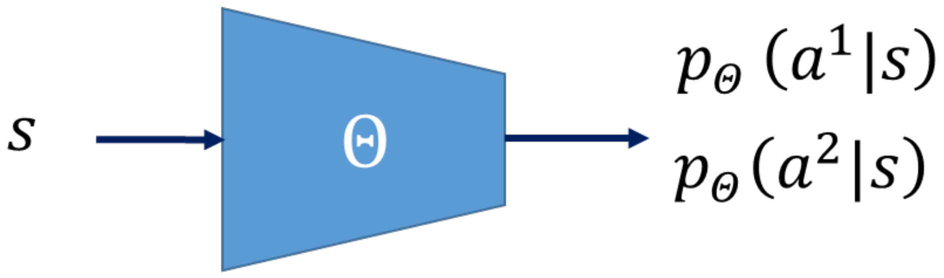
Recap: what and how DQN and PPO learn



DQN

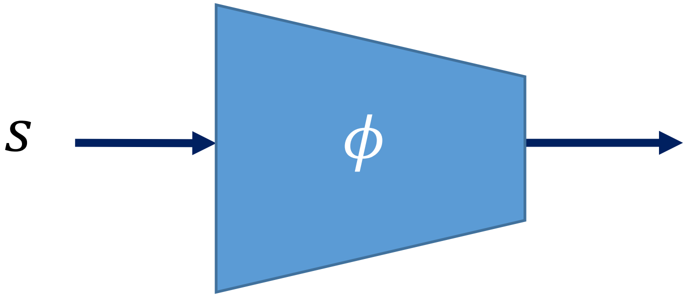
- Learn expected value of long-term reward of discrete actions $Q(s, a)$
- Use Bellman eq. to recursively learn $Q^*(s, a)$ from $Q^*(s', a')$



PPO

- Learn policy of continuous action $p_\Theta\,(a|s)$
- Use expected value of long-term reward to adjust probability $p_\Theta\,(a|s)$

Can we use DQN to learn continuous actions?



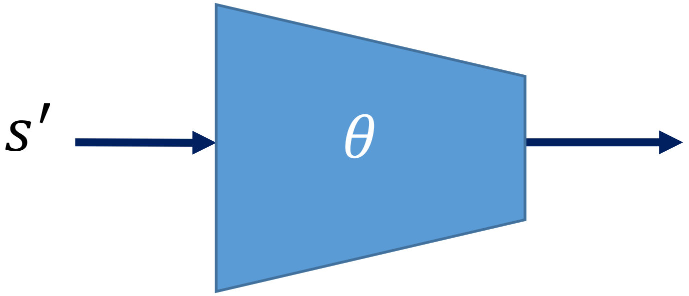
Critic Q_ϕ

$$Q_\phi(s, a^1)$$
$$Q_\phi(s, a^2)$$
$$\vdots$$

$$Loss = \mathrm{E}\left[\left(r_s^a + \gamma \max_{a'} Q_\phi(s', a') - Q_\phi(s, a)\right)^2\right]$$

For continuous actions, it is impossible to calculate $Q_\phi(s', a')$ for every possible a' value

Solution: train a NN to learn $\max_{a'} Q(s', a')$

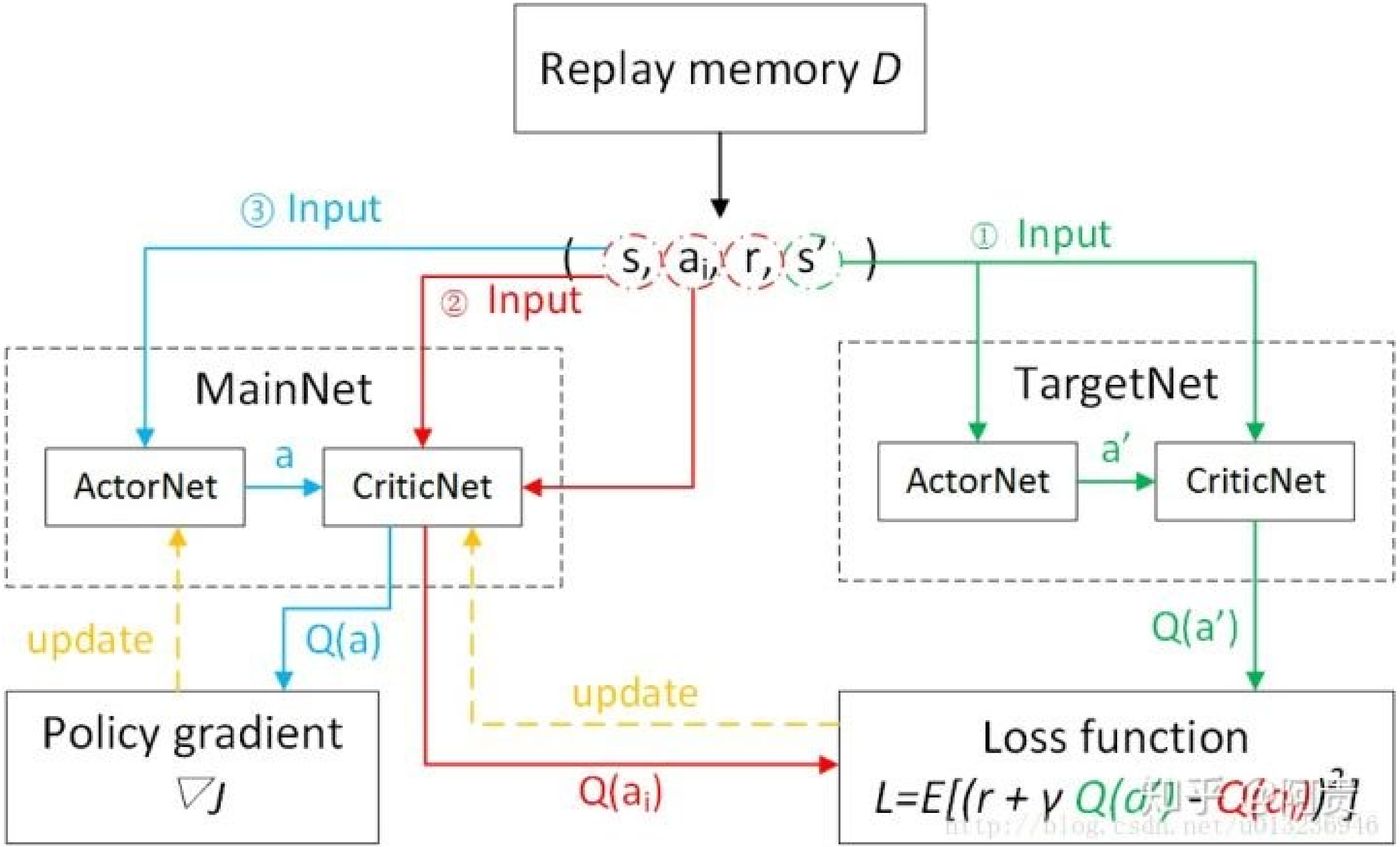


Actor μ_θ

$$\max_{a'_1} Q(s', a'_1)$$
$$\max_{a'_2} Q(s', a'_2)$$
$$\vdots$$

$$Loss = \mathrm{E}\left[\left((r_s^a + \gamma Q_\phi(s', \mu_\theta(s')) - Q_\phi(s, a)\right)^2\right]$$

Deep deterministic policy gradient



圖片來源: https://zhuanlan.zhihu.com/p/47873624

Update critic net

9. DDPG_Agent.ipynb

$$Loss = \mathrm{E}\left[\left((r_s^a + \gamma Q_\phi(s', \mu_\theta(s')) - Q_\phi(s,a)\right)^2\right]$$

```
actions_next = self.actor_target(next_states)
Q_targets_next = self.critic_target(next_states, actions_next)

# Compute Q targets for current states (y_i)
Q_targets = rewards + (GAMMA * Q_targets_next * (1 - dones))

# Compute critic loss
Q_expected = self.critic_local(states, actions)
critic_loss = F.mse_loss(Q_expected, Q_targets)
```

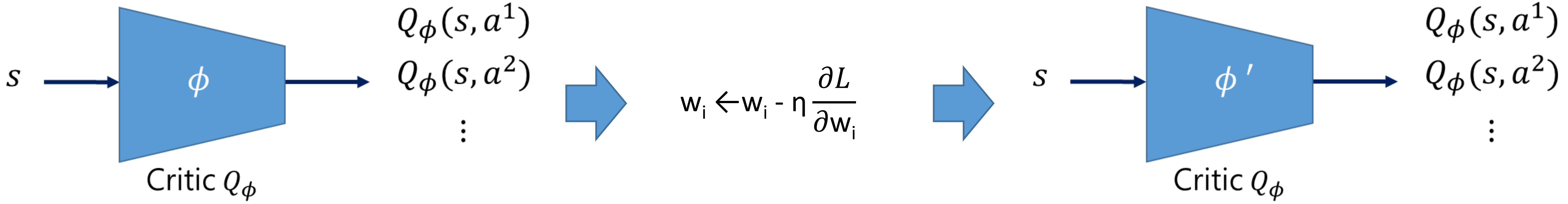
$\mu_{\theta_{target}}(s') \leftarrow \max_{a'} Q_\phi(s', a')$

$Q_{\phi_{target}}(s', \mu_\theta(s'))$

$r_s^a + \gamma Q_{\phi_{target}}(s', \mu_\theta(s'))$

$Q_\phi(s,a)$

$\mathrm{E}\left[\left((r_s^a + \gamma Q_\phi(s', \mu_\theta(s')) - Q_\phi(s,a)\right)^2\right]$



Update actor net

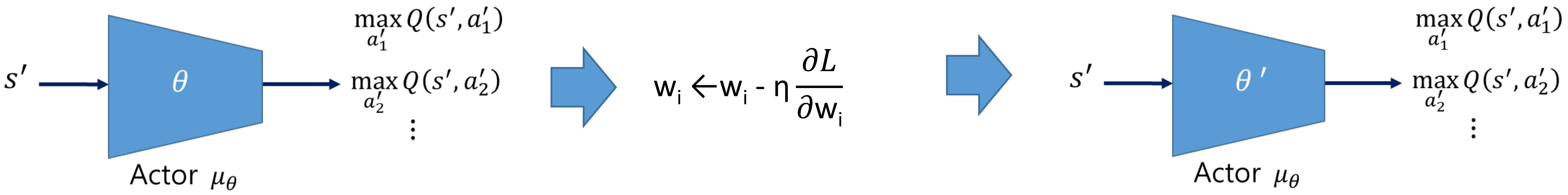
$$\max_{\theta} \mathbb{E}_{s \sim \mathcal{D}}\left[Q_{\phi}(s, \mu_{\theta}(s))\right]$$

9. DDPG_Agent.ipynb

```
# Compute actor loss
actions_pred = self.actor_local(states)
actor_loss = -self.critic_local(states, actions_pred).mean()
```

$\mu_{\theta}(s)$

$Q_{\phi}(s, \mu_{\theta}(s))$



Update target nets

$$\phi_{target} \leftarrow \rho\phi_{target} + (1-\rho)\phi$$

$$\theta_{target} \leftarrow \rho\theta_{target} + (1-\rho)\theta$$

ρ is a hyperparameter between 0 and 1 (usually close to 1). (This hyperparameter is called polyak in our code).

9. DDPG_Agent.ipynb

```
# ----------------------- update target networks ----------------------- #
self.soft_update(self.critic_local, self.critic_target, TAU)
self.soft_update(self.actor_local, self.actor_target, TAU)
```

```
for target_param, local_param in zip(target_model.parameters(), local_model.parameters()):
    target_param.data.copy_(tau*local_param.data + (1.0-tau)*target_param.data)
```

Introduction

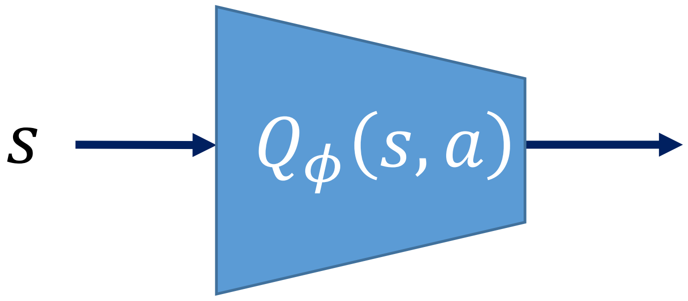
Deep Deterministic Policy Gradient (DDPG) is an algorithm which concurrently learns a Q-function and a policy. It uses off-policy data and the Bellman equation to learn the Q-function, and uses the Q-function to learn the policy.

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

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

Reference: https://spinningup.openai.com/en/latest/algorithms/ddpg.html#deep-deterministic-policy-gradient

The Q-learning side of DDPG – Critic Net



$Q(s,a^1)$

$Q(s,a^2)$

$\vdots$

Critic Net $\quad Q_\phi(s,a), Q_{\phi_{targ}}(s',a')$

$$L(\phi,\mathcal{D}) = \underset{(s,a,r,s',a')\backsim\mathcal{D}}{\mathbb{E}}\left[\left(Q_\phi(s,a) - \ (r_s^a + \gamma(1-d)\max_{a'} Q_{\phi_{targ}}(s',a')\right)^2\right]$$

Trick one – replay buffer $\mathcal{D}$

$$L(\phi, \mathcal{D}) = \mathop{\mathbb{E}}_{(s,a,r,s',a')\boxed{\sim\mathcal{D}}}\left[\left(Q_\phi(s,a) - \ (r_s^a + \gamma(1-d)\max_{a'} Q_\phi(s',a')\right)^2\right])$$

In order for the algorithm to have stable behavior, the replay buffer should be large enough to contain a wide range of experiences, but it may not always be good to keep everything. If you only use the very-most recent data, you will overfit to that and things will break; if you use too much experience, you may slow down your learning.

Trick two – Target Network ϕ_{targ}

$$L(\phi, \mathcal{D}) = \underset{(s,a,r,s',a') \backsim \mathcal{D}}{\mathbb{E}} \left[\left(Q_\phi(s,a) - \ (r_s^a + \boxed{\gamma(1-d) \max_{a'} Q_{\phi_{targ}}(s',a')} \right)^2 \right])$$

Eval network ϕ

Target network ϕ_{targ}

DDPG: calculating Max over actions in the Target

Computing the maximum over actions in the target is a challenge in continuous action spaces. DDPG deals with this by using a target policy network to compute an action which approximately maximizes $Q_{\phi_{targ}}$. The target policy network is found the same way as the target Q-function: by polyak averaging the policy parameters over the course of training.

$$L(\phi, \mathcal{D}) = \underset{(s,a,r,s',a') \sim \mathcal{D}}{\mathbb{E}} \left[\left(Q_\phi(s,a) - \ (r_s^a + \gamma(1-d) Q_{\phi_{targ}}(s', \mu_{\theta_{targ}}(s') \right)^2 \right])$$

$\mu_{\theta_{targ}}$: target policy network

The policy learning side of DDPG

$$\max_{\theta} \mathop{\mathbb{E}}_{s \backsim \mathcal{D}} \left[Q_{\phi}(s, \mu_{\theta}(s)) \right]$$

Exploration vs. Exploitation

Because the policy is deterministic, if the agent were to explore on-policy, in the beginning it would probably not try a wide enough variety of actions to find useful learning signals. To make DDPG policies explore better, we add noise to their actions at training time.

9. DDPG_Agent.ipynb

```
#adding noise for exploration!
if add_noise:
    acts += self.noise.sample()
```