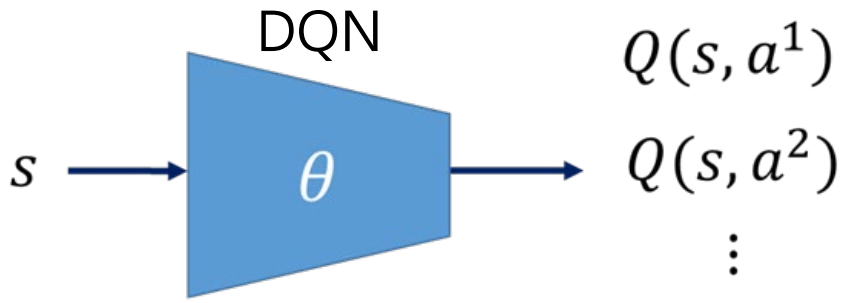
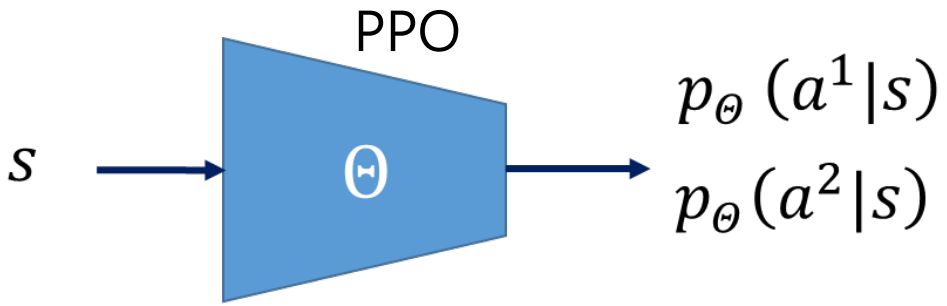
Recap: PPO, DQN and DDPG



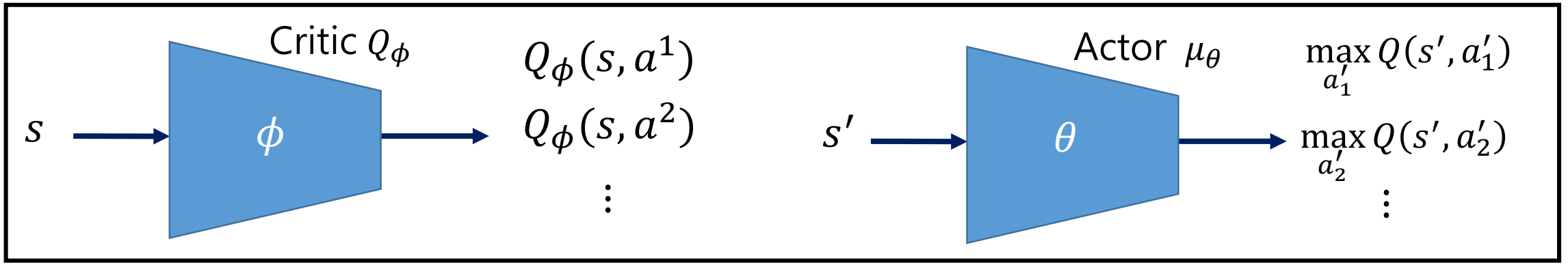
- Learn expected long-term reward $Q(s,a)$
- Use Bellman eq. to update $Q^*(s,a)$ based on $Q^*(s',a')$



- Learn policy $p_\Theta(a|s)$
- Use policy gradient (gradient of expected value of long-term rewards) to adjust $p_\Theta(a|s)$

DDPG

- Learn $Q(s,a)$ and a' that max $Q(s',a')$
- Use Bellman eq. to update $Q^*(s,a)$ and use $Q(s,a)$ to update policy network



Introduction – Twin Delayed DDPG (TD3)

While DDPG can achieve great performance sometimes, it is frequently brittle with respect to hyperparameters and other kinds of tuning. A common failure mode for DDPG is that the learned Q-function begins to dramatically overestimate Q-values, which then leads to the policy breaking, because it exploits the errors in the Q-function. Twin Delayed DDPG (TD3) is an algorithm that addresses this issue by introducing three critical tricks.

Reference: Twin Delayed DDPG — Spinning Up documentation (openai.com)

Tricks to overcome DDPG problems

Trick one – Clipped Double-Q Learning

TD3 learns two Q-functions instead of one (hence "twin"), and uses the smaller of the two Q-values to form the targets in the Bellman error loss functions.

Trick two – "Delayed" Policy Updates

TD3 updates the policy (and target networks) less frequently than the Q-function. The paper recommends one policy update for every two Q-function updates.

Trick three – Target Policy Smoothing

TD3 adds noise to the target action, to make it harder for the policy to exploit Q-function errors by smoothing out Q along changes in action.

Target Policy Smoothing

Actions used to form the Q-learning target are based on the target policy network $\mu_{\theta_{target'}}$ but with clipped noise added on each dimension of the action. After adding the clipped noise, the target action is then clipped to lie in the valid action range (all valid actions, a, satisfy $a_{Low} \leq a \leq a_{High}$).

$$a'(s') = clip\left(\mu_{\theta_{target}}(s') + clip(\epsilon, -c, c), a_{Low}, a_{High}\right) \qquad \epsilon \sim \mathcal{N}(0, \sigma)$$

Target policy network in DDPG

$$\mu_{\theta_{target}}(s') \approx arg\max_{a'} Q(s', a')$$

Clipped double-Q learning

Both Q-functions use a single target, calculated using whichever of the two Q-functions gives a smaller target value.

$$y(r, s', d) = r + \gamma(1 - d) \min_{j=1,2} Q_{\phi_{targ,j}}(s', a'(s'))$$

$$L(\phi_1, \mathcal{D}) = \mathop{\mathrm{E}}_{(s,a,r,s',a') \backsim \mathcal{D}} \left[\left(Q_{\phi_1}(s, a) - y(r, s', d) \right)^2 \right])$$

$$L(\phi_2, \mathcal{D}) = \mathop{\mathrm{E}}_{(s,a,r,s',a') \backsim \mathcal{D}} \left[\left(Q_{\phi_2}(s, a) - y(r, s', d) \right)^2 \right])$$

Learning policy

The policy is learned just by maximizing Q_{ϕ_1}.

$$\max_{\theta} \mathop{\mathbb{E}}_{s \sim \mathcal{D}} \left[Q_{\phi_1}(s, \mu_\theta(s)) \right]$$

In TD3, the policy is updated less frequently than the Q-functions are. This helps damp the volatility that normally arises in DDPG because of how a policy update changes the target.

Exploration vs. Exploitation

TD3 trains a deterministic policy in an off-policy way. 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 TD3 policies explore better, we add noise to their actions at training time, typically uncorrelated mean-zero Gaussian noise. To facilitate getting higher-quality training data, you may reduce the scale of the noise over the course of training.