Policy Gradient & Actor-Critic

Apr 23th, 2019

Applied Deep Learning

YUN-NUNG (VIVIAN) CHEN HTTP://ADL.MIULAB.TW

國立臺灣大學
National Taiwan University

Slides credited from Dr. David Silver & Hung-Yi Lee

Reinforcement Learning Approach

Value-based RL

- Estimate the optimal value function $Q^*(s, a)$

$Q^*(s, a)$ is maximum value achievable under any policy

Policy-based RL

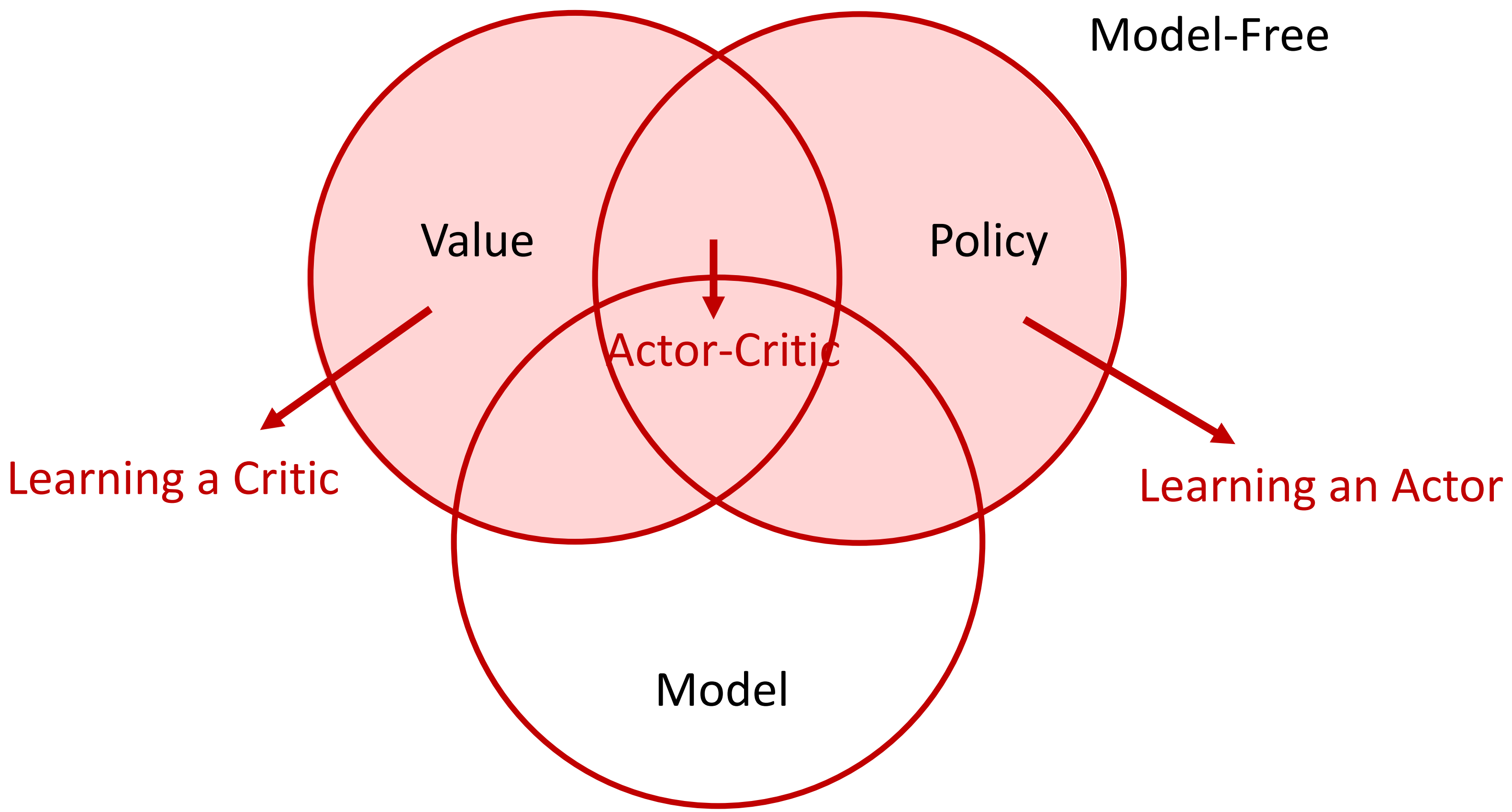
- Search directly for optimal policy π^*

π^* is the policy achieving maximum future reward

Model-based RL

- Build a model of the environment
- Plan (e.g. by lookahead) using model

RL Agent Taxonomy



Policy-Based Approach

LEARNING AN ACTOR

Policy

A policy is the agent's behavior

A policy maps from state to action

- Deterministic policy: $a = \pi(s)$
- Stochastic policy: $\pi(a) = P(a \mid s)$

Policy Networks

Represent policy by a network with parameters θ

$$a = \pi(a \mid s, \theta) \qquad a = \pi(s, \theta)$$

stochastic policy | deterministic policy

Objective is to maximize total discounted reward by SGD

$$O(\theta) = \mathrm{E}[r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots \mid \pi(\cdot, \theta)]$$

On-Policy v.s. Off-Policy

On-policy: The agent learned and the agent interacting with the environment is the same

Off-policy: The agent learned and the agent interacting with the environment is different

Goodness of Actor

An episode is considered as a trajectory τ

- $\tau = \{s_1, a_1, r_1, s_2, a_2, r_2, \cdots, s_T, a_T, r_T\}$
- Reward: $R(\tau) = \sum_{t=1}^{T} \gamma^{t-1} r_t$

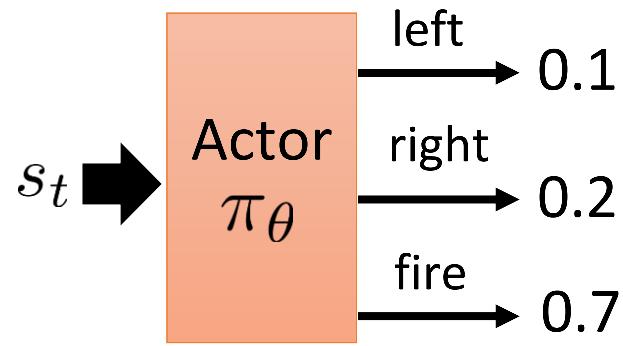
$$P(\tau \mid \theta) =$$

$$p(s_1)p(a_1 \mid s_1, \theta)p(r_1, s_2 \mid s_1, a_1)p(a_2 \mid s_2, \theta)p(r_2, s_3 \mid s_2, a_2)\cdots$$

$$= p(s_1) \prod_{t=1}^{T} p(a_t \mid s_t, \theta) p(r_t, s_{t+1} \mid s_t, a_t)$$

not related to your actor

control by your actor



$$p(a_t = \text{fire} \mid s_t, \theta) = 0.7$$

Goodness of Actor

An episode is considered as a trajectory τ

- $\tau = \{s_1, a_1, r_1, s_2, a_2, r_2, \cdots, s_T, a_T, r_T\}$
- Reward: $R(\tau) = \sum_{t=1}^{T} \gamma^{t-1} r_t$

We define $\mathcal{R}(\theta)$ as the *expected value* of reward

- If you use an actor to play game, each τ has $P(\tau|\theta)$ to be sampled

$$\mathcal{R}(\theta) = \sum_{\tau} R(\tau) P(\tau \mid \theta) \approx \frac{1}{N} \sum_{n=1}^{N} R(\tau^n)$$

  - Use π_θ to play the game N times, obtain $\{\tau^1, \tau^2, \cdots, \tau^N\}$
  - Sampling τ from $P(\tau|\theta)$ N times

sum over all possible trajectory

Deep Policy Networks

Represent policy by deep network with weights

Objective is to maximize total discounted reward by SGD

$$\mathcal{R}(\theta) = \mathbb{E}\Big[r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots \mid \pi(\cdot, \theta)\Big]$$

Update the model parameters iteratively

$$\theta^* = \arg\max_{\theta} \mathcal{R}(\theta)$$

$$\theta' \leftarrow \theta + \eta \nabla \mathcal{R}(\theta)$$

Policy Gradient $\bar{R}_\theta = \sum_\tau R(\tau)P(\tau \mid \theta)$

Gradient assent to maximize the expected reward

$$\nabla \bar{R}_\theta = \sum_\tau R(\tau) \nabla P(\tau \mid \theta) = \sum_\tau R(\tau) P(\tau \mid \theta) \frac{\nabla P(\tau \mid \theta)}{P(\tau \mid \theta)}$$

do not have to be differentiable

can even be a black box

$$= \sum_\tau R(\tau) P(\tau \mid \theta) \nabla \log P(\tau \mid \theta)$$

$$\frac{d \log f(x)}{dx} = \frac{1}{f(x)} \frac{df(x)}{dx}$$

use π_θ to play the game N times, obtain $\{\tau^1, \tau^2, \cdots, \tau^N\}$

$$\approx \frac{1}{N} \sum_{n=1}^{N} R(\tau^n) \underline{\nabla \log P(\tau^n \mid \theta)}$$

Policy Gradient $\nabla \log P(\tau \mid \theta)$

An episode trajectory $\tau = \{s_1, a_1, r_1, s_2, a_2, r_2, \cdots, s_T, a_T, r_T\}$

$$P(\tau \mid \theta) = p(s_1) \prod_{t=1}^{T} p(a_t \mid s_t, \theta) p(r_t, s_{t+1} \mid s_t, a_t)$$

$$\log P(\tau \mid \theta)$$

$$= \log p(s_1) \sum_{t=1}^{T} \log p(a_t \mid s_t, \theta) + \log p(r_t, s_{t+1} \mid s_t, a_t)$$

$$\nabla \log P(\tau \mid \theta) = \sum_{t=1}^{T} \nabla \log p(a_t \mid s_t, \theta)$$

ignore the terms not related to θ

Policy Gradient

Gradient assent for iteratively updating the parameters

$\theta' \leftarrow \theta + \eta \nabla \mathcal{R}(\theta)$

$$\nabla \mathcal{R}(\theta) \approx \frac{1}{N} \sum_{n=1}^{N} R(\tau^n) \nabla \log P(\tau^n \mid \theta)$$

$$= \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} R(\tau^n) \nabla \log p(a_t^n \mid s_t^n, \theta)$$

- If τ^n machine takes a_t^n when seeing s_t^n

$R(\tau^n) > 0$ ➡ Tuning θ to increase $p(a_t^n \mid s_t^n)$

$R(\tau^n) < 0$ ➡ Tuning θ to decrease $p(a_t^n \mid s_t^n)$

Important: use *cumulative* reward $R(\tau^n)$ of the whole trajectory τ^n instead of *immediate* reward r_t^n

Policy Gradient

Given actor parameter θ

$$\tau^1: \ (s_1^1, a_1^1) \quad R(\tau^1) \qquad \tau^2: \ (s_1^2, a_1^2) \quad R(\tau^2)$$

$$(s_2^1, a_2^1) \quad R(\tau^1) \qquad (s_2^2, a_2^2) \quad R(\tau^2)$$

$$\vdots \qquad \vdots \qquad \vdots \qquad \vdots$$

data collection

model update

$$\theta' \leftarrow \theta + \eta \nabla \mathcal{R}(\theta)$$

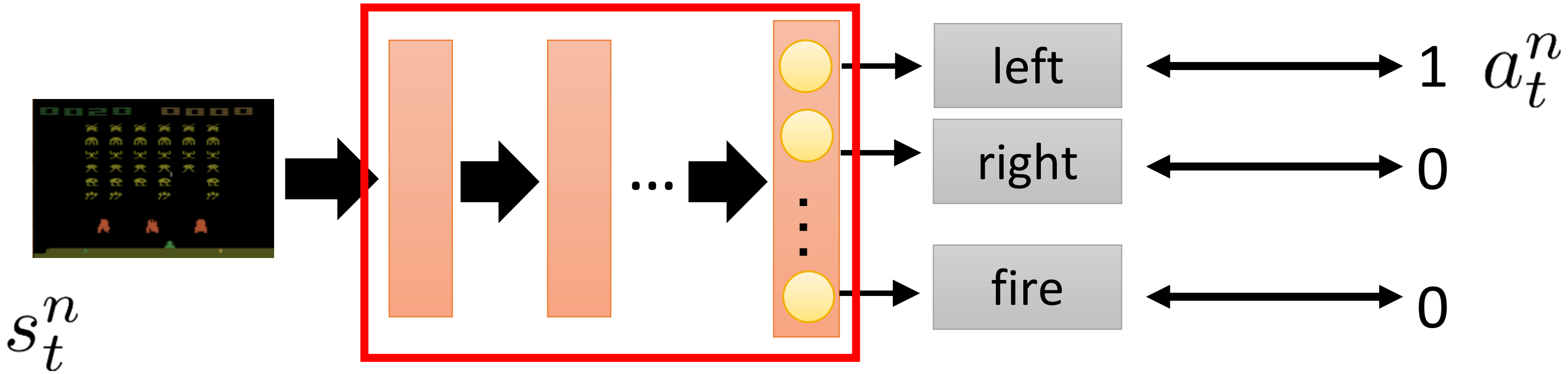
$$\nabla \mathcal{R}(\theta) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} R(\tau^n) \nabla \log p(a_t^n \mid s_t^n, \theta)$$

Implementation

$$\theta' \leftarrow \theta + \eta \nabla \mathcal{R}(\theta)$$

$$\nabla \mathcal{R}(\theta) = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n} R(\tau^n)\nabla \log p(a_t^n \mid s_t^n, \theta)$$

Treat it as a classification problem



$$\frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n} \log p(a_t^n \mid s_t^n) \longrightarrow \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n} \nabla \log p(a_t^n \mid s_t^n)$$

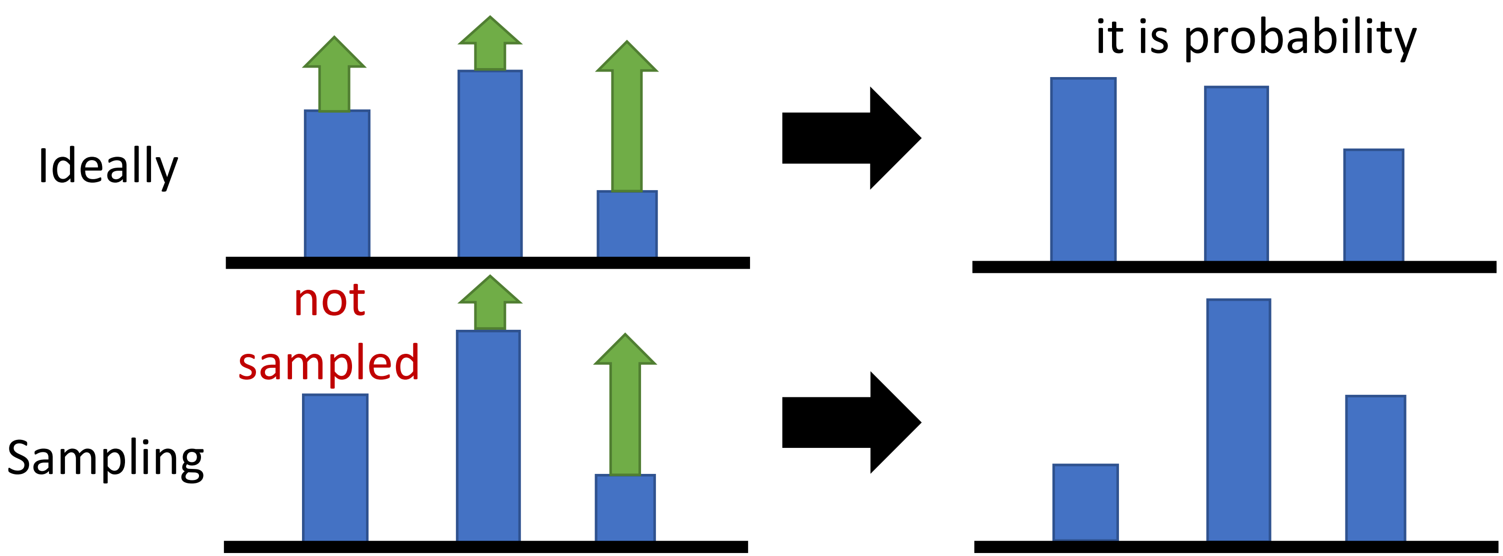
TF, PyTorch …

$$\frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n} R(\tau^n) \log p(a_t^n \mid s_t^n) \longrightarrow \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n} R(\tau^n)\nabla \log p(a_t^n \mid s_t^n)$$

Improvement: Adding Baseline

$$\theta' \leftarrow \theta + \eta \nabla \mathcal{R}(\theta)$$

$$\nabla \mathcal{R}(\theta) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} (R(\tau^n) - b) \nabla \log p(a_t^n \mid s_t^n, \theta)$$



Issue: the probability of the actions not sampled will decrease

Actor-Critic Approach

LEARNING AN ACTOR & A CRITIC

Actor-Critic (Value-Based + Policy-Based)

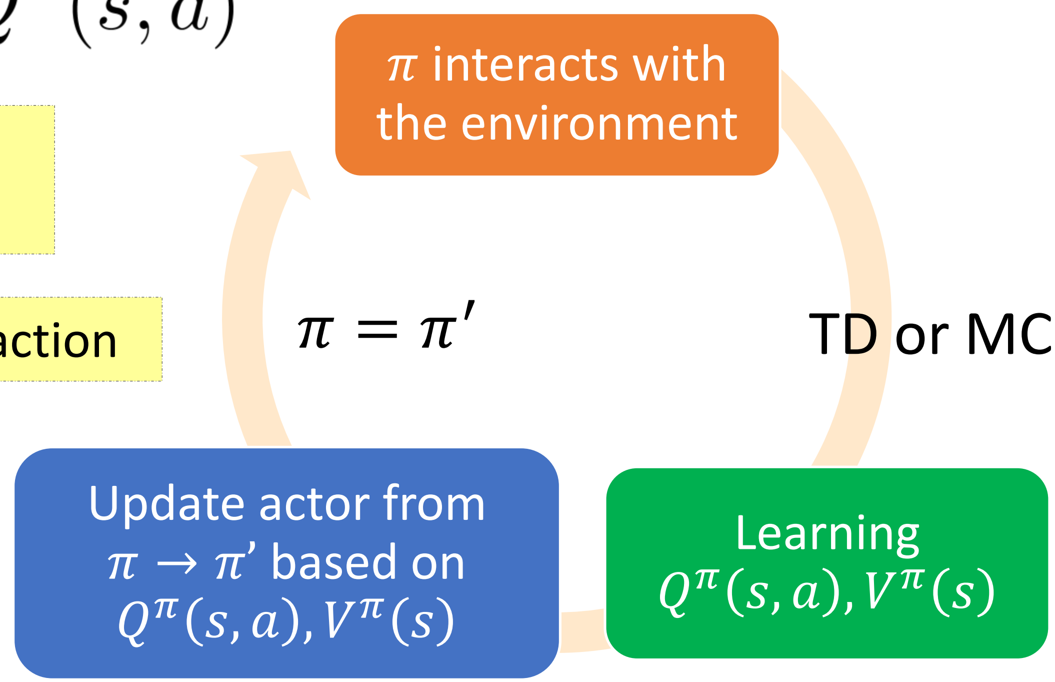
Estimate value function $Q^{\pi}(s,a), V^{\pi}(s)$

Update policy based on the value function evaluation π

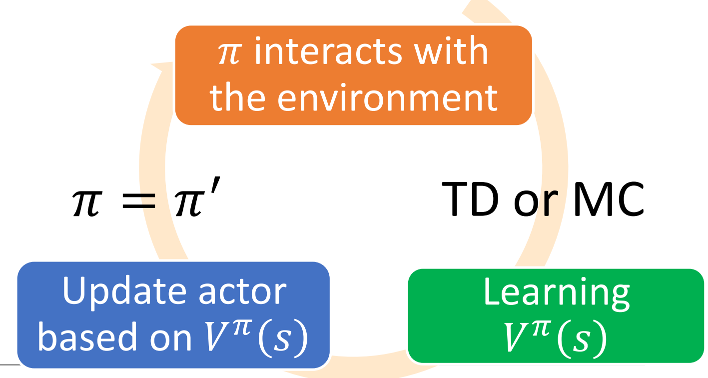
$$\pi'(s) = \arg\max_{a} Q^{\pi}(s,a)$$

π is a actual function that maximizes the value

may works for continuous action



Advantage Actor-Critic



Learning the policy (actor) using the value evaluated by critic

$$\theta^{\pi'} \leftarrow \theta^{\pi} + \eta \nabla \mathcal{R}(\theta^{\pi})$$

$$\nabla \mathcal{R}(\theta^{\pi}) = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n} R(\tau^n) \nabla \log p(a_t^n \mid s_t^n, \theta^{\pi})$$ baseline is added

evaluated by critic

Advantage function: $r_t^n - \left(V^{\pi}(s_t^n) - V^{\pi}(s_{t+1}^n)\right)$

the reward r_t^n we truly obtain when taking action a_t^n

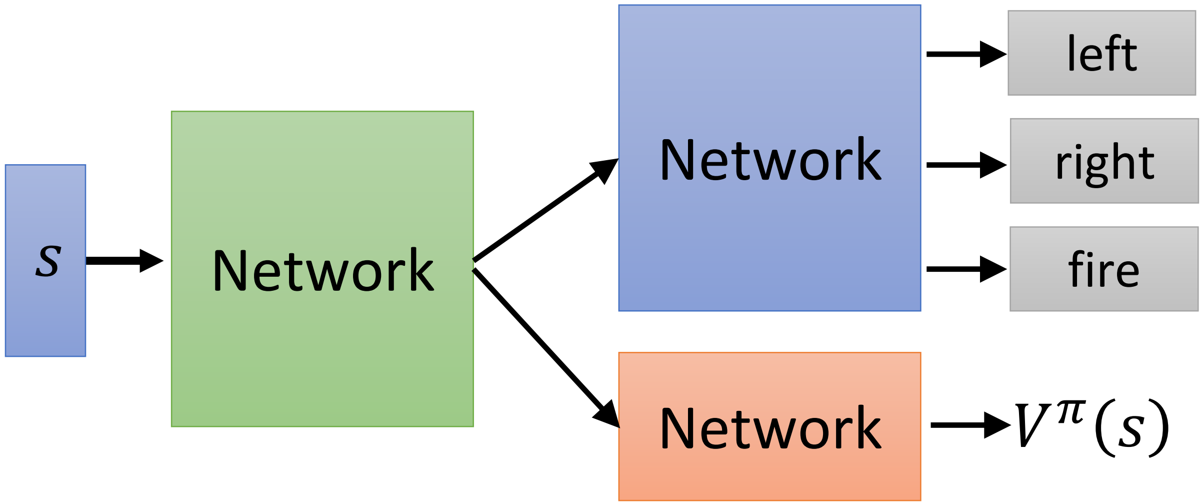
expected reward r_t^n we obtain if we use actor π

- Positive advantage function $\leftrightarrow$ increasing the prob. of action a_t^n
- Negative advantage function $\leftrightarrow$ decreasing the prob. of action a_t^n

Advantage Actor-Critic

Tips

- The parameters of actor $\pi(s)$ and critic $V^{\pi}(s)$ can be shared



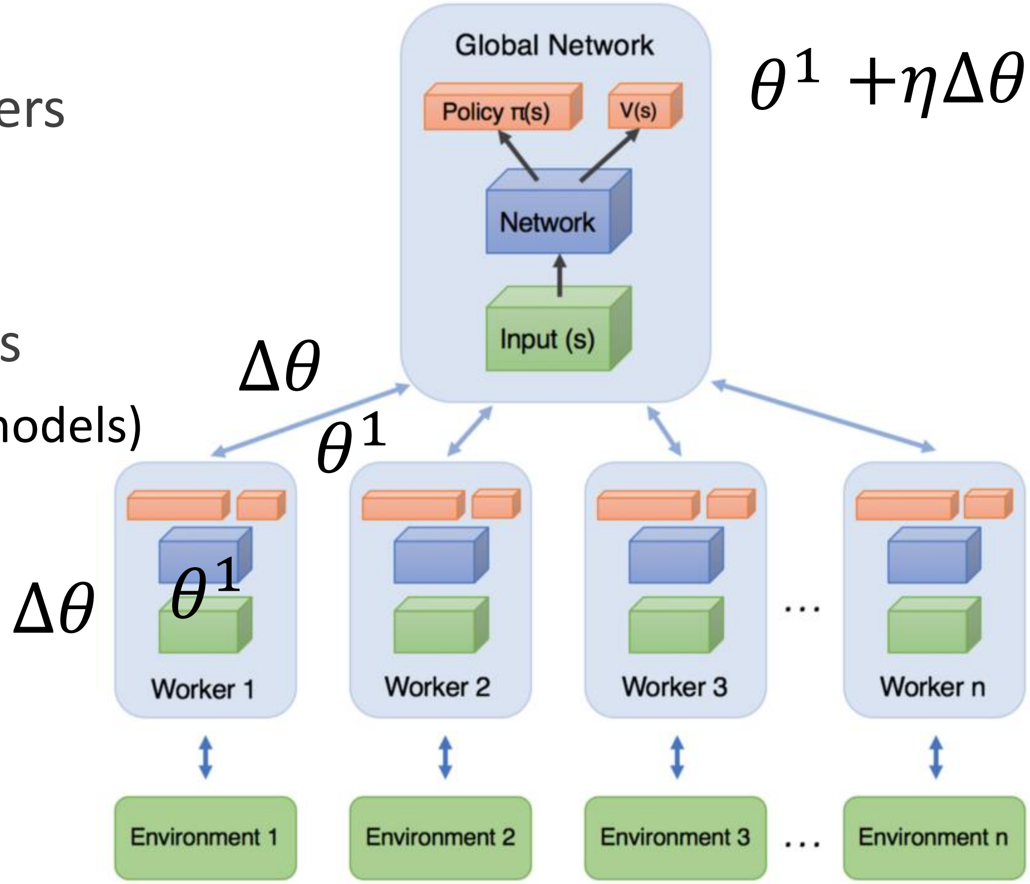
- Use output entropy as regularization for $\pi(s)$
  - Larger entropy is preferred → exploration

Asynchronous Advantage Actor-Critic (A3C)

Asynchronous

1. Copy global parameters
2. Sampling some data
3. Compute gradients
4. Update global models

(other workers also update models)



Pathwise Derivative Policy Gradient

Original actor-critic tells that a given action is good or bad

Pathwise derivative policy gradient tells that which action is good

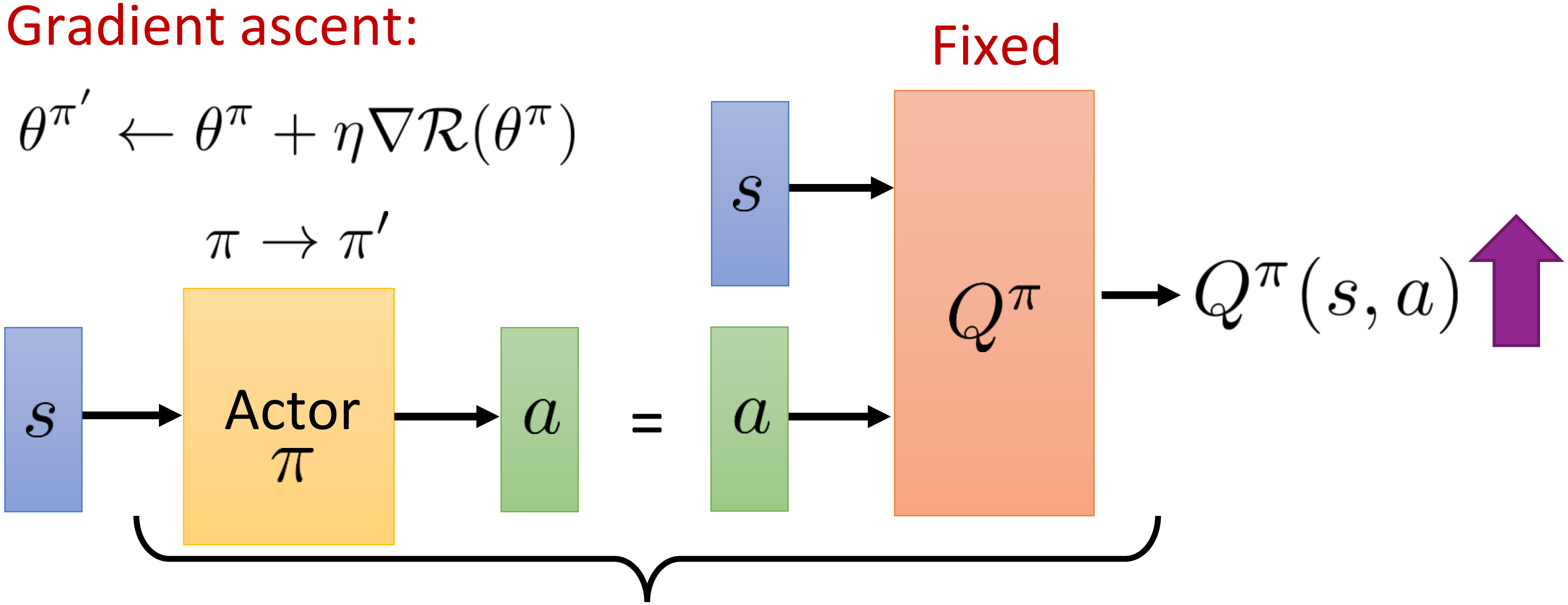
Pathwise Derivative Policy Gradient

$$\pi'(s) = \arg\max_a Q^\pi(s, a)$$ ⬅ an actor's output

Gradient ascent:

$$\theta^{\pi'} \leftarrow \theta^\pi + \eta \nabla \mathcal{R}(\theta^\pi)$$

$$\pi \rightarrow \pi'$$



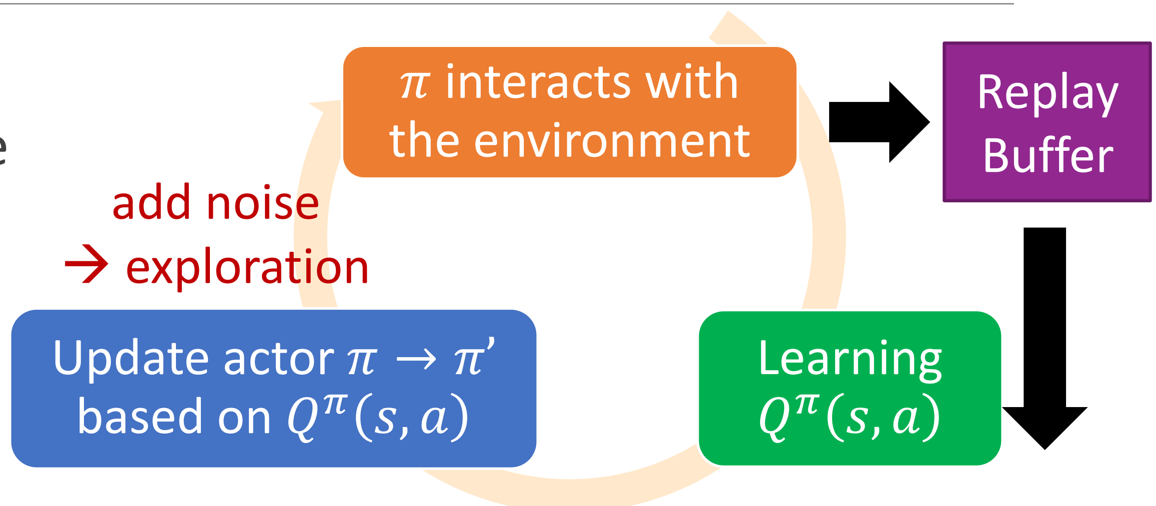
This is a large network

Silver et al., "Deterministic Policy Gradient Algorithms", ICML, 2014.
Lillicrap et al., "Continuous Control with Deep Reinforcement Learning", ICLR, 2016.

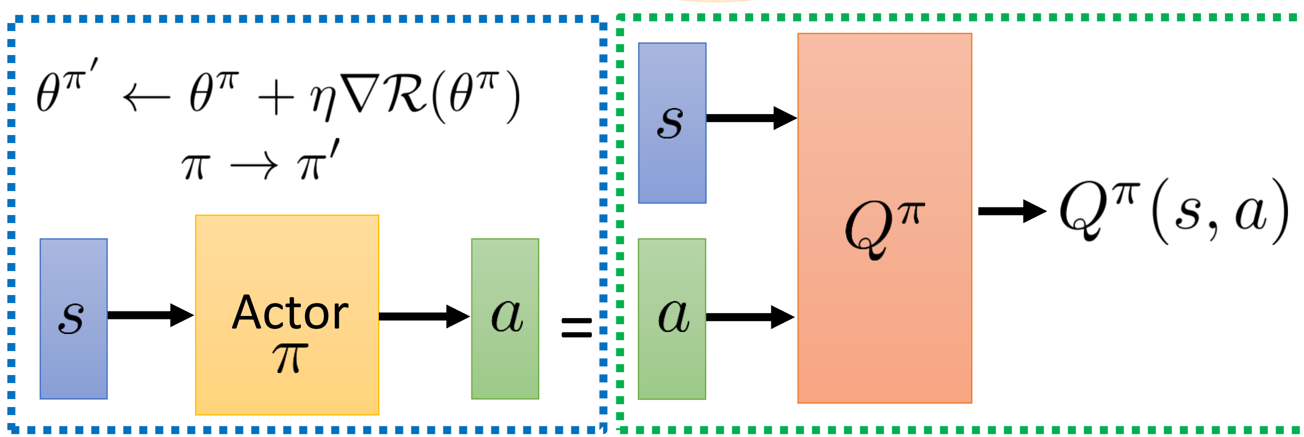
Deep Deterministic Policy Gradient (DDPG)

Idea

- Critic estimates value of current policy by DQN
- Actor updates policy in direction that improves Q



Critic provides loss function for actor



DDPG Algorithm

Initialize critic network θ^Q and actor network θ^π

Initialize target critic network $\theta^{Q'} = \theta^Q$ and target actor network $\theta^{\pi'} = \theta^\pi$

Initialize replay buffer R

In each iteration

- Use $\pi(s) + \text{noise}$ to interact with the environment, collect a set of $\{s_t, a_t, r_t, s_{t+1}\}$, put them in R
- Sample N examples $\{s_n, a_n, r_n, s_{n+1}\}$ from R
- Update critic Q to minimize $\sum_n (\hat{y}_n - Q(s_n, a_n))^2$

  $\hat{y}_n = r_n + Q'(s_{n+1}, \pi'(s_{n+1}))$ using target networks

- Update actor π to maximize $\sum_n Q(s_n, \pi(s_n))$
- Update target networks: $\theta^{\pi'} \leftarrow m\theta^\pi + (1-m)\theta^{\pi'}$

  $\theta^{Q'} \leftarrow m\theta^Q + (1-m)\theta^{Q'}$

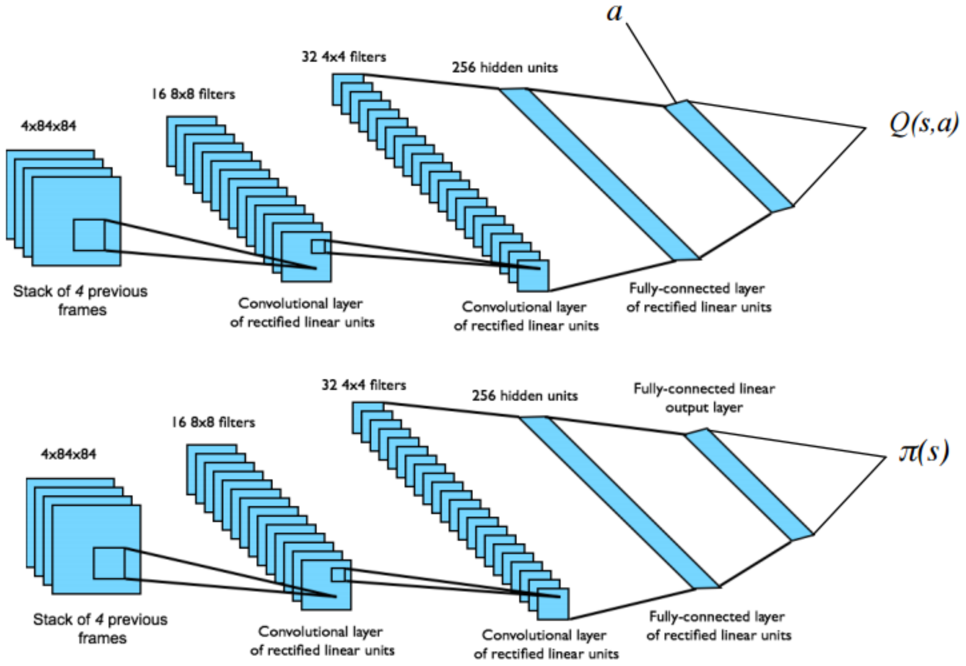
  the target networks update slower

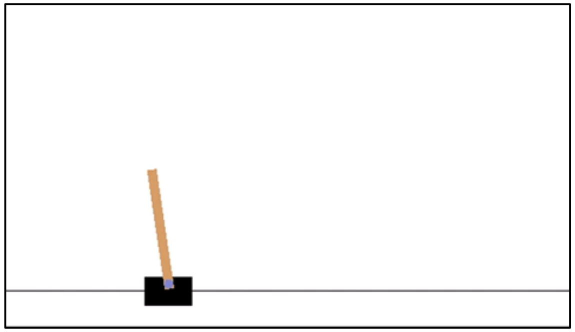
Lillicrap et al., "Continuous Control with Deep Reinforcement Learning," ICLR, 2016.

DDPG in Simulated Physics

Goal: end-to-end learning of control policy from pixels

- Input: state is stack of raw pixels from last 4 frames
- Output: two separate CNNs for Q and π





Concluding Remarks

RL is a general purpose framework for **decision making** under interactions between agent and environment

Policy gradient

- learns a policy that maps from state to action

Actor-critic

- estimates value function $Q^{\pi}(s, a), V^{\pi}(s)$
- updates policy based on the value function evaluation π