

# Week 6: Policy Optimization I: Policy Gradient

Bolei Zhou

*The Chinese University of Hong Kong*

October 13, 2020

# Annoucement

- ① Homework 2 due soon
- ② Homework 3 will come out

# This Week's Plan

- ① Introduction
- ② Policy-based Reinforcement Learning
- ③ Monte-Carlo Policy Gradient

# Value-based RL versus Policy-based RL

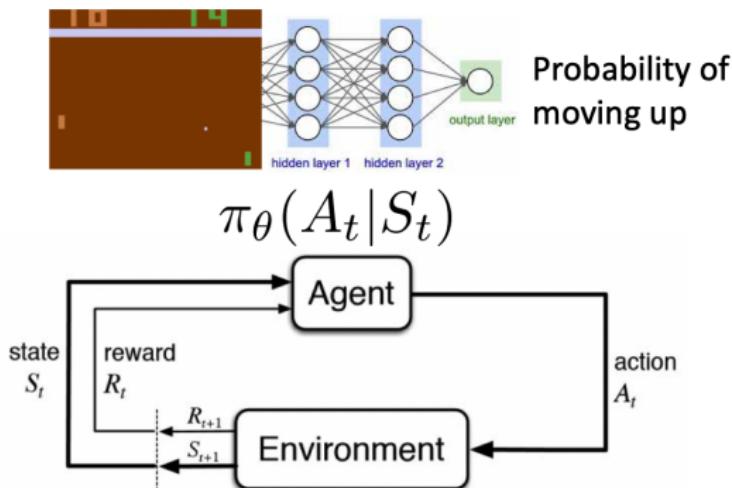
- ① In previous lectures, we focused on value-based RL and had value function approximation with parameter  $\theta$

$$V_\theta(s) \approx v^\pi(s)$$
$$q_\theta(s, a) \approx q^\pi(s, a)$$

- ② A policy is generated directly from the value function using  $\epsilon$ -greedy or greedy  $a_t = \arg \max_a Q(a, s_t)$
- ③ Instead, we can also parameterize the policy function as  $\pi_\theta(a|s)$  where  $\theta$  is the learnable policy parameter

# RL Diagram in the View of Policy Optimization

- ① Action from the policy is all we need, then let's optimize the policy directly!



# Value-based RL versus Policy-based RL

## ① Value-based RL

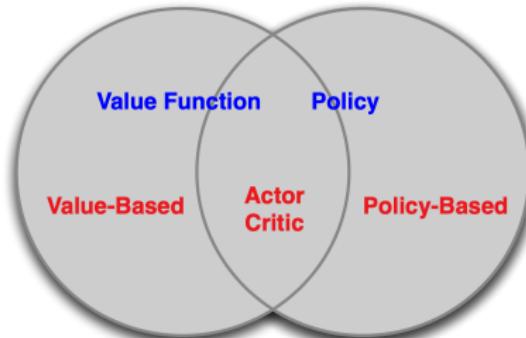
- ① to learn value function
- ② implicit policy based on the value function

## ② Policy-based RL

- ① no value function
- ② to learn policy directly

## ③ Actor-critic

- ① to learn both policy and value function



# Advantages of Policy-based RL

## ① Advantages:

- ① better convergence properties: we are guaranteed to converge on a local optimum (worst case) or global optimum (best case)
- ② Policy gradient is more effective in high-dimensional action space
- ③ Policy gradient can learn stochastic policies, while value function can't

## ② Disadvantages:

- ① typically converges to a local optimum
- ② evaluating a policy is inefficient and has high variance

## Two Types of Policies

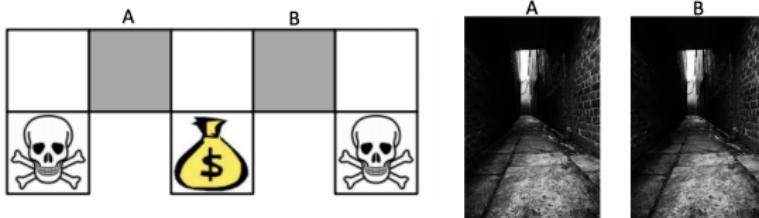
- ① Deterministic: given a state, the policy returns the action to take
- ② Stochastic: given a state, the policy returns a probability distribution of the actions (e.g., 40% chance to turn left, 60% chance to turn right)

# Example: Rock-Paper-Scissors



- ① Two-player game
- ② What is the best policy?
  - ① It is easy to beat a deterministic policy
  - ② A uniform random policy is optimal (Nash equilibrium)

## Example: Aliased Gridworld



- ① The agent cannot differentiate the two grey states (look the same to the agent)
- ② Considers the following features (for all N, E, S, W)

$$\psi(s, a) = [\mathbf{1}(\text{wall to N}, a = \text{move E}), \mathbf{1}(\text{wall to S}, a = \text{move W}), \dots]$$

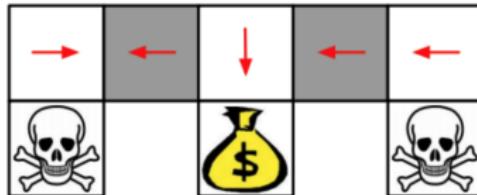
- ③ If it is value-based RL, value function approximation as:

$$Q_\theta(s, a) = f(\psi(s, a), \theta)$$

- ④ If it is policy-based RL, policy function approximation as:

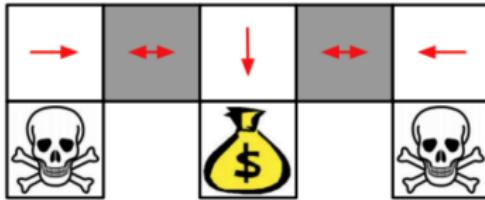
$$\pi_\theta(s, a) = g(\psi(s, a), \theta)$$

## Example: Aliased Gridworld



- ① Value-based RL learns a near-deterministic policy, e.g., greedy or  $\epsilon$ -greedy
- ② Because of the aliasing (the agent cannot differentiate two states), an optimal deterministic policy from value-based RL will either
  - ① move W in both grey states (shown by red arrows)
  - ② move E in both grey states
- ③ Either way, 50% chance will get stuck

## Example: Aliased Gridworld



- ① Policy-based RL can learn the optimal stochastic policy
- ② An optimal stochastic policy will randomly move E or W in two grey states

$$\pi_{\theta}(\text{move E} | \text{wall to N and S}) = 0.5$$

$$\pi_{\theta}(\text{move W} | \text{wall to N and S}) = 0.5$$

- ③ For any starting point, it will reach the goal state in a few steps with high probability

# Objective of Optimizing Policy

- ① Objective: Given a policy approximator  $\pi_\theta(s, a)$  with parameter  $\theta$ , find the best  $\theta$
- ② How do we measure the quality of a policy  $\pi_\theta$ ?
- ③ In episodic environments we can use the start value

$$J_1(\theta) = V^{\pi_\theta}(s_1)$$

- ④ In continuing environments
  - ① we can use the average value

$$J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$$

- ② or the average reward per time-step

$$J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s, a) R(s, a)$$

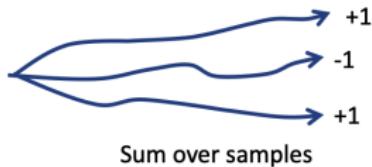
where  $d^{\pi_\theta}$  is stationary distribution of Markov chain for  $\pi_\theta$

# Objective of Optimizing Policy

- ① The value of the policy is defined as

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_t r(s_t, a_t^\tau) \right] \\ &\approx \frac{1}{m} \sum_m \sum_t r(s_{t,m}, a_{t,m}) \end{aligned}$$

It is the same as the value function we defined in the value-based RL



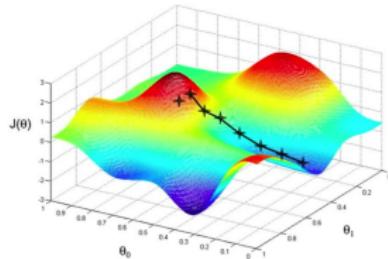
- ①  $\tau$  is a trajectory sampled from the policy function  $\pi_\theta$
- ② The goal of policy-based RL

$$\theta^* = \arg \max_{\theta} \mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_t r(s_t, a_t^\tau) \right]$$

# Objective of Optimizing Policy

- ① Policy-based RL is an optimization problem that find  $\theta$  that maximizes  $J(\theta)$
- ② If  $J(\theta)$  is differentiable, we can use gradient-based methods:
  - ① gradient ascend
  - ② conjugate gradient
  - ③ quasi-newton
- ③ If  $J(\theta)$  is non-differentiable or hard to compute the derivative, some derivative-free black-box optimization methods:
  - ① Cross-entropy method (CEM)
  - ② Evolution strategy
  - ③ Finite difference

# Policy Optimization using Derivative



- ① Consider a function  $J(\theta)$  to be any policy objective function
- ② Goal is to find parameter  $\theta^*$  that maximizes  $J(\theta)$  by ascending the gradient of the policy, w.r.t parameter  $\theta$

$$\Delta\theta = \alpha \nabla_{\theta} J(\theta)$$

- ③ Adjust  $\theta$  in the direction of the gradient, where  $\alpha$  is step-size
- ④ Define the gradient of  $J(\theta)$  to be

$$\nabla_{\theta} J(\theta) = \left( \frac{\partial J(\theta)}{\partial \theta_1}, \frac{\partial J(\theta)}{\partial \theta_2}, \dots, \frac{\partial J(\theta)}{\partial \theta_n} \right)^T$$

# Policy Optimization using Derivative-free Methods

- ① Sometimes we cannot compute the derivative, i.e.,  $\nabla_{\theta} J(\theta)$
- ② Derivative free methods:
  - ① Evolution Strategy (ES)
  - ② Cross Entropy Method (CEM)
  - ③ Finite Difference

# Derivative-free Method: Evolution Strategy (ES)

- ①  $\theta^* = \arg \max J(\theta)$
- ② Treat  $J(\theta)$  as a black box score function (not differentiable)

---

## Algorithm 1 Evolution Strategy

---

```
1: for iter  $t = 0, 1, 2, \dots$  do
2:   sample  $\epsilon_1, \dots, \epsilon_n \sim \mathcal{N}(0, I)$  to generate perturbed  $\theta_i = \theta_t + \delta\epsilon_i$ 
3:   execute roll-out under  $\theta_i$  for  $i = 1, \dots, n$  to compute  $J(\theta_i)$ 
4:   set  $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\delta} \sum_{i=1}^n J(\theta_i) \epsilon_i$ 
5: end for
```

---

- ③ ES is easy to implement and scale. Running on a computing cluster of 80 machines and 1,440 CPU cores, the implementation is able to train a 3D MuJoCo humanoid walker in only 10 minutes
  - ① <https://openai.com/blog/evolution-strategies/>
- ④ Example code: [https://github.com/cuhkrlcourse/RLexample/blob/master/derivativefree/es\\_cem\\_agent.py](https://github.com/cuhkrlcourse/RLexample/blob/master/derivativefree/es_cem_agent.py)

# Derivative-free Method: Cross-Entropy Method

- ①  $\theta^* = \arg \max J(\theta)$
- ② Treat  $J(\theta)$  as a black box score function (not differentiable)

---

## Algorithm 2 CEM

---

```
1: for iter  $t = 1, 2, 3, \dots$  do
2:    $\mathcal{C} = \{\}$ 
3:   for parameter set  $e = 1$  to  $n$  do
4:     sample  $\theta^{(e)} \sim P_{\mu^{(t)}}(\theta)$ 
5:     execute roll-outs under  $\theta^{(e)}$  to evaluate  $J(\theta^{(e)})$ 
6:     store  $(\theta^e, J(\theta^{(e)}))$  in  $\mathcal{C}$ 
7:   end for
8:    $\mu^{(i+1)} = \arg \max_{\mu} \sum_{k \in \hat{\mathcal{C}}} \log P_{\mu}(\theta^{(k)})$ 
    where  $\hat{\mathcal{C}}$  are the top 10% of  $\mathcal{C}$  ranked by  $J(\theta^{(e)})$ 
9: end for
```

- 
- ③ Example code: [https://github.com/cuhkrlcourse/RLexample/  
blob/master/derivativefree/es\\_cem\\_agent.py](https://github.com/cuhkrlcourse/RLexample/blob/master/derivativefree/es_cem_agent.py)

# Approximate Gradients by Finite Difference

- ① To evaluate policy gradient of  $\pi_\theta(s, a)$
- ② For each dimension  $k \in [1, n]$ 
  - ① estimate  $k$ th partial derivative of objective function by perturbing  $\theta$  by a small amount  $\delta$  in  $k$ th dimension

$$\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta + \delta u_k) - J(\theta)}{\delta}$$

where  $u_k$  is unit vector with 1 in  $k$ th component, 0 elsewhere

- ③ use  $n$  evaluations to compute policy gradient in total  $n$  dimensions
- ④ Though noisy and inefficient, it works for arbitrary policies, even if policy is not differentiable.

# Compute the Policy Gradient Analytically

- ①  $\pi_\theta$  is a probability function or density function with parameter  $\theta$
- ② Assume policy  $\pi_\theta$  is differentiable whenever it is no-zero
- ③ We can compute the gradient  $\nabla_\theta \pi_\theta(s, a)$
- ④ Likelihood ratio exploits the following trick

$$\begin{aligned}\nabla_\theta \pi_\theta(s, a) &= \pi_\theta(s, a) \frac{\nabla_\theta \pi_\theta(s, a)}{\pi_\theta(s, a)} \\ &= \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a)\end{aligned}$$

- ⑤ The score function is  $\nabla_\theta \log \pi_\theta(s, a)$ 
  - ① Score indicates the steepness of the log-likelihood function
  - ② [https://en.wikipedia.org/wiki/Score\\_\(statistics\)](https://en.wikipedia.org/wiki/Score_(statistics))

# Policy Gradient for One-Step MDPs

- ① Consider a simple class of one-step MDPs
  - ① Start in state  $s \sim d(s)$
  - ② Terminate after one time-step with reward  $r = R(s, a)$
- ② Use likelihood ratios to compute the policy gradient

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\pi_\theta}[r] \\ &= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s, a) r \end{aligned}$$

- ③ The gradient is as

$$\begin{aligned} \nabla_\theta J(\theta) &= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a) r \\ &= \mathbb{E}_{\pi_\theta}[r \nabla_\theta \log \pi_\theta(s, a)] \end{aligned}$$

# Discrete Policy Function: Softmax Policy

- ① Simple policy model: weight actions using linear combination of features  $\phi(s, a)^T \theta$
- ② Probability of action is proportional to the exponentiated weight

$$\pi_\theta(s, a) = \frac{\exp^{\phi(s, a)^T \theta}}{\sum_{a'} \exp^{\phi(s, a')^T \theta}}$$

- ③ The score function is

$$\nabla_\theta \log \pi_\theta(s, a) = \phi(s, a) - \mathbb{E}_{\pi_\theta}[\phi(s, .)]$$

# Continuous Policy Function: Gaussian Policy

- ① In continuous action spaces, a Gaussian policy can be naturally defined
- ② Mean is a linear combination of state features  $\mu(s) = \phi(s)^T \theta$
- ③ Variance may be fixed  $\sigma^2$  or can also be parameterized
- ④ Policy is Gaussian value  $a \sim \mathcal{N}(\mu(s), \sigma^2)$
- ⑤ The score function is

$$\nabla_{\theta} \log \pi_{\theta}(s, a) = \frac{(a - \mu(s))\phi(s)}{\sigma^2}$$

# Policy Gradient for Multi-step MDPs

- ① Denote a state-action trajectory from one episode as  
 $\tau = (s_0, a_0, r_1, \dots, s_{T-1}, a_{T-1}, r_T, s_T) \sim (\pi_\theta, P(s_{t+1}|s_t, a_t))$
- ② Use  $R(\tau) = \sum_{t=0}^T R(s_t, a_t)$  to be the sum of rewards for a trajectory  $\tau$
- ③ The policy objective is

$$J(\theta) = \mathbb{E}_{\pi_\theta} \left[ \sum_{t=0}^T R(s_t, a_t) \right] = \sum_{\tau} P(\tau; \theta) R(\tau)$$

where  $P(\tau; \theta)$  denotes the probability over trajectories when executing the policy  $\pi_\theta$

- ④ Then our goal is to find the policy parameter  $\theta$

$$\theta^* = \arg \max_{\theta} J(\theta) = \arg \max_{\theta} \sum_{\tau} P(\tau; \theta) R(\tau)$$

# Policy Gradient for Multi-step MDPs

- ① Our goal is to find the policy parameter  $\theta$

$$\theta^* = \arg \max_{\theta} J(\theta) = \arg \max_{\theta} \sum_{\tau} P(\tau; \theta) R(\tau)$$

- ② Take the gradient with respect to  $\theta$ :

$$\begin{aligned}\nabla_{\theta} J(\theta) &= \nabla_{\theta} \sum_{\tau} P(\tau; \theta) R(\tau) \\ &= \sum_{\tau} \nabla_{\theta} P(\tau; \theta) R(\tau) \\ &= \sum_{\tau} \frac{P(\tau; \theta)}{P(\tau; \theta)} \nabla_{\theta} P(\tau; \theta) R(\tau) \\ &= \sum_{\tau} P(\tau; \theta) R(\tau) \frac{\nabla_{\theta} P(\tau; \theta)}{P(\tau; \theta)} \\ &= \sum_{\tau} P(\tau; \theta) R(\tau) \nabla_{\theta} \log P(\tau; \theta)\end{aligned}$$

# Policy Gradient for Multi-step MDPs

- ① Our goal is to find the policy parameter  $\theta$

$$\theta^* = \arg \max_{\theta} J(\theta) = \arg \max_{\theta} \sum_{\tau} P(\tau; \theta) R(\tau)$$

- ② Take the gradient with respect to  $\theta$ :

$$\nabla_{\theta} J(\theta) = \sum_{\tau} P(\tau; \theta) R(\tau) \nabla_{\theta} \log P(\tau; \theta)$$

- ③ Approximate the gradient of objective with empirical estimate for  $m$  sample paths under policy  $\pi_{\theta}$ :

$$\nabla_{\theta} J(\theta) \approx \frac{1}{m} \sum_{i=1}^m R(\tau_i) \nabla_{\theta} \log P(\tau_i; \theta)$$

# Decompose a Trajectory

- ① Approximate with empirical estimate for  $m$  sample paths under policy  $\pi_\theta$ :

$$\nabla_\theta J(\theta) \approx \frac{1}{m} \sum_{i=1}^m R(\tau_i) \nabla_\theta \log P(\tau_i; \theta)$$

- ② Decompose  $\nabla_\theta \log P(\tau; \theta)$

$$\begin{aligned}\nabla_\theta \log P(\tau; \theta) &= \nabla_\theta \log \left[ \mu(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t | s_t) p(s_{t+1} | s_t, a_t) \right] \\ &= \nabla_\theta \left[ \log \mu(s_0) + \sum_{t=0}^{T-1} \log \pi_\theta(a_t | s_t) + \log p(s_{t+1} | s_t, a_t) \right] \\ &= \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t)\end{aligned}$$

# Likelihood Ratio Policy Gradient

- ① Our goal is to find the policy parameter  $\theta$

$$\theta^* = \arg \max_{\theta} J(\theta) = \arg \max_{\theta} \sum_{\tau} P(\tau; \theta) R(\tau)$$

- ② Approximate with empirical estimate for  $m$  sample paths under policy  $\pi_\theta$ :

$$\nabla_{\theta} J(\theta) \approx \frac{1}{m} \sum_{i=1}^m R(\tau_i) \nabla_{\theta} \log P(\tau_i; \theta)$$

- ③ Thus we have  $\nabla_{\theta} \log P(\tau_i; \theta) = \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$

$$\nabla_{\theta} J(\theta) \approx \frac{1}{m} \sum_{i=1}^m R(\tau_i) \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t^i | s_t^i)$$

- ④ It does not need to know the transition dynamics!

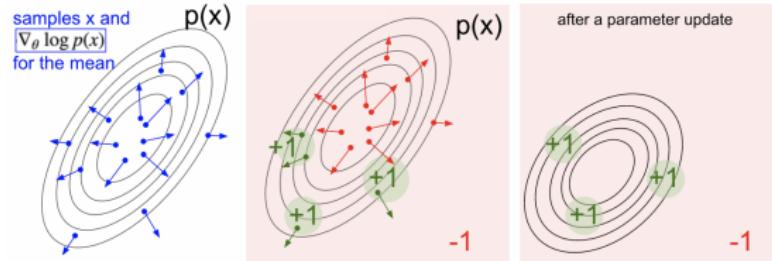
# Understand Score Function Gradient Estimator

- ① Consider the generic form of  $E_{\tau \sim \pi_\theta}[R(\tau)]$  as

$$\nabla_\theta \mathbb{E}_{x \sim p(x; \theta)}[f(x)] = \mathbb{E}_x[f(x) \nabla_\theta \log p(x; \theta)]$$

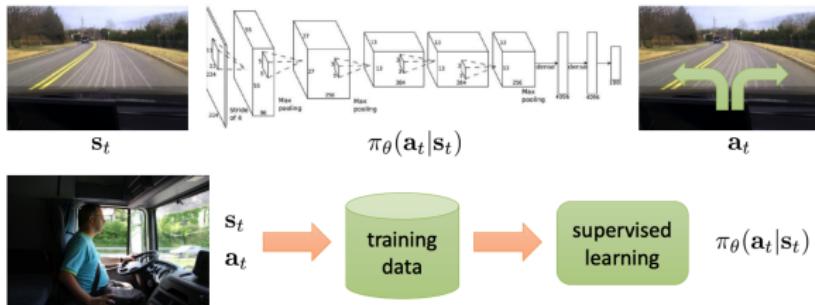
- ② The above gradient can be understood as:

- ① Shift the distribution  $p$  through its parameter  $\theta$  to let its future samples  $x$  achieve higher scores as judged by  $f(x)$
- ② The direction of  $f(x) \nabla_\theta \log p(x; \theta)$  pushes up the log probability of the sample, in proportion to how good it is



# Comparison to Maximum Likelihood

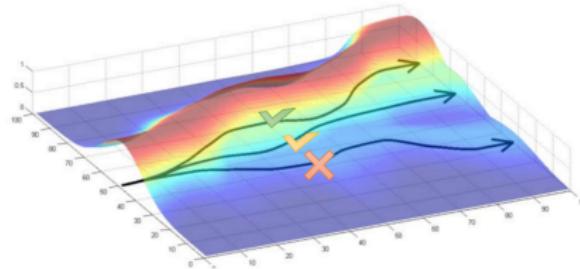
- ① Policy gradient estimator:  $\nabla_{\theta} J(\theta) \approx \frac{1}{M} \sum_{m=1}^M \left( \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(a_{t,m}, s_{t,m}) \right) \left( \sum_{t=1}^T r(s_{t,m}, a_{t,m}) \right)$
- ② Maximum likelihood estimator:  
 $\nabla_{\theta} J_{ML}(\theta) \approx \frac{1}{M} \sum_{m=1}^M \left( \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(a_{t,m}, s_{t,m}) \right)$
- ③ good action is made more likely, bad action is made less likely



# Comparison to Maximum Likelihood

$$\nabla_{\theta} J(\theta) \approx \frac{1}{m} \sum_{i=1}^m R(\tau_i) \nabla_{\theta} \log P(\tau_i; \theta)$$

- ① If going up the hill leads to higher reward, change the policy parameters to increase the likelihood of trajectories that move higher



# Large Variance of Policy Gradient

- ① We have the following approximate update

$$\nabla_{\theta} J(\theta) \approx \frac{1}{m} \sum_{i=1}^m R(\tau_i) \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t^i | s_t^i)$$

- ② Unbiased but having high variance as we use the MC method
- ③ Two fixes:
  - ① Use temporal causality
  - ② Include a baseline

# Reduce Variance of Policy Gradient using Causality

- ① Previously  $\nabla_{\theta} \mathbb{E}_{\tau}[R] = \mathbb{E}_{\tau} \left[ \left( \sum_{t=0}^{T-1} r_t \right) \left( \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right) \right]$
- ② We can derive the gradient estimator for a single reward term  $r_{t'}$  as

$$\nabla_{\theta} \mathbb{E}_{\tau}[r_{t'}] = \mathbb{E}_{\tau} \left[ r_{t'} \sum_{t=0}^{t'} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]$$

- ③ Summing this formula over  $t$ , we obtain

$$\begin{aligned}\nabla_{\theta} J(\theta) &= \nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}}[R] = \mathbb{E}_{\tau} \left[ \sum_{t'=0}^{T-1} r_{t'} \sum_{t=0}^{t'} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right] \\ &= \mathbb{E}_{\tau} \left[ \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \sum_{t'=t}^{T-1} r_{t'} \right] \\ &= \mathbb{E}_{\tau} \left[ \sum_{t=0}^{T-1} G_t \cdot \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]\end{aligned}$$

# Reduce Variance of Policy Gradient using Causality

- ① Therefore we have

$$\nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}}[R] = \mathbb{E}_{\tau} \left[ \sum_{t=0}^{T-1} G_t \cdot \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]$$

- ②  $G_t = \sum_{t'=t}^{T-1} r_{t'}$  is the return for a trajectory at step  $t$
- ③ Causality: policy at time  $t'$  cannot affect reward at time  $t$  when  $t < t'$
- ④ Then we can have the following estimated update

$$\nabla_{\theta} \mathbb{E}[R] \approx \frac{1}{m} \sum_{i=1}^m \sum_{t=0}^{T-1} G_t^{(i)} \cdot \nabla_{\theta} \log \pi_{\theta}(a_t^i | s_t^i)$$

# REINFORCE: a Monte-Carlo policy gradient algorithm

- ① The algorithm simply samples multiple trajectories following the policy  $\pi_\theta$  while updating  $\theta$  using the estimated gradient

## REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization  $\pi(a|s, \theta)$

Initialize policy parameter  $\theta \in \mathbb{R}^{d'}$

Repeat forever:

    Generate an episode  $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$ , following  $\pi(\cdot| \cdot, \theta)$

    For each step of the episode  $t = 0, \dots, T - 1$ :

$G \leftarrow$  return from step  $t$

$\theta \leftarrow \theta + \alpha \gamma^t G \nabla_\theta \ln \pi(A_t | S_t, \theta)$

- ② Classic paper: Williams (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning: introduces REINFORCE algorithm

# Reduce Variance Using a Baseline

## ① The original update

$$\nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}}[R] = \mathbb{E}_{\tau} \left[ \sum_{t=0}^{T-1} G_t \cdot \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]$$

- ①  $G_t = \sum_{t'=t}^{T-1} r_{t'}$  is the return for a trajectory which might have high variance
- ② We subtract a baseline  $b(s)$  from the policy gradient to reduce variance

$$\nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}}[R] = \mathbb{E}_{\tau} \left[ \sum_{t=0}^{T-1} (G_t - b(s_t)) \cdot \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]$$

- ③ A good baseline is the expected return

$$b(s_t) = \mathbb{E}[r_t + r_{t+1} + \dots + r_{T-1}]$$

## Reduce Variance Using a Baseline

$$\nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}}[R] = \mathbb{E}_{\tau} \left[ \sum_{t=0}^{T-1} (\textcolor{red}{G_t} - b(s_t)) \cdot \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]$$

- ① Interpretation: increase the logprob of action  $a_t$  proportional to how much returns  $G_t$  are better than the expected return
- ② We can **prove** that baseline  $b(s)$  can reduce variance, without changing the expectation:

$$\mathbb{E}_{\tau} \left[ \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) b(s_t) \right] = 0, \quad (1)$$

$$E_{\tau} \left[ \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) (G_t - b(s_t)) \right] = E_{\tau} \left[ \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) G_t \right] \quad (2)$$

$$Var_{\tau} \left[ \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) (G_t - b(s_t)) \right] < Var_{\tau} \left[ \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) G_t \right] \quad (3)$$

- ③ Thus subtracting a baseline is unbiased in expectation but reduces variance

# Vanilla Policy Gradient Algorithm with Baseline

**procedure** POLICY GRADIENT( $\alpha$ )

    Initialize policy parameters  $\theta$  and baseline values  $b(s)$  for all  $s$ , e.g. to 0

**for** iteration = 1, 2, ... **do**

        Collect a set of  $m$  trajectories by executing the current policy  $\pi_\theta$

**for** each time step  $t$  of each trajectory  $\tau^{(i)}$  **do**

            Compute the *return*  $G_t^{(i)} = \sum_{t'=t}^{T-1} r_{t'}$

            Compute the *advantage estimate*  $\hat{A}_t^{(i)} = G_t^{(i)} - b(s_t)$

        Re-fit the baseline to the empirical returns by updating  $\mathbf{w}$  to minimize

$$\sum_{i=1}^m \sum_{t=0}^{T-1} \|b(s_t) - G_t^{(i)}\|^2$$

    Update policy parameters  $\theta$  using the policy gradient estimate  $\hat{g}$

$$\hat{g} = \sum_{i=1}^m \sum_{t=0}^{T-1} \hat{A}_t^{(i)} \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)})$$

    with an optimizer like SGD ( $\theta \leftarrow \theta + \alpha \cdot \hat{g}$ ) or Adam  
    **return**  $\theta$  and baseline values  $b(s)$

- ① Classic paper: Sutton, McAllester, Singh, Mansour (1999). Policy gradient methods for reinforcement learning with function approximation: actor-critic algorithms with value function approximation

# Practical Implementation of the Algorithm

- ① In practice we usually do not compute the gradients  $\sum_t \hat{A}_t \nabla_\theta \log \pi_\theta(a_t|s_t)$  individually
- ② Instead, we accumulate data from current batch as

$$L(\theta) = \sum_t \hat{A}_t \log \pi_\theta(a_t|s_t; \theta)$$

- ③ Then the policy gradient estimator  $\hat{g} = \nabla_\theta L(\theta)$
- ④ We also could have a joint loss with value function approximation as

$$L(\theta, \mathbf{w}) = \sum_t \left( \hat{A}_t \log \pi_\theta(a_t|s_t; \theta) - \|b(s_t) - \hat{R}_t\|^2 \right)$$

- ⑤ Then solve this using auto diff

# Short Summary of Policy Optimization

- ① Softmax policy: weight actions using linear combination of features  
 $\phi(s, a)^T \theta$

$$\pi_\theta(s, a) = \frac{\exp^{\phi(s, a)^T \theta}}{\sum_{a'} \exp^{\phi(s, a')^T \theta}}$$

- ② Gaussian policy:
  - ① Mean is a linear combination of state features  $\mu(s) = \phi(s)^T \theta$
  - ② Variance may be fixed  $\sigma^2$  or can also be parameterized
  - ③ Policy is Gaussian, the continuous  $a \sim \mathcal{N}(\mu(s), \sigma^2)$
- ③ Neural network policy:  $\pi_\theta(a|s) = F(s)$  where  $F(\cdot)$  is a neural network

# Short Summary of Policy Optimization

- ① In policy optimization, for the policy function  $\pi_\theta$  the objective is to maximize

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$$

- ①  $R(\tau)$  could be any reasonable reward function on  $\tau$
- ② So the gradient is

$$\begin{aligned}\nabla_\theta J(\theta) &= \nabla_\theta \sum_{\tau} P(\tau; \theta) R(\tau) \\ &= \mathbb{E}_\tau [R(\tau) \nabla_\theta \log P(\tau; \theta)]\end{aligned}$$

- ① **log likelihood trick:**  $\nabla_\theta f_\theta(x) = f_\theta(x) \nabla_\theta \log f_\theta(x)$
- ② Trajectory distribution:  $P(\tau; \theta) = \mu(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t | s_t) p(s_{t+1} | s_t, a_t)$
- ③ **cancelling trick:**  $\nabla_\theta \log P(\tau; \theta) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t)$

# Short Summary of Policy Gradient

- ① After decomposing  $\tau$  we derived that

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau} \left[ \left( \sum_{t=0}^{T-1} r_t \right) \left( \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right) \right]$$

- ② After applying the causality, we have

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau} \left[ \sum_{t=0}^{T-1} G_t \cdot \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]$$

- ①  $G_t = \sum_{t'=t}^{T-1} r_{t'}$  is the return for a trajectory at step  $t$

# Policy Gradient is On-Policy RL

- ①  $\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta}} [\nabla_{\theta} \log P_{\theta}(\tau) r(\tau)]$
- ② In REINFORCE algorithm: there is the on-policy sampling (sample  $\{\tau\}$  from  $\pi_{\theta}$ )
- ③ On-policy learning is stable but might be inefficient

# Off-Policy Policy Gradient with Importance Sampling

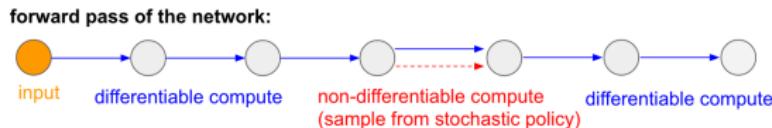
- ①  $\theta^* = \arg \max_{\theta} J(\theta)$  where  $J(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta}} [r(\tau)]$
- ② Let's say we have a behavior policy  $\hat{\pi}$  where we can generate samples, then

$$J(\theta) = \mathbb{E}_{\tau \sim \hat{\pi}} \left[ \frac{\pi_{\theta}(\tau)}{\hat{\pi}(\tau)} r(\tau) \right]$$

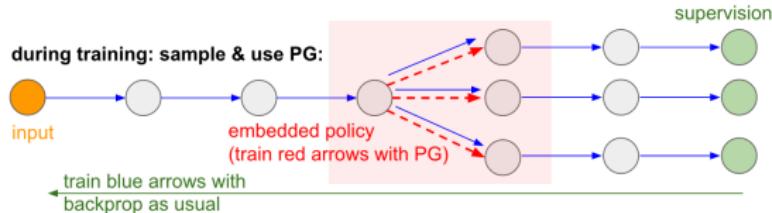
- ③ Then we can derive the off-policy policy gradient:
  - ① The basic idea lies in the TRPO and PPO algorithms

# Overcoming Non-differentiable Computation

- ① Another interesting strength of Policy Gradient is that it allows us to overcome the non-differentiable computation



- ② During training we will produce several samples (indicated by the branches below), and then we'll encourage samples that eventually led to good outcomes (in this case for example measured by the loss at the end)



# Extension of Policy Gradient

- ① State-of-the-art RL methods are almost all policy-based
  - ① **A2C and A3C:** Asynchronous Methods for Deep Reinforcement Learning, ICML'16. Representative high-performance actor-critic algorithm: <https://openai.com/blog/baselines-acktr-a2c/>
  - ② **TRPO:** Schulman, L., Moritz, Jordan, Abbeel (2015). Trust region policy optimization: deep RL with natural policy gradient and adaptive step size
  - ③ **PPO:** Schulman, Wolski, Dhariwal, Radford, Klimov (2017). Proximal policy optimization algorithms: deep RL with importance sampled policy gradient

# Summary

- ① Derive the policy gradient by yourself to get a deeper understanding!