

# Lecture 11: Policy Optimization I

Bolei Zhou

The Chinese University of Hong Kong

*bzhou@ie.cuhk.edu.hk*

March 3, 2020

# Today's Plan

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

# Value-based RL versus Policy-based RL

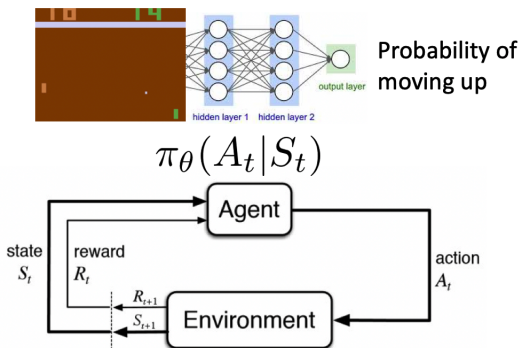
- 1 In previous lectures, we focused on value-based RL and had value function approximation with parameter  $\theta$

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

- 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

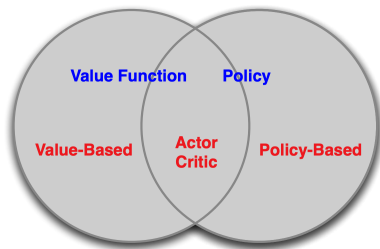
# RL Diagram in the View of Policy Optimization

- 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)

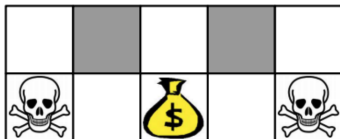
# Example: Rock-Paper-Scissors



- ① Two-player game
- ② What is the best policy?
  - ① A deterministic policy is easily beaten
  - ② A uniform random policy is optimal (Nash equilibrium)



## Example: Aliased Gridworld 1



- 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)

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

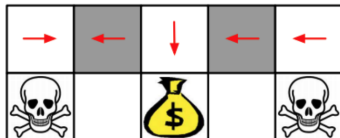
- 3 If it is value-based RL, value function approximation as:

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

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

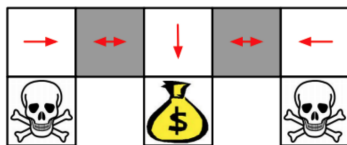
$$\pi_{\theta}(s, a) = g(\psi(s, a), \theta)$$

## Example: Aliased Gridworld 2



- 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 3



- 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

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

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

- 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

$$J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[v_1]$$

- 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

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

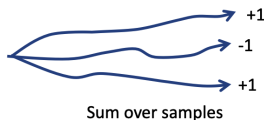
where  $d^{\pi_\theta}$  is stationary distribution of Markov chain for  $\pi_\theta$

# 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



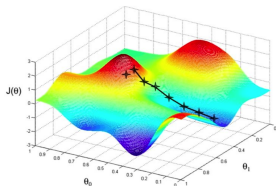
- 1  $\tau$  is a trajectory sampled from the policy function  $\pi_{\theta}$
- 2 The goal of policy-based RL

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

# 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 Hill climbing
  - 3 Evolution algorithm

# 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  $J(\mathbf{w})$  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 Cross Entropy Method (CEM)
  - ② Finite Difference



# 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 1** CEM for black-box function optimization

---

```
1: for iter  $i = 1$  to  $N$  do
2:    $\mathcal{C} = \{\}$ 
3:   for parameter set  $e = 1$  to  $N$  do
4:     sample  $\theta^{(e)} \sim P_{\mu^{(i)}}(\theta)$ 
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 of CEM for a simple RL problem: [https://github.com/metalbubble/RLexample/blob/master/my\\_learning\\_agent.py](https://github.com/metalbubble/RLexample/blob/master/my_learning_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  $kth$  partial derivative of objective function by perturbing  $\theta$  by a small amount  $\epsilon$  in  $kth$  dimension

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

where  $u_k$  is unit vector with 1 in  $kth$  component, 0 else where

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

# Computing the Policy Gradient Analytically

- ① Assume policy  $\pi_\theta$  is differentiable whenever it is no-zero
- ② and we can compute the gradient  $\nabla_\theta \pi_\theta(s, a)$
- ③ Likelihood ratios exploit the following tricks

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

- ④ The score function is  $\nabla_\theta \log \pi_\theta(s, a)$

# Policy Gradient for One-Step MDPs

- 1 Consider a simple class of one-step MDPs
  - 1 Starting in state  $s \sim d(s)$
  - 2 Terminating 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

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

# Policy Example: 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

$$\pi_{\theta}(s, a) = \frac{\exp^{\phi(s, a)^T \theta}}{\sum_b \exp^{\phi(s, b)^T \theta}}$$

- 3 The score function is

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

# Policy Example: 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, the continuous  $a \sim \mathcal{N}(\mu(s), \sigma^2)$
- 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 with empirical estimate for  $m$  sample paths under policy  $\pi_{\theta}$ :

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

# Decomposing the Trajectories into States and Actions

- 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  $\theta$

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

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

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

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

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

- ① It do not need to know the dynamics model!

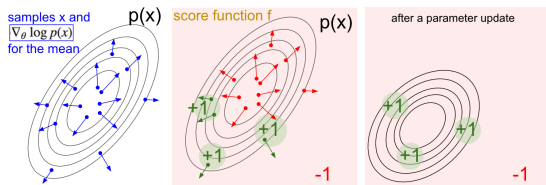
# Understanding Score Function Gradient Estimator

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

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

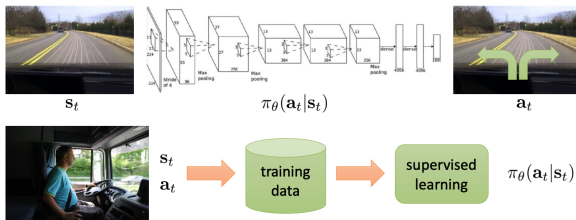
- 2 The above gradient can be understood as:

- 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

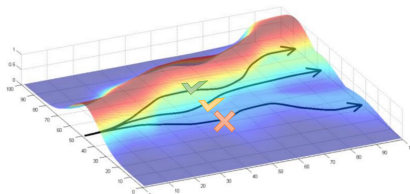
- 1 Policy gradient estimator:  $\nabla_{\theta} J(\theta) \approx \frac{1}{M} \sum_{m=1}^M \left( \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(a_{t,m} | s_{t,m}) \right) \left( \sum_{t=1}^T r(s_{t,m}, a_{t,m}) \right)$
- 2 Maximum likelihood estimator:  $\nabla_{\theta} J_{ML}(\theta) \approx \frac{1}{M} \sum_{m=1}^M \left( \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(a_{t,m} | s_{t,m}) \right)$
- 3 Interpretation: **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 very noisy
- 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

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



# 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

# Reducing 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

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

# Reducing Variance Using a Baseline

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

- 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:

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

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

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

- 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  $b(s)$

# 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

# Concluding Remark

- ① Derive the policy gradient by yourself to get a deeper understanding!!!
- ② Next lecture will be on actor-critic policy gradient and some variants