

Policy Gradient

CMPUT 366: Intelligent Systems

S&B §13.0-13.3

Lecture Overview

1. Recap
2. Parameterized Policies
3. Policy Gradient Theorem
4. REINFORCE Algorithm

Recap:

Parameterized Value Functions

- A **parameterized value function**'s values are set by setting the values of a **weight vector** $\mathbf{w} \in \mathbb{R}^d$:

$$\hat{v}(s, \mathbf{w}) \approx v_{\pi}(s)$$

- \hat{v} could be a **linear function**: \mathbf{w} is the feature weights
- \hat{v} could be a **neural network**: \mathbf{w} is the weights, biases, kernels, etc.
- Many fewer weights than states: $d \ll |\mathcal{S}|$
 - Changing **one weight** changes the estimated value of **many states**
 - Updating a single state **generalizes** to affect many other states' values

Recap: Stochastic Gradient Descent

- **Stochastic Gradient Descent:** After each example, adjust weights a tiny bit in **direction** that would most **reduce error** on **that example**:

$$\begin{aligned}\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t - \frac{1}{2}\alpha \nabla \left[\boxed{v_\pi(S_t)} - \hat{v}(S_t, \mathbf{w}_t) \right]^2 \text{ error} \\ &= \mathbf{w}_t + \alpha \left[\boxed{v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)} \right] \nabla \hat{v}(s, \mathbf{w}_t)\end{aligned}$$

target

error

- We don't know $v_\pi(S_t)$, so we update toward **approximate target** U_t :

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha \left[U_t - \hat{v}(S_t, \mathbf{w}_t) \right] \nabla \hat{v}(s, \mathbf{w}_t)$$

Approaches to Control

1. **Action-value methods** (all previous approaches)

- Learn the value of **each action** in **each state**: $q_{\pi}(s,a)$
- Pick the **max-value action** (usually): $\arg \max_a q_{\pi}(s,a)$

2. **Function approximation** (last lecture)

- *Prediction*: Learn the **parameters** \mathbf{w} of state-value function $\hat{v}(s, \mathbf{w})$
- *Control*: Learn the **parameters** \mathbf{w} of action-value function $\hat{q}(s, \mathbf{w})$

3. **Policy-gradient methods** (today)

- Learn the **parameters** θ of a policy $\pi(a|s, \theta)$
- Update by gradient ascent in performance

Parameterized Policies

- The action probabilities of a **parameterized policy** $\pi(a | s, \theta)$ are set by setting the values of a **parameter vector** $\theta \in \mathbb{R}^{d'}$
- Common approach: **softmax in action preferences**
 - Learn an **action preference function** $h(s, a, \theta)$
 - **Softmax** over action preferences gives action probabilities:

$$\pi(a | s, \theta) \doteq \frac{e^{h(s, a, \theta)}}{\sum_{a'} e^{h(s, a', \theta)}}$$

Action Preferences

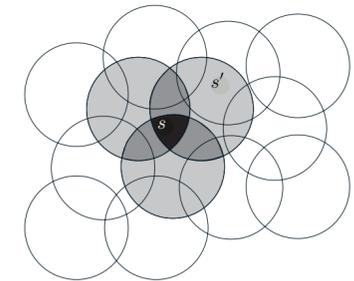
- **Question:** What functional forms can we use for action preferences?

- Anything we could have used for \hat{v} :

- **Linear approximations:** $h(s, a, \theta) \doteq \theta^T \mathbf{x}(s) = \sum_{i=1}^d \theta_i x_i(s)$

- Including coarse coding, tile coding

- **Neural network:** θ are weights, biases, kernels, etc.



Parameterized Policies

Advantage: Deterministic Action

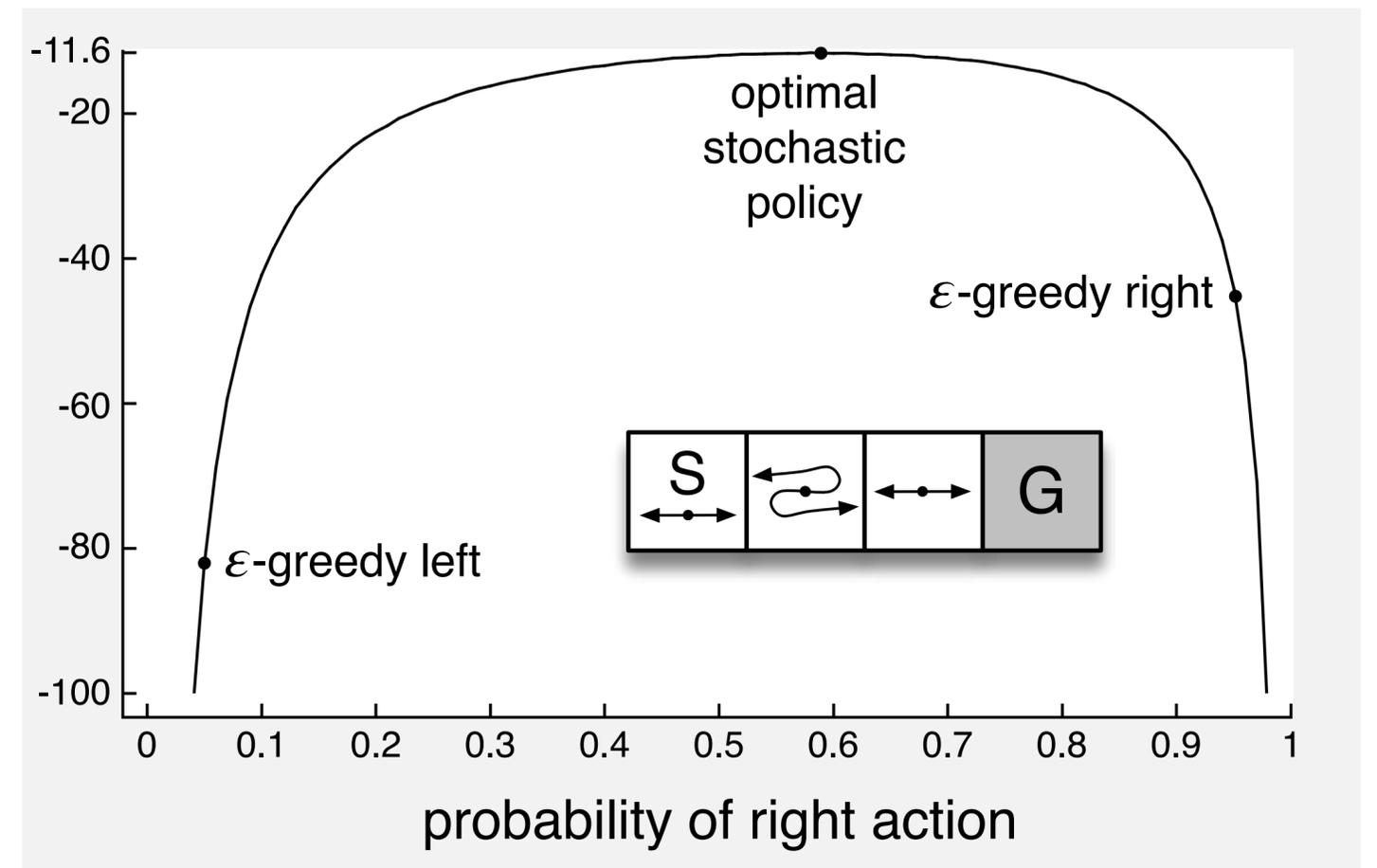
- The **optimal action** $\pi^*(a|s) = \arg \max q^*(s,a)$ is typically **deterministic**
- If we run an ε -soft policy, we cannot get to an optimal policy
 - **Every action** is played either with probability ε or $(1-\varepsilon)$
- Softmax in action preference policies can learn **arbitrary probabilities**, because $h(s,a,\theta)$ is completely **unconstrained**:

$$\pi(a | s, \theta) \doteq \frac{e^{h(s,a,\theta)}}{\sum_{a'} e^{h(s,a',\theta)}}$$

- **Question:** How can a softmax in action preferences policy converge to a deterministic policy?
- **Question:** Can you get the same results with $h(s,a,\theta) = \hat{q}(s,a,\theta)$? (**why?**)

Example: Switcheroo Corridor

- Actions **left** and **right** have usual effect
- Except in one state they are **reversed!**
- Function approximation makes all the states **look identical**
- **Optimal policy** is **stochastic**, with $\Pr(\text{right}) \approx 0.59$
- But ϵ -greedy policies can only pick $\Pr(\text{right})$ of ϵ or $(1-\epsilon)$!



Parameterized Policies

Advantage: Stochastic Actions

- Optimal policies are **deterministic**, but only when there is no **state aggregation**
- When **function approximation** makes states look the same, or when states are **imperfectly observable**, the optimal policy might be an **arbitrary probability distribution**
- Parameterized policies can represent **arbitrary** distributions
 - Although not necessarily arbitrary distributions in **every possible state (why not?)**

Policy Performance

- We choose the policy parameters θ in order to maximize the **performance** of the policy: $J(\theta)$
- **Question:** What should $J(\theta)$ be in episodic cases?
- **Expected returns** to the policy specified by θ :

$$J(\theta) \doteq \mathbb{E}_{\pi_{\theta}} [G_0]$$

- With special **single starting state** s_0 :

$$J(\theta) \doteq v_{\pi_{\theta}}(s_0)$$

Policy Gradient Ascent

1. Want to **maximize performance**: $J(\theta) = v_{\pi\theta}(S_0)$
2. Gradient gives direction that **J increases**: $\nabla J(\theta)$
3. Update parameters in **direction of the gradient**:

$$\begin{aligned}\theta_{t+1} &\leftarrow \theta_t + \alpha \nabla J(\theta_t) \\ &= \theta_t + \alpha \nabla v_{\pi\theta}(S_t)\end{aligned}$$

Oops!

Policy Gradient Theorem

- The **gradient of the policy** $\nabla J(\theta)$ is just the gradient of the value function with respect to the policy $v_{\pi\theta}(s_0)$
- But we **don't know** the gradient of the **value function**!

Policy Gradient Theorem:

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a q_{\pi}(s, a) \nabla \pi(a | s, \theta)$$

on-policy stationary distribution true action values gradient of policy

Monte Carlo Policy Gradient

$$\begin{aligned}\nabla J(\theta) &\propto \sum_s \mu(s) \sum_a q_\pi(s, a) \nabla \pi(a | s, \theta) \\ &= \mathbb{E}_\pi \left[\sum_a q_\pi(S_t, a) \nabla \pi(a | S_t, \theta) \right] \\ &= \mathbb{E}_\pi \left[\sum_a q_\pi(S_t, a) \nabla \pi(a | S_t, \theta) \frac{\pi(a | S_t, \theta)}{\pi(a | S_t, \theta)} \right] \\ &= \mathbb{E}_\pi \left[\sum_a \pi(a | S_t, \theta) q_\pi(S_t, a) \frac{\nabla \pi(a | S_t, \theta)}{\pi(a | S_t, \theta)} \right] \\ &= \mathbb{E}_\pi \left[q_\pi(S_t, A_t) \frac{\nabla \pi(A_t | S_t, \theta)}{\pi(A_t | S_t, \theta)} \right] \\ &= \mathbb{E}_\pi \left[G_t \frac{\nabla \pi(A_t | S_t, \theta)}{\pi(A_t | S_t, \theta)} \right]\end{aligned}$$

Monte Carlo Policy Gradient Algorithm: REINFORCE

$$\text{REINFORCE Update: } \theta_{t+1} \leftarrow \theta_t + \alpha G_t \frac{\nabla \pi(A_t | S_t, \theta_t)}{\pi(A_t | S_t, \theta_t)}$$

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

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

Algorithm parameter: step size $\alpha > 0$

Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):

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

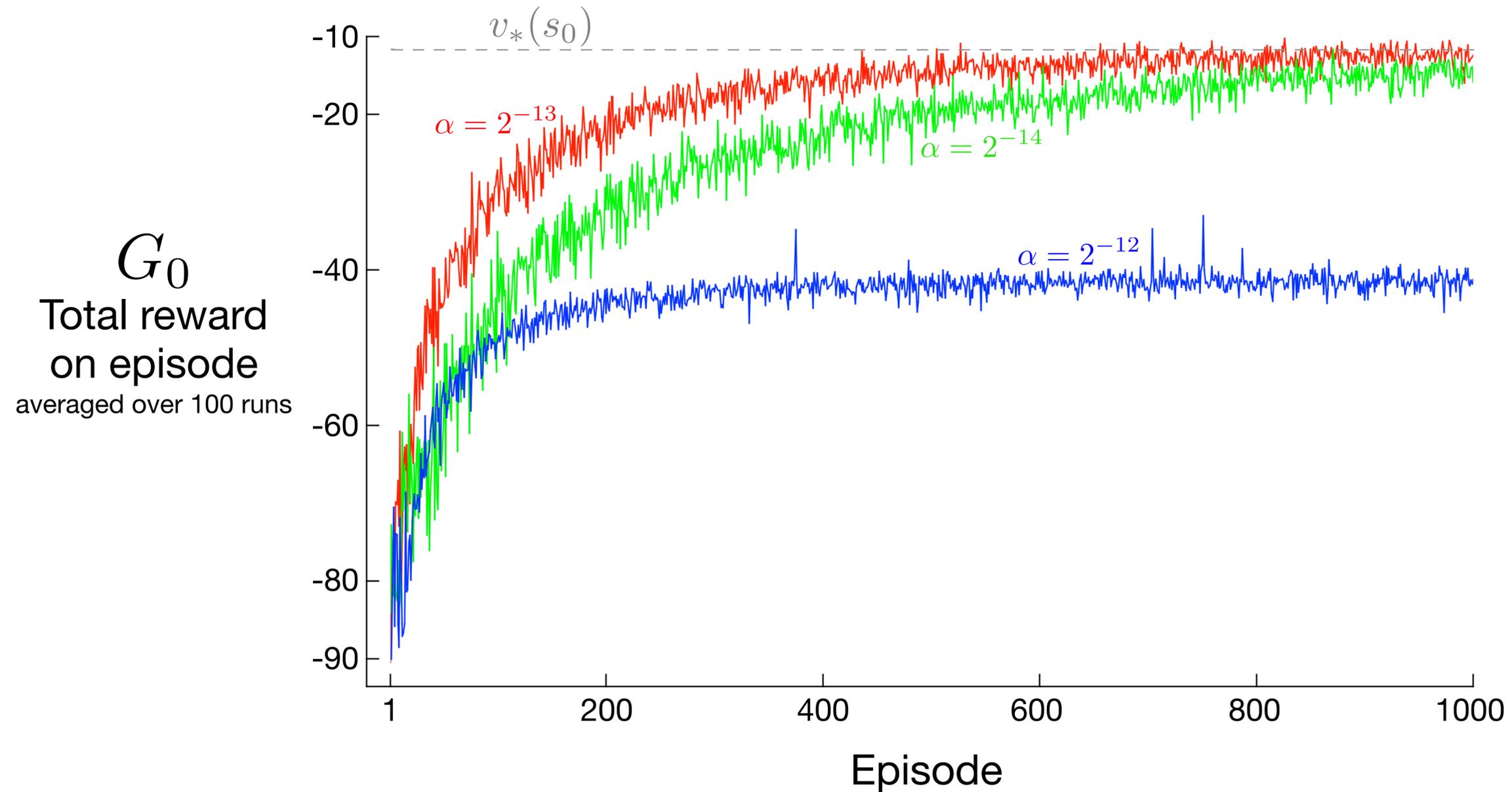
Loop for each step of the episode $t = 0, 1, \dots, T - 1$:

$$G \leftarrow \sum_{k=t+1}^T \gamma^{k-t-1} R_k \quad (G_t)$$

$$\theta \leftarrow \theta + \alpha \gamma^t G \underbrace{\nabla \ln \pi(A_t | S_t, \theta)}$$

$\frac{\nabla \pi(A_t | S_t, \theta)}{\pi(A_t | S_t, \theta)}$ "eligibility function" $\left(\nabla \ln x = \frac{\nabla x}{x} \right)$

REINFORCE Performance in Switcheroo Corridor



Summary

- All our previous control algorithms were **action-value** methods
 1. Approximate the action-value $q^*(s,a)$
 2. Choose maximal-value action at every state
- **Policy gradient** methods:
 1. Represent policies using **parametric function** $\pi(s|a,\theta)$
 2. **Directly optimize** performance $J(\theta)$ by adjusting θ
- **Policy Gradient Theorem** lets us restate $J(\theta)$ in terms of quantities that we **know** ($\nabla \Pi$) or can **approximate** (q_π)
- REINFORCE uses a particular **estimation scheme** for policy gradients