In the Name of God

Iran University
of Science and
Technology

Reinforcement Learning in Control

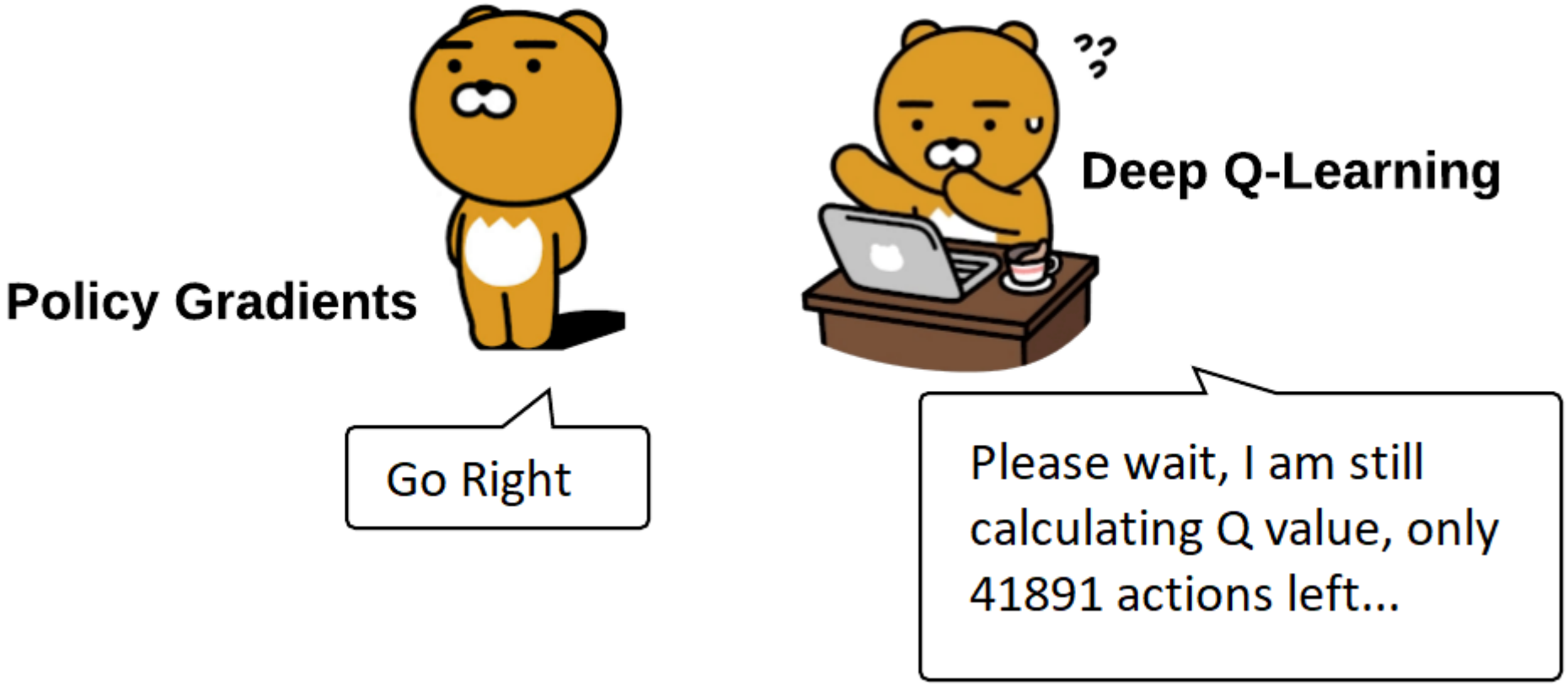
Dr. Saeed Shamaghdari

Electrical Engineering Department
Control Group

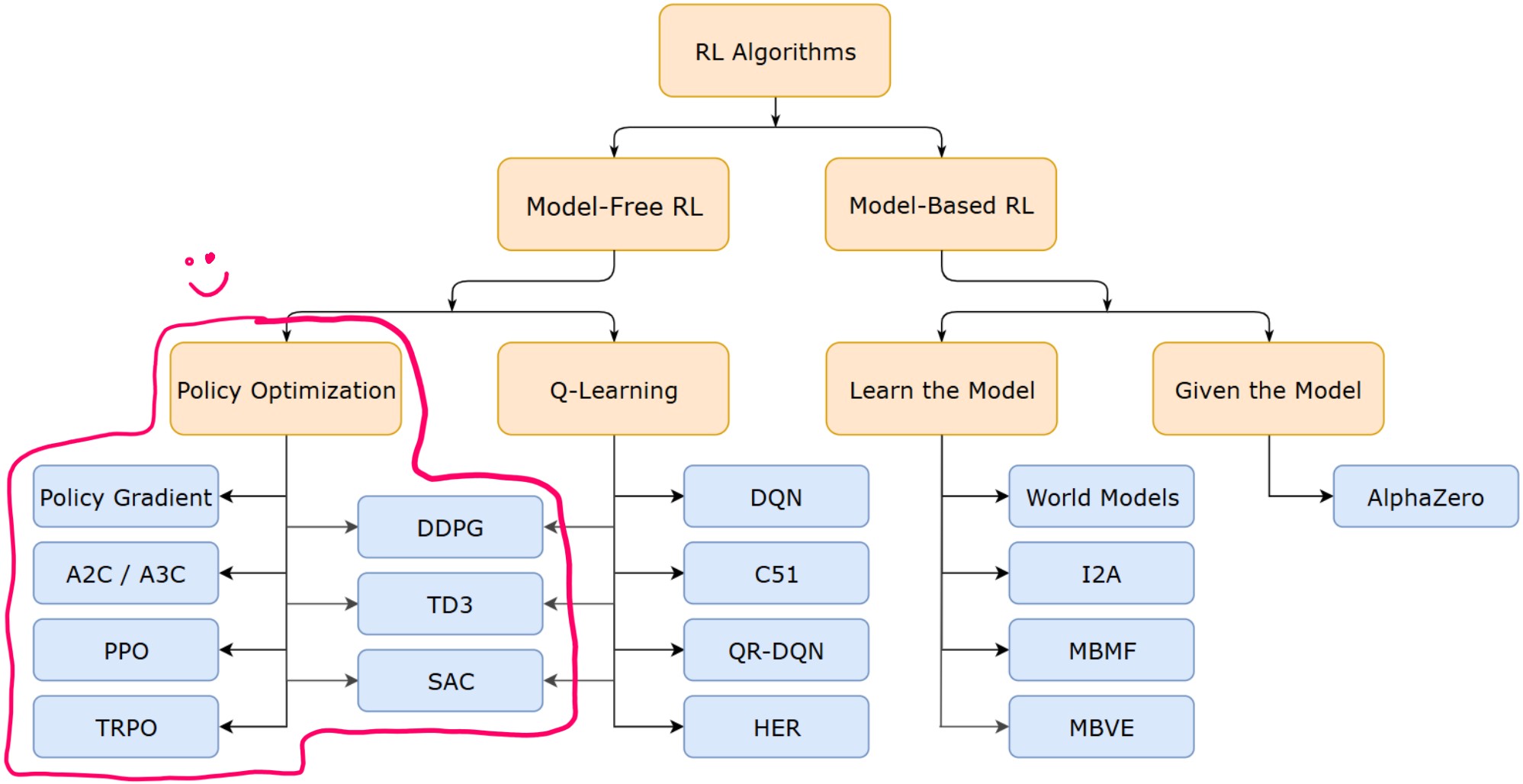
Fall 2025 | 4041

Policy Gradient Methods

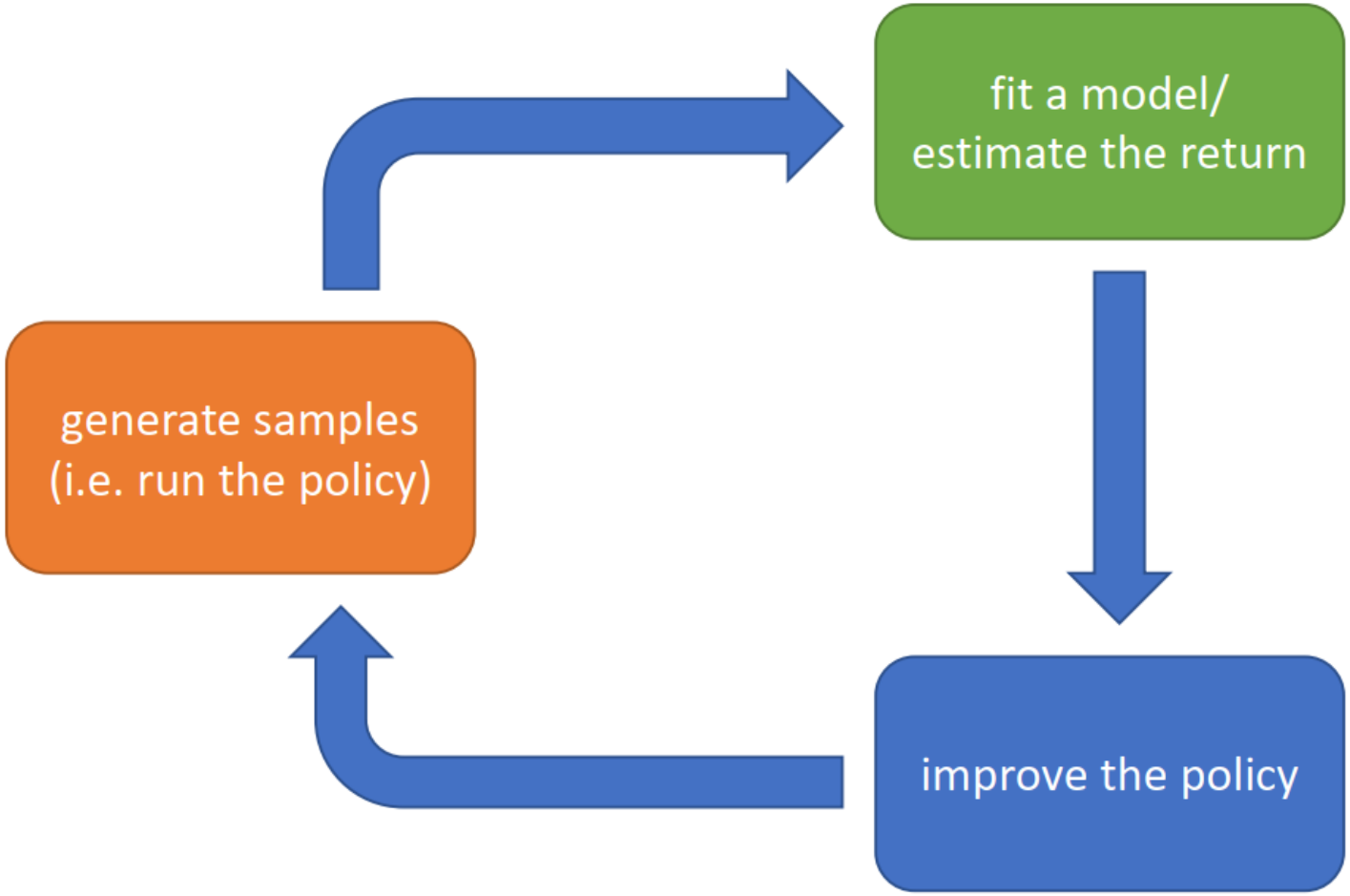
Quick Review

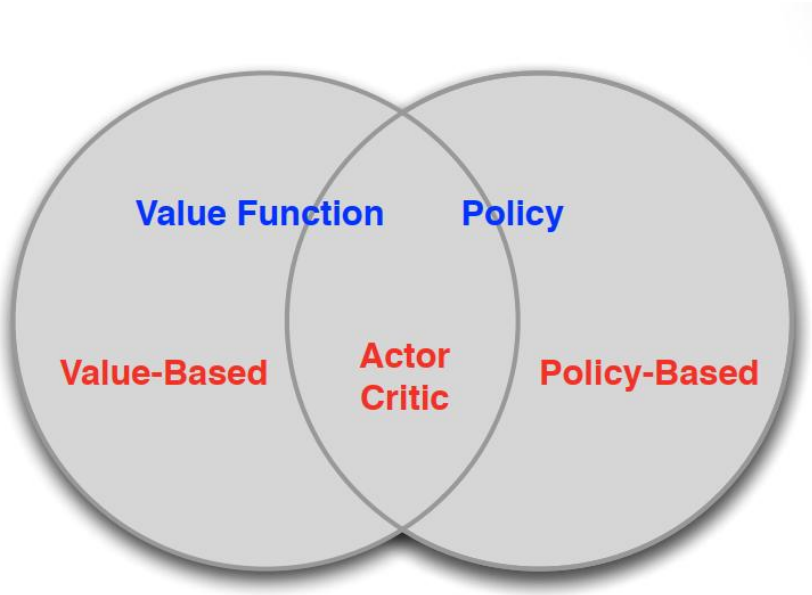


RL Algorithms

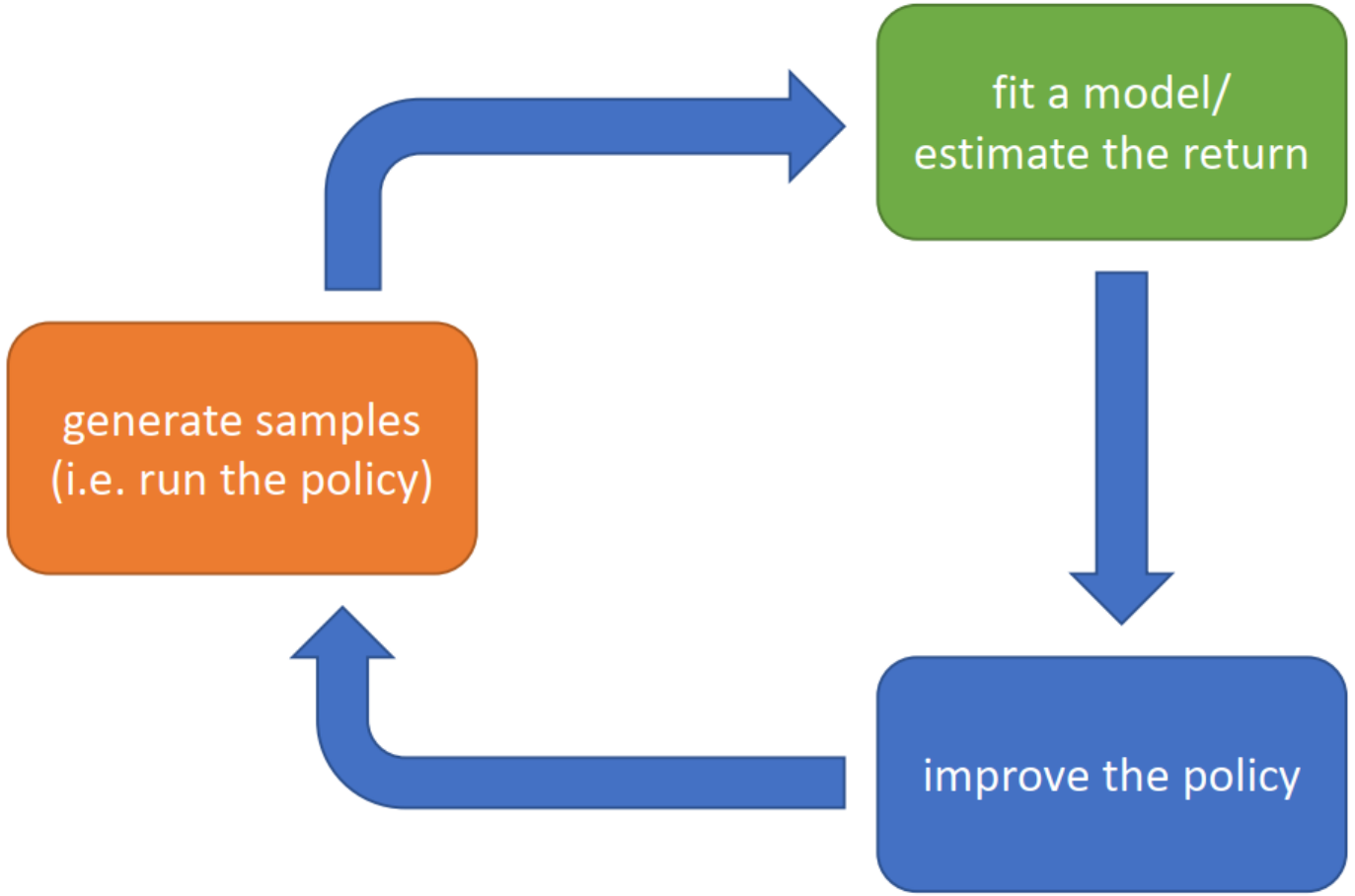


Anatomy of RL Algorithms





Anatomy of RL Algorithms: Value Function Based Methods



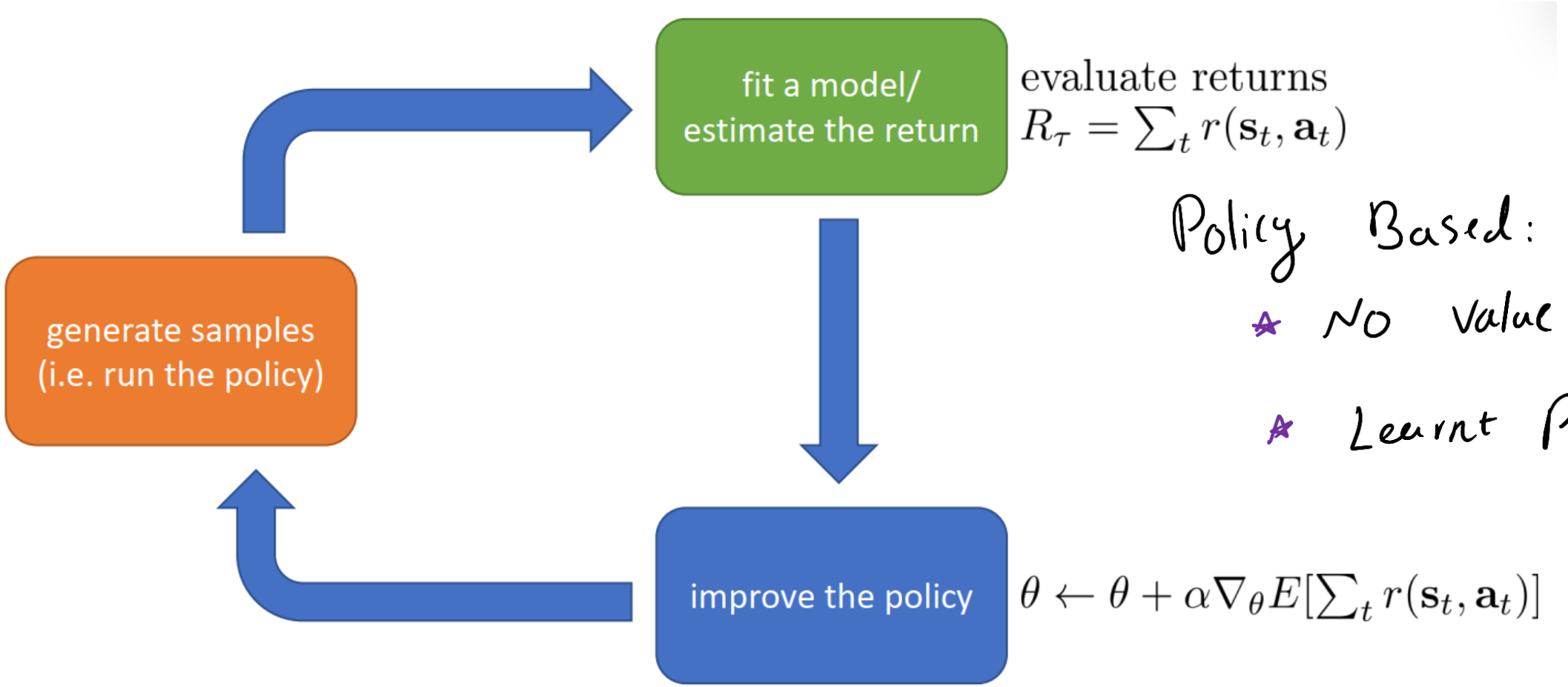
fit $V(\mathbf{s})$ or $Q(\mathbf{s}, \mathbf{a})$

Value Based:

- Learnt Value Function
- Implicit Policy (e.g ε-greedy)

set $\pi(\mathbf{s}) = \arg\max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a})$

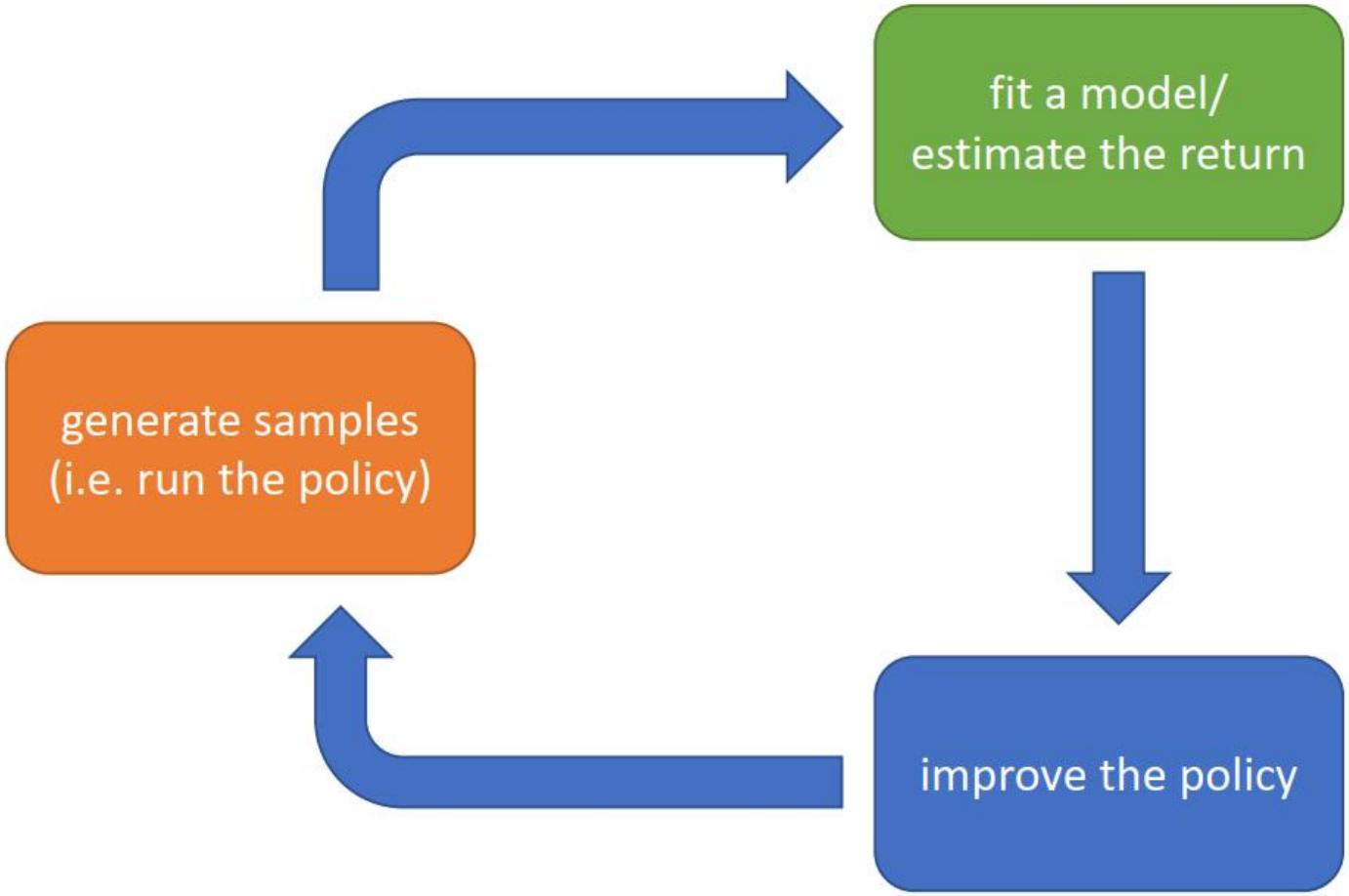
Anatomy of RL Algorithms: Policy Gradients



Policy Based:

- No value function
- Learnt Policy

Anatomy of RL Algorithms: Actor-Critic



fit $V(\mathbf{s})$ or $Q(\mathbf{s}, \mathbf{a})$

$\theta \leftarrow \theta + \alpha \nabla_\theta E[Q(\mathbf{s}_t, \mathbf{a}_t)]$

Actor-Critic:

- Learnt Value Function
- Learnt Policy

Policy Gradients

So far in this class almost all the methods have been *action-value methods*; they learned the values of actions and then selected actions based on their estimated action values.

Link between Policy and Value:

$$\pi^*(s) = \arg\max_a Q^*(s, a)$$

This Chapter:

we consider methods that instead learn a *parameterized policy* that can select actions without consulting a value function.

A value function may still be used to *learn* the policy parameter, but is not required for action selection.

Policy Gradients

Definition

We write

$$\pi(a|s,\boldsymbol{\theta}) = \Pr\{A_t = a \mid S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}\}$$

for the probability that action *a* is taken at time *t* given that the environment is in state *s* at time *t* with parameter θ.

$\boldsymbol{\theta} \in \mathbb{R}^{d'}$ → policy's parameter vector

If a method uses a learned value function as well, then the value function's weight vector is denoted $w \in \mathbb{R}^d$ as usual, as in $\hat{v}(s,w)$.

Policy Gradients

Policy gradients are methods for learning the policy parameter based on the gradient of some scalar performance measure

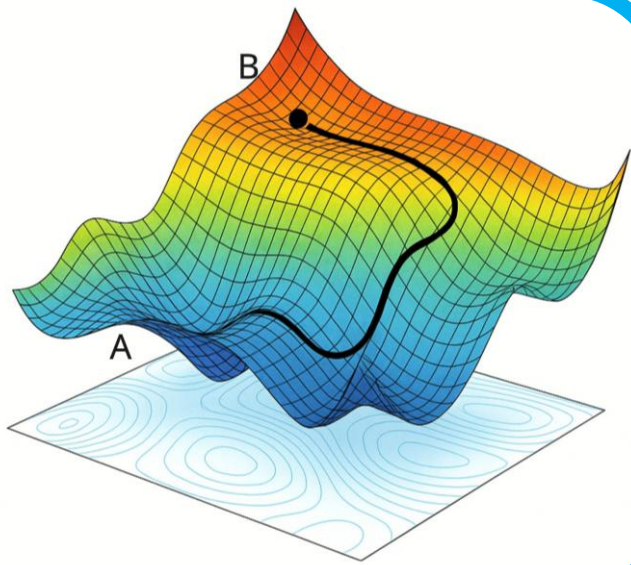
$$J(\theta)$$

with respect to the policy parameter.

Maximize performance using *gradient ascent*

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \widehat{\nabla J(\boldsymbol{\theta}_t)}$$

Where $\widehat{\nabla J(\theta_t)} \in \mathbb{R}^{d'}$ is a stochastic estimate whose expectation approximates the gradient of the performance measure with respect to its argument θ_t.



Policy Gradients

- All methods that follow this general schema we call *policy gradient methods*, whether or not they also learn an approximate value function.

- Methods that learn approximations to both policy and value functions are often called *actor–critic methods*, where ‘actor’ is a reference to the learned policy, and ‘critic’ refers to the learned value function, usually a state-value function.

Policy Approximation and its Advantages

In this section we introduce the most common parameterization for discrete action spaces and point out the advantages it offers over action-value methods.

If the action space is discrete and not too large, then a natural and common kind of parameterization is to form parameterized numerical preferences $h(s, a, \theta) \in \mathbb{R}$ for each state-action pair.

Actions with the highest preferences -> Highest probabilities of being selected

Policy Approximation: Soft-max in Action Preferences

$$\pi(a|s,\boldsymbol{\theta}) \doteq \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_b e^{h(s,b,\boldsymbol{\theta})}}$$

The action preferences themselves can be parameterized arbitrarily. For example, they might be computed by a deep artificial neural network (ANN), Or the preferences could simply be linear in features,

$$h(s,a,\boldsymbol{\theta}) = \boldsymbol{\theta}^\top \mathbf{x}(s,a)$$

using feature vectors $\mathrm{x}(\mathrm{s},\mathrm{a}) \in \mathbb{R}^{d'}$.

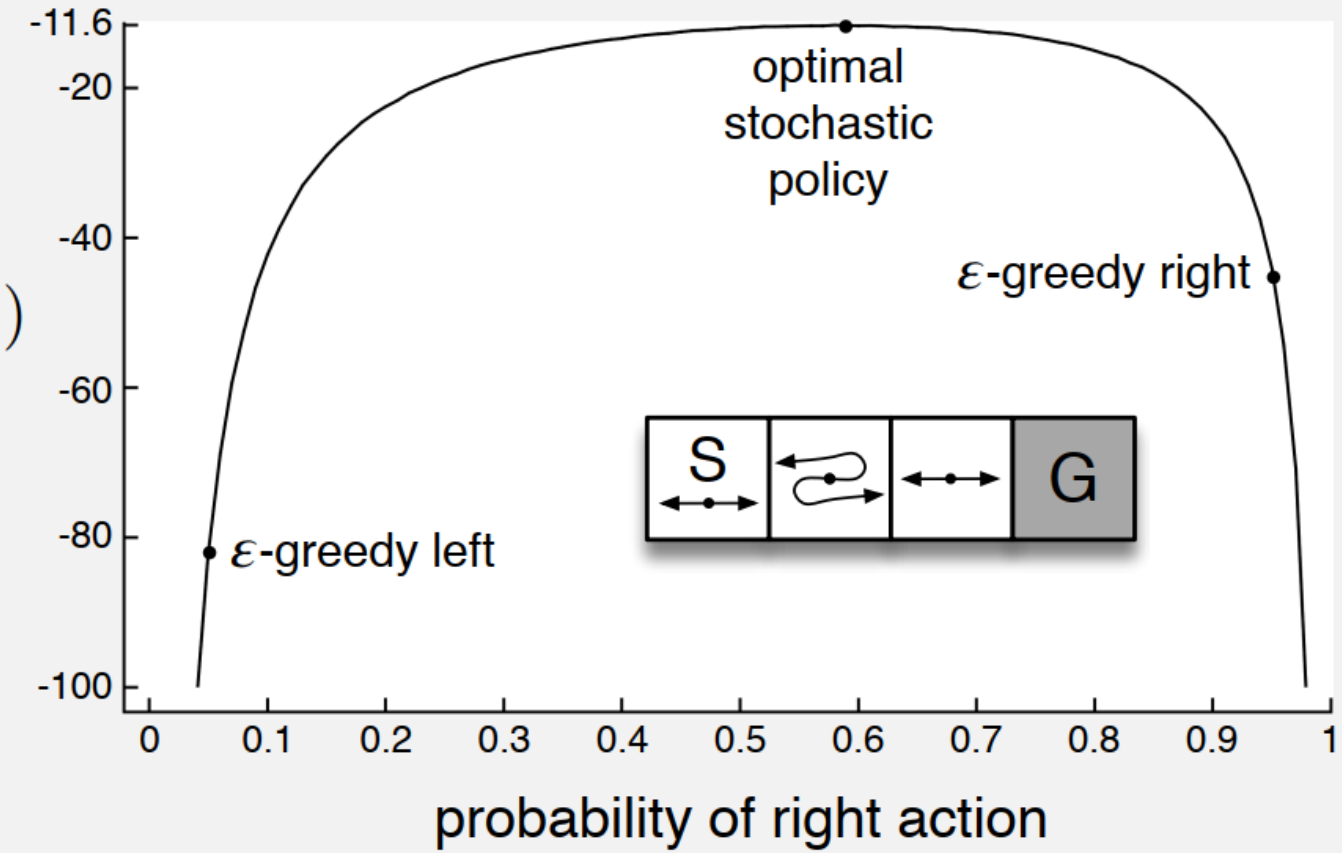
Policy Approximation: Soft-max in Action Preferences

One advantage of parameterizing policies according to the soft-max in action preferences is that the approximate policy can approach a deterministic policy, whereas with ε-greedy action selection over action values there is always an ε probability of selecting a random action.

A second advantage of parameterizing policies according to the soft-max in action preferences is that it enables the selection of actions with arbitrary probabilities. Action-value methods have no natural way of finding stochastic optimal policies, whereas policy approximating methods can.

Example: Short Corridor with Switched Actions

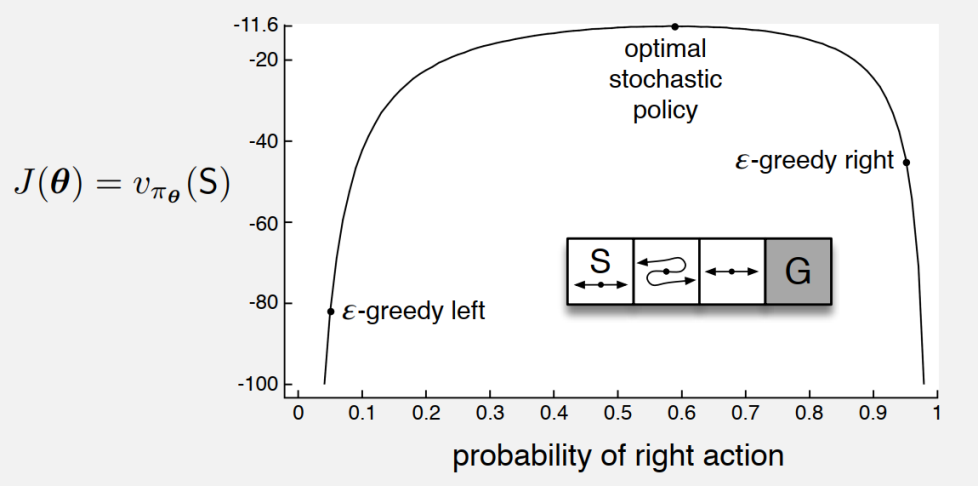
$$J(\boldsymbol{\theta}) = v_{\pi_{\boldsymbol{\theta}}}(\mathrm{S})$$



Example: Short Corridor with Switched Actions

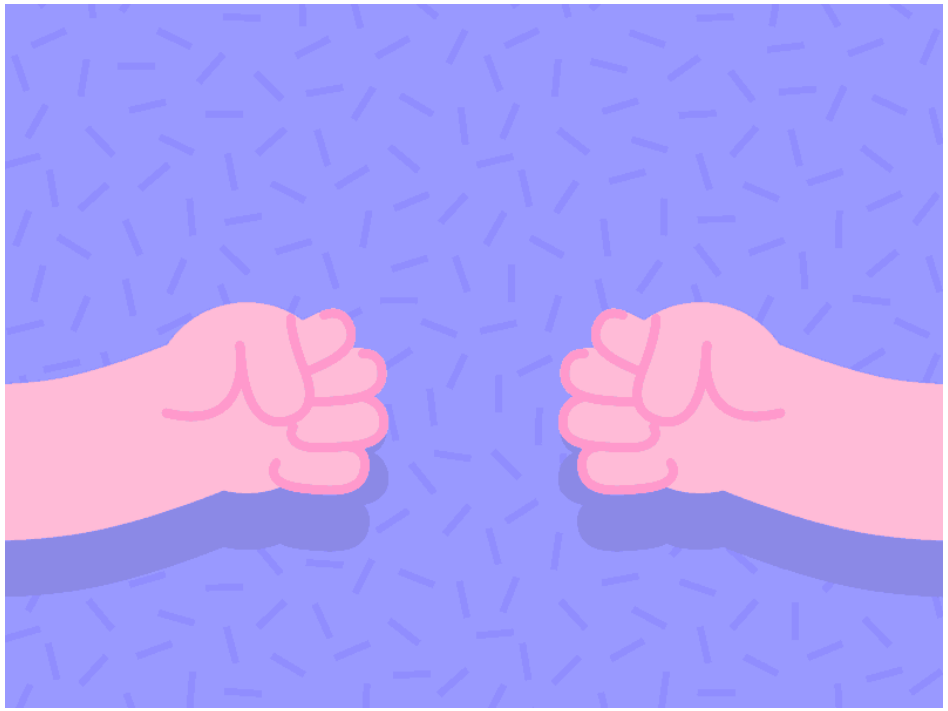
- Value-based RL learns near deterministic policy (greedy, e-greedy)
- The agent can get stuck and never reach the goal!

- Stochastic Policy:
  The agent reach the goal state in a few steps with high probability!
- Policy-based RL can learn the optimal stochastic policy



Another Example: Rock-Paper-Scissors

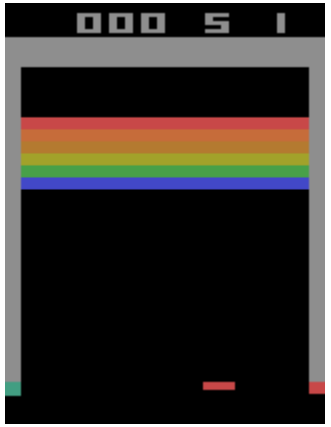
- Consider policies for *iterated* rock-paper-scissors
  - A deterministic policy is easily exploited
  - A uniform random policy is optimal (i.e. Nash equilibrium)



Policy Approximation: Soft-max in Action Preferences

Perhaps the simplest advantage that policy parameterization may have over action-value parameterization is that the policy may be a simpler function to approximate. A policy-based method will typically learn faster and yield a superior asymptotic policy.

Advantage of Policy-Based RL: Summary



- Advantages:
  - Better convergence properties
  - Effective in high-dimensional or continuous action spaces
  - Can learn stochastic policies
- Disadvantages:
  - Typically converge to a local rather than global optimum
  - Evaluating a policy is typically inefficient and high variance

The Policy Gradient Theorem

Stronger convergence guarantees are available for policy-gradient methods than for action-value methods.

By assuming that every episode starts in some particular (non-random) state *s*0; Then, in the episodic case we define performance as

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

the true value function for π_θ, the policy determined by θ

Assume no discounting

The Policy Gradient Theorem

With function approximation it may seem challenging to change the policy parameter in a way that ensures improvement. The problem is that performance depends on both the action selections and the distribution of states in which those selections are made, and that both of these are affected by the policy parameter.

How can we estimate the performance gradient with respect to the policy parameter when the gradient depends on the unknown effect of policy changes on the state distribution?

The Policy Gradient Theorem

Proof (Episodic Case)

To keep the notation simple, we leave it implicit in all cases that π is a function of θ, and all gradients are also implicitly with respect to θ. First note that the gradient of the state-value function can be written in terms of the action-value function as

$$\nabla v_\pi(s) = \nabla \left[\sum_a \pi(a|s) q_\pi(s,a) \right], \quad \text{for all } s \in \mathcal{S}$$

The Policy Gradient Theorem

Proof (Episodic Case)

$$
\begin{aligned}
\nabla v_\pi(s) &= \nabla \left[\sum_a \pi(a|s) q_\pi(s,a) \right], \quad \text{for all } s \in \mathcal{S} \\
&= \sum_a \Big[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \nabla q_\pi(s,a) \Big] \\
&= \sum_a \Big[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \nabla \sum_{s',r} p(s',r|s,a)\big(r + v_\pi(s')\big) \Big] \\
&= \sum_a \Big[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \sum_{s'} p(s'|s,a) \nabla v_\pi(s') \Big]
\end{aligned}
$$

The Policy Gradient Theorem

Proof (Episodic Case)

$$
\begin{aligned}
&= \sum_a \Big[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \sum_{s'} p(s'|s,a) \nabla v_\pi(s') \Big] \\
&= \sum_a \Big[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \sum_{s'} p(s'|s,a) && \text{(unrolling)} \\
&\qquad \sum_{a'} \big[\nabla \pi(a'|s') q_\pi(s',a') + \pi(a'|s') \sum_{s''} p(s''|s',a') \nabla v_\pi(s'') \big] \Big] \\
&= \sum_{x \in \mathcal{S}} \sum_{k=0}^{\infty} \Pr(s \to x, k, \pi) \sum_a \nabla \pi(a|x) q_\pi(x,a),
\end{aligned}
$$

The Policy Gradient Theorem

Proof (Episodic Case)

$$= \sum_{x \in \mathcal{S}} \sum_{k=0}^{\infty} \Pr(s \to x, k, \pi) \sum_{a} \nabla \pi(a|x) q_\pi(x, a),$$

after repeated unrolling, where $\Pr(s \to x, k, \pi)$ is the probability of transitioning from state *s* to state *x* in *k* steps under policy π. It is then immediate that

$$\nabla J(\boldsymbol{\theta}) = \nabla v_\pi(s_0)$$

The Policy Gradient Theorem

Proof (Episodic Case)

$$
\begin{aligned}
\nabla J(\boldsymbol{\theta}) &= \nabla v_\pi(s_0) \\
&= \sum_s \left(\sum_{k=0}^{\infty} \Pr(s_0 \to s, k, \pi) \right) \sum_a \nabla \pi(a|s) q_\pi(s,a) \\
&= \sum_s \eta(s) \sum_a \nabla \pi(a|s) q_\pi(s,a) \\
&= \sum_{s'} \eta(s') \sum_s \frac{\eta(s)}{\sum_{s'} \eta(s')} \sum_a \nabla \pi(a|s) q_\pi(s,a)
\end{aligned}
$$

The Policy Gradient Theorem

Proof (Episodic Case)

$$
\begin{aligned}
&= \sum_{s'} \eta(s') \sum_{s} \frac{\eta(s)}{\sum_{s'} \eta(s')} \sum_{a} \nabla \pi(a|s) q_\pi(s,a) \\
&= \sum_{s'} \eta(s') \sum_{s} \mu(s) \sum_{a} \nabla \pi(a|s) q_\pi(s,a) \\
&\propto \sum_{s} \mu(s) \sum_{a} \nabla \pi(a|s) q_\pi(s,a)
\end{aligned}
$$

that does *not* involve the derivative of the state distribution.

The Policy Gradient Theorem

The policy gradient theorem for the episodic case establishes that

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

where the gradients are column vectors of partial derivatives with respect to the components of θ, and π denotes the policy corresponding to parameter vector θ. The distribution μ here is the on-policy distribution under π.

In the episodic case, the constant of proportionality is the average length of an episode, and in the continuing case it is 1, so that the relationship is actually an equality.

REINFORCE: Monte Carlo Policy Gradient

Notice that the right-hand side of the policy gradient theorem is a sum over states weighted by how often the states occur under the target policy π; if π is followed, then states will be encountered in these proportions. Thus

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

$$= \mathbb{E}_\pi \left[\sum_a q_\pi(S_t,a) \nabla \pi(a|S_t,\boldsymbol{\theta}) \right] .$$

Maximize performance using *gradient ascent*

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \widehat{\nabla J(\boldsymbol{\theta}_t)}$$

REINFORCE: Monte Carlo Policy Gradient

We could stop here and instantiate our stochastic gradient-ascent algorithm as

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \sum_a \hat{q}(S_t, a, \mathbf{w}) \nabla \pi(a|S_t, \boldsymbol{\theta})$$

where $\hat{q}$ is some learned approximation to q_π.

Our current interest is the classical REINFORCE algorithm (Willams, 1992) whose update at time *t* involves just A_t, the one action actually taken at time *t*.

Maximize performance using *gradient ascent*

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \widehat{\nabla J(\boldsymbol{\theta}_t)}$$

REINFORCE: Monte Carlo Policy Gradient

$$\nabla J(\boldsymbol{\theta}) \propto \mathbb{E}_\pi \left[\sum_a q_\pi(S_t, a) \nabla \pi(a|S_t, \boldsymbol{\theta}) \right]$$

$$\propto \mathbb{E}_\pi \left[\sum_a \pi(a|S_t, \boldsymbol{\theta}) q_\pi(S_t, a) \frac{\nabla \pi(a|S_t, \boldsymbol{\theta})}{\pi(a|S_t, \boldsymbol{\theta})} \right]$$

$$= \mathbb{E}_\pi \left[q_\pi(S_t, A_t) \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})} \right] \qquad \text{(replacing } a \text{ by the sample } A_t \sim \pi\text{)}$$

$$= \mathbb{E}_\pi \left[G_t \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})} \right], \qquad \text{(because } \mathbb{E}_\pi[G_t|S_t, A_t] = q_\pi(S_t, A_t)\text{)}$$

The REINFORCE Update:

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha G_t \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)}$$

REINFORCE: Monte Carlo Policy Gradient

The REINFORCE Update:

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha G_t \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)}$$

$$\nabla \ln x = \frac{\nabla x}{x}$$

$$\nabla \ln \pi(A_t|S_t, \boldsymbol{\theta}_t) = \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)}$$

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha G_t \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta}_t)$$

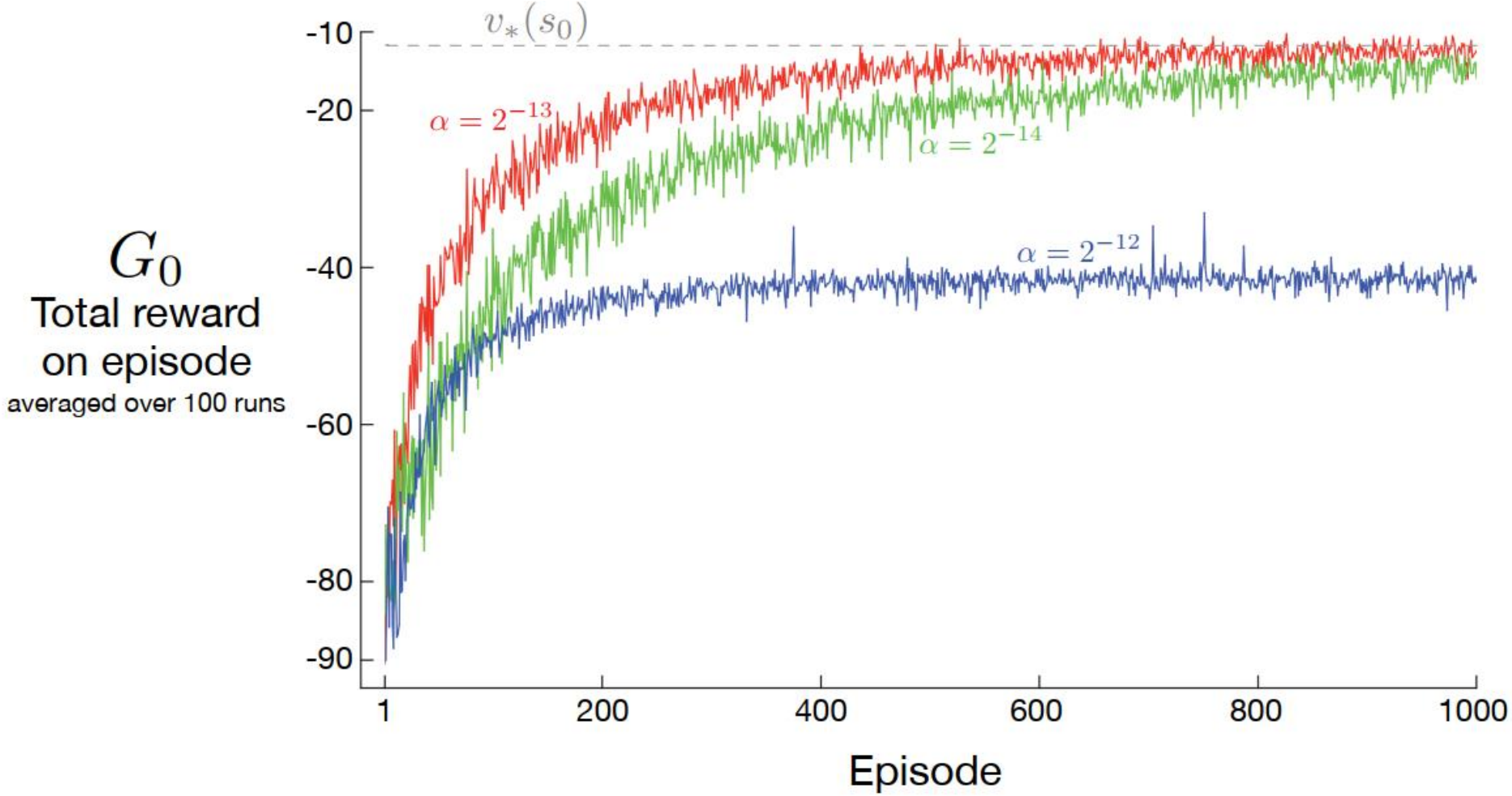
REINFORCE: Algorithm (Discounted)

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

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

REINFORCE: Short Corridor Example



REINFORCE: Problem!

Good convergence properties, However …

Q: What problem does the REINFORCE method have as a Monte Carlo approach?

REINFORCE with Baseline

The policy gradient theorem can be generalized to include a comparison of the action value to an arbitrary *baseline* $b(s)$:

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

The baseline can be any function, even a random variable, as long as it does not vary with a; the equation remains valid because the subtracted quantity is zero:

$$\sum_a b(s) \nabla \pi(a|s,\boldsymbol{\theta}) \;=\; b(s) \nabla \sum_a \pi(a|s,\boldsymbol{\theta}) \;=\; b(s) \nabla 1 \;=\; 0$$

REINFORCE with Baseline: Update

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha\Big(G_t - b(S_t)\Big)\frac{\nabla\pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)}$$

One natural choice for the baseline is an estimate of the state value

$$\hat{v}(S_t,\mathbf{w})$$

$\mathbf{w} \in \mathbb{R}^d$ is a weight vector

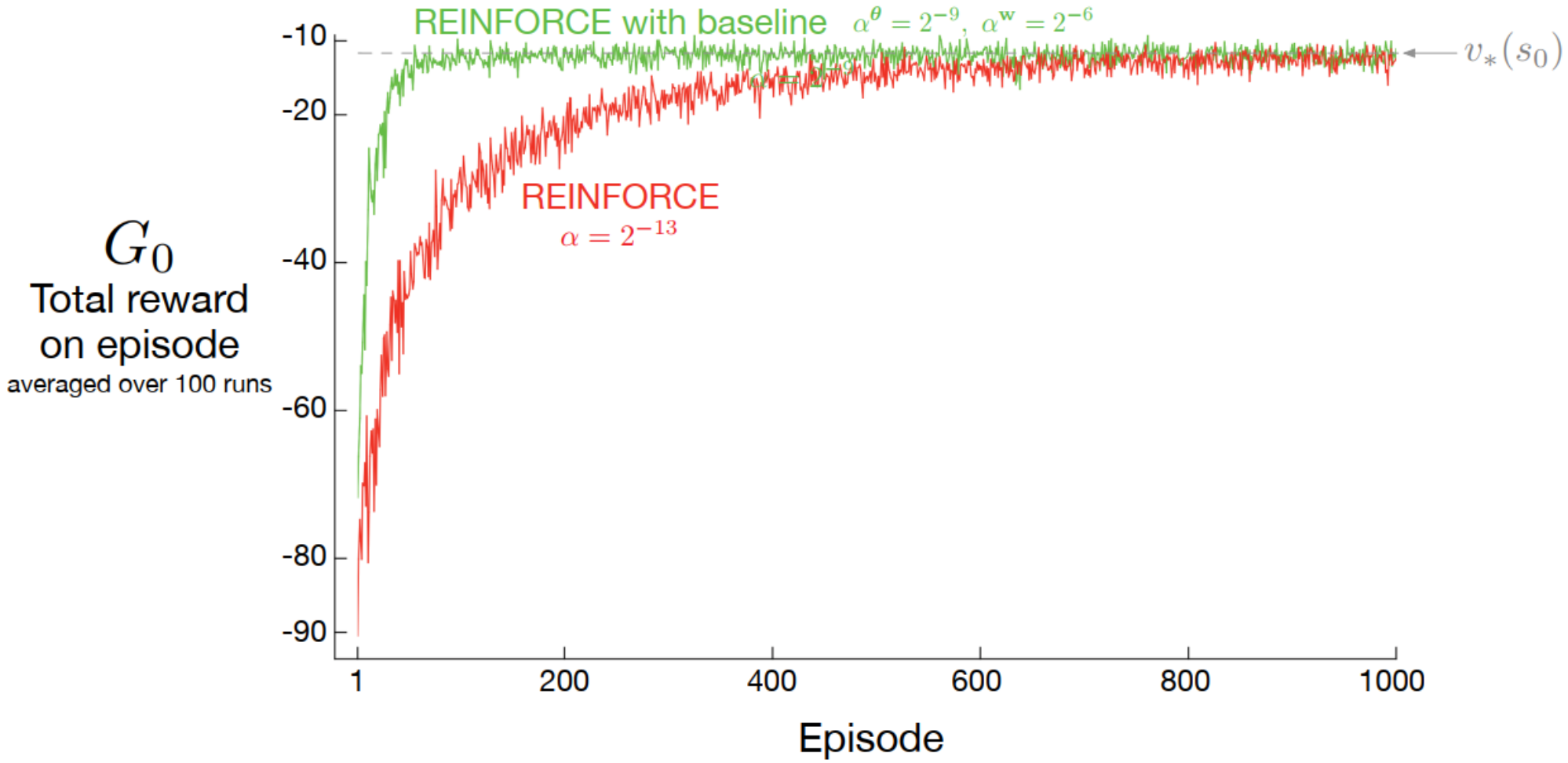
REINFORCE with Baseline: Algorithm

REINFORCE with Baseline (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Algorithm parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\delta \leftarrow G - \hat{v}(S_t,\mathbf{w})$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S_t,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

REINFORCE with Baseline: Short Corridor Example



Actor-Critic Methods

In REINFORCE with baseline, the learned state-value function estimates the value of the *first* state of each state transition. This estimate sets a baseline for the subsequent return, but is made prior to the transition's action and thus cannot be used to assess that action. In actor–critic methods, on the other hand, the state-value function is applied also to the *second* state of the transition.

When the state-value function is used to assess actions in this way it is called a *critic*, and the overall policy-gradient method is termed an *actor–critic* method. Note that the bias in the gradient estimate is not due to bootstrapping as such; the actor would be biased even if the critic was learned by a Monte Carlo method.

Actor-Critic Methods

First consider one-step actor–critic methods, the analog of the TD methods introduced in chapter 6.

$$\begin{aligned}\boldsymbol{\theta}_{t+1} &\doteq \boldsymbol{\theta}_t + \alpha\Big(G_{t:t+1} - \hat{v}(S_t,\mathbf{w})\Big)\frac{\nabla\pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)} \\ &= \boldsymbol{\theta}_t + \alpha\Big(R_{t+1} + \gamma\hat{v}(S_{t+1},\mathbf{w}) - \hat{v}(S_t,\mathbf{w})\Big)\frac{\nabla\pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)} \\ &= \boldsymbol{\theta}_t + \alpha\delta_t\frac{\nabla\pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)}.\end{aligned}$$

Actor-Critic Methods

The natural state-value-function learning method to pair with this is semi-gradient TD(0). Pseudocode for the complete algorithm is given in the box at the top of the next page. Note that it is now a fully online, incremental algorithm, with states, actions, and rewards processed as they occur and then never revisited.

Actor-Critic Methods: Algorithm

One-step Actor–Critic (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$