Actor-Critic Algorithms

CMPT 729 G100

Jason Peng

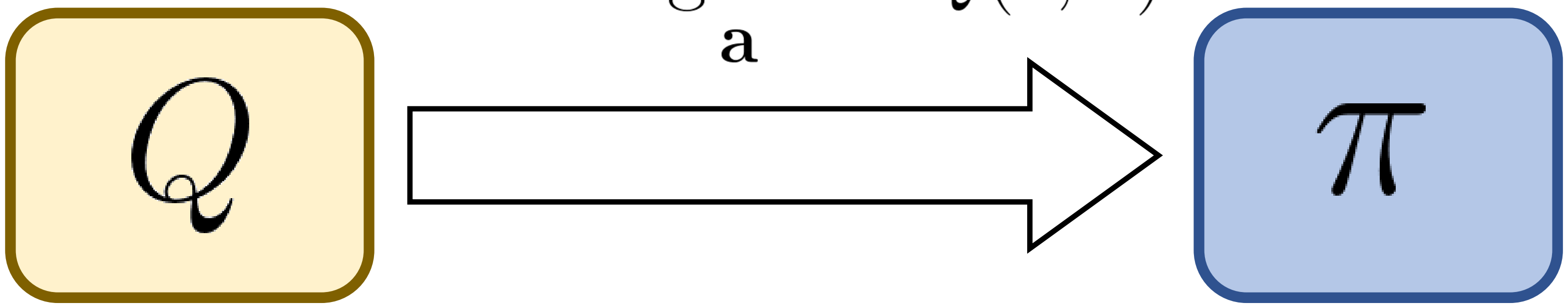
Overview

- Actor-Critic Algorithms
- Deterministic Policy Gradient
- Soft Actor-Critic
- Surrogate Objective

Value-Based Methods

- Learn only the Q-function
- Q-function implicitly encodes policy

$$\mathbf{a}^* = \arg\max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a})$$



Q-Learning

- ✓ Often much more sample efficient than policy gradient
- ✓ Off-policy learning

- ✗ Limited to relatively small discrete action spaces
- ✗ Does not directly optimize performance
  - Lower Bellman error ≠ better performance
- ✗ No convergence guarantees with function approximators

$$Q^{k+1} = \arg\min_{Q} \mathbb{E}_{(\mathbf{s},\mathbf{a},r,\mathbf{s}')\sim\mathcal{D}}\left[\left(\left(r + \gamma \max_{\mathbf{a}'} Q^k(\mathbf{s}',\mathbf{a}')\right) - Q(\mathbf{s},\mathbf{a})\right)^2\right]$$

Intractable in large/continuous action spaces

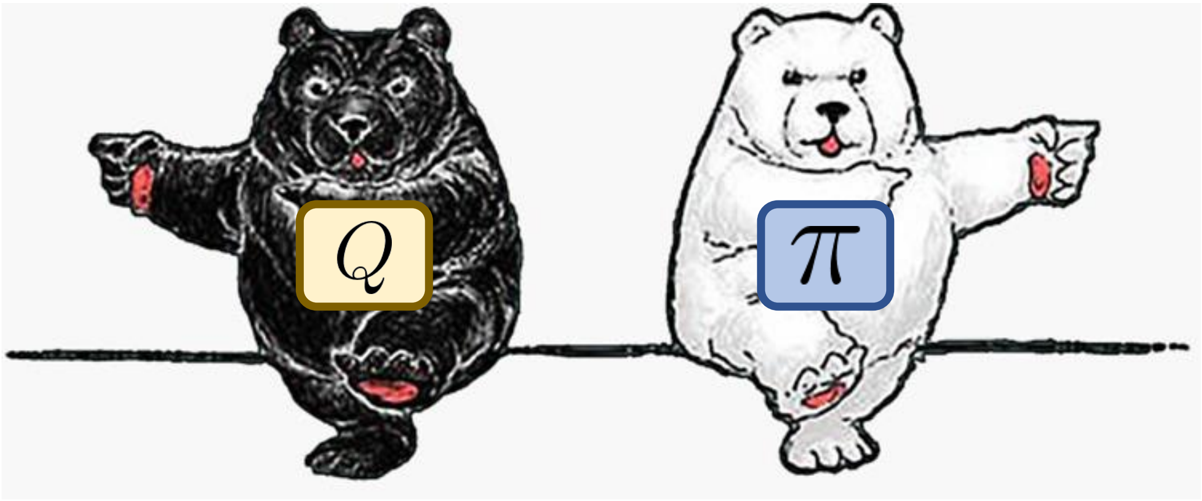
Policy Gradient

- ✓ Directly optimize $J(\pi)$ by estimating gradient $\nabla_\pi J(\pi)$
- ✓ General: can be applied to continuous and discrete states and actions
- ✗ High-variance gradient estimator → unstable/slow convergence
- ✗ Very sample inefficient

$$\nabla_\pi J(\pi) = \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})}\mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}\mathbb{E}_{\tau\sim p(\tau|\pi,\mathbf{s}_0=\mathbf{s},\mathbf{a}_0=\mathbf{a})}\left[\nabla_\pi \log\pi(\mathbf{a}|\mathbf{s})\left(\sum_{t'=0}^{T-1}\gamma^{t'}r_{t'} - V^\pi(\mathbf{s})\right)\right]$$

Taxonomy of RL Algorithms

- Policy-Based Methods
- Value-Based Methods
- Actor-Critic Methods
- Model-Based Methods



Taxonomy of RL Algorithms

- Policy-Based Methods
- Value-Based Methods
- Actor-Critic Methods
- Model-Based Methods

Taxonomy of RL Algorithms

- Policy-Based Methods
- Value-Based Methods
- Actor-Critic Methods
- Model-Based Methods

Nondifferentiable Objective

$$\pi^* = \arg\max_{\pi} J(\pi)$$

nondifferentiable

Policy Gradient

$$\pi^* = \arg\max_{\pi} J(\pi)$$

make this differentiable?

$$\nabla_\pi J(\pi) = \mathbb{E}_{\tau \sim p(\tau|\pi)} \left[R(\tau) \sum_{t=0}^{T-1} \nabla_\pi \log \pi(\mathbf{a}_t | \mathbf{s}_t) \right]$$

Policy Gradient

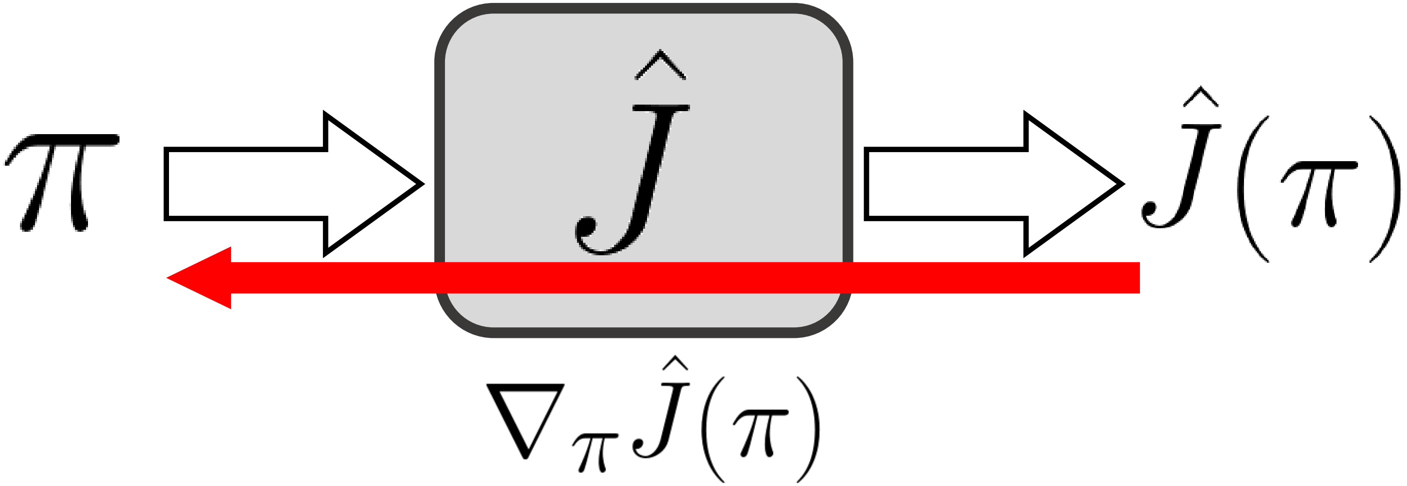
$$\pi^* = \arg\max_{\pi} \; J(\pi)$$

$$\hat{J}(\pi) \approx J(\pi)$$

differentiable

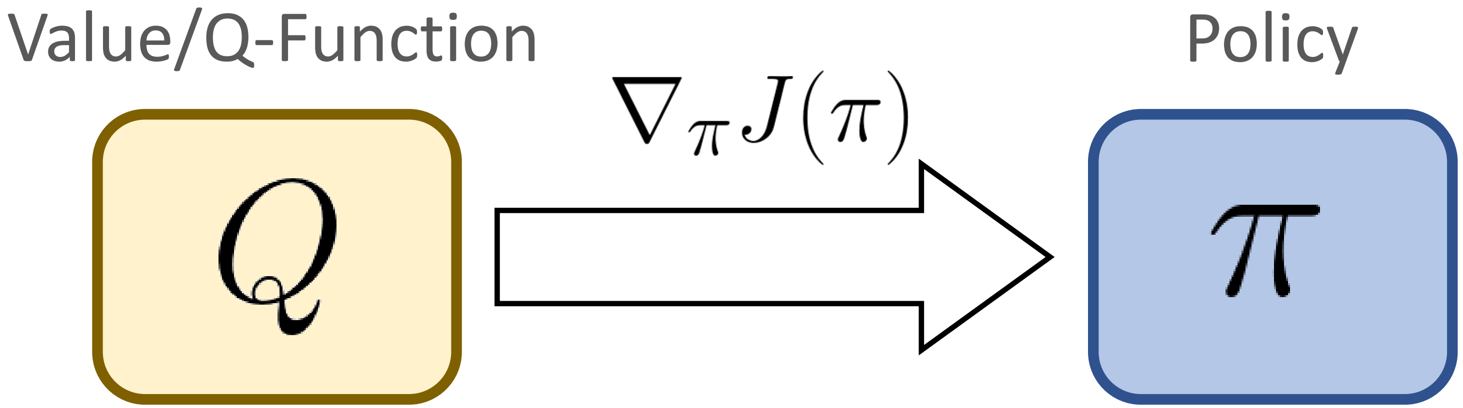
Surrogate Objective

Differentiable surrogate objective → just use gradient ascent!



Actor-Critic Methods

- Jointly learn both policy and value function
- Use value function to improve policy



Actor-Critic Algorithms

$$\nabla_\pi J(\pi) = \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})}\mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}\mathbb{E}_{p(\tau|\pi,\mathbf{s}_0=\mathbf{s},\mathbf{a}_0=\mathbf{a})}\left[\nabla_\pi \log\pi(\mathbf{a}|\mathbf{s})\left(\sum_{t=0}\gamma^t r_t - V^\pi(\mathbf{s})\right)\right]$$

Variance reduction via bootstrapping

n-step return: $r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots + \gamma^{k-1} r_{k-1} + \gamma^k V^\pi(\mathbf{s}_k)$

bootstrap

Bootstrapped Policy Gradient

$$\nabla_\pi J(\pi) = \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})}\mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}\mathbb{E}_{p(\tau|\pi,\mathbf{s}_0=\mathbf{s},\mathbf{a}_0=\mathbf{a})}\left[\nabla_\pi \log\pi(\mathbf{a}|\mathbf{s})\left(\sum_{t=0}\gamma^t r_t - V^\pi(\mathbf{s})\right)\right]$$

⇓

$$\nabla_\pi J(\pi) = \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})}\mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}\mathbb{E}_{p(\mathbf{s}'|\mathbf{s},\mathbf{a})}\left[\nabla_\pi \log\pi(\mathbf{a}|\mathbf{s})\left(r + \gamma \underline{V^\pi(\mathbf{s}')} - \underline{V^\pi(\mathbf{s})}\right)\right]$$

estimate return — $\gamma V^\pi(\mathbf{s}')$

baseline — $V^\pi(\mathbf{s})$

Bootstrapped Policy Gradient

$$\nabla_\pi J(\pi) = \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})}\mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}\mathbb{E}_{p(\tau|\pi,\mathbf{s}_0=\mathbf{s},\mathbf{a}_0=\mathbf{a})}\left[\nabla_\pi \log\pi(\mathbf{a}|\mathbf{s})\left(\sum_{t=0}\gamma^t r_t - V^\pi(\mathbf{s})\right)\right]$$

need to rollout entire episode

$$\nabla_\pi J(\pi) = \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})}\mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}\mathbb{E}_{p(\mathbf{s}'|\mathbf{s},\mathbf{a})}\left[\nabla_\pi \log\pi(\mathbf{a}|\mathbf{s})\left(r + \gamma V^\pi(\mathbf{s}') - V^\pi(\mathbf{s})\right)\right]$$

only need to execute a single timestep

Actor-Critic Algorithm

Actor = Policy

Critic = Value/Q-function

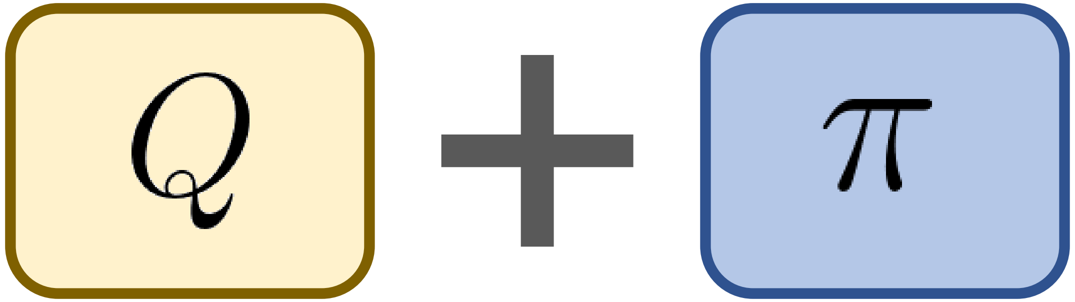
Actor

feedback

Critic

Actor-Critic Algorithm

- Combine Q-learning and policy gradient
- General and much more efficient algorithm



Policy Gradient

$$\nabla_\pi J(\pi) = \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})}\mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}\mathbb{E}_{p(\tau|\pi,\mathbf{s}_0=\mathbf{s},\mathbf{a}_0=\mathbf{a})}\left[\nabla_\pi \log\pi(\mathbf{a}|\mathbf{s})\left(\sum_{t=0}\gamma^t r_t - V^\pi(\mathbf{s})\right)\right]$$

Policy Gradient

$$\nabla_\pi J(\pi) = \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})}\mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}\mathbb{E}_{p(\tau|\pi,\mathbf{s}_0=\mathbf{s},\mathbf{a}_0=\mathbf{a})}\left[\nabla_\pi \log\pi(\mathbf{a}|\mathbf{s})\left(\sum_{t=0}\gamma^t r_t\right)\right]$$

Policy Gradient

$$\nabla_\pi J(\pi) = \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})}\mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}\underline{\mathbb{E}_{p(\tau|\pi,\mathbf{s}_0=\mathbf{s},\mathbf{a}_0=\mathbf{a})}}\left[\nabla_\pi \log\pi(\mathbf{a}|\mathbf{s})\left(\sum_{t=0}\gamma^t r_t\right)\right]$$

Policy Gradient

$$\nabla_\pi J(\pi) = \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})}\mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}\left[\nabla_\pi \log\pi(\mathbf{a}|\mathbf{s})\underbrace{\mathbb{E}_{p(\tau|\pi,\mathbf{s}_0=\mathbf{s},\mathbf{a}_0=\mathbf{a})}\left[\sum_{t=0}\gamma^t r_t\right]}\right]$$

"reward-to-go"

$$= Q^\pi(\mathbf{s},\mathbf{a})$$

Policy Gradient

$$\nabla_\pi J(\pi) = \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})}\mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}\left[\nabla_\pi \log\pi(\mathbf{a}|\mathbf{s}) \underline{Q^\pi(\mathbf{s},\mathbf{a})}\right]$$

Policy Gradient

Why is policy gradient so inefficient?

- Estimating gradient requires estimating the return of policy
- Estimating the return requires rolling out the policy

$$\nabla_\pi J(\pi) = \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})}\mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}\left[\nabla_\pi \log \pi(\mathbf{a}|\mathbf{s}) \underline{Q^\pi(\mathbf{s},\mathbf{a})}\right]$$

$$\mathbb{E}_{\tau\sim p(\tau|\pi,\mathbf{s}_0=\mathbf{s},\mathbf{a}_0=\mathbf{a})}\left(\sum_{t=0}\gamma^t r_t\right)$$

Actor-Critic Algorithm

Idea: Replace Monte-Carlo return estimator with a learned Q-function

Can estimate gradients without collecting new data

$$\nabla_\pi J(\pi) \approx \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})}\mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}\left[\nabla_\pi \log \pi(\mathbf{a}|\mathbf{s})\hat{Q}^\pi(\mathbf{s},\mathbf{a})\right]$$

$$\hat{Q}^\pi(\mathbf{s},\mathbf{a}) \approx Q^\pi(\mathbf{s},\mathbf{a})$$

Surrogate Objective

$$\hat{Q}^{\pi}(\mathbf{s}, \mathbf{a}) \approx Q^{\pi}(\mathbf{s}, \mathbf{a})$$

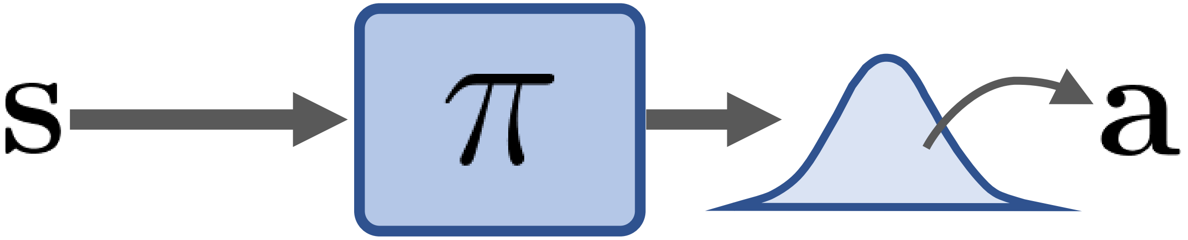
$$\Downarrow$$

$$\hat{J}(\pi) \approx J(\pi)$$

✓ differentiable ✗ nondifferentiable

Deterministic Policy Gradient (DPG)

$$\nabla_\pi \hat{J}(\pi) = \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})}\mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}\left[\nabla_\pi \log\pi(\mathbf{a}|\mathbf{s})\hat{Q}^\pi(\mathbf{s},\mathbf{a})\right]$$



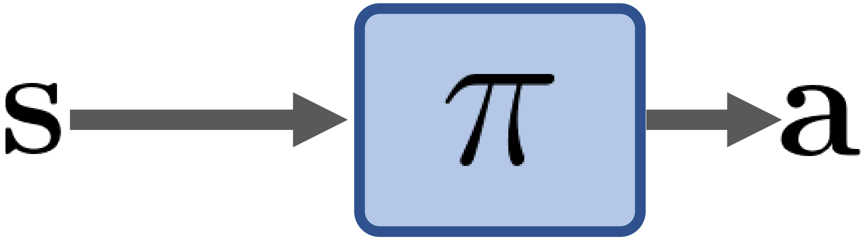
Stochastic Policy: $\pi(\mathbf{a}|\mathbf{s})$

Deterministic Policy Gradient Algorithms [Silver et al. 2014]
Continuous control with deep reinforcement learning [Lillicrap et al. 2016]

Deterministic Policy Gradient (DPG)

$$\nabla_{\pi} \hat{J}(\pi) = \mathbb{E}_{\mathbf{s} \sim d_{\pi}(\mathbf{s})} \mathbb{E}_{\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})} \left[\underline{\nabla_{\pi} \log \pi(\mathbf{a}|\mathbf{s})} \hat{Q}^{\pi}(\mathbf{s}, \mathbf{a}) \right]$$

nondifferentiable

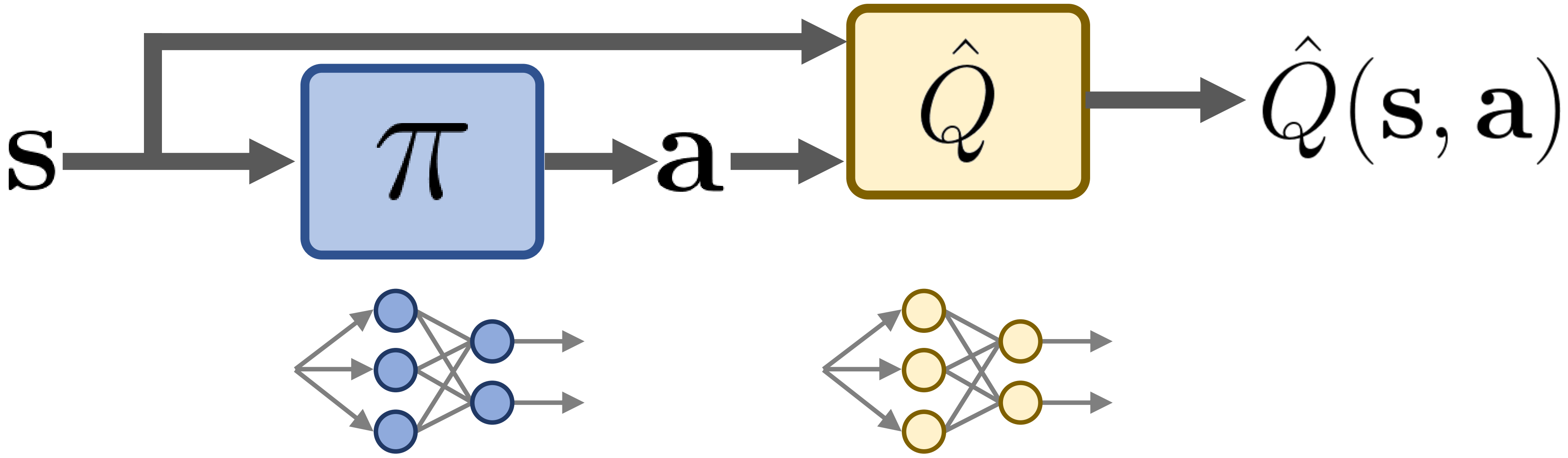


Deterministic Policy: $\mathbf{a} = \pi(\mathbf{s})$

Deterministic Policy Gradient Algorithms [Silver et al. 2014]
Continuous control with deep reinforcement learning [Lillicrap et al. 2016]

Deterministic Policy Gradient (DPG)

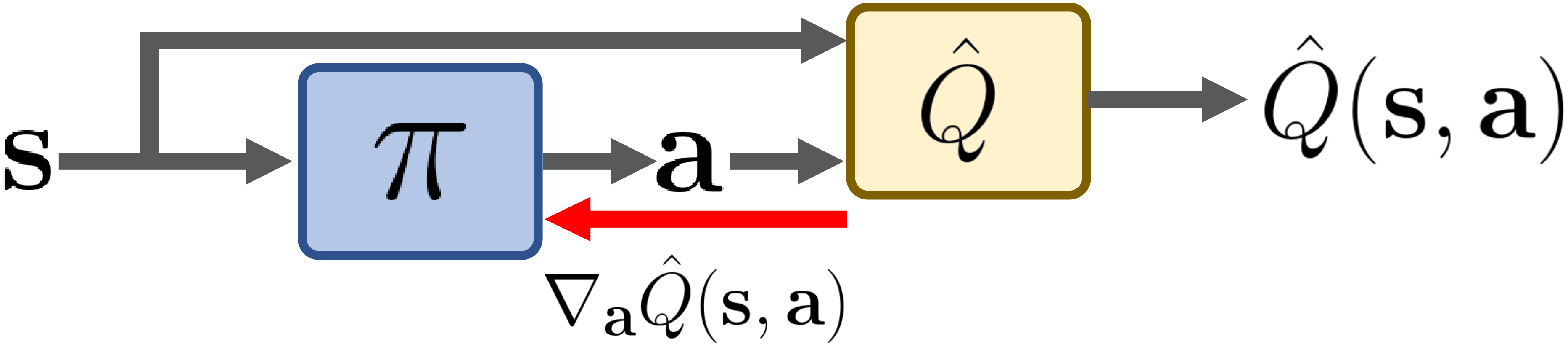
$$\nabla_\pi \hat{J}(\pi) = \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})}\left[\nabla_\pi \hat{Q}^\pi(\mathbf{s}, \pi(\mathbf{s}))\right]$$



Deterministic Policy Gradient Algorithms [Silver et al. 2014]
Continuous control with deep reinforcement learning [Lillicrap et al. 2016]

Deterministic Policy Gradient (DPG)

$$\nabla_\pi \hat{J}(\pi) = \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})}\left[\nabla_\pi \hat{Q}^\pi(\mathbf{s}, \pi(\mathbf{s}))\right]$$



Directly backprop from $\hat{Q}$ to π

Deterministic Policy Gradient Algorithms [Silver et al. 2014]
Continuous control with deep reinforcement learning [Lillicrap et al. 2016]

Deterministic Policy Gradient (DPG)

$$\nabla_\pi \hat{J}(\pi) = \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})}\left[\nabla_\pi \underline{\hat{Q}^\pi(\mathbf{s}, \pi(\mathbf{s}))}\right]$$

deterministic
no variance

Monte-Carlo Return Estimator:

$$\nabla_\pi J(\pi) = \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})}\mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}\left[\nabla_\pi \log \pi(\mathbf{a}|\mathbf{s}) \underline{\mathbb{E}_{p(\tau|\pi,\mathbf{s}_0=\mathbf{s},\mathbf{a}_0=\mathbf{a})}\left[\sum_{t=0} \gamma^t r_t\right]}\right]$$

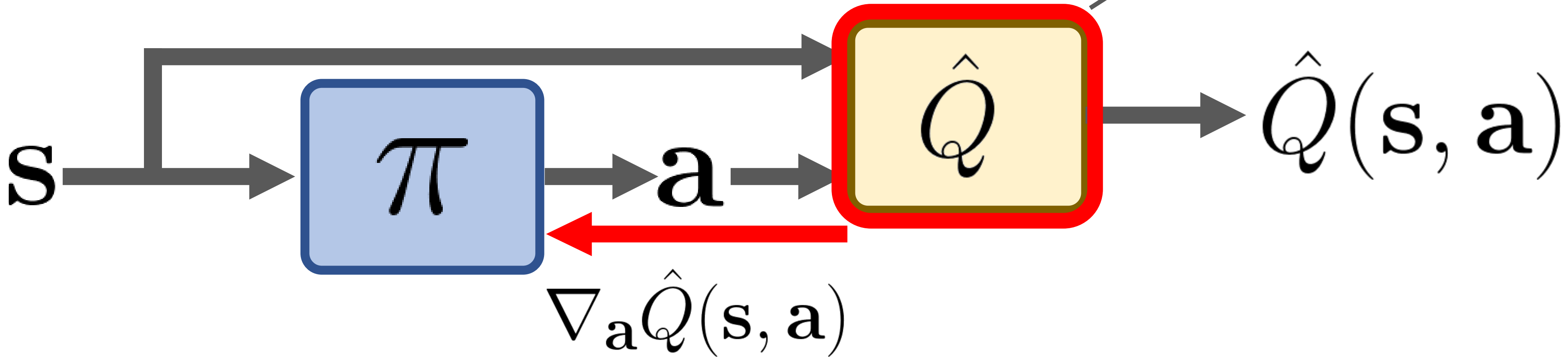
stochastic
high variance

Deterministic Policy Gradient Algorithms [Silver et al. 2014]
Continuous control with deep reinforcement learning [Lillicrap et al. 2016]

Deterministic Policy Gradient (DPG)

$$\nabla_{\pi}\hat{J}(\pi) = \mathbb{E}_{\mathbf{s}\sim d_{\pi}(\mathbf{s})}\left[\nabla_{\pi}\hat{Q}^{\pi}(\mathbf{s}, \pi(\mathbf{s}))\right]$$

How to train Q-function?



Directly backprop from $\hat{Q}$ to π

Deterministic Policy Gradient Algorithms [Silver et al. 2014]
Continuous control with deep reinforcement learning [Lillicrap et al. 2016]

Deterministic Policy Gradient (DPG)

$$Q^{k+1} = \arg\min_{Q} \mathbb{E}_{(\mathbf{s},\mathbf{a},r,\mathbf{s}')\sim\mathcal{D}} \left[\left(\left(r + \gamma \max_{\mathbf{a}'} Q^k(\mathbf{s}', \mathbf{a}') \right) - Q(\mathbf{s}, \mathbf{a}) \right)^2 \right]$$

Intractable in continuous action spaces

- Max over actions needed for learning the optimal Q-function Q^*
- Learn Q^π instead of Q^*

Deterministic Policy Gradient Algorithms [Silver et al. 2014]
Continuous control with deep reinforcement learning [Lillicrap et al. 2016]

Recursive Definition

Optimal Q-function:

$$Q^*(\mathbf{s},\mathbf{a}) = \mathbb{E}_{\mathbf{s}'\sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})}\left[r(\mathbf{s},\mathbf{a},\mathbf{s}') + \gamma \underline{\mathbb{E}_{\mathbf{a}'\sim\pi^*(\mathbf{a}'|\mathbf{s}')}}\left[Q^*(\mathbf{s}',\mathbf{a}')\right]\right]$$

True for all policies

$$Q^*(\mathbf{s},\mathbf{a}) = \mathbb{E}_{\mathbf{s}'\sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})}\left[r(\mathbf{s},\mathbf{a},\mathbf{s}') + \gamma \underline{\max_{\mathbf{a}'}}\left(Q^*(\mathbf{s}',\mathbf{a}')\right)\right]$$

Only true for optimal Q-function

Recursive Definition

Optimal Q-function:

$$Q^*(\mathbf{s}, \mathbf{a}) = \mathbb{E}_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})} \left[r(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma \mathbb{E}_{\mathbf{a}' \sim \pi^*(\mathbf{a}'|\mathbf{s}')} \left[Q^*(\mathbf{s}', \mathbf{a}') \right] \right]$$

General policy:

$$Q^\pi(\mathbf{s}, \mathbf{a}) = \mathbb{E}_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})} \left[r(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma \mathbb{E}_{\mathbf{a}' \sim \pi(\mathbf{a}'|\mathbf{s}')} \left[Q^\pi(\mathbf{s}', \mathbf{a}') \right] \right]$$

Deterministic Policy Gradient (DPG)

$$Q^{k+1} = \arg\min_{Q} \mathbb{E}_{(\mathbf{s},\mathbf{a},r,\mathbf{s}')\sim\mathcal{D}}\left[\left(\left(r + \gamma \underline{\mathbb{E}_{\mathbf{a}'\sim\pi(\mathbf{a}'|\mathbf{s}')}}\left[Q^k(\mathbf{s}',\mathbf{a}')\right]\right) - Q(\mathbf{s},\mathbf{a})\right)^2\right]$$

Deterministic Policy Gradient (DPG)

ALGORITHM: DPG

$Q^0 \leftarrow$ initialize Q-function
$\pi^0 \leftarrow$ initialize policy
$\mathcal{D} \leftarrow \{\emptyset\}$ initialize dataset

for iteration $k = 0, ..., n-1$ **do**
  Sample trajectory τ according to $\pi^k(\mathbf{s})$
  Add transitions to dataset $\mathcal{D} = \mathcal{D} \cup \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}$

  Calculate target values for each sample i:
  $y_i = r_i + \gamma Q^k(\mathbf{s}'_i, \pi(\mathbf{s}'_i))$

  Update Q-function:
  $Q^{k+1} = \arg\min_Q \mathbb{E}_{(\mathbf{s}_i,\mathbf{a}_i,\mathbf{r}_i,\mathbf{s}'_i)\sim\mathcal{D}} \left[(y_i - Q(\mathbf{s}_i, \mathbf{a}_i))^2\right]$

  Update policy:
  $\pi^{k+1} = \arg\max_\pi \mathbb{E}_{\mathbf{s}_i\sim\mathcal{D}} \left[Q^{k+1}(\mathbf{s}_i, \pi(\mathbf{s}_i))\right]$
end for

return π^n

Continuous control with deep reinforcement learning [Lillicrap et al. 2016]

Deterministic Policy Gradient (DPG)

ALGORITHM: DPG

1: $Q^0 \leftarrow$ initialize Q-function
2: $\pi^0 \leftarrow$ initialize policy
3: $\mathcal{D} \leftarrow \{\emptyset\}$ initialize dataset

4: **for** iteration $k = 0, ..., n-1$ **do**
5: Sample trajectory τ according to $\pi^k(\mathbf{s})$
6: Add transitions to dataset $\mathcal{D} = \mathcal{D} \cup \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}$

7: Calculate target values for each sample i:
$y_i = r_i + \gamma Q^k(\mathbf{s}'_i, \pi(\mathbf{s}'_i))$

8: Update Q-function:
$Q^{k+1} = \arg\min_Q \ \mathbb{E}_{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{r}_i, \mathbf{s}'_i) \sim \mathcal{D}} \left[(y_i - Q(\mathbf{s}_i, \mathbf{a}_i))^2 \right]$

9: Update policy:
$\pi^{k+1} = \arg\max_\pi \ \mathbb{E}_{\mathbf{s}_i \sim \mathcal{D}} \left[Q^{k+1}(\mathbf{s}_i, \pi(\mathbf{s}_i)) \right]$
10: **end for**

11: return π^n

Continuous control with deep reinforcement learning [Lillicrap et al. 2016]

Deterministic Policy Gradient (DPG)

ALGORITHM: DPG

$Q^0 \leftarrow$ initialize Q-function
$\pi^0 \leftarrow$ initialize policy
$\mathcal{D} \leftarrow \{\emptyset\}$ initialize dataset

for iteration $k = 0, ..., n-1$ **do**
  Sample trajectory τ according to $\pi^k(\mathbf{s})$
  Add transitions to dataset $\mathcal{D} = \mathcal{D} \cup \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}$

  Calculate target values for each sample i:
  $y_i = r_i + \gamma Q^k(\mathbf{s}'_i, \pi(\mathbf{s}'_i))$

  Update Q-function:
  $Q^{k+1} = \arg\min_Q \ \mathbb{E}_{(\mathbf{s}_i,\mathbf{a}_i,\mathbf{r}_i,\mathbf{s}'_i)\sim\mathcal{D}} \left[(y_i - Q(\mathbf{s}_i, \mathbf{a}_i))^2\right]$

  Update policy:
  $\pi^{k+1} = \arg\max_\pi \ \mathbb{E}_{\mathbf{s}_i\sim\mathcal{D}} \left[Q^{k+1}(\mathbf{s}_i, \pi(\mathbf{s}_i))\right]$
end for

return π^n

Continuous control with deep reinforcement learning [Lillicrap et al. 2016]

Deterministic Policy Gradient (DPG)

ALGORITHM: DPG

$Q^0 \leftarrow$ initialize Q-function
$\pi^0 \leftarrow$ initialize policy
$\mathcal{D} \leftarrow \{\emptyset\}$ initialize dataset

for iteration $k = 0, ..., n-1$ **do**
  Sample trajectory τ according to $\pi^k(\mathbf{s})$
  Add transitions to dataset $\mathcal{D} = \mathcal{D} \cup \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}$

  Calculate target values for each sample i:
  $y_i = r_i + \gamma Q^k(\mathbf{s}'_i, \pi(\mathbf{s}'_i))$

  Update Q-function:
  $Q^{k+1} = \arg\min_Q \ \mathbb{E}_{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{r}_i, \mathbf{s}'_i) \sim \mathcal{D}} \left[(y_i - Q(\mathbf{s}_i, \mathbf{a}_i))^2 \right]$

  Update policy:
  $\pi^{k+1} = \arg\max_\pi \ \mathbb{E}_{\mathbf{s}_i \sim \mathcal{D}} \left[Q^{k+1}(\mathbf{s}_i, \pi(\mathbf{s}_i)) \right]$
end for

return π^n

Continuous control with deep reinforcement learning [Lillicrap et al. 2016]

Deterministic Policy Gradient (DPG)

ALGORITHM: DPG

$Q^0 \leftarrow$ initialize Q-function
$\pi^0 \leftarrow$ initialize policy
$\mathcal{D} \leftarrow \{\emptyset\}$ initialize dataset

for iteration $k = 0, ..., n-1$ **do**
  Sample trajectory τ according to $\pi^k(\mathbf{s})$
  Add transitions to dataset $\mathcal{D} = \mathcal{D} \cup \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}$

  Calculate target values for each sample i:
  $y_i = r_i + \gamma Q^k(\mathbf{s}'_i, \pi(\mathbf{s}'_i))$

  Update Q-function:
  $Q^{k+1} = \arg\min_Q \ \mathbb{E}_{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{r}_i, \mathbf{s}'_i) \sim \mathcal{D}} \left[(y_i - Q(\mathbf{s}_i, \mathbf{a}_i))^2 \right]$

  Update policy:
  $\pi^{k+1} = \arg\max_\pi \ \mathbb{E}_{\mathbf{s}_i \sim \mathcal{D}} \left[Q^{k+1}(\mathbf{s}_i, \pi(\mathbf{s}_i)) \right]$
end for

return π^n

Continuous control with deep reinforcement learning [Lillicrap et al. 2016]

Deterministic Policy Gradient (DPG)

ALGORITHM: DPG

$Q^0 \leftarrow$ initialize Q-function
$\pi^0 \leftarrow$ initialize policy
$\mathcal{D} \leftarrow \{\emptyset\}$ initialize dataset

for iteration $k = 0, ..., n-1$ **do**
  Sample trajectory τ according to $\pi^k(\mathbf{s})$
  Add transitions to dataset $\mathcal{D} = \mathcal{D} \cup \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}$

  Calculate target values for each sample i:
  $y_i = r_i + \gamma Q^k(\mathbf{s}'_i, \pi(\mathbf{s}'_i))$

  Update Q-function:
  $Q^{k+1} = \arg\min_Q \ \mathbb{E}_{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{r}_i, \mathbf{s}'_i) \sim \mathcal{D}} \left[(y_i - Q(\mathbf{s}_i, \mathbf{a}_i))^2 \right]$

  Update policy:
  $\pi^{k+1} = \arg\max_\pi \ \mathbb{E}_{\mathbf{s}_i \sim \mathcal{D}} \left[Q^{k+1}(\mathbf{s}_i, \pi(\mathbf{s}_i)) \right]$
end for

return π^n

Continuous control with deep reinforcement learning [Lillicrap et al. 2016]

Deterministic Policy Gradient (DPG)

ALGORITHM: DPG

$Q^0 \leftarrow$ initialize Q-function
$\pi^0 \leftarrow$ initialize policy
$\mathcal{D} \leftarrow \{\emptyset\}$ initialize dataset

for iteration $k = 0, ..., n-1$ **do**
  Sample trajectory τ according to $\pi^k(\mathbf{s})$
  Add transitions to dataset $\mathcal{D} = \mathcal{D} \cup \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}$

  Calculate target values for each sample i:
  $y_i = r_i + \gamma Q^k(\mathbf{s}'_i, \pi(\mathbf{s}'_i))$

  Update Q-function:
  $Q^{k+1} = \arg\min_Q \; \mathbb{E}_{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{r}_i, \mathbf{s}'_i) \sim \mathcal{D}} \left[(y_i - Q(\mathbf{s}_i, \mathbf{a}_i))^2 \right]$

  Update policy:
  $\pi^{k+1} = \arg\max_\pi \; \mathbb{E}_{\mathbf{s}_i \sim \mathcal{D}} \left[Q^{k+1}(\mathbf{s}_i, \pi(\mathbf{s}_i)) \right]$
end for

return π^n

Continuous control with deep reinforcement learning [Lillicrap et al. 2016]

Deterministic Policy Gradient (DPG)

ALGORITHM: DPG

$Q^0 \leftarrow$ initialize Q-function
$\pi^0 \leftarrow$ initialize policy
$\mathcal{D} \leftarrow \{\emptyset\}$ initialize dataset

for iteration $k = 0, ..., n-1$ **do**
    Sample trajectory τ according to $\pi^k(\mathbf{s})$
    Add transitions to dataset $\mathcal{D} = \mathcal{D} \cup \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}$

    Calculate target values for each sample i:
    $y_i = r_i + \gamma Q^k(\mathbf{s}'_i, \pi(\mathbf{s}'_i))$

    Update Q-function:
    $Q^{k+1} = \arg\min_Q \; \mathbb{E}_{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{r}_i, \mathbf{s}'_i) \sim \mathcal{D}} \left[(y_i - Q(\mathbf{s}_i, \mathbf{a}_i))^2 \right]$

    Update policy:
    $\pi^{k+1} = \arg\max_\pi \; \mathbb{E}_{\mathbf{s}_i \sim \mathcal{D}} \left[Q^{k+1}(\mathbf{s}_i, \pi(\mathbf{s}_i)) \right]$
end for

return π^n

Continuous control with deep reinforcement learning [Lillicrap et al. 2016]

Deterministic Policy Gradient (DPG)

ALGORITHM: DPG

$Q^0 \leftarrow$ initialize Q-function
$\pi^0 \leftarrow$ initialize policy
$\mathcal{D} \leftarrow \{\emptyset\}$ initialize dataset

for iteration $k = 0, ..., n-1$ **do**
  Sample trajectory τ according to $\pi^k(\mathbf{s})$
  Add transitions to dataset $\mathcal{D} = \mathcal{D} \cup \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}$

  Calculate target values for each sample i:
  $y_i = r_i + \gamma Q^k(\mathbf{s}'_i, \pi(\mathbf{s}'_i))$

  Update Q-function:
  $Q^{k+1} = \arg\min_Q \; \mathbb{E}_{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{r}_i, \mathbf{s}'_i) \sim \mathcal{D}} \left[(y_i - Q(\mathbf{s}_i, \mathbf{a}_i))^2 \right]$

  Update policy:
  $\pi^{k+1} = \arg\max_\pi \; \mathbb{E}_{\mathbf{s}_i \sim \mathcal{D}} \left[Q^{k+1}(\mathbf{s}_i, \pi(\mathbf{s}_i)) \right]$
end for

return π^n

Continuous control with deep reinforcement learning [Lillicrap et al. 2016]

Deterministic Policy Gradient (DPG)

ALGORITHM: DPG

$Q^0 \leftarrow$ initialize Q-function
$\pi^0 \leftarrow$ initialize policy
$\mathcal{D} \leftarrow \{\emptyset\}$ initialize dataset

for iteration $k = 0, ..., n-1$ **do**
    Sample trajectory τ according to $\pi^k(\mathbf{s})$
    Add transitions to dataset $\mathcal{D} = \mathcal{D} \cup \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}$

    Calculate target values for each sample i:
    $y_i = r_i + \gamma Q^k(\mathbf{s}'_i, \pi(\mathbf{s}'_i))$

    Update Q-function:
    $Q^{k+1} = \arg\min_Q \; \mathbb{E}_{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{r}_i, \mathbf{s}'_i) \sim \mathcal{D}} \left[(y_i - Q(\mathbf{s}_i, \mathbf{a}_i))^2 \right]$

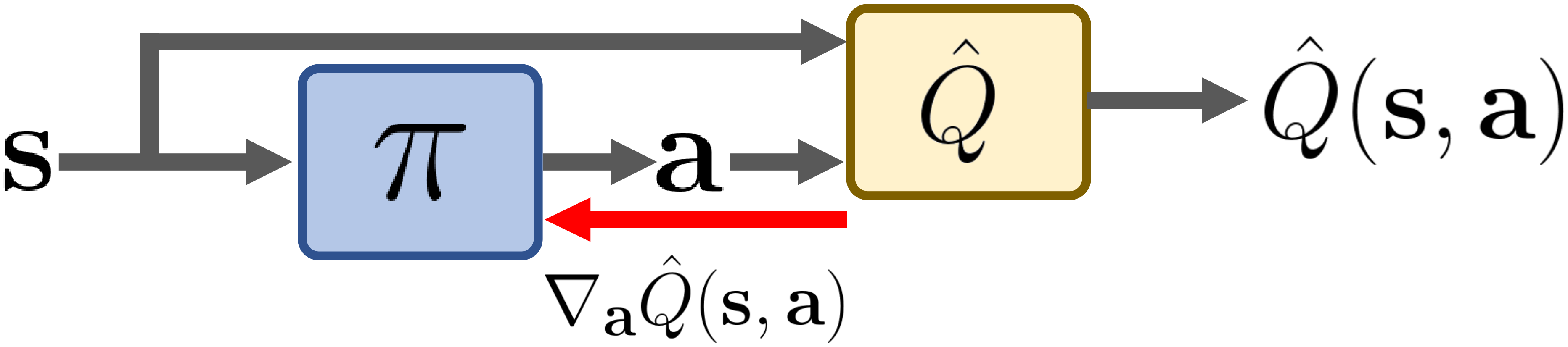
    Update policy:
    $\pi^{k+1} = \arg\max_\pi \; \mathbb{E}_{\mathbf{s}_i \sim \mathcal{D}} \left[Q^{k+1}(\mathbf{s}_i, \pi(\mathbf{s}_i)) \right]$
end for

return π^n

Continuous control with deep reinforcement learning [Lillicrap et al. 2016]

Deterministic Policy

$$\nabla_\pi \hat{J}(\pi) = \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})}\left[\nabla_\pi \hat{Q}^\pi(\mathbf{s}, \pi(\mathbf{s}))\right]$$

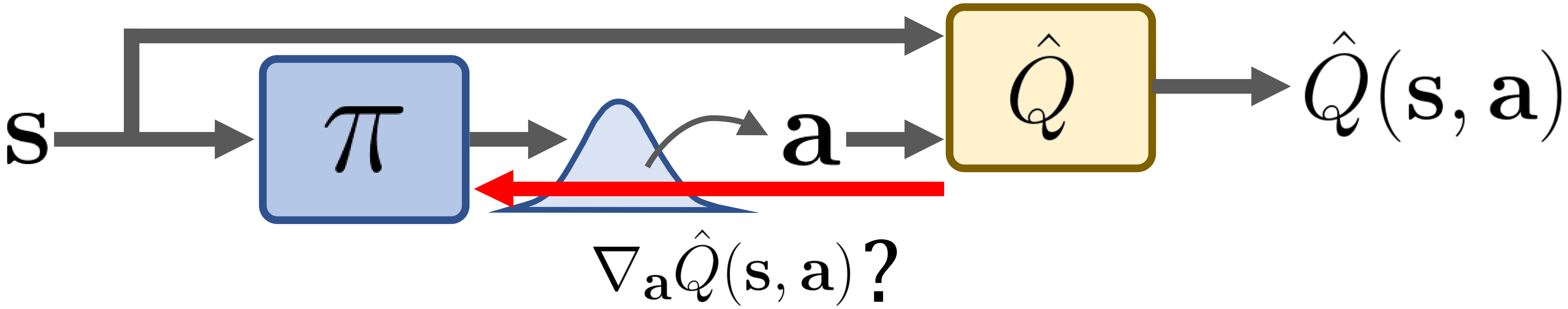


Directly backprop from $\hat{Q}$ to π

Deterministic Policy Gradient Algorithms [Silver et al. 2014]
Continuous control with deep reinforcement learning [Lillicrap et al. 2016]

Stochastic Policy

$$\nabla_\pi \hat{J}(\pi) = \nabla_\pi \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})} \mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})} \left[\hat{Q}^\pi(\mathbf{s},\mathbf{a}) \right]$$



Stochastic Policy

$$\nabla_\pi \hat{J}(\pi) = \nabla_\pi \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})}\mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}\left[\hat{Q}^\pi(\mathbf{s},\mathbf{a})\right]$$

⇓ Score Function

$$\nabla_\pi \hat{J}(\pi) = \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})}\mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}\left[\underline{\nabla_\pi \log \pi(\mathbf{a}|\mathbf{s})\hat{Q}^\pi(\mathbf{s},\mathbf{a})}\right]$$

high variance

Better method: reparameterization trick

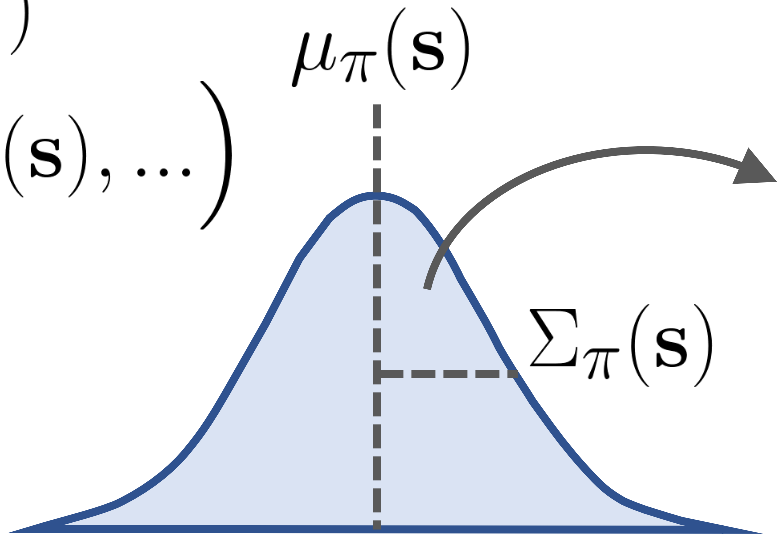
Reparameterization Trick

$$\nabla_\pi \hat{J}(\pi) = \nabla_\pi \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})} \underline{\mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}} \left[\hat{Q}^\pi(\mathbf{s}, \mathbf{a}) \right]$$

Gaussian policy:

$$\pi(\cdot|\mathbf{s}) = N(\mu_\pi(\mathbf{s}), \Sigma_\pi(\mathbf{s}))$$

$$\Sigma_\pi(\mathbf{s}) = \mathrm{diag}\left(\sigma_\pi^1(\mathbf{s}), \sigma_\pi^2(\mathbf{s}), \ldots\right)$$



$$\mathbf{a} = \mu_\pi(\mathbf{s}) + \Sigma_\pi(\mathbf{s})\epsilon$$

$$\epsilon \sim N(0, I)$$

Reparameterization Trick

$$\nabla_\pi \hat{J}(\pi) = \nabla_\pi \mathbb{E}_{\mathbf{s} \sim d_\pi(\mathbf{s})} \underline{\mathbb{E}_{\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})}} \left[\hat{Q}^\pi(\mathbf{s}, \mathbf{a}) \right]$$

Reparameterization Trick

$$\nabla_\pi \hat{J}(\pi) = \nabla_\pi \mathbb{E}_{\mathbf{s} \sim d_\pi(\mathbf{s})} \underline{\mathbb{E}_{\epsilon \sim N(0,I)}} \left[\hat{Q}^\pi \left(\mathbf{s}, \mu_\pi(\mathbf{s}) + \Sigma_\pi(\mathbf{s}) \epsilon \right) \right]$$

Reparameterization Trick

$$\nabla_\pi \hat{J}(\pi) = \nabla_\pi \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})} \underline{\mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}} \left[\hat{Q}^\pi(\mathbf{s},\mathbf{a})\right]$$

Reparameterization Trick

$$\nabla_\pi \hat{J}(\pi) = \nabla_\pi \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})} \mathbb{E}_{\epsilon\sim N(0,I)} \left[\hat{Q}^\pi\left(\mathbf{s}, \underbrace{\mu_\pi(\mathbf{s}) + \Sigma_\pi(\mathbf{s})\epsilon}_{=\,\mathbf{a}}\right)\right]$$

Reparameterization Trick

$$\nabla_\pi \hat{J}(\pi) = \nabla_\pi \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})} \underline{\mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}} \left[\hat{Q}^\pi(\mathbf{s}, \mathbf{a}) \right]$$

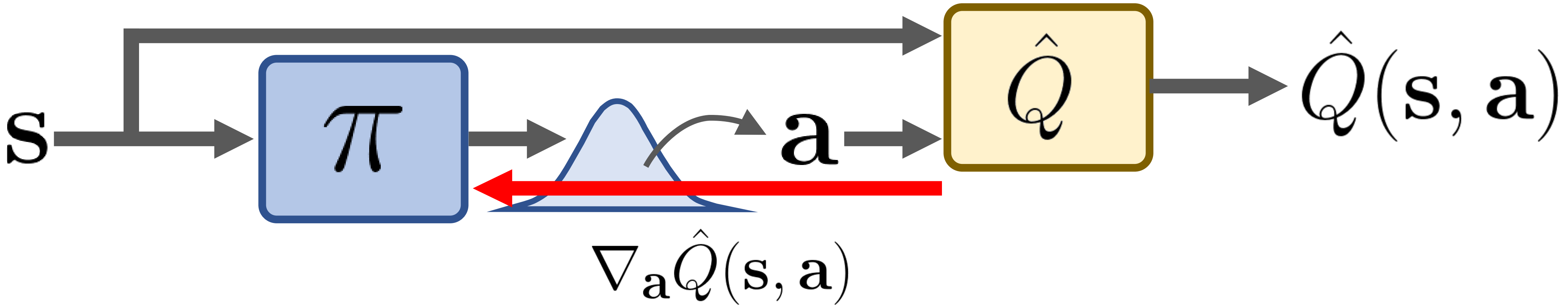
Reparameterization Trick

$$\nabla_\pi \hat{J}(\pi) = \nabla_\pi \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})} \underline{\mathbb{E}_{\epsilon\sim N(0,I)}} \left[\hat{Q}^\pi \left(\mathbf{s}, \mu_\pi(\mathbf{s}) + \Sigma_\pi(\mathbf{s})\epsilon \right) \right]$$

$$\nabla_\pi \hat{J}(\pi) = \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})} \mathbb{E}_{\epsilon\sim N(0,I)} \left[\underline{\nabla_\pi} \hat{Q}^\pi \left(\mathbf{s}, \mu_\pi(\mathbf{s}) + \Sigma_\pi(\mathbf{s})\epsilon \right) \right]$$

Soft Actor-Critic (SAC)

$$\nabla_\pi \hat{J}(\pi) = \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})}\mathbb{E}_{\epsilon\sim N(0,I)}\left[\nabla_\pi \hat{Q}^\pi\left(\mathbf{s}, \mu_\pi(\mathbf{s}) + \Sigma_\pi(\mathbf{s})\epsilon\right)\right]$$



Reparameterization Trick

Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor [Haarnoja et al. 2018]

Soft Actor-Critic (SAC)

ALGORITHM: SAC

1: $Q^0 \leftarrow$ initialize Q-function
2: $\pi^0 \leftarrow$ initialize policy
3: $\mathcal{D} \leftarrow \{\emptyset\}$ initialize dataset

4: **for** iteration $k = 0, ..., n-1$ **do**
5: Sample trajectory τ according to $\pi^k(\mathbf{a}|\mathbf{s})$
6: Add transitions to dataset $\mathcal{D} = \mathcal{D} \cup \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}$

7: Calculate target values for each sample i:
$y_i = r_i + \gamma \mathbb{E}_{\mathbf{a}' \sim \pi^k(\mathbf{a}'|\mathbf{s}'_i)} \left[Q^k(\mathbf{s}'_i, \mathbf{a}') \right]$

8: Update Q-function:
$Q^{k+1} = \arg\min_Q \ \mathbb{E}_{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{r}_i, \mathbf{s}'_i) \sim \mathcal{D}} \left[(y_i - Q(\mathbf{s}_i, \mathbf{a}_i))^2 \right]$

9: Update policy:
$\pi^{k+1} = \arg\max_\pi \ \mathbb{E}_{\mathbf{s}_i \sim \mathcal{D}} \mathbb{E}_{\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s}_i)} \left[Q^{k+1}(\mathbf{s}_i, \mathbf{a}) \right]$
10: **end for**

11: return π^n

ALGORITHM: DPG

1: $Q^0 \leftarrow$ initialize Q-function
2: $\pi^0 \leftarrow$ initialize policy
3: $\mathcal{D} \leftarrow \{\emptyset\}$ initialize dataset

4: **for** iteration $k = 0, ..., n-1$ **do**
5: Sample trajectory τ according to $\pi^k(\mathbf{s})$
6: Add transitions to dataset $\mathcal{D} = \mathcal{D} \cup \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}$

7: Calculate target values for each sample i:
$y_i = r_i + \gamma Q^k(\mathbf{s}'_i, \pi(\mathbf{s}'_i))$

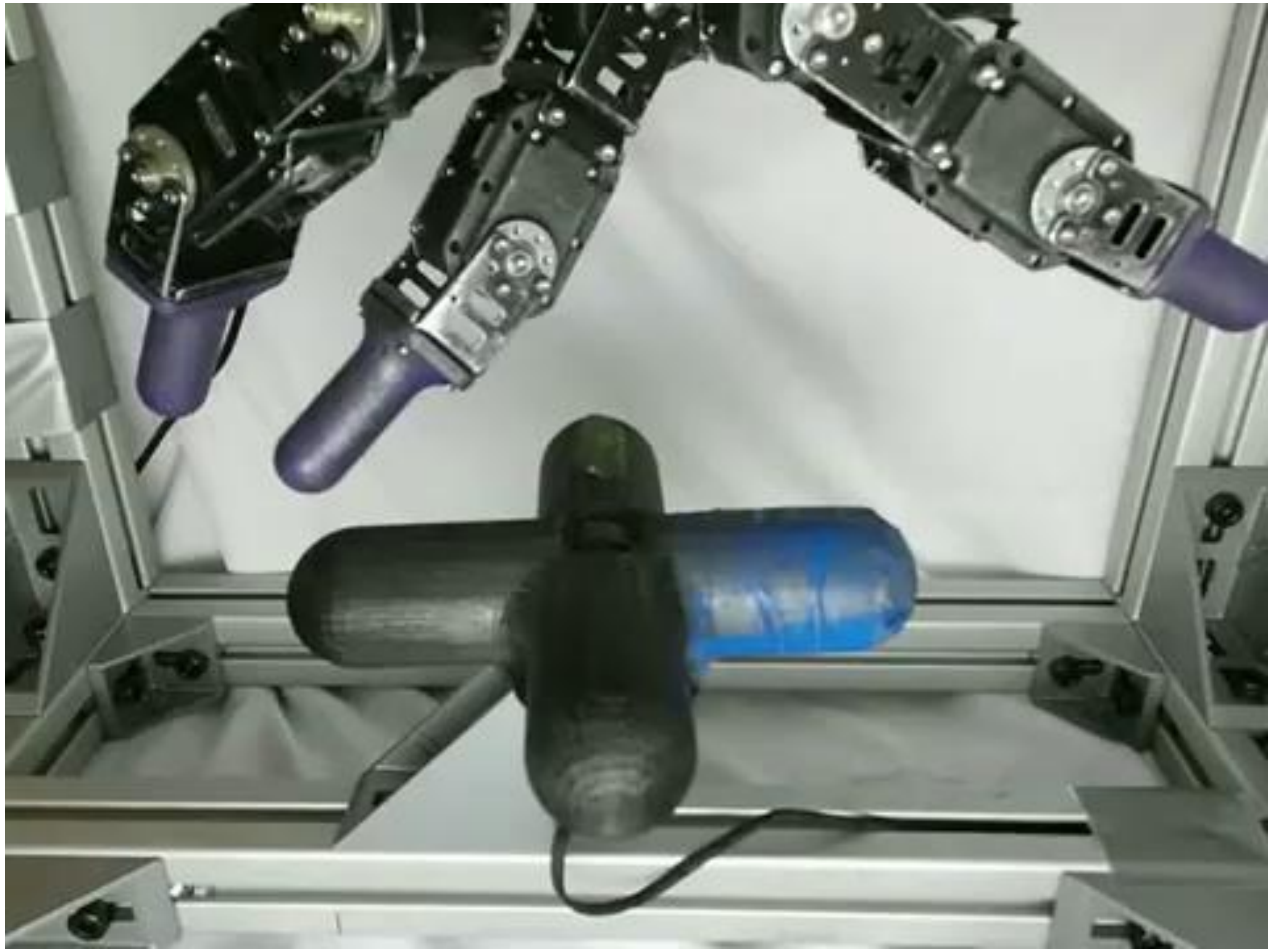
8: Update Q-function:
$Q^{k+1} = \arg\min_Q \ \mathbb{E}_{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{r}_i, \mathbf{s}'_i) \sim \mathcal{D}} \left[(y_i - Q(\mathbf{s}_i, \mathbf{a}_i))^2 \right]$

9: Update policy:
$\pi^{k+1} = \arg\max_\pi \ \mathbb{E}_{\mathbf{s}_i \sim \mathcal{D}} \left[Q^{k+1}(\mathbf{s}_i, \pi(\mathbf{s}_i)) \right]$
10: **end for**

11: return π^n

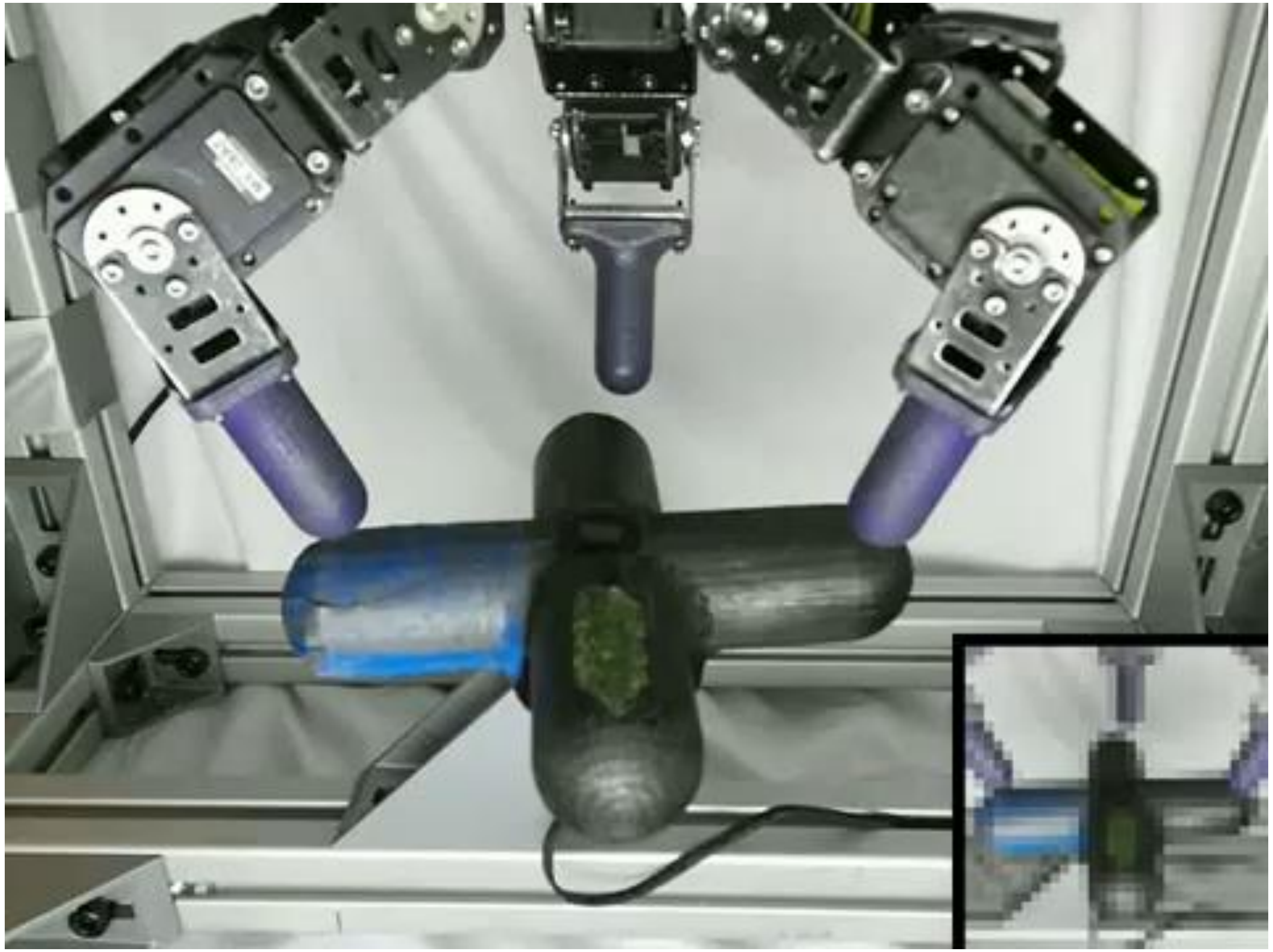
Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor [Haarnoja et al. 2018]

Soft Actor-Critic (SAC)



Soft Actor-Critic Algorithms and Applications
[Haarnoja et al. 2019]

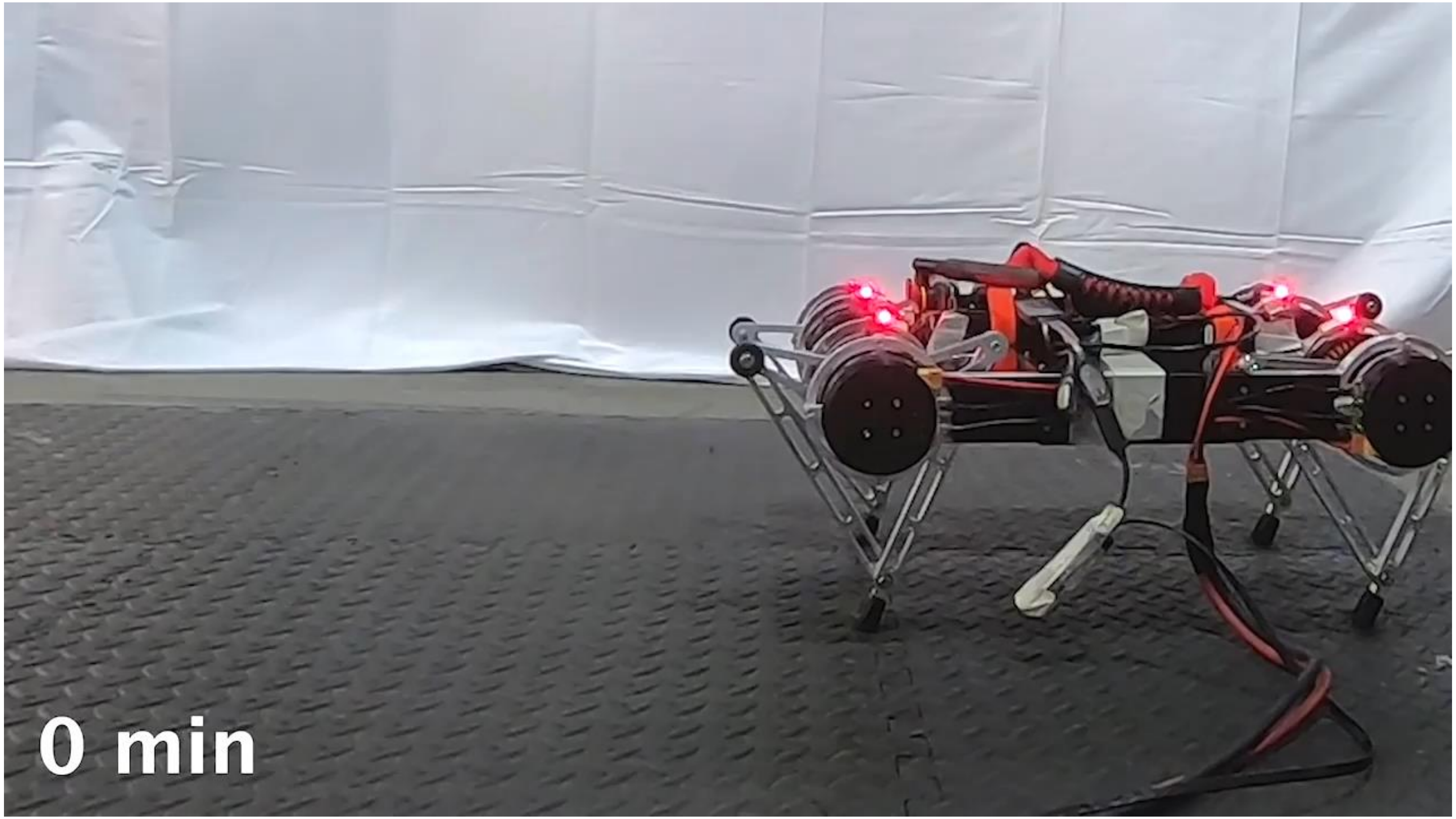
Soft Actor-Critic (SAC)



20 hours later
300k samples

Soft Actor-Critic Algorithms and Applications
[Haarnoja et al. 2019]

Soft Actor-Critic (SAC)



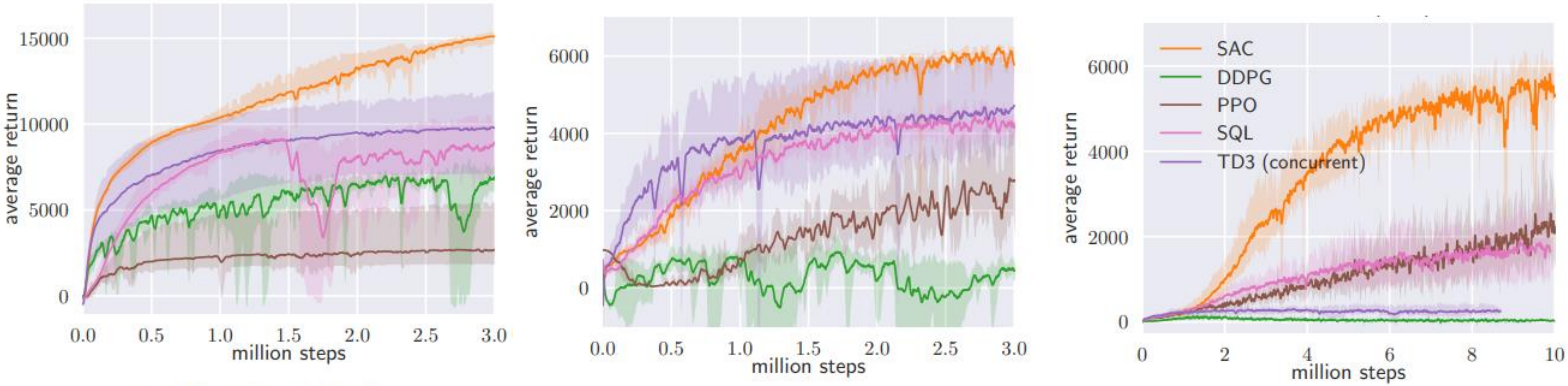
Soft Actor-Critic Algorithms and Applications
[Haarnoja et al. 2019]

Soft Actor-Critic (SAC)

2 hours later
160k samples

Soft Actor-Critic Algorithms and Applications
[Haarnoja et al. 2019]

Soft Actor-Critic (SAC)

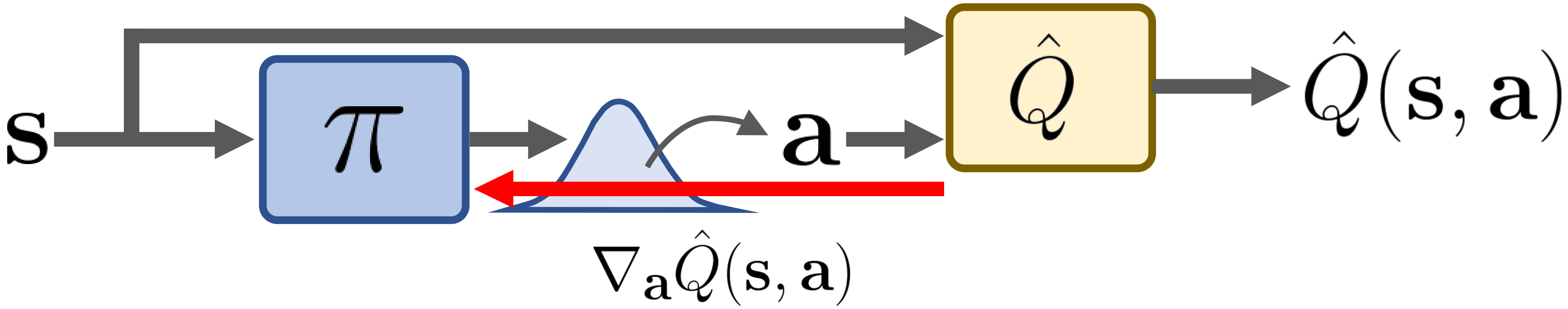


(c) HalfCheetah-v1 (d) Ant-v1 (f) Humanoid (rllab)

Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor [Haarnoja et al. 2018]

Continuous Actions

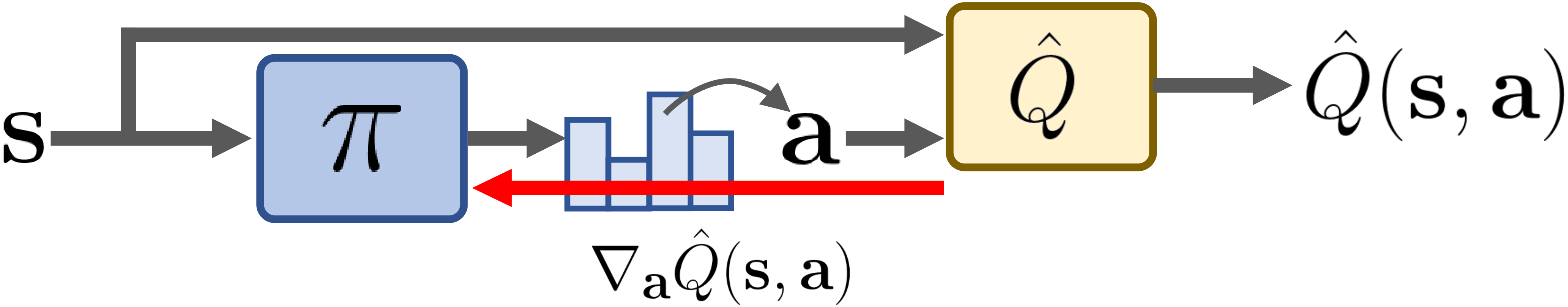
$$\nabla_\pi \hat{J}(\pi) = \nabla_\pi \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})} \mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})} \left[\hat{Q}^\pi(\mathbf{s}, \mathbf{a}) \right]$$



Reparameterization Trick

Discrete Actions

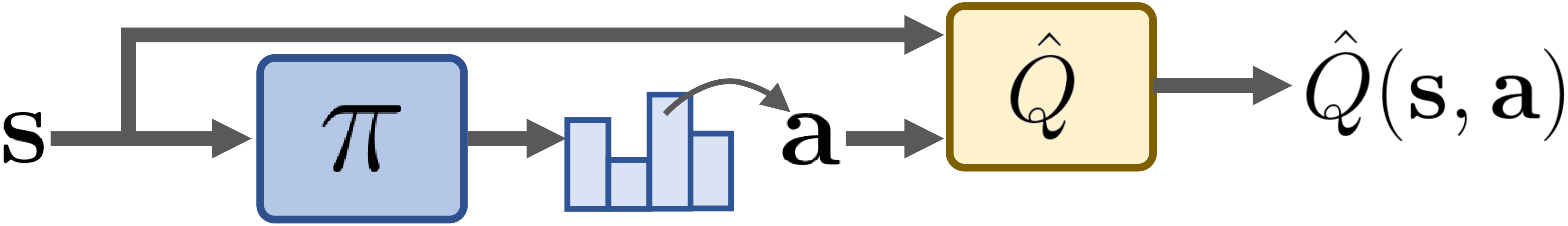
$$\nabla_\pi \hat{J}(\pi) = \nabla_\pi \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})} \mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})} \left[\hat{Q}^\pi(\mathbf{s}, \mathbf{a}) \right]$$



✗ Reparameterization Trick

Discrete Actions

$$\nabla_\pi \hat{J}(\pi) = \nabla_\pi \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})} \underline{\mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})} \left[\hat{Q}^\pi(\mathbf{s},\mathbf{a})\right]}$$



✗ Reparameterization Trick

✓ Directly compute expectation

Discrete Actions

$$\nabla_\pi \hat{J}(\pi) = \nabla_\pi \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})} \mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})} \left[\hat{Q}^\pi(\mathbf{s}, \mathbf{a}) \right]$$

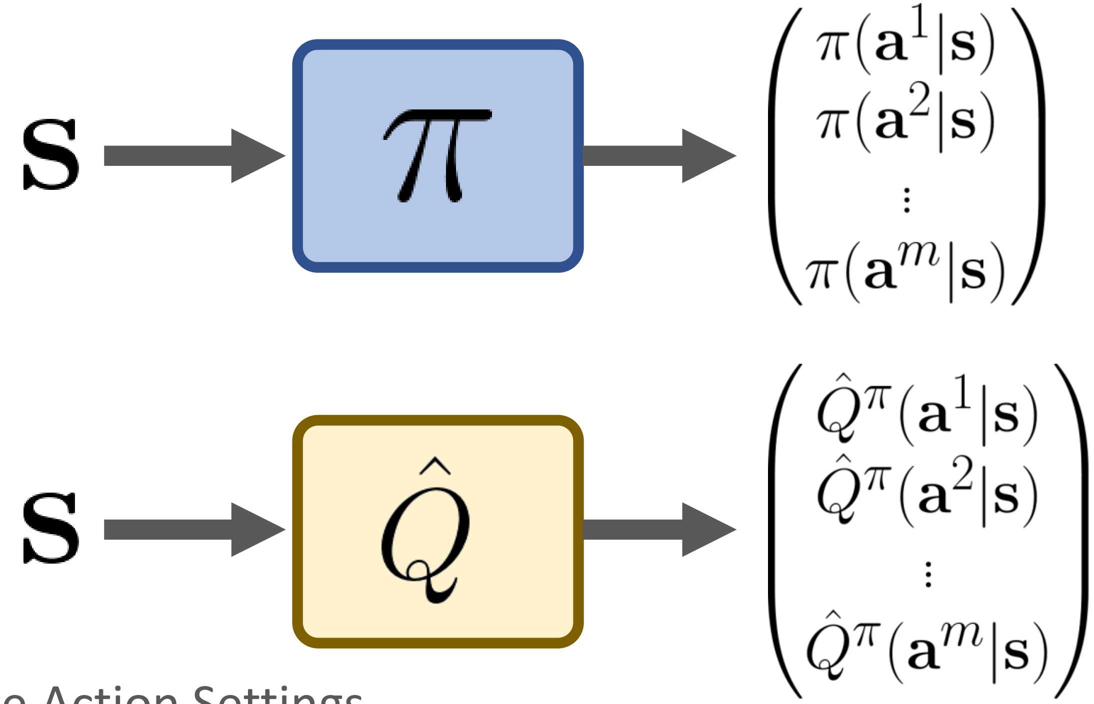
$$\Downarrow$$

$$\nabla_\pi \hat{J}(\pi) = \nabla_\pi \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})} \left[\sum_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s}) \hat{Q}^\pi(\mathbf{s}, \mathbf{a}) \right]$$

Soft Actor-Critic for Discrete Action Settings
[Petros et al. 2019]

Discrete Actions

$$\nabla_\pi \hat{J}(\pi) = \nabla_\pi \mathbb{E}_{\mathbf{s}\sim d_\pi(\mathbf{s})}\left[\sum_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s})\hat{Q}^\pi(\mathbf{s},\mathbf{a})\right]$$



Soft Actor-Critic for Discrete Action Settings
[Petros et al. 2019]

Discrete Actions

$$\nabla_\pi \hat{J}(\pi) = \nabla_\pi \mathbb{E}_{\mathbf{s} \sim d_\pi(\mathbf{s})} \left[\sum_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s}) \hat{Q}^\pi(\mathbf{s}, \mathbf{a}) \right]$$

dot product

$$\nabla_\pi \left[\begin{pmatrix} \log\pi(\mathbf{a}^1|\mathbf{s}) \\ \log\pi(\mathbf{a}^2|\mathbf{s}) \\ \vdots \\ \log\pi(\mathbf{a}^m|\mathbf{s}) \end{pmatrix} \bullet \begin{pmatrix} \hat{Q}^\pi(\mathbf{a}^1|\mathbf{s}) \\ \hat{Q}^\pi(\mathbf{a}^2|\mathbf{s}) \\ \vdots \\ \hat{Q}^\pi(\mathbf{a}^m|\mathbf{s}) \end{pmatrix} \right]$$

Soft Actor-Critic for Discrete Action Settings
[Petros et al. 2019]

Surrogate Objective

Surrogate Objective

$$\nabla_\pi \hat{J}(\pi) = \mathbb{E}_{\underline{\mathbf{s} \sim d_\pi(\mathbf{s})}} \mathbb{E}_{\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})} \left[\nabla_\pi \log \pi(\mathbf{a}|\mathbf{s}) \hat{Q}^\pi(\mathbf{s}, \mathbf{a}) \right]$$

Surrogate Objective

$$\hat{Q}^{\pi}(\mathbf{s}, \mathbf{a}) \approx Q^{\pi}(\mathbf{s}, \mathbf{a})$$

Original objective:

$$J(\pi) = \mathbb{E}_{\mathbf{s} \sim d_{\pi}(\mathbf{s})} \mathbb{E}_{\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})} \left[Q^{\pi}(\mathbf{s}, \mathbf{a}) \right]$$

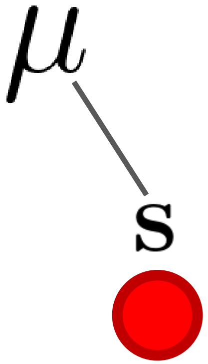
Surrogate objective:

$$\hat{J}(\pi) = \mathbb{E}_{\mathbf{s} \sim d_{\mu}(\mathbf{s})} \mathbb{E}_{\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})} \left[Q^{\pi}(\mathbf{s}, \mathbf{a}) \right]$$

$\mu(\mathbf{a}|\mathbf{s})$: behavior policy

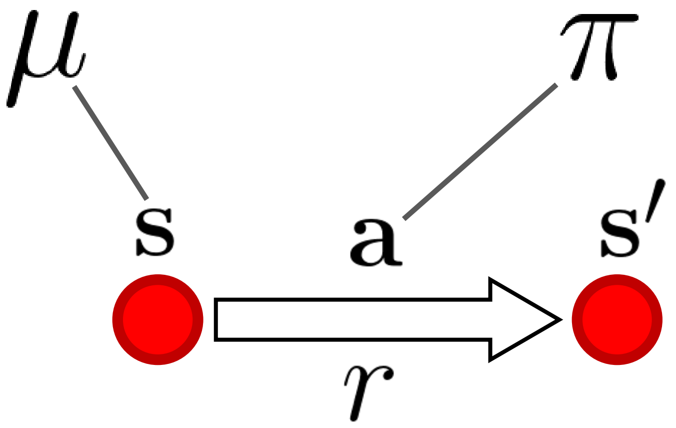
Surrogate Objective

$$\hat{J}(\pi) = \mathbb{E}_{\mathbf{s} \sim d_\mu(\mathbf{s})} \mathbb{E}_{\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})} \left[Q^\pi(\mathbf{s}, \mathbf{a}) \right]$$



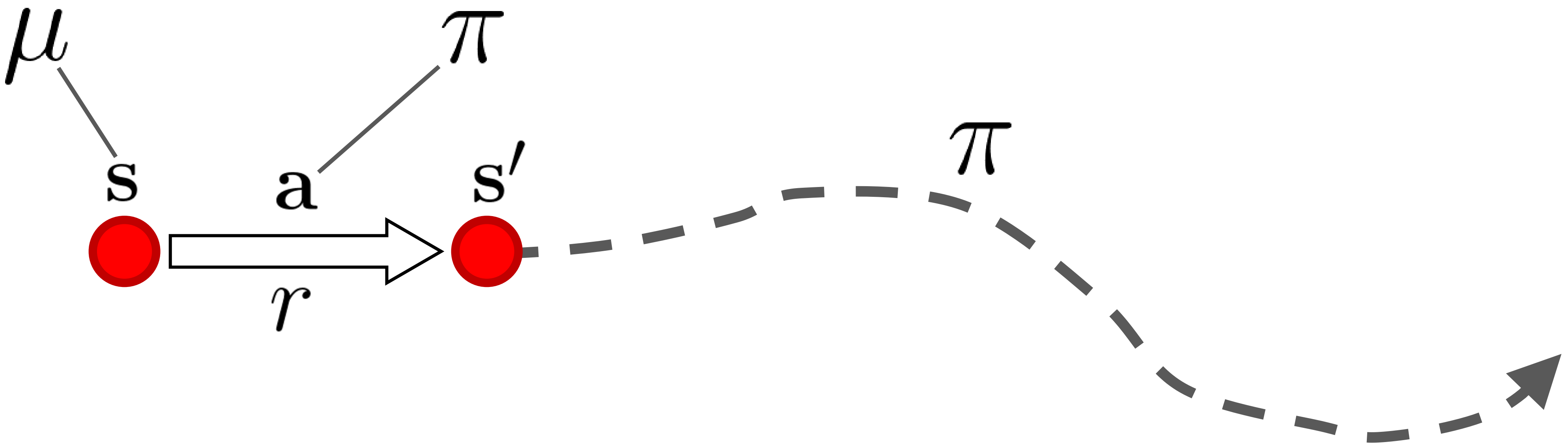
Surrogate Objective

$$\hat{J}(\pi) = \mathbb{E}_{\mathbf{s} \sim d_\mu(\mathbf{s})} \mathbb{E}_{\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})} \left[Q^\pi(\mathbf{s}, \mathbf{a}) \right]$$



Surrogate Objective

$$\hat{J}(\pi) = \mathbb{E}_{\mathbf{s}\sim d_\mu(\mathbf{s})}\mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}\left[Q^\pi(\mathbf{s},\mathbf{a})\right]$$



π is trying to maximize return starting in states visited by μ

Surrogate Objective

$$\hat{J}(\pi) = \mathbb{E}_{\mathbf{s}\sim d_\mu(\mathbf{s})}\mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}\left[Q^\pi(\mathbf{s},\mathbf{a})\right]$$

$$\mu \approx \pi$$

$$\Downarrow$$

$$d_\mu(\mathbf{s}) \approx d_\pi(\mathbf{s})$$

$$\Downarrow$$

$$\hat{J}(\pi) \approx J(\pi)$$

Soft Actor-Critic (SAC)

ALGORITHM: SAC

$Q^0 \leftarrow$ initialize Q-function
$\pi^0 \leftarrow$ initialize policy
$\mathcal{D} \leftarrow \{\emptyset\}$ initialize dataset

for iteration $k = 0, ..., n-1$ **do**
  Sample trajectory τ according to $\pi^k(\mathbf{a}|\mathbf{s})$
  Add transitions to dataset $\mathcal{D} = \mathcal{D} \cup \{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i)\}$

  Calculate target values for each sample i:
  $y_i = r_i + \gamma \mathbb{E}_{\mathbf{a}' \sim \pi^k(\mathbf{a}'|\mathbf{s}'_i)} \left[Q^k(\mathbf{s}'_i, \mathbf{a}') \right]$

  Update Q-function:
  $Q^{k+1} = \arg\min_Q \; \mathbb{E}_{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{r}_i, \mathbf{s}'_i) \sim \mathcal{D}} \left[(y_i - Q(\mathbf{s}_i, \mathbf{a}_i))^2 \right]$

  Update policy:
  $\pi^{k+1} = \arg\max_\pi \; \mathbb{E}_{\mathbf{s}_i \sim \mathcal{D}} \mathbb{E}_{\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s}_i)} \left[Q^{k+1}(\mathbf{s}_i, \mathbf{a}) \right]$
end for

return π^n

Only take a few grad steps

Soft Actor-Critic (SAC)

ALGORITHM: SAC

$Q^0 \leftarrow$ initialize Q-function
$\pi^0 \leftarrow$ initialize policy
$\mathcal{D} \leftarrow \{\emptyset\}$ initialize dataset

for iteration $k = 0, ..., n-1$ **do**
    Sample trajectory τ according to $\pi^k(\mathbf{a}|\mathbf{s})$
    Add transitions to dataset $\mathcal{D} = \mathcal{D} \cup \{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i)\}$

    Calculate target values for each sample i:
    $y_i = r_i + \gamma \mathbb{E}_{\mathbf{a}' \sim \pi^k(\mathbf{a}'|\mathbf{s}'_i)} \left[Q^k(\mathbf{s}'_i, \mathbf{a}') \right]$

    Update Q-function:
    $Q^{k+1} = \arg\min_Q \ \mathbb{E}_{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{r}_i, \mathbf{s}'_i) \sim \mathcal{D}} \left[(y_i - Q(\mathbf{s}_i, \mathbf{a}_i))^2 \right]$

    Update policy:
    $\pi^{k+1} = \arg\max_\pi \ \mathbb{E}_{\mathbf{s}_i \sim \mathcal{D}} \mathbb{E}_{\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s}_i)} \left[Q^{k+1}(\mathbf{s}_i, \mathbf{a}) \right]$
end for

return π^n

Behavior Policy

- Behavior policy doesn't have to correspond to just a single policy

$$\mu^k = \{\pi^0, \pi^1, ..., \pi^k\}$$

- Keep data from all previous iterations
- Train policy using data collected from all previous policies
- Much more sample efficient

Summary

- Actor-Critic Algorithms
- Deterministic Policy Gradient
- Soft Actor-Critic
- Surrogate Objective