Deep Reinforcement Learning via Policy Optimization

John Schulman

OpenAI

July 3, 2017
Introduction
Deep Reinforcement Learning: What to Learn?

- Policies (select next action)
Deep Reinforcement Learning: What to Learn?

- Policies (select next action)
- Value functions (measure goodness of states or state-action pairs)
Deep Reinforcement Learning: What to Learn?

- Policies (select next action)
- Value functions (measure goodness of states or state-action pairs)
- Models (predict next states and rewards)
Model Free RL: (Rough) Taxonomy

- Policy Optimization
  - DFO / Evolution
  - Policy Gradients
- Dynamic Programming
  - Policy Iteration
  - Value Iteration
  - Q-Learning
  - modified policy iteration
- Actor-Critic Methods
Policy Optimization vs Dynamic Programming

- Conceptually...

Policy optimization: optimize what you care about

Dynamic programming: indirect, exploit the problem structure, self-consistency

Empirically...

Policy optimization more versatile, dynamic programming methods more sample-efficient when they work

Policy optimization methods more compatible with rich architectures (including recurrence) which add tasks other than control (auxiliary objectives), dynamic programming methods more compatible with exploration and off-policy learning
Policy Optimization vs Dynamic Programming

- Conceptually . .
  - Policy optimization: optimize what you care about
Policy Optimization vs Dynamic Programming

- Conceptually...
  - Policy optimization: optimize what you care about
  - Dynamic programming: indirect, exploit the problem structure, self-consistency
Policy Optimization vs Dynamic Programming

- **Conceptually . . .**
  - Policy optimization: optimize what you care about
  - Dynamic programming: indirect, exploit the problem structure, self-consistency

- **Empirically . . .**
Policy Optimization vs Dynamic Programming

- Conceptually . . .
  - Policy optimization: optimize what you care about
  - Dynamic programming: indirect, exploit the problem structure, self-consistency

- Empirically . . .
  - Policy optimization more versatile, dynamic programming methods more sample-efficient when they work
Policy Optimization vs Dynamic Programming

- Conceptually . . .
  - Policy optimization: optimize what you care about
  - Dynamic programming: indirect, exploit the problem structure, self-consistency

- Empirically . . .
  - Policy optimization more versatile, dynamic programming methods more sample-efficient when they work
  - Policy optimization methods more compatible with rich architectures (including recurrence) which add tasks other than control (auxiliary objectives), dynamic programming methods more compatible with exploration and off-policy learning
Parameterized Policies

- A family of policies indexed by parameter vector $\theta \in \mathbb{R}^d$
Parameterized Policies

- A family of policies indexed by parameter vector $\theta \in \mathbb{R}^d$
  - Deterministic: $a = \pi(s, \theta)$
Parameterized Policies

- A family of policies indexed by parameter vector $\theta \in \mathbb{R}^d$
  - Deterministic: $a = \pi(s, \theta)$
  - Stochastic: $\pi(a \mid s, \theta)$
Parameterized Policies

- A family of policies indexed by parameter vector $\theta \in \mathbb{R}^d$
  - Deterministic: $a = \pi(s, \theta)$
  - Stochastic: $\pi(a | s, \theta)$
- Analogous to classification or regression with input $s$, output $a$. 
Parameterized Policies

- A family of policies indexed by parameter vector $\theta \in \mathbb{R}^d$
  - Deterministic: $a = \pi(s, \theta)$
  - Stochastic: $\pi(a | s, \theta)$
- Analogous to classification or regression with input $s$, output $a$.
  - Discrete action space: network outputs vector of probabilities
  - Continuous action space: network outputs mean and diagonal covariance of Gaussian
Parameterized Policies

- A family of policies indexed by parameter vector \( \theta \in \mathbb{R}^d \)
  - Deterministic: \( a = \pi(s, \theta) \)
  - Stochastic: \( \pi(a | s, \theta) \)
- Analogous to classification or regression with input \( s \), output \( a \).
  - Discrete action space: network outputs vector of probabilities
  - Continuous action space: network outputs mean and diagonal covariance of Gaussian
In each episode, the initial state is sampled from $\mu$, and the agent acts until the terminal state is reached. For example:

- Taxi robot reaches its destination (termination = good)
- Waiter robot finishes a shift (fixed time)
- Walking robot falls over (termination = bad)

Goal: maximize expected return per episode

$$\maximize_{\pi} \mathbb{E}[R | \pi]$$
In each episode, the initial state is sampled from \( \mu \), and the agent acts until the terminal state is reached. For example:

- Taxi robot reaches its destination (termination = good)
- Waiter robot finishes a shift (fixed time)
- Walking robot falls over (termination = bad)
Episodic Setting

- In each episode, the initial state is sampled from $\mu$, and the agent acts until the *terminal state* is reached. For example:
  - Taxi robot reaches its destination (termination = good)
  - Waiter robot finishes a shift (fixed time)
Episodic Setting

- In each episode, the initial state is sampled from $\mu$, and the agent acts until the terminal state is reached. For example:
  - Taxi robot reaches its destination (termination = good)
  - Waiter robot finishes a shift (fixed time)
  - Walking robot falls over (termination = bad)
In each episode, the initial state is sampled from $\mu$, and the agent acts until the *terminal state* is reached. For example:

- Taxi robot reaches its destination (termination $= \text{good}$)
- Waiter robot finishes a shift (fixed time)
- Walking robot falls over (termination $= \text{bad}$)

Goal: maximize expected return per episode

$$\maximize_{\pi} \mathbb{E} \left[ R \mid \pi \right]$$
Derivative Free Optimization / Evolution
Cross Entropy Method

Initialize $\mu \in \mathbb{R}^d, \sigma \in \mathbb{R}^d$

for iteration $= 1, 2, \ldots$ do

Collect $n$ samples of $\theta_i \sim N(\mu, \text{diag}(\sigma))$
Perform one episode with each $\theta_i$, obtaining reward $R_i$
Select the top $p\%$ of $\theta$ samples (e.g. $p = 20$), the elite set
Fit a Gaussian distribution, to the elite set, updating $\mu, \sigma$.

end for

Return the final $\mu$. 
Cross Entropy Method

- Sometimes works embarrassingly well
Cross Entropy Method

- Sometimes works embarrassingly well
Sometimes works embarrassingly well

<table>
<thead>
<tr>
<th>Method</th>
<th>Mean Score</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nonreinforcement learning</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hand-coded</td>
<td>631,167</td>
<td>Dellacherie (Fahey, 2003)</td>
</tr>
<tr>
<td>Genetic algorithm</td>
<td>586,103</td>
<td>(Böhm et al., 2004)</td>
</tr>
<tr>
<td>Reinforcement learning</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relational reinforcement</td>
<td>≈50</td>
<td>Ramon and Driessens (2004)</td>
</tr>
<tr>
<td>learning-kernel-based regression</td>
<td>3183</td>
<td>Bertsekas and Tsitsiklis (1996)</td>
</tr>
<tr>
<td>Policy iteration</td>
<td>&lt;3000</td>
<td>Lagoudakis, Parr, and Littman</td>
</tr>
<tr>
<td>Least squares policy iteration</td>
<td></td>
<td>(2002)</td>
</tr>
<tr>
<td>Linear programming + Bootstrap</td>
<td>4274</td>
<td>Farias and van Roy (2006)</td>
</tr>
<tr>
<td>Natural policy gradient</td>
<td>≈6800</td>
<td>Kakade (2001)</td>
</tr>
<tr>
<td>CE+RL</td>
<td>21,282</td>
<td></td>
</tr>
<tr>
<td>CE+RL, constant noise</td>
<td>72,705</td>
<td></td>
</tr>
<tr>
<td>CE+RL, decreasing noise</td>
<td>348,895</td>
<td></td>
</tr>
</tbody>
</table>

- Sometimes works embarrassingly well

<table>
<thead>
<tr>
<th>Method</th>
<th>Mean Score</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nonreinforcement learning</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hand-coded</td>
<td>631,167</td>
<td>Dellacherie (Fabey, 2003)</td>
</tr>
<tr>
<td>Genetic algorithm</td>
<td>586,103</td>
<td>Böhm et al., 2004</td>
</tr>
<tr>
<td>Reinforcement learning</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relational reinforcement</td>
<td>≈50</td>
<td>Ramon and Driessens (2004)</td>
</tr>
<tr>
<td>learning+kernel-based regression</td>
<td>3183</td>
<td>Bertsekas and Tsitsiklis (1996)</td>
</tr>
<tr>
<td>Least squares policy iteration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linear programming + Bootstrap</td>
<td>4274</td>
<td>Farias and van Roy (2006)</td>
</tr>
<tr>
<td>Natural policy gradient</td>
<td>≈6800</td>
<td>Kakade (2001)</td>
</tr>
<tr>
<td>CE+RL</td>
<td>21,282</td>
<td></td>
</tr>
<tr>
<td>CE+RL, constant noise</td>
<td>72,705</td>
<td></td>
</tr>
<tr>
<td>CE+RL, decreasing noise</td>
<td>348,895</td>
<td></td>
</tr>
</tbody>
</table>

Let $\mu$ define distribution for policy $\pi_\theta$: $\theta \sim P_\mu(\theta)$
Stochastic Gradient Ascent on Distribution

- Let $\mu$ define distribution for policy $\pi_\theta$: $\theta \sim P_\mu(\theta)$
- Return $R$ depends on policy parameter $\theta$ and noise $\zeta$

$$\text{maximize } \mathbb{E}_{\theta,\zeta} [R(\theta, \zeta)]$$

$R$ is unknown and possibly nondifferentiable
Stochastic Gradient Ascent on Distribution

- Let $\mu$ define distribution for policy $\pi_\theta: \theta \sim P_\mu(\theta)$
- Return $R$ depends on policy parameter $\theta$ and noise $\zeta$

$$\maximize_\mu \mathbb{E}_{\theta,\zeta} [R(\theta, \zeta)]$$

$R$ is unknown and possibly nondifferentiable
- “Score function” gradient estimator:

$$\nabla_\mu \mathbb{E}_{\theta,\zeta} [R(\theta, \zeta)] = \mathbb{E}_{\theta,\zeta} [\nabla_\mu \log P_\mu(\theta)R(\theta, \zeta)]$$

$$\approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\mu \log P_\mu(\theta_i)R_i$$
Stochastic Gradient Ascent on Distribution

- Compare with cross-entropy method
Stochastic Gradient Ascent on Distribution

- Compare with cross-entropy method
  - Score function grad:

  \[
  \nabla_{\mu} \mathbb{E}_{\theta, \zeta} [R(\theta, \zeta)] \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_{\mu} \log P_{\mu}(\theta_i) R_i
  \]

- Cross entropy method:
  \[
  \maximize \mu \quad \frac{1}{N} \sum_{i=1}^{N} \log P_{\mu}(\theta_i) f(R_i)
  \]
  where \( f(r) = 1 \) if \( r \) above threshold
Stochastic Gradient Ascent on Distribution

- Compare with cross-entropy method
  - Score function $\text{grad}$:
    \[
    \nabla_\mu \mathbb{E}_{\theta, \zeta} [R(\theta, \zeta)] \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\mu \log P_\mu(\theta_i) R_i
    \]

- Cross entropy method:
  
  \[
  \text{maximize} \quad \frac{1}{N} \sum_{i=1}^{N} \log P_\mu(\theta_i) f(R_i) \quad \text{(cross entropy method)}
  \]

  where $f(r) = \mathbb{1}[r \text{ above threshold}]$
Connection to Finite Differences

- Suppose $P_\mu$ is Gaussian distribution with mean $\mu$, covariance $\sigma^2 I$

\[
\log P_\mu(\theta) = -\|\mu - \theta\|^2 / 2\sigma^2 + \text{const}
\]

\[
\nabla_\mu \log P_\mu(\theta) = (\theta - \mu) / \sigma^2
\]

\[
R_i \nabla_\mu \log P_\mu(\theta_i) = R_i (\theta_i - \mu) / \sigma^2
\]

- Suppose we do antithetic sampling, where we use pairs of samples $\theta_+ = \mu + \sigma z$, $\theta_- = \mu - \sigma z$

\[
\nabla_\mu \log P_\mu(\theta_+) = R(\mu + \sigma z, \zeta) - R(\mu - \sigma z, \zeta')
\]

Using same noise $\zeta$ for both evaluations reduces variance
Connection to Finite Differences

- Suppose $P_\mu$ is Gaussian distribution with mean $\mu$, covariance $\sigma^2 I$

\[
\log P_\mu(\theta) = -\|\mu - \theta\|^2 / 2\sigma^2 + \text{const}
\]
\[
\nabla_\mu \log P_\mu(\theta) = (\theta - \mu) / \sigma^2
\]
\[
R_i \nabla_\mu \log P_\mu(\theta_i) = R_i (\theta_i - \mu) / \sigma^2
\]

- Suppose we do antithetic sampling, where we use pairs of samples $\theta_+ = \mu + \sigma z$, $\theta_- = \mu - \sigma z$

\[
\frac{1}{2} \left( R(\mu + \sigma z, \zeta) \nabla_\mu \log P_\mu(\theta_+) + R(\mu - \sigma z, \zeta') \nabla_\mu \log P_\mu(\theta_-) \right)
\]
\[
= \frac{1}{\sigma} (R(\mu + \sigma z, \zeta) - R(\mu - \sigma z, \zeta')) z
\]
Suppose $P_\mu$ is Gaussian distribution with mean $\mu$, covariance $\sigma^2 I$

$$\log P_\mu(\theta) = -\|\mu - \theta\|^2 / 2\sigma^2 + \text{const}$$

$$\nabla_\mu \log P_\mu(\theta) = (\theta - \mu) / \sigma^2$$

$$R_i \nabla_\mu \log P_\mu(\theta_i) = R_i (\theta_i - \mu) / \sigma^2$$

Suppose we do antithetic sampling, where we use pairs of samples $\theta_+ = \mu + \sigma z$, $\theta_- = \mu - \sigma z$

$$\frac{1}{2} \left( R(\mu + \sigma z, \zeta) \nabla_\mu \log P_\mu(\theta_+) + R(\mu - \sigma z, \zeta') \nabla_\mu \log P_\mu(\theta_-) \right)$$

$$= \frac{1}{\sigma} \left( R(\mu + \sigma z, \zeta) - R(\mu - \sigma z, \zeta') \right) z$$

Using same noise $\zeta$ for both evaluations reduces variance
Deriving the Score Function Estimator

- “Score function” gradient estimator:

\[ \nabla_\mu \mathbb{E}_{\theta, \zeta} [R(\theta, \zeta)] = \mathbb{E}_{\theta, \zeta} [\nabla_\mu \log P_\mu(\theta) R(\theta, \zeta)] \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\mu \log P_\mu(\theta_i) R_i \]
Deriving the Score Function Estimator

- “Score function” gradient estimator:

\[ \nabla_\mu \mathbb{E}_{\theta, \zeta} [R(\theta, \zeta)] = \mathbb{E}_{\theta, \zeta} [\nabla_\mu \log P_\mu(\theta) R(\theta, \zeta)] \]

\[ \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\mu \log P_\mu(\theta_i) R_i \]

- Derive by writing expectation as an integral

\[ \nabla_\mu \int d\mu d\zeta P_\mu(\theta) R(\theta, \zeta) \]

\[ = \int d\mu d\zeta \nabla_\mu P_\mu(\theta) R(\theta, \zeta) \]

\[ = \int d\mu d\zeta P_\mu(\theta) \nabla_\mu \log P_\mu(\theta) R(\theta, \zeta) \]

\[ = \mathbb{E}_{\theta, \zeta} [\nabla_\mu \log P_\mu(\theta) R(\theta, \zeta)] \]
Literature on DFO

- Evolution strategies (Rechenberg and Eigen, 1973)
Literature on DFO

- Evolution strategies (Rechenberg and Eigen, 1973)
- Simultaneous perturbation stochastic approximation (Spall, 1992)
Literature on DFO

- Evolution strategies (Rechenberg and Eigen, 1973)
- Simultaneous perturbation stochastic approximation (Spall, 1992)
- Covariance matrix adaptation: popular relative of CEM (Hansen, 2006)
Literature on DFO

- Evolution strategies (Rechenberg and Eigen, 1973)
- Simultaneous perturbation stochastic approximation (Spall, 1992)
- Covariance matrix adaptation: popular relative of CEM (Hansen, 2006)
- Reward weighted regression (Peters and Schaal, 2007), PoWER (Kober and Peters, 2007)
Success Stories

- CMA is very effective for optimizing low-dimensional locomotion controllers
Success Stories

- CMA is very effective for optimizing low-dimensional locomotion controllers
  - UT Austin Villa: RoboCup 2012 3D Simulation League Champion
Success Stories

- CMA is very effective for optimizing low-dimensional locomotion controllers
  - UT Austin Villa: RoboCup 2012 3D Simulation League Champion

- Evolution Strategies was shown to perform well on Atari, competitive with policy gradient methods (Salimans et al., 2017)
Policy Gradient Methods
Problem:

maximize $E[R \mid \pi_\theta]$

- Here, we’ll use a fixed policy parameter $\theta$ (instead of sampling $\theta \sim P_\mu$) and estimate gradient with respect to $\theta$
Problem:

\[ \text{maximize } E[R \mid \pi_\theta] \]

- Here, we’ll use a fixed policy parameter \( \theta \) (instead of sampling \( \theta \sim P_\mu \)) and estimate gradient with respect to \( \theta \)
- Noise is in action space rather than parameter space
Overview

Problem:

\[
\text{maximize } E[R \mid \pi_\theta]
\]

Intuitions: collect a bunch of trajectories, and ...

1. Make the good trajectories more probable
Overview

Problem:

\[
\text{maximize } E[R \mid \pi_{\theta}]
\]

Intuitions: collect a bunch of trajectories, and ...

1. Make the good trajectories more probable
2. Make the good actions more probable
Overview

Problem:

\[
\text{maximize } E[R \mid \pi_\theta]
\]

Intuitions: collect a bunch of trajectories, and ...

1. Make the good trajectories more probable
2. Make the good actions more probable
3. Push the actions towards better actions
Now random variable is a whole trajectory
\( \tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_{T-1}, a_{T-1}, r_{T-1}, s_T) \)

\[
\nabla_\theta E_\tau [R(\tau)] = E_\tau [\nabla_\theta \log P(\tau | \theta)R(\tau)]
\]
Score Function Gradient Estimator for Policies

- Now random variable is a whole trajectory
  \[ \tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_{T-1}, a_{T-1}, r_{T-1}, s_T) \]
  \[ \nabla_\theta E_\tau[R(\tau)] = E_\tau[\nabla_\theta \log P(\tau | \theta) R(\tau)] \]

- Just need to write out \( P(\tau | \theta) \):
  \[
P(\tau | \theta) = \mu(s_0) \prod_{t=0}^{T-1} [\pi(a_t | s_t, \theta) P(s_{t+1}, r_t | s_t, a_t)]
  \]
  \[
  \log P(\tau | \theta) = \log \mu(s_0) + \sum_{t=0}^{T-1} [\log \pi(a_t | s_t, \theta) + \log P(s_{t+1}, r_t | s_t, a_t)]
  \]
  \[
  \nabla_\theta \log P(\tau | \theta) = \nabla_\theta \sum_{t=0}^{T-1} \log \pi(a_t | s_t, \theta)
  \]
  \[
  \nabla_\theta \mathbb{E}_\tau[R] = \mathbb{E}_\tau \left[ R \nabla_\theta \sum_{t=0}^{T-1} \log \pi(a_t | s_t, \theta) \right]
  \]
Policy Gradient: Use Temporal Structure

Previous slide:

\[ \nabla_\theta \mathbb{E}_\tau [R] = \mathbb{E}_\tau \left[ \left( \sum_{t=0}^{T-1} r_t \right) \left( \sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t | s_t, \theta) \right) \right] \]
Policy Gradient: Use Temporal Structure

- Previous slide:

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

- We can repeat the same argument to derive the gradient estimator for a single reward term \( r_{t'} \).

\[ \nabla_\theta \mathbb{E}[r_{t'}] = \mathbb{E} \left[ r_{t'} \sum_{t=0}^{t'} \nabla_\theta \log \pi(a_t | s_t, \theta) \right] \]
Policy Gradient: Use Temporal Structure

- Previous slide:

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

- We can repeat the same argument to derive the gradient estimator for a single reward term \( r_t' \).

\[ \nabla_\theta \mathbb{E}[r_t'] = \mathbb{E} \left[ r_t' \sum_{t'=0}^{t'} \nabla_\theta \log \pi(a_t | s_t, \theta) \right] \]

- Sum this formula over \( t \), we obtain

\[ \nabla_\theta \mathbb{E}[R] = \mathbb{E} \left[ \sum_{t'=0}^{T-1} \sum_{t=0}^{t'} \nabla_\theta \log \pi(a_t | s_t, \theta) \right] \]

\[ = \mathbb{E} \left[ \sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t | s_t, \theta) \sum_{t'=t}^{T-1} r_t' \right] \]
Further reduce variance by introducing a baseline $b(s)$

$$
\nabla_\theta \mathbb{E}_\tau [R] = \mathbb{E}_\tau \left[ \sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t \mid s_t, \theta) \left( \sum_{t'=t}^{T-1} r_{t'} - b(s_t) \right) \right]
$$
Policy Gradient: Introduce Baseline

- Further reduce variance by introducing a baseline $b(s)$

$$
\nabla_\theta \mathbb{E}_T[R] = \mathbb{E}_T \left[ \sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t | s_t, \theta) \left( \sum_{t'=t}^{T-1} r_{t'} - b(s_t) \right) \right]
$$

- For any choice of $b$, gradient estimator is unbiased.
Policy Gradient: Introduce Baseline

- Further reduce variance by introducing a baseline $b(s)$

$$\nabla_{\theta} \mathbb{E}_T [R] = \mathbb{E}_T \left[ \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi(a_t \mid s_t, \theta) \left( \sum_{t'=t}^{T-1} r_{t'} - b(s_t) \right) \right]$$

- For any choice of $b$, gradient estimator is unbiased.
- Near optimal choice is expected return,
  $$b(s_t) \approx \mathbb{E} [r_t + r_{t+1} + r_{t+2} + \cdots + r_{T-1}]$$
Policy Gradient: Introduce Baseline

- Further reduce variance by introducing a baseline $b(s)$

$$
\nabla_\theta \mathbb{E}_\tau [R] = \mathbb{E}_\tau \left[ \sum_{t=0}^{T-1} \nabla_\theta \log \pi (a_t \mid s_t, \theta) \left( \sum_{t'=t}^{T-1} r_{t'} - b(s_t) \right) \right] 
$$

- For any choice of $b$, gradient estimator is unbiased.
- Near optimal choice is expected return,
  \[
  b(s_t) \approx \mathbb{E} [r_t + r_{t+1} + r_{t+2} + \cdots + r_{T-1}] 
  \]
- Interpretation: increase logprob of action $a_t$ proportionally to how much returns $\sum_{t=t'}^{T-1} r_{t'}$ are better than expected
Discounts for Variance Reduction

- Introduce discount factor $\gamma$, which ignores delayed effects between actions and rewards

$$\nabla_\theta \mathbb{E}_\tau [R] \approx \mathbb{E}_\tau \left[ \sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t | s_t, \theta) \left( \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'} - b(s_t) \right) \right]$$
 Discounts for Variance Reduction

- Introduce discount factor $\gamma$, which ignores delayed effects between actions and rewards

$$ \nabla_\theta \mathbb{E}_T [R] \approx \mathbb{E}_T \left[ \sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t | s_t, \theta) \left( \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'} - b(s_t) \right) \right] $$

- Now, we want $b(s_t) \approx \mathbb{E} \left[ r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-1-t} r_{T-1} \right]$
Discounts for Variance Reduction

- Introduce discount factor $\gamma$, which ignores delayed effects between actions and rewards

$$
\nabla_\theta \mathbb{E}_T [R] \approx \mathbb{E}_T \left[ \sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t | s_t, \theta) \left( \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'} - b(s_t) \right) \right]
$$

- Now, we want $b(s_t) \approx \mathbb{E} \left[ r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-1-t} r_{T-1} \right]$  

- Write gradient estimator more generally as

$$
\nabla_\theta \mathbb{E}_T [R] \approx \mathbb{E}_T \left[ \sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t | s_t, \theta) \hat{A}_t \right]
$$

$\hat{A}_t$ is the advantage estimate
“Vanilla” Policy Gradient Algorithm

Initialize policy parameter $\theta$, baseline $b$

for iteration = 1, 2, ... do

Collect a set of trajectories by executing the current policy

At each timestep in each trajectory, compute

the return $\hat{R}_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$, and

the advantage estimate $\hat{A}_t = \hat{R}_t - b(s_t)$.

Re-fit the baseline, by minimizing $\|b(s_t) - R_t\|^2$, summed over all trajectories and timesteps.

Update the policy, using a policy gradient estimate $\hat{g}$, which is a sum of terms $\nabla_\theta \log \pi(a_t | s_t, \theta) \hat{A}_t$

end for
Advantage Actor-Critic

- Use neural network that represents policy $\pi_\theta$ and value function $V_\theta$ (approximating $V^{\pi_\theta}$)

- Pseudocode

```
for iteration=1, 2, ... do
    Agent acts for $T$ timesteps (e.g., $T = 20$),
    For each timestep $t$, compute
    $$\hat{R}_t = r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V_\theta(s_t)$$
    $$\hat{A}_t = \hat{R}_t - V_\theta(s_t)$$
    $\hat{R}_t$ is target value function, in regression problem
    $\hat{A}_t$ is estimated advantage function
    Compute loss gradient $g = \nabla_\theta \sum_{t=1}^{T} \left[ -\log \pi_\theta(a_t | s_t) \hat{A}_t + c(V_\theta(s) - \hat{R}_t)^2 \right]$ 
    $g$ is plugged into a stochastic gradient ascent algorithm, e.g., Adam.
```

---


Trust Region Policy Optimization

- Motivation: make policy gradients more robust and sample efficient

\[ L_{\pi_{\text{old}}} (\pi) = \frac{1}{N} \sum_{i=1}^{N} \pi(a_i | s_i) \pi_{\text{old}}(a_i | s_i) \hat{A}_i(1) \]

Differentiating this objective gives the policy gradient

- \( L_{\pi_{\text{old}}} (\pi) \) is only accurate when state distribution of \( \pi \) is close to \( \pi_{\text{old}} \), thus it makes sense to constrain or penalize the distance \( D_{KL}[\pi || \pi] \)
Trust Region Policy Optimization

- **Motivation:** make policy gradients more robust and sample efficient
  - Unlike in supervised learning, policy affects distribution of inputs, so a large bad update can be disastrous

\[
L_{\pi_{\text{old}}} (\pi) = \frac{1}{N} \sum_{i=1}^{N} \pi (a_i | s_i) \pi_{\text{old}} (a_i | s_i) \hat{A}_i (1)
\]

Differentiating this objective gives the policy gradient

- \( L_{\pi_{\text{old}}} (\pi) \) is only accurate when state distribution of \( \pi \) is close to \( \pi_{\text{old}} \), thus it makes sense to constrain or penalize the distance \( D_{\text{KL}} [\pi || \pi_{\text{old}}] \).
Trust Region Policy Optimization

- Motivation: make policy gradients more robust and sample efficient
  - Unlike in supervised learning, policy affects distribution of inputs, so a large bad update can be disastrous
- Makes use of a “surrogate objective” that estimates the performance of the policy around $\pi_{old}$ used for sampling

$$L_{\pi_{old}}(\pi) = \frac{1}{N} \sum_{i=1}^{N} \frac{\pi(a_i \mid s_i)}{\pi_{old}(a_i \mid s_i)} \hat{A}_i$$  \hspace{1cm} (1)

Differentiating this objective gives the policy gradient
Trust Region Policy Optimization

- Motivation: make policy gradients more robust and sample efficient
  - Unlike in supervised learning, policy affects distribution of inputs, so a large bad update can be disastrous
- Makes use of a “surrogate objective” that estimates the performance of the policy around $\pi_{\text{old}}$ used for sampling

\[
L_{\pi_{\text{old}}} (\pi) = \frac{1}{N} \sum_{i=1}^{N} \frac{\pi(a_i \mid s_i)}{\pi_{\text{old}}(a_i \mid s_i)} \hat{A}_i
\]  

(1)

Differentiating this objective gives the policy gradient

- $L_{\pi_{\text{old}}} (\pi)$ is only accurate when state distribution of $\pi$ is close to $\pi_{\text{old}}$, thus it makes sense to constrain or penalize the distance $D_{KL} [\pi_{\text{old}} \parallel \pi]$
Trust Region Policy Optimization

- **Pseudocode:**
  ```
  for iteration=1, 2, . . . do
    Run policy for $T$ timesteps or $N$ trajectories
    Estimate advantage function at all timesteps
    \[
    \max_{\theta} \sum_{n=1}^{N} \frac{\pi_\theta(a_n | s_n)}{\pi_{\theta_{old}}(a_n | s_n)} \hat{A}_n
    \]
    subject to $KL_{\pi_{\theta_{old}}} (\pi_\theta) \leq \delta$
  end for
  ```

- Can solve constrained optimization problem efficiently by using conjugate gradient

- Closely related to natural policy gradients (Kakade, 2002), natural actor critic (Peters and Schaal, 2005), REPS (Peters et al., 2010)
“Proximal” Policy Optimization

▶ Use penalty instead of constraint

$$\max_{\theta} \sum_{n=1}^{N} \frac{\pi_{\theta}(a_n | s_n)}{\pi_{\theta_{old}}(a_n | s_n)} \hat{A}_n - C \cdot \text{KL}_{\pi_{\theta_{old}}} (\pi_{\theta})$$

▶ Pseudocode:

for iteration=1, 2, . . . do
    Run policy for $T$ timesteps or $N$ trajectories
    Estimate advantage function at all timesteps
    Do SGD on above objective for some number of epochs
    If KL too high, increase $\beta$. If KL too low, decrease $\beta$
end for

▶ $\approx$ same performance as TRPO, but only first-order optimization
Variance Reduction for Policy Gradients
Reward Shaping

Chain MDP

$A = \{\{\} \}$

$S = \{-m, -m+1, \ldots, n-1, n\}$, $|S| = m + n + 1$

$m$ and $n$ are terminal

$R(s,a,s') = \begin{cases} 1 & (s,a,s') = (n-1, \emptyset, n) \\ 0 & \text{otherwise} \end{cases}$

Initial state $s = 0$. 
Reward Shaping

- Reward shaping: $\delta(s, a, s') = r(s, a, s') + \gamma \Phi(s') - \Phi(s)$ for arbitrary “potential” $\Phi$

---

Reward Shaping

- Reward shaping: \( \delta(s, a, s') = r(s, a, s') + \gamma \Phi(s') - \Phi(s) \) for arbitrary “potential” \( \Phi \)
- Theorem: \( \delta \) admits the same optimal policies as \( r \).

---

**Reward Shaping**

- Reward shaping: \( \delta(s, a, s') = r(s, a, s') + \gamma \Phi(s') - \Phi(s) \) for arbitrary “potential” \( \Phi \)

- Theorem: \( \delta \) admits the same optimal policies as \( r \).
  - Proof sketch: suppose \( Q^* \) satisfies Bellman equation \( (TQ = Q) \). If we transform \( r \to \delta \), policy's value function satisfies \( \tilde{Q}(s, a) = Q^*(s, a) - \Phi(s) \)

---

Reward Shaping

- Reward shaping: \( \delta(s, a, s') = r(s, a, s') + \gamma\Phi(s') - \Phi(s) \) for arbitrary “potential” \( \Phi \)
- Theorem: \( \delta \) admits the same optimal policies as \( r \).\(^1\)
  - Proof sketch: suppose \( Q^* \) satisfies Bellman equation (\( TQ = Q \)). If we transform \( r \rightarrow \delta \), policy’s value function satisfies \( \tilde{Q}(s, a) = Q^*(s, a) - \Phi(s) \)
  - \( Q^* \) satisfies Bellman equation \( \Rightarrow \tilde{Q} \) also satisfies Bellman equation

Reward Shaping

- Theorem: \( \delta \) admits the same optimal policies as \( R \). A. Y. Ng, D. Harada, and S. Russell. “Policy invariance under reward transformations: Theory and application to reward shaping”. *ICML*. 1999
Reward Shaping

- Theorem: $\delta$ admits the same optimal policies as $R$. A. Y. Ng, D. Harada, and S. Russell. “Policy invariance under reward transformations: Theory and application to reward shaping”. ICML. 1999

- Alternative proof: advantage function is invariant. Let’s look at effect on $V^\pi$ and $Q^\pi$:

  $\mathbb{E} [\delta_0 + \gamma \delta_1 + \gamma^2 \delta_2 + \ldots]$

  $= \mathbb{E} [(r_0 + \gamma \Phi(s_1) - \Phi(s_0)) + \gamma (r_1 + \gamma \Phi(s_2) - \Phi(s_1)) + \gamma^2 (r_2 + \gamma \Phi(s_3) - \Phi(s_2)) + \ldots]$

  $= \mathbb{E} [r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots - \Phi(s_0)]$

  Thus,

  $\tilde{V}^\pi(s) = V^\pi(s) - \Phi(s)$

  $\tilde{Q}^\pi(s) = Q^\pi(s, a) - \Phi(s)$

  $\tilde{A}^\pi(s) = A^\pi(s, a)$

  $A^\pi(s, \pi(s)) = 0$ at all states $\Rightarrow \pi$ is optimal
Reward Shaping and Problem Difficulty

- Shape with $\Phi = V^* \Rightarrow$ problem is solved in one step of value iteration
Reward Shaping and Problem Difficulty

- Shape with $\Phi = V^* \Rightarrow$ problem is solved in one step of value iteration
- Shaping leaves policy gradient invariant (and just adds baseline to estimator)

\[
E[\nabla_\theta \log \pi_\theta(a_0 | s_0)(r_0 + \gamma \Phi(s_1) - \Phi(s_0)) + \gamma (r_1 + \gamma \Phi(s_2) - \Phi(s_1)) \\
+ \gamma^2 (r_2 + \gamma \Phi(s_3) - \Phi(s_2)) + \ldots]
\]

\[
= E[\nabla_\theta \log \pi_\theta(a_0 | s_0)(r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots - \Phi(s_0))]
\]

\[
= E[\nabla_\theta \log \pi_\theta(a_0 | s_0)(r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots)]
\]
First note the connection between shaped reward and advantage function:

$$\mathbb{E}_{s_1} \left[ r_0 + \gamma V^\pi(s_1) - V^\pi(s_0) \mid s_0 = s, a_0 = a \right] = A^\pi(s, a)$$

Now considering the policy gradient and ignoring all but first shaped reward (i.e., pretend $\gamma = 0$ after shaping) we get

$$\mathbb{E} \left[ \sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t) \delta_t \right] = \mathbb{E} \left[ \sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t) (r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)) \right]$$

$$= \mathbb{E} \left[ \sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t) A^\pi(s_t, a_t) \right]$$
Reward Shaping and Policy Gradients

- Compromise: use more aggressive discount $\gamma \lambda$, with $\lambda \in (0, 1)$: called generalized advantage estimation

$$\sum_t \nabla_\theta \log \pi_\theta(a_t | s_t) \sum_{k=0}^{\infty} (\gamma \lambda)^k \delta_{t+k}$$
Reward Shaping and Policy Gradients

- Compromise: use more aggressive discount $\gamma \lambda$, with $\lambda \in (0, 1)$: called generalized advantage estimation

$$\sum_t \nabla_\theta \log \pi_\theta(a_t | s_t) \sum_{k=0}^{\infty} (\gamma \lambda)^k \delta_{t+k}$$

- Or alternatively, use hard cutoff as in A3C

$$\sum_t \nabla_\theta \log \pi_\theta(a_t | s_t) \sum_{k=0}^{n-1} \gamma^k \delta_{t+k}$$

$$= \sum_t \nabla_\theta \log \pi_\theta(a_t | s_t) \left( \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n \Phi(s_{t+n}) - \Phi(s_t) \right)$$
Reward Shaping—Summary

- Reward shaping transformation leaves policy gradient and optimal policy invariant
Reward Shaping—Summary

- Reward shaping transformation leaves policy gradient and optimal policy invariant.
- Shaping with $\Phi \approx V^\pi$ makes consequences of actions more immediate.
Reward Shaping—Summary

- Reward shaping transformation leaves policy gradient and optimal policy invariant
- Shaping with $\Phi \approx V^\pi$ makes consequences of actions more immediate
- Shaping, and then ignoring all but first term, gives policy gradient
Aside: Reward Shaping is Crucial in Practice


\[ L(s) = L_{CI}(s) + L_{Physics}(s) + L_{Task}(s) + L_{Hint}(s) \]
Aside: Reward Shaping is Crucial in Practice


\[
L(s) = L_{CI}(s) + L_{Physics}(s) + L_{Task}(s) + L_{Hint}(s)
\]


The state-cost is composed of 4 terms. The first term penalizes the horizontal distance (in the \(xy\)-plane) between the center-of-mass (CoM) and the mean of the feet positions. The second term penalizes the horizontal distance between the torso and the CoM. The third penalizes the vertical distance between the torso and a point 1.3m over the mean of the feet. All three terms use the smooth-abs norm (Figure 2).
Choosing parameters $\gamma, \lambda$

Performance as $\gamma, \lambda$ are varied

(Cart-pole performance after 20 iterations)

(3D Biped)

(Generalized Advantage Estimation for Policy Gradients, S. et al., ICLR 2016)
Pathwise Derivative Methods
Episodic MDP:

Want to compute $\nabla_{\theta} \mathbb{E}[R_T]$. We’ll use $\nabla_{\theta} \log \pi(a_t | s_t; \theta)$
Deriving the Policy Gradient, Reparameterized

- Episodic MDP:

Want to compute $\nabla_\theta \mathbb{E}[R_T]$. We’ll use $\nabla_\theta \log \pi(a_t | s_t; \theta)$

- Reparameterize: $a_t = \pi(s_t, z_t; \theta)$. $z_t$ is noise from fixed distribution.
Deriving the Policy Gradient, Reparameterized

- Episodic MDP:

Want to compute $\nabla_{\theta} \mathbb{E}[R_T]$. We’ll use $\nabla_{\theta} \log \pi(a_t \mid s_t; \theta)$

- Reparameterize: $a_t = \pi(s_t, z_t; \theta)$. $z_t$ is noise from fixed distribution.

- Only works if $P(s_2 \mid s_1, a_1)$ is known 😞
Using a $Q$-function

\[
\frac{d}{d\theta} \mathbb{E}[R_T] = \mathbb{E} \left[ \sum_{t=1}^{T} \frac{dR_T}{da_t} \frac{da_t}{d\theta} \right] = \mathbb{E} \left[ \sum_{t=1}^{T} \frac{d}{da_t} \mathbb{E}[R_T | a_t] \frac{da_t}{d\theta} \right] \\
= \mathbb{E} \left[ \sum_{t=1}^{T} \frac{d}{da_t} Q(s_t, a_t) \frac{da_t}{d\theta} \right] = \mathbb{E} \left[ \sum_{t=1}^{T} \frac{d}{d\theta} Q(s_t, \pi(s_t, z_t; \theta)) \right]
\]
SVG(0) Algorithm

- Learn $Q_\phi$ to approximate $Q^\pi,\gamma$, and use it to compute gradient estimates.
**SVG(0) Algorithm**

- Learn $Q_{\phi}$ to approximate $Q^{\pi, \gamma}$, and use it to compute gradient estimates.
- Pseudocode:

```plaintext
for iteration=1, 2, ... do
    Execute policy $\pi_{\theta}$ to collect $T$ timesteps of data
    Update $\pi_{\theta}$ using $g \propto \nabla_{\theta} \sum_{t=1}^{T} Q(s_t, \pi(s_t, z_t; \theta))$
    Update $Q_{\phi}$ using $g \propto \nabla_{\phi} \sum_{t=1}^{T} (Q_{\phi}(s_t, a_t) - \hat{Q}_t)^2$, e.g. with TD($\lambda$)
end for
```

Instead of learning $Q$, we learn

- State-value function $V \approx V_{\pi,\gamma}$
- Dynamics model $f$, approximating $s_{t+1} = f(s_t, a_t) + \zeta_t$
- Given transition $(s_t, a_t, s_{t+1})$, infer $\zeta_t = s_{t+1} - f(s_t, a_t)$
- $Q(s_t, a_t) = \mathbb{E}[r_t + \gamma V(s_{t+1})] = \mathbb{E}[r_t + \gamma f(s_t, a_t) + \zeta_t]$, and $a_t = \pi(s_t, \theta, \zeta_t)$
Instead of learning $Q$, we learn

- State-value function $V \approx V^{\pi,\gamma}$
SVG(1) Algorithm

- Instead of learning $Q$, we learn
  - State-value function $V \approx V^{\pi, \gamma}$
  - Dynamics model $f$, approximating $s_{t+1} = f(s_t, a_t) + \zeta_t$
SVG(1) Algorithm

- Instead of learning $Q$, we learn
  - State-value function $V \approx V^{\pi, \gamma}$
  - Dynamics model $f$, approximating $s_{t+1} = f(s_t, a_t) + \zeta_t$
- Given transition $(s_t, a_t, s_{t+1})$, infer $\zeta_t = s_{t+1} - f(s_t, a_t)$
SVG(1) Algorithm

- Instead of learning $Q$, we learn
  - State-value function $V \approx V^\pi, \gamma$
  - Dynamics model $f$, approximating $s_{t+1} = f(s_t, a_t) + \zeta_t$

- Given transition $(s_t, a_t, s_{t+1})$, infer $\zeta_t = s_{t+1} - f(s_t, a_t)$

- $Q(s_t, a_t) = \mathbb{E}[r_t + \gamma V(s_{t+1})] = \mathbb{E}[r_t + \gamma V(f(s_t, a_t) + \zeta_t)]$, and $a_t = \pi(s_t, \theta, \zeta_t)$
SVG(∞) Algorithm

- Just learn dynamics model $f$
SVG(∞) Algorithm

- Just learn dynamics model $f$
- Given whole trajectory, infer all noise variables
SVG(∞) Algorithm

- Just learn dynamics model $f$
- Given whole trajectory, infer all noise variables
- Freeze all policy and dynamics noise, differentiate through entire deterministic computation graph
SVG Results

- Applied to 2D robotics tasks

SVG Results

- Applied to 2D robotics tasks

- Overall: different gradient estimators behave similarly

---

Deterministic Policy Gradient

- For Gaussian actions, variance of score function policy gradient estimator goes to infinity as variance goes to zero.

Deterministic Policy Gradient

- For Gaussian actions, variance of score function policy gradient estimator goes to infinity as variance goes to zero
  - Intuition: finite difference gradient estimators

Problem: there's no exploration.

Solution: add noise to the policy, but estimate $Q$ with TD(0), so it's valid

Policy gradient is a little biased (even with $Q = Q_\pi$), but only because state distribution is off—it gets the right gradient at every state

Deterministic Policy Gradient

- For Gaussian actions, variance of score function policy gradient estimator goes to infinity as variance goes to zero
  - Intuition: finite difference gradient estimators
- But SVG(0) gradient is fine when $\sigma \to 0$

$$\nabla_\theta \sum_t Q(s_t, \pi(s_t, \theta, \zeta_t))$$

- Problem: there’s no exploration.
- Solution: add noise to the policy, but estimate $Q$ with TD(0), so it’s valid
- Policy gradient is a little biased (even with $Q = Q_\pi$), but only because state distribution is off—it gets the right gradient at every state

---

Deterministic Policy Gradient

- For Gaussian actions, variance of score function policy gradient estimator goes to infinity as variance goes to zero
  - Intuition: finite difference gradient estimators
- But SVG(0) gradient is fine when $\sigma \to 0$
  \[
  \nabla_{\theta} \sum_t Q(s_t, \pi(s_t, \theta, \zeta_t))
  \]
- Problem: there’s no exploration.

Deterministic Policy Gradient

- For Gaussian actions, variance of score function policy gradient estimator goes to infinity as variance goes to zero
  - Intuition: finite difference gradient estimators
- But SVG(0) gradient is fine when $\sigma \to 0$
  \[
  \nabla_{\theta} \sum_{t} Q(s_t, \pi(s_t, \theta, \zeta_t))
  \]
- Problem: there’s no exploration.
- Solution: add noise to the policy, but estimate $Q$ with TD(0), so it’s valid off-policy
Deterministic Policy Gradient

- For Gaussian actions, variance of score function policy gradient estimator goes to infinity as variance goes to zero
  - Intuition: finite difference gradient estimators
- But SVG(0) gradient is fine when $\sigma \to 0$
  \[
  \nabla_{\theta} \sum_{t} Q(s_t, \pi(s_t, \theta, \zeta_t))
  \]
- Problem: there’s no exploration.
- Solution: add noise to the policy, but estimate $Q$ with TD(0), so it’s valid off-policy
- Policy gradient is a little biased (even with $Q = Q^\pi$), but only because state distribution is off—it gets the right gradient at every state

Deep Deterministic Policy Gradient

- Incorporate replay buffer and target network ideas from DQN for increased stability
- Use lagged (Polyak-averaging) version of $Q_{\phi}$ and $\pi_{\theta}$ for fitting $Q_{\phi}$ (towards $Q_{\pi,\gamma}$) with TD(0)

$$\hat{Q}_t = r_t + \gamma Q_{\phi'}(s_{t+1}, \pi(s_{t+1}; \theta'))$$

- Pseudocode:

  ```python
  for iteration=1, 2, ... do
      Act for several timesteps, add data to replay buffer
      Sample minibatch
      Update $\pi_{\theta}$ using $g \propto \nabla_{\theta} \sum_{t=1}^{T} Q(s_t, \pi(s_t, z_t; \theta))$
      Update $Q_{\phi}$ using $g \propto \nabla_{\phi} \sum_{t=1}^{T} (Q_{\phi}(s_t, a_t) - \hat{Q}_t)^2$
  end for
  ```

DDPG Results

Applied to 2D and 3D robotics tasks and driving with pixel input

---

Policy Gradient Methods: Comparison

- Two kinds of policy gradient estimator
  - REINFORCE / score function estimator: $\nabla \log \pi(a|s) \hat{A}$
  - Pathwise derivative estimators (differentiate wrt action)
    - SVG(0) / DPG: $\frac{d}{da} Q(s, a)$ (learn $Q$)
    - SVG(1): $\frac{d}{da} (r + \gamma V(s'))$ (learn $f, V$)
    - SVG($\infty$): $\frac{d}{da} (r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...)$ (learn $f$)
  - Pathwise derivative methods more sample-efficient when they work (maybe), but work less generally due to high bias
Policy Gradient Methods: Comparison

- Two kinds of policy gradient estimator
  - REINFORCE / score function estimator: $\nabla \log \pi(a \mid s) \hat{A}$.

Pathwise derivative estimators (differentiate wrt action)

- SVG(0) / DPG: $\frac{d}{da} Q(s, a)$ (learn $Q$)
- SVG(1): $\frac{d}{da} (r + \gamma V(s'))$ (learn $f, V$)
- SVG(∞): $\frac{d}{da} (r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...)$ (learn $f$)

Pathwise derivative methods more sample-efficient when they work (maybe), but work less generally due to high bias.
Policy Gradient Methods: Comparison

- Two kinds of policy gradient estimator
  - REINFORCE / score function estimator: \( \nabla \log \pi(a \mid s) \hat{A} \).
    - Learn \( Q \) or \( V \) for variance reduction, to estimate \( \hat{A} \)
Policy Gradient Methods: Comparison

- Two kinds of policy gradient estimator
  - REINFORCE / score function estimator: $\nabla \log \pi(a \mid s) \hat{A}$.
    - Learn $Q$ or $V$ for variance reduction, to estimate $\hat{A}$
  - Pathwise derivative estimators (differentiate wrt action)
Policy Gradient Methods: Comparison

- Two kinds of policy gradient estimator
  - REINFORCE / score function estimator: $\nabla \log \pi(a \mid s) \hat{A}$.
    - Learn $Q$ or $V$ for variance reduction, to estimate $\hat{A}$
  - Pathwise derivative estimators (differentiate wrt action)
    - SVG(0) / DPG: $\frac{d}{da} Q(s, a)$ (learn $Q$)
Policy Gradient Methods: Comparison

- Two kinds of policy gradient estimator
  - REINFORCE / score function estimator: $\nabla \log \pi(a | s) \hat{A}$.
    - Learn $Q$ or $V$ for variance reduction, to estimate $\hat{A}$
  - Pathwise derivative estimators (differentiate wrt action)
    - SVG(0) / DPG: $\frac{d}{da} Q(s, a)$ (learn $Q$)
    - SVG(1): $\frac{d}{da} (r + \gamma V(s'))$ (learn $f, V$)

Pathwise derivative methods more sample-efficient when they work (maybe), but work less generally due to high bias.
Policy Gradient Methods: Comparison

- Two kinds of policy gradient estimator
  - REINFORCE / score function estimator: $\nabla \log \pi(a \mid s) \hat{A}$.
    - Learn $Q$ or $V$ for variance reduction, to estimate $\hat{A}$
  - Pathwise derivative estimators (differentiate wrt action)
    - SVG(0) / DPG: $\frac{d}{da} Q(s, a)$ (learn $Q$)
    - SVG(1): $\frac{d}{da} (r + \gamma V(s'))$ (learn $f, V$)
    - SVG($\infty$): $\frac{d}{da_t} (r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots)$ (learn $f$)

Pathwise derivative methods more sample-efficient when they work (maybe), but work less generally due to high bias
Policy Gradient Methods: Comparison

- Two kinds of policy gradient estimator
  - REINFORCE / score function estimator: $\nabla \log \pi(a \mid s) \hat{A}$.
    - Learn $Q$ or $V$ for variance reduction, to estimate $\hat{A}$
  - Pathwise derivative estimators (differentiate wrt action)
    - SVG(0) / DPG: $\frac{d}{da} Q(s, a)$ (learn $Q$)
    - SVG(1): $\frac{d}{da} (r + \gamma V(s'))$ (learn $f$, $V$)
    - SVG($\infty$): $\frac{d}{da_t} (r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots)$ (learn $f$)

- Pathwise derivative methods more sample-efficient when they work (maybe), but work less generally due to high bias
Thanks

Questions?