Deep Reinforcement Learning via Policy Optimization

John Schulman

OpenAI

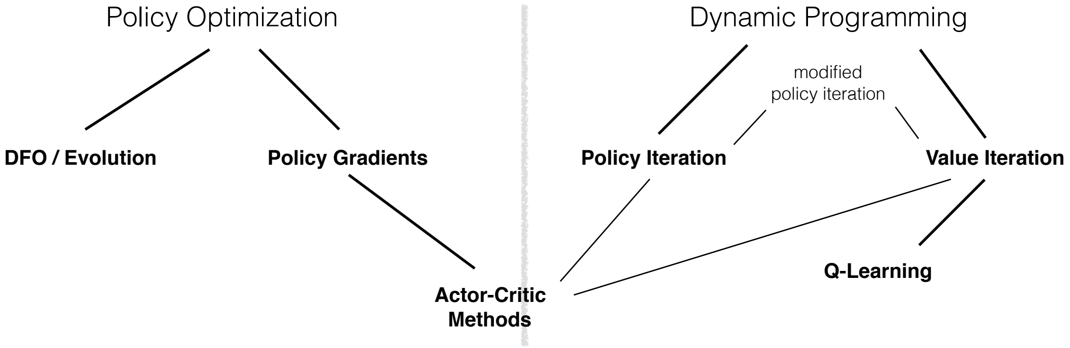
June 11, 2017

Introduction

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 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 $\theta \in \mathbb{R}^d$
  - Deterministic: $a = \pi(s, \theta)$
  - Stochastic: $\pi(a \mid s, \theta)$
- 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

Episodic Setting

- In each episode, the initial state is sampled from μ, 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

$$\underset{\pi}{\text{maximize}}\, \mathbb{E}\left[R \mid \pi\right]$$

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 $\theta_i \sim N(\mu, \mathrm{diag}(\sigma))$
  Perform one episode with each θ_i, obtaining reward R_i
  Select the top $p\%$ of θ samples (e.g. $p = 20$), the **elite set**
  Fit a Gaussian distribution, to the elite set, updating μ, σ.
end for
Return the final μ.

Cross Entropy Method

- Sometimes works embarrassingly well

Method	Mean Score	Reference
Nonreinforcement learning		
Hand-coded	631,167	Dellacherie (Fahey, 2003)
Genetic algorithm	586,103	(Böhm et al., 2004)
Reinforcement learning		
Relational reinforcement learning+kernel-based regression	≈50	Ramon and Driessens (2004)
Policy iteration	3183	Bertsekas and Tsitsiklis (1996)
Least squares policy iteration	<3000	Lagoudakis, Parr, and Littman (2002)
Linear programming + Bootstrap	4274	Farias and van Roy (2006)
Natural policy gradient	≈6800	Kakade (2001)
CE+RL	21,252	
CE+RL, constant noise	72,705	
CE+RL, decreasing noise	348,895	

I. Szita and A. Lörincz. "Learning Tetris using the noisy cross-entropy method". *Neural computation* (2006)

Approximate Dynamic Programming Finally Performs Well in the Game of Tetris

Victor Gabillon
INRIA Lille - Nord Europe, Team SequeL, FRANCE
victor.gabillon@inria.fr

Mohammad Ghavamzadeh*
INRIA Lille - Team SequeL & Adobe Research
mohammad.ghavamzadeh@inria.fr

Bruno Scherrer
INRIA Nancy - Grand Est, Team Maia, FRANCE
bruno.scherrer@inria.fr

Stochastic Gradient Ascent on Distribution

- Let μ define distribution for policy π_θ: $\theta \sim P_\mu(\theta)$
- Return R depends on policy parameter θ and noise ζ

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

  R is unknown and possibly nondifferentiable
- "Score function" gradient estimator:

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

Stochastic Gradient Ascent on Distribution

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

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

  - Cross entropy method:

$$\underset{\mu}{\text{maximize}} \frac{1}{N}\sum_{i=1}^{N} \log P_\mu(\theta_i) f(R_i) \qquad \text{(cross entropy method)}$$

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

Connection to Finite Differences

- Suppose P_μ is Gaussian distribution with mean μ, covariance $\sigma^2 I$

$$\begin{aligned}
\log P_\mu(\theta) &= -\|\mu - \theta\|^2/2\sigma^2 + const \\
\nabla_\mu \log P_\mu(\theta) &= (\theta - \mu)/\sigma^2 \\
R_i \nabla_\mu \log P_\mu(\theta_i) &= R_i(\theta_i - \mu)/\sigma^2
\end{aligned}$$

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

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

- Using same noise ζ for both evaluations reduces variance

Deriving the Score Function Estimator

- "Score function" gradient estimator:

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

- Derive by writing expectation as an integral

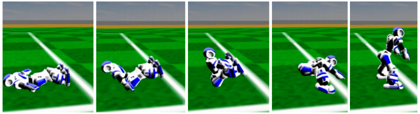
$$\nabla_\mu \int \mathrm{d}\mu \mathrm{d}\zeta \; P_\mu(\theta) R(\theta,\zeta)$$
$$= \int \mathrm{d}\mu \mathrm{d}\zeta \; \nabla_\mu P_\mu(\theta) R(\theta,\zeta)$$
$$= \int \mathrm{d}\mu \mathrm{d}\zeta \; P_\mu(\theta) \nabla_\mu \log P_\mu(\theta) R(\theta,\zeta)$$
$$= \mathbb{E}_{\theta,\zeta}\left[\nabla_\mu \log P_\mu(\theta) R(\theta,\zeta)\right]$$

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

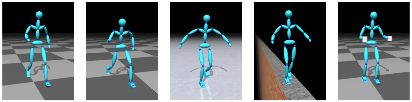
- CMA is very effective for optimizing low-dimensional locomotion controllers
  - UT Austin Villa: RoboCup 2012 3D Simulation League Champion



Optimizing Walking Controllers for Uncertain Inputs and Environments

Jack M. Wang David J. Fleet Aaron Hertzmann

University of Toronto



- Evolution Strategies was shown to perform well on Atari, competetive with policy gradient methods (Salimans et al., 2017)

Policy Gradient Methods

Overview

Problem:

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

- Here, we'll use a fixed policy parameter θ (instead of sampling $\theta \sim P_\mu$) and estimate gradient with respect to θ
- 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
2. Make the good actions more probable
3. Push the actions towards better actions

Score Function Gradient Estimator for Policies

- Now random variable is a whole trajectory $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_{T-1}, a_{T-1}, r_{T-1}, s_T)$

$$\nabla_\theta E_\tau[R(\tau)] = E_\tau[\nabla_\theta \log P(\tau \mid \theta) R(\tau)]$$

- Just need to write out $P(\tau \mid \theta)$:

$$P(\tau \mid \theta) = \mu(s_0) \prod_{t=0}^{T-1} [\pi(a_t \mid s_t, \theta) P(s_{t+1}, r_t \mid s_t, a_t)]$$

$$\log P(\tau \mid \theta) = \log \mu(s_0) + \sum_{t=0}^{T-1} [\log \pi(a_t \mid s_t, \theta) + \log P(s_{t+1}, r_t \mid s_t, a_t)]$$

$$\nabla_\theta \log P(\tau \mid \theta) = \nabla_\theta \sum_{t=0}^{T-1} \log \pi(a_t \mid s_t, \theta)$$

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

Policy Gradient: Use Temporal Structure

- Previous slide:

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

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

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

- Sum this formula over t, we obtain

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

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,
  $b(s_t) \approx \mathbb{E}\left[r_t + r_{t+1} + r_{t+2} + \cdots + r_{T-1}\right]$
- 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 γ, which ignores delayed effects between actions and rewards

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

- Now, we want $b(s_t) \approx \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots + \gamma^{T-1-t} r_{T-1}\right]$
- Write gradient estimator more generally as

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

$\hat{A}_t$ is the *advantage estimate*

"Vanilla" Policy Gradient Algorithm

Initialize policy parameter θ, baseline b
for iteration$=1, 2, \dots$ **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 $\nabla_\theta \log \pi(a_t \mid s_t, \theta) \hat{A}_t$
end for

Advantage Actor-Critic

- Use neural network that represents policy π_θ and value function V_θ (approximating V^{π_θ})
- Pseudocode

for iteration$=1, 2, \ldots$ **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 $g = \nabla_\theta \sum_{t=1}^{T} \left[-\log \pi_\theta(a_t \mid s_t) \hat{A}_t + c(V_\theta(s) - \hat{R}_t)^2 \right]$

g is plugged into a stochastic gradient ascent algorithm, e.g., Adam.

end for

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, et al. "Asynchronous methods for deep reinforcement learning". (2016)

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 π_{old} used for sampling

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

  Differentiating this objective gives the policy gradient
- $L_{\pi_{\text{old}}}(\pi)$ is only accurate when state distribution of π is close to π_{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$=1, 2, \ldots$ **do**
  Run policy for T timesteps or N trajectories
  Estimate advantage function at all timesteps

  $$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & \sum_{n=1}^{N} \frac{\pi_\theta(a_n \mid s_n)}{\pi_{\theta_{\text{old}}}(a_n \mid s_n)} \hat{A}_n \\ \text{subject to} \quad & \overline{\mathrm{KL}}_{\pi_{\theta_{\text{old}}}}(\pi_\theta) \leq \delta \end{aligned}$$

  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

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

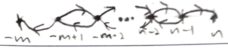
- Pseudocode:

  for iteration=$1, 2, \ldots$ **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 β. If KL too low, decrease β.
  end for

- $\approx$ same performance as TRPO, but only first-order optimization

Variance Reduction for Policy Gradients

Reward Shaping



Date

Chain MDP

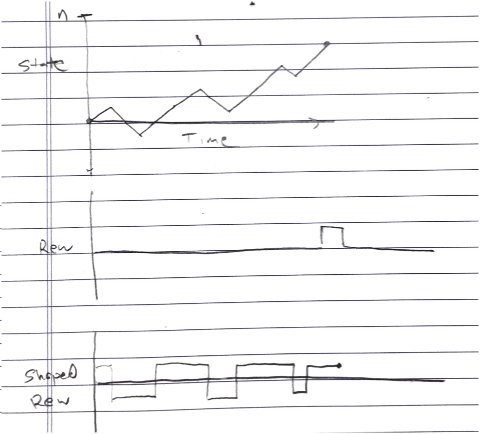
$A = \{\leftarrow, \rightarrow\}$

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

$-m$ and n are terminal

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

Initial state $s = 0$.



Reward Shaping

- Reward shaping: $\tilde{r}(s, a, s') = r(s, a, s') + \gamma\Phi(s') - \Phi(s)$ for arbitrary "potential" Φ
- Theorem: $\tilde{r}$ admits the same optimal policies as r.[1]
  - Proof sketch: suppose Q^* satisfies Bellman equation ($\mathcal{T}Q = Q$). If we transform $r \to \tilde{r}$, 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

[1] A. Y. Ng, D. Harada, and S. Russell. "Policy invariance under reward transformations: Theory and application to reward shaping". *ICML*. 1999.

Reward Shaping

- Theorem: $\tilde{R}$ 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^π and Q^π:

$$\mathbb{E}\left[r_0 + \gamma r_1 + \gamma^2 r_2 + \dots\right] \qquad \text{condition on either } s_0 \text{ or } (s_0, a_0)$$
$$= \mathbb{E}\left[\tilde{r}_0 + \gamma \tilde{r}_1 + \gamma^2 \tilde{r}_2 + \dots\right]$$
$$= \mathbb{E}\left[(r_0 + \gamma\Phi(s_1) - \Phi(s_0)) + \gamma(r_1 + \gamma\Phi(s_2) - \Phi(s_1)) + \gamma^2(r_2 + \gamma\Phi(s_3) - \Phi(s_2)) + \dots\right]$$
$$= \mathbb{E}\left[r_0 + \gamma r_1 + \gamma^2 r_2 + \dots - \Phi(s_0)\right]$$

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
- Shaping leaves policy gradient invariant (and just adds baseline to estimator)

$$
\begin{aligned}
&\mathbb{E}\big[\nabla_\theta \log \pi_\theta(a_0 \mid s_0)(r_0 + \gamma\Phi(s_1) - \Phi(s_0)) + \gamma(r_1 + \gamma\Phi(s_2) - \Phi(s_1)) \\
&\qquad + \gamma^2(r_2 + \gamma\Phi(s_3) - \Phi(s_2)) + \dots\big] \\
&= \mathbb{E}\left[\nabla_\theta \log \pi_\theta(a_0 \mid s_0)(r_0 + \gamma r_1 + \gamma^2 r_2 + \dots - \Phi(s_0))\right] \\
&= \mathbb{E}\left[\nabla_\theta \log \pi_\theta(a_0 \mid s_0)(r_0 + \gamma r_1 + \gamma^2 r_2 + \dots)\right]
\end{aligned}
$$

Reward Shaping and Policy Gradients

- First note 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

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

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

Reward Shaping and Policy Gradients

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

$$\sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t) \sum_{k=0}^{\infty} (\gamma\lambda)^k \tilde{r}_{t+k}$$

- Or alternatively, use hard cutoff as in A3C

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

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

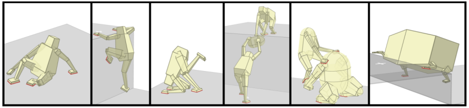
Reward Shaping—Summary

- Reward shaping transformation leaves policy gradient and optimal policy invariant
- 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

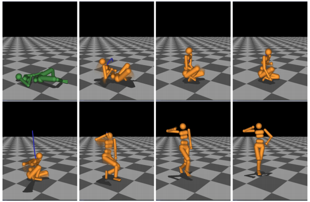
- I. Mordatch, E. Todorov, and Z. Popović. "Discovery of complex behaviors through contact-invariant optimization". *ACM Transactions on Graphics (TOG)* 31.4 (2012), p. 43

$$L(\mathbf{s}) = L_{\mathrm{CI}}(\mathbf{s}) + L_{\mathrm{Physics}}(\mathbf{s}) + L_{\mathrm{Task}}(\mathbf{s}) + L_{\mathrm{Hint}}(\mathbf{s})$$



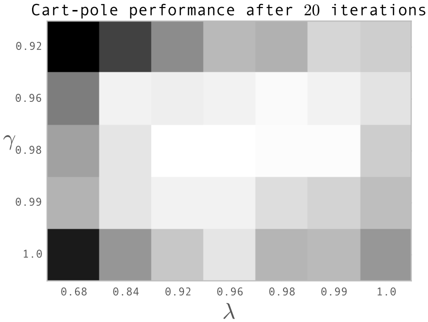
- Y. Tassa, T. Erez, and E. Todorov. "Synthesis and stabilization of complex behaviors through online trajectory optimization". *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE. 2012, pp. 4906–4913

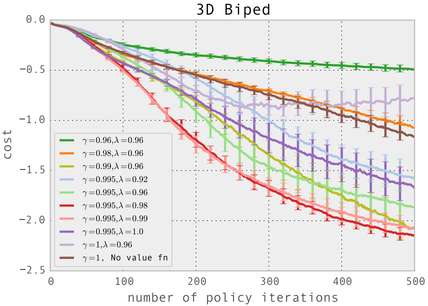
The state-cost is composed of 4 terms. The first term penalizes the horizontal distance (in the xy-plane) between the center-of-mass (CoM) and the mean of the feet positions. The second term penalizes the horizontal distance between the torso and the CoM. The third penalizes the vertical distance between 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 γ, λ

Performance as γ, λ are varied



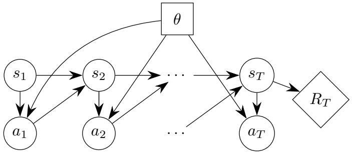


(*Generalized Advantage Estimation for Policy Gradients*, S. et al., ICLR 2016)

Pathwise Derivative Methods

Deriving the Policy Gradient, Reparameterized

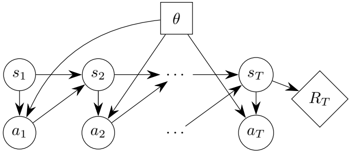
- Episodic MDP:



  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 $\ddot\frown$

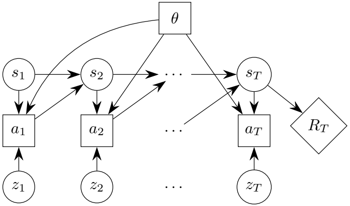
Deriving the Policy Gradient, Reparameterized

- Episodic MDP:



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

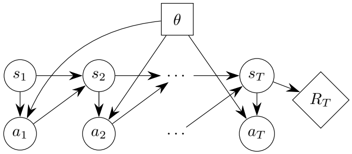
- Reparameterize: $a_t = \pi(s_t, z_t; \theta)$. z_t is noise from fixed distribution.



- Only works if $P(s_2 \mid s_1, a_1)$ is known $\ddot{\frown}$

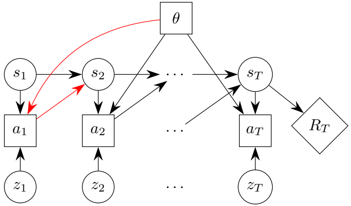
Deriving the Policy Gradient, Reparameterized

- Episodic MDP:



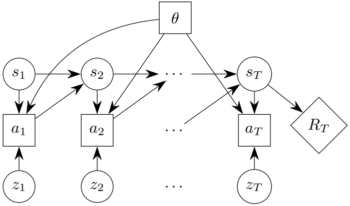
Want to compute $\nabla_\theta \mathbb{E}[R_T]$. We'll use $\nabla_\theta \log \pi(a_t \mid s_t; \theta)$

- 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 $\ddot{\frown}$

Using a Q-function



$$\frac{\mathrm{d}}{\mathrm{d}\theta}\mathbb{E}\left[R_T\right] = \mathbb{E}\left[\sum_{t=1}^{T}\frac{\mathrm{d}R_T}{\mathrm{d}a_t}\frac{\mathrm{d}a_t}{\mathrm{d}\theta}\right] = \mathbb{E}\left[\sum_{t=1}^{T}\frac{\mathrm{d}}{\mathrm{d}a_t}\mathbb{E}\left[R_T \mid a_t\right]\frac{\mathrm{d}a_t}{\mathrm{d}\theta}\right]$$

$$= \mathbb{E}\left[\sum_{t=1}^{T}\frac{\mathrm{d}Q(s_t, a_t)}{\mathrm{d}a_t}\frac{\mathrm{d}a_t}{\mathrm{d}\theta}\right] = \mathbb{E}\left[\sum_{t=1}^{T}\frac{\mathrm{d}}{\mathrm{d}\theta}Q(s_t, \pi(s_t, z_t; \theta))\right]$$

SVG(0) Algorithm

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

N. Heess, G. Wayne, D. Silver, T. Lillicrap, Y. Tassa, et al. "Learning Continuous Control Policies by Stochastic Value Gradients". *arXiv preprint arXiv:1510.09142* (2015)

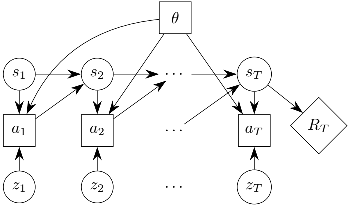
SVG(0) Algorithm

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

for iteration$=1, 2, \ldots$ **do**
  Execute policy π_θ to collect T timesteps of data
  Update π_θ using $g \propto \nabla_\theta \sum_{t=1}^{T} Q(s_t, \pi(s_t, z_t; \theta))$
  Update Q_ϕ 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

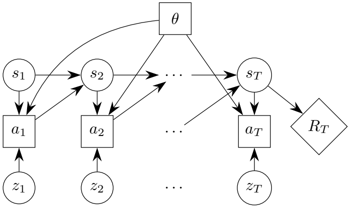
N. Heess, G. Wayne, D. Silver, T. Lillicrap, Y. Tassa, et al. "Learning Continuous Control Policies by Stochastic Value Gradients". *arXiv preprint arXiv:1510.09142* (2015)

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 $a_t = \pi(s_t, \theta, \zeta_t)$

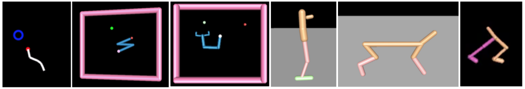
SVG(∞) Algorithm



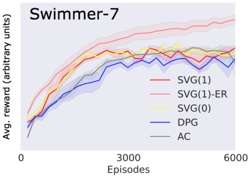
- Just learn dynamics model f
- Given whole trajectory, infer all noise variables
- Freeze all policy and dynamics noise, differentiate through entire deterministic computation graph

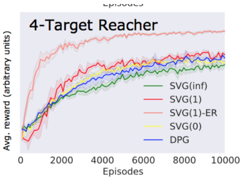
SVG Results

- Applied to 2D robotics tasks



- Overall: different gradient estimators behave similarly





N. Heess, G. Wayne, D. Silver, T. Lillicrap, Y. Tassa, et al. "Learning Continuous Control Policies by Stochastic Value Gradients". *arXiv preprint arXiv:1510.09142* (2015)

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

D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, et al. "Deterministic Policy Gradient Algorithms". *ICML*. 2014

Deep Deterministic Policy Gradient

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

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

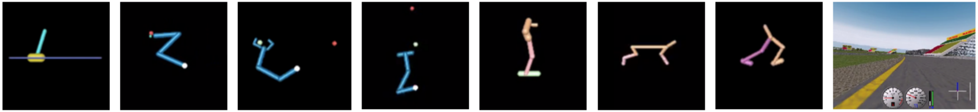
- Pseudocode:

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

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, et al. "Continuous control with deep reinforcement learning". *arXiv preprint arXiv:1509.02971* (2015)

DDPG Results

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



T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, et al. "Continuous control with deep reinforcement learning". *arXiv preprint arXiv:1509.02971* (2015)

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: $\frac{\mathrm{d}}{\mathrm{d}a} Q(s, a)$ (learn Q)
    - SVG(1): $\frac{\mathrm{d}}{\mathrm{d}a}(r + \gamma V(s'))$ (learn f, V)
    - SVG(∞): $\frac{\mathrm{d}}{\mathrm{d}a_t}(r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots)$ (learn f)
- Pathwise derivative methods more sample-efficient when they work (maybe), but work less generally due to high bias

Thanks

Questions?