Trust Region Policy Optimization

Desiderata

Desiderata for policy optimization method:

- Stable, monotonic improvement. (How to choose stepsizes?)
- Good sample efficiency

Step Sizes

Why are step sizes a big deal in RL?

- Supervised learning
  - Step too far $\rightarrow$ next updates will fix it
- Reinforcement learning
  - Step too far $\rightarrow$ bad policy
  - Next batch: collected under bad policy
  - Can't recover, collapse in performance!

Surrogate Objective

- Let $\eta(\pi)$ denote the expected return of π
- We collect data with π_{old}. Want to optimize some objective to get a new policy π
- Define $L_{\pi_{\text{old}}}(\pi)$ to be the "surrogate objective"[1]

$$L(\pi) = \mathbb{E}_{\pi_{\text{old}}}\left[\frac{\pi(a \mid s)}{\pi_{\text{old}}(a \mid s)} A^{\pi_{\text{old}}}(s, a)\right]$$

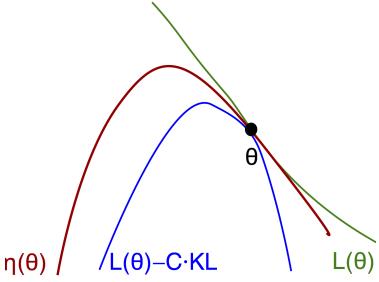
$$\nabla_\theta L(\pi_\theta)\big|_{\theta_{\text{old}}} = \nabla_\theta \eta(\pi_\theta)\big|_{\theta_{\text{old}}} \quad \text{(policy gradient)}$$

- Local approximation to the performance of the policy; does not depend on parameterization of π

[1] S. Kakade and J. Langford. "Approximately optimal approximate reinforcement learning". In: *ICML*. vol. 2. 2002, pp. 267–274.

Improvement Theory

- Theory: bound the difference between $L_{\pi_{\text{old}}}(\pi)$ and $\eta(\pi)$, the performance of the policy
- Result: $\eta(\pi) \geq L_{\pi_{\text{old}}}(\pi) - C \cdot \max_s \text{KL}[\pi_{\text{old}}(\cdot \mid s), \pi(\cdot \mid s)]$, where $c = 2\epsilon\gamma/(1-\gamma)^2$
- Monotonic improvement guaranteed (MM algorithm)



Practical Algorithm: TRPO

- Constrained optimization problem

$$\max_{\pi} L(\pi), \text{ subject to } \overline{\mathrm{KL}}[\pi_{\mathrm{old}}, \pi] \leq \delta$$

$$\text{where } L(\pi) = \mathbb{E}_{\pi_{\mathrm{old}}} \left[\frac{\pi(a \mid s)}{\pi_{\mathrm{old}}(a \mid s)} A^{\pi_{\mathrm{old}}}(s, a) \right]$$

- Construct loss from empirical data

$$\hat{L}(\pi) = \sum_{n=1}^{N} \frac{\pi(a_n \mid s_n)}{\pi_{\mathrm{old}}(a_n \mid s_n)} \hat{A}_n$$

- Make quadratic approximation and solve with conjugate gradient algorithm

J. Schulman, S. Levine, P. Moritz, et al. "Trust Region Policy Optimization". In: *ICML*. 2015

Practical Algorithm: TRPO

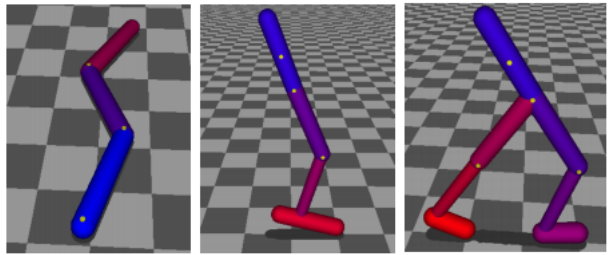
for iteration=$1, 2, \ldots$ **do**
  Run policy for T timesteps or N trajectories
  Estimate advantage function at all timesteps
  Compute policy gradient g
  Use CG (with Hessian-vector products) to compute $F^{-1}g$
  Do line search on surrogate loss and KL constraint
end for

J. Schulman, S. Levine, P. Moritz, et al. "Trust Region Policy Optimization". In: *ICML*. 2015

Practical Algorithm: TRPO

Applied to

- Locomotion controllers in 2D



- Atari games with pixel input

J. Schulman, S. Levine, P. Moritz, et al. "Trust Region Policy Optimization". In: *ICML*. 2015

"Proximal" Policy Optimization

- Use penalty instead of constraint

$$\underset{\theta}{\text{minimize}} \sum_{n=1}^{N} \frac{\pi_\theta(a_n \mid s_n)}{\pi_{\theta_{\text{old}}}(a_n \mid s_n)} \hat{A}_n - \beta \overline{\text{KL}}[\pi_{\theta_{\text{old}}}, \pi_\theta]$$

"Proximal" Policy Optimization

- Use penalty instead of constraint

$$\underset{\theta}{\text{minimize}} \sum_{n=1}^{N} \frac{\pi_\theta(a_n \mid s_n)}{\pi_{\theta_{\text{old}}}(a_n \mid s_n)} \hat{A}_n - \beta \overline{\text{KL}}[\pi_{\theta_{\text{old}}}, \pi_\theta]$$

- 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

"Proximal" Policy Optimization

- Use penalty instead of constraint

$$\underset{\theta}{\text{minimize}} \sum_{n=1}^{N} \frac{\pi_\theta(a_n \mid s_n)}{\pi_{\theta_{\text{old}}}(a_n \mid s_n)} \hat{A}_n - \beta \overline{\text{KL}}[\pi_{\theta_{\text{old}}}, \pi_\theta]$$

- 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 Using Value Functions

Variance Reduction

- Now, we have the following policy gradient formula:

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

- A^π is not known, but we can plug in $\hat{A}_t$, an *advantage estimator*
- Previously, we showed that taking

$$\hat{A}_t = r_t + r_{t+1} + r_{t+2} + \cdots - b(s_t)$$

for any function $b(s_t)$, gives an unbiased policy gradient estimator. $b(s_t) \approx V^\pi(s_t)$ gives variance reduction.

The Delayed Reward Problem

- With policy gradient methods, we are confounding the effect of multiple actions:

$$\hat{A}_t = r_t + r_{t+1} + r_{t+2} + \cdots - b(s_t)$$

  mixes effect of $a_t, a_{t+1}, a_{t+2}, \ldots$
- SNR of $\hat{A}_t$ scales roughly as $1/T$
  - Only a_t contributes to *signal* $A^\pi(s_t, a_t)$, but $a_{t+1}, a_{t+2}, \ldots$ contribute to noise.

Variance Reduction with Discounts

- Discount factor γ, $0 < \gamma < 1$, downweights the effect of rewars that are far in the future—ignore long term dependencies
- We can form an advantage estimator using the *discounted return*:

$$\hat{A}_t^{\gamma} = \underbrace{r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots}_{\text{discounted return}} - b(s_t)$$

  reduces to our previous estimator when $\gamma = 1$.
- So advantage has expectation zero, we should fit baseline to be *discounted value function*

$$V^{\pi,\gamma}(s) = \mathbb{E}_{\tau}\left[r_0 + \gamma r_1 + \gamma^2 r_2 + \dots \mid s_0 = s\right]$$

- Discount γ is similar to using a horizon of $1/(1-\gamma)$ timesteps
- $\hat{A}_t^{\gamma}$ is a biased estimator of the advantage function

Value Functions in the Future

- Baseline accounts for and removes the effect of *past* actions
- Can also use the value function to estimate future rewards

$$
\begin{aligned}
&r_t + \gamma V(s_{t+1}) && \text{cut off at one timestep} \\
&r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) && \text{cut off at two timesteps} \\
&\dots \\
&r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots && \infty \text{ timesteps (no } V\text{)}
\end{aligned}
$$

Value Functions in the Future

- Subtracting out baselines, we get advantage estimators

$$\hat{A}_t^{(1)} = r_t + \gamma V(s_{t+1}) - V(s_t)$$
$$\hat{A}_t^{(2)} = r_t + r_{t+1} + \gamma^2 V(s_{t+2}) - V(s_t)$$
$$\dots$$
$$\hat{A}_t^{(\infty)} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots - V(s_t)$$

- $\hat{A}_t^{(1)}$ has low variance but high bias, $\hat{A}_t^{(\infty)}$ has high variance but low bias.
- Using intermediate k (say, 20) gives an intermediate amount of bias and variance

Finite-Horizon Methods: Advantage Actor-Critic

- A2C / A3C uses this fixed-horizon advantage estimator

V. Mnih, A. P. Badia, M. Mirza, et al. "Asynchronous Methods for Deep Reinforcement Learning". In: *ICML* (2016).

Finite-Horizon Methods: Advantage Actor-Critic

- A2C / A3C uses this fixed-horizon advantage estimator
- Pseudocode

for iteration$=1, 2, \ldots$ **do**
Agent acts for T timesteps (e.g., $T = 20$),
For each timestep t, compute

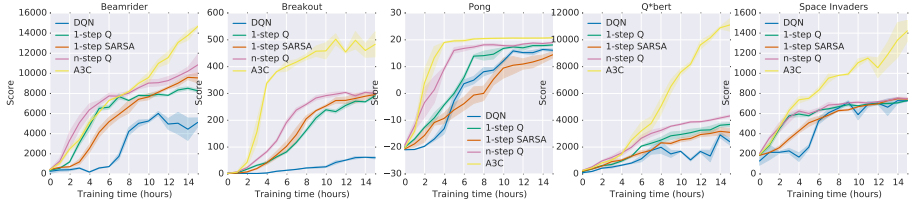
$$\hat{R}_t = r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V(s_t)$$
$$\hat{A}_t = \hat{R}_t - V(s_t)$$

$\hat{R}_t$ is target value function, in regression problem
$\hat{A}_t$ is estimated advantage function
Compute loss gradient $g = \nabla_\theta \sum_{t=1}^{T} \left[-\log \pi_\theta(a_t \mid s_t) \hat{A}_t + c(V(s) - \hat{R}_t)^2 \right]$
g is plugged into a stochastic gradient descent variant, e.g., Adam.
end for

V. Mnih, A. P. Badia, M. Mirza, et al. "Asynchronous Methods for Deep Reinforcement Learning". In: *ICML* (2016).

A3C Video

A3C Results


Beamrider
Breakout
Pong
Q*bert
Space Invaders
DQN
1-step Q
1-step SARSA
n-step Q
A3C
Score
Training time (hours)

TD(λ) Methods: Generalized Advantage Estimation

- Recall, finite-horizon advantage estimators

$$\hat{A}_t^{(k)} = r_t + \gamma r_{t+1} + \cdots + \gamma^{k-1} r_{t+k-1} + \gamma^k V(s_{t+k}) - V(s_t)$$

- Define the TD error $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$
- By a telescoping sum,

$$\hat{A}_t^{(k)} = \delta_t + \gamma \delta_{t+1} + \cdots + \gamma^{k-1} \delta_{t+k-1}$$

- Take exponentially weighted average of finite-horizon estimators:

$$\hat{A}^\lambda = \hat{A}_t^{(1)} + \lambda \hat{A}_t^{(2)} + \lambda^2 \hat{A}_t^{(3)} + \ldots$$

- We obtain

$$\hat{A}_t^\lambda = \delta_t + (\gamma\lambda)\delta_{t+1} + (\gamma\lambda)^2 \delta_{t+2} + \ldots$$

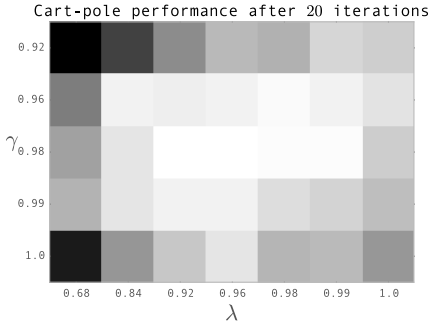
- This scheme named *generalized advantage estimation* (GAE) in [1], though versions have appeared earlier, e.g., [2]. Related to TD(λ)

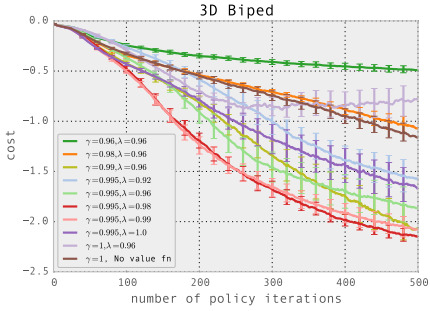
J. Schulman, P. Moritz, S. Levine, et al. "High-dimensional continuous control using generalized advantage estimation". In: *ICML*. 2015

H. Kimura and S. Kobayashi. "An Analysis of Actor/Critic Algorithms Using Eligibility Traces: Reinforcement Learning with Imperfect Value Function." In: *ICML*. 1998, pp. 278–286

Choosing parameters γ, λ

Performance as γ, λ are varied



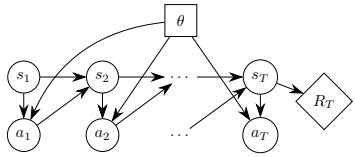


TRPO+GAE Video

Pathwise Derivative Policy Gradient Methods

Deriving the Policy Gradient, Reparameterized

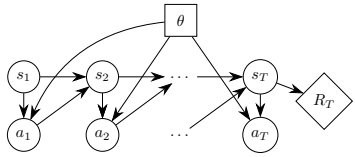
- Episodic MDP:



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

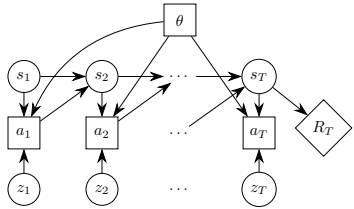
Deriving the Policy Gradient, Reparameterized

- Episodic MDP:



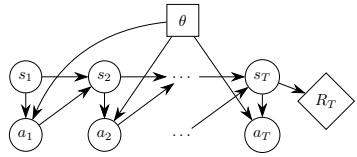
  Want to compute $\nabla_\theta \mathbb{E}[R_T]$. We'll use $\nabla_\theta \log \pi(a_t \mid s_t; \theta)$

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



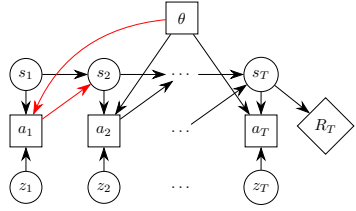
Deriving the Policy Gradient, Reparameterized

- Episodic MDP:



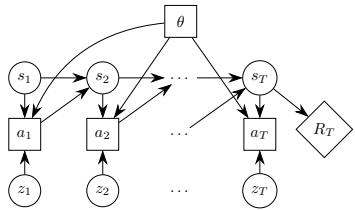
  Want to compute $\nabla_\theta \mathbb{E}[R_T]$. We'll use $\nabla_\theta \log \pi(a_t \mid s_t; \theta)$

- 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, et al. "Learning continuous control policies by stochastic value gradients". In: *NIPS*. 2015

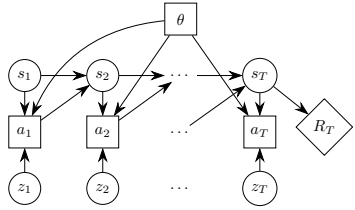
SVG(0) Algorithm

- 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

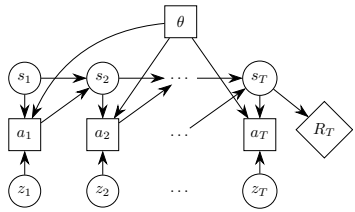
N. Heess, G. Wayne, D. Silver, et al. "Learning continuous control policies by stochastic value gradients". In: *NIPS*. 2015

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

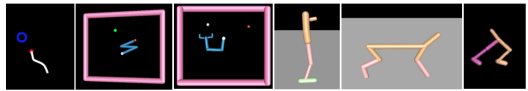
SVG(∞) Algorithm



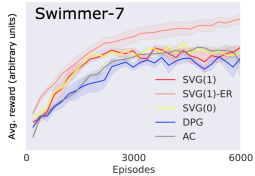
- Just learn dynamics model f
- Given whole trajectory, infer all noise variables
- Freeze all policy and dynamics noise, differentiate through entire deterministic computation graph

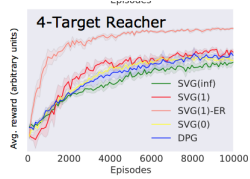
SVG Results

- Applied to 2D robotics tasks



- Overall: different gradient estimators behave similarly





N. Heess, G. Wayne, D. Silver, et al. "Learning continuous control policies by stochastic value gradients". In: *NIPS*. 2015

Deterministic Policy Gradient

- For Gaussian actions, variance of score function policy gradient estimator goes to infinity as variance goes to zero
- 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, et al. "Deterministic policy gradient algorithms". In: *ICML*. 2014

Deep Deterministic Policy Gradient

- Incorporate replay buffer and target network ideas from DQN for increased stability

T. P. Lillicrap, J. J. Hunt, A. Pritzel, et al. "Continuous control with deep reinforcement learning". In: *ICLR* (2015).

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)

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

T. P. Lillicrap, J. J. Hunt, A. Pritzel, et al. "Continuous control with deep reinforcement learning". In: *ICLR* (2015).

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, et al. "Continuous control with deep reinforcement learning". In: *ICLR* (2015).

DDPG Results

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

T. P. Lillicrap, J. J. Hunt, A. Pritzel, et al. "Continuous control with deep reinforcement learning". In: *ICLR* (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

Policy Gradient Methods: Comparison

Task	Random	REINFORCE	TNPG	RWR	REPS	TRPO	CEM	CMA-ES	DDPG
Cart-Pole Balancing	77.1 ± 0.0	4693.7 ± 14.0	**3986.4 ± 748.9**	**4861.5 ± 12.3**	565.6 ± 137.6	**4869.8 ± 37.6**	4815.4 ± 4.8	2440.4 ± 568.3	4634.4 ± 87.8
Inverted Pendulum*	−153.4 ± 0.2	13.4 ± 18.0	**209.7 ± 55.5**	84.7 ± 13.8	−113.3 ± 4.6	**247.2 ± 76.1**	38.2 ± 25.7	−40.1 ± 5.7	40.0 ± 244.6
Mountain Car	−415.4 ± 0.0	−67.1 ± 1.0	**-66.5 ± 4.5**	−79.4 ± 1.1	−275.6 ± 166.3	**-61.7 ± 0.9**	−66.0 ± 2.4	−85.0 ± 7.7	−288.4 ± 170.3
Acrobot	−1904.5 ± 1.0	−508.1 ± 91.0	−395.8 ± 121.2	−352.7 ± 35.9	−1001.5 ± 10.8	−326.0 ± 24.4	−436.8 ± 14.7	−785.6 ± 13.1	**-223.6 ± 5.8**
Double Inverted Pendulum*	149.7 ± 0.1	4116.5 ± 65.2	**4455.4 ± 37.6**	3614.8 ± 368.1	446.7 ± 114.8	**4412.4 ± 50.4**	2566.2 ± 178.9	1576.1 ± 51.3	2863.4 ± 154.0
Swimmer*	−1.7 ± 0.1	92.3 ± 0.1	**96.0 ± 0.2**	60.7 ± 5.5	3.8 ± 3.3	**96.0 ± 0.2**	68.8 ± 2.4	64.9 ± 1.4	85.8 ± 1.8
Hopper	8.4 ± 0.0	714.0 ± 29.3	**1155.1 ± 57.9**	553.2 ± 71.0	86.7 ± 17.6	**1183.3 ± 150.0**	63.1 ± 7.8	20.3 ± 14.3	267.1 ± 43.5
2D Walker	−1.7 ± 0.0	506.5 ± 78.8	**1302.6 ± 108.2**	136.0 ± 15.9	−37.0 ± 38.1	**1353.8 ± 85.0**	84.5 ± 19.2	77.1 ± 24.3	318.4 ± 181.6
Half-Cheetah	−90.8 ± 0.3	1183.1 ± 69.2	**1729.5 ± 184.6**	376.1 ± 28.2	34.5 ± 38.0	**1914.0 ± 120.1**	330.4 ± 274.8	441.3 ± 107.6	**2148.6 ± 702.7**
Ant*	13.4 ± 0.7	548.3 ± 55.5	**706.0 ± 127.7**	37.6 ± 3.1	39.0 ± 9.8	**730.2 ± 61.3**	49.2 ± 5.9	17.8 ± 15.5	326.2 ± 20.8
Simple Humanoid	41.5 ± 0.2	128.1 ± 34.0	**255.0 ± 24.5**	93.3 ± 17.4	28.3 ± 4.7	**269.7 ± 40.3**	60.6 ± 12.9	28.7 ± 3.9	99.4 ± 28.1
Full Humanoid	13.2 ± 0.1	262.2 ± 10.5	**288.4 ± 25.2**	46.7 ± 5.6	41.7 ± 6.1	**287.9 ± 23.4**	36.9 ± 2.9	N/A ± N/A	119.0 ± 31.2
Cart-Pole Balancing (LS)*	77.1 ± 0.0	420.9 ± 265.5	**945.1 ± 27.8**	68.9 ± 1.5	898.1 ± 22.1	**960.2 ± 46.0**	227.0 ± 223.0	68.0 ± 1.6	
Inverted Pendulum (LS)	−122.1 ± 0.1	−13.4 ± 3.2	**0.7 ± 6.1**	−107.4 ± 0.2	−87.2 ± 8.0	**4.5 ± 4.1**	−81.2 ± 33.2	−62.4 ± 3.4	
Mountain Car (LS)	−83.0 ± 0.0	−81.2 ± 0.6	**-65.7 ± 9.0**	−81.7 ± 0.1	−82.6 ± 0.4	**-64.2 ± 9.5**	**-68.9 ± 1.3**	**-73.2 ± 0.6**	
Acrobot (LS)*	−393.2 ± 0.0	−128.9 ± 11.6	**-84.6 ± 2.9**	−235.9 ± 5.3	−379.5 ± 1.4	**-83.3 ± 9.9**	−149.5 ± 15.3	−159.9 ± 7.5	
Cart-Pole Balancing (NO)*	101.4 ± 0.1	616.0 ± 210.8	**916.3 ± 23.0**	93.8 ± 1.2	99.6 ± 7.2	606.2 ± 122.2	181.4 ± 32.1	104.4 ± 16.0	
Inverted Pendulum (NO)	−122.2 ± 0.1	6.5 ± 1.1	**11.5 ± 0.5**	−110.0 ± 1.4	−119.3 ± 4.2	**10.4 ± 2.2**	−55.6 ± 16.7	−80.3 ± 2.8	
Mountain Car (NO)	−83.0 ± 0.0	−74.7 ± 7.8	**-64.5 ± 8.6**	−81.7 ± 0.1	−82.9 ± 0.1	**-60.2 ± 2.0**	−67.4 ± 1.4	−73.5 ± 0.5	
Acrobot (NO)*	−393.5 ± 0.0	**-186.7 ± 31.3**	**-164.5 ± 13.4**	−233.1 ± 0.4	−258.5 ± 14.0	**-149.6 ± 8.6**	−213.4 ± 6.3	−236.6 ± 6.2	
Cart-Pole Balancing (SI)*	76.3 ± 0.1	431.7 ± 274.1	**980.5 ± 7.3**	69.0 ± 2.8	702.4 ± 196.4	**980.3 ± 5.1**	746.6 ± 93.2	71.6 ± 2.9	
Inverted Pendulum (SI)	−121.8 ± 0.2	−5.3 ± 5.6	**14.8 ± 1.7**	−108.7 ± 4.7	−92.8 ± 23.9	**14.1 ± 0.9**	−51.8 ± 10.6	−63.1 ± 4.8	
Mountain Car (SI)	−82.7 ± 0.0	−63.9 ± 0.2	**-61.8 ± 0.4**	−81.4 ± 0.1	−80.7 ± 2.3	**-61.6 ± 0.4**	−63.9 ± 1.0	−66.9 ± 0.6	
Acrobot (SI)*	−387.8 ± 1.0	**-169.1 ± 32.3**	**-156.6 ± 38.9**	−233.2 ± 2.6	−216.1 ± 7.7	**-170.9 ± 40.3**	−250.2 ± 13.7	−245.0 ± 5.5	
Swimmer + Gathering	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
Ant + Gathering	−5.8 ± 5.0	−0.1 ± 0.1	−0.4 ± 0.1	−5.5 ± 0.5	−6.7 ± 0.7	−0.4 ± 0.0	−4.7 ± 0.7	N/A ± N/A	−0.3 ± 0.3
Swimmer + Maze	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
Ant + Maze	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	N/A ± N/A	0.0 ± 0.0

Y. Duan, X. Chen, R. Houthooft, et al. "Benchmarking Deep Reinforcement Learning for Continuous Control". In: *ICML* (2016).

Stochastic Computation Graphs

Gradients of Expectations

Want to compute $\nabla_\theta \mathbb{E}[F]$. Where's θ?

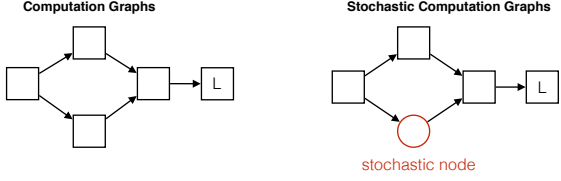
- In distribution, e.g., $\mathbb{E}_{x \sim p(\cdot \mid \theta)}[F(x)]$
  - $\nabla_\theta \mathbb{E}_x[f(x)] = \mathbb{E}_x[f(x) \nabla_\theta \log p_x(x;\ \theta)]$.
  - Score function estimator
  - Example: REINFORCE policy gradients, where x is the trajectory
- Outside distribution: $\mathbb{E}_{z \sim \mathcal{N}(0,1)}[F(\theta, z)]$

$$\nabla_\theta \mathbb{E}_z[f(x(z, \theta))] = \mathbb{E}_z[\nabla_\theta f(x(z, \theta))].$$

  - Pathwise derivative estimator
  - Example: SVG policy gradient
- Often, we can reparametrize, to change from one form to another
- What if F depends on θ in complicated way, affecting distribution and F?

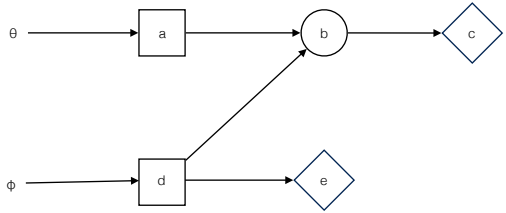
Stochastic Computation Graphs

- Stochastic computation graph is a DAG, each node corresponds to a deterministic or stochastic operation
- Can automatically derive unbiased gradient estimators, with variance reduction



J. Schulman, N. Heess, T. Weber, et al. "Gradient Estimation Using Stochastic Computation Graphs". In: *NIPS*. 2015

Worked Example



- $L = c + e$. Want to compute $\frac{\mathrm{d}}{\mathrm{d}\theta}\mathbb{E}[L]$ and $\frac{\mathrm{d}}{\mathrm{d}\phi}\mathbb{E}[L]$.
- Treat stochastic nodes (b, d) as constants, and introduce losses $\mathit{logprob} * (\mathit{futurecost})$ at each stochastic node
- Obtain unbiased gradient estimate by differentiating surrogate:

$$\mathrm{Surrogate}(\theta, \psi) = \underbrace{c + e}_{(1)} + \underbrace{\log p(\hat{b} \mid a, d)\hat{c}}_{(2)}$$

(1): how parameters influence cost through deterministic dependencies

(2): how parameters affect distribution over random variables.