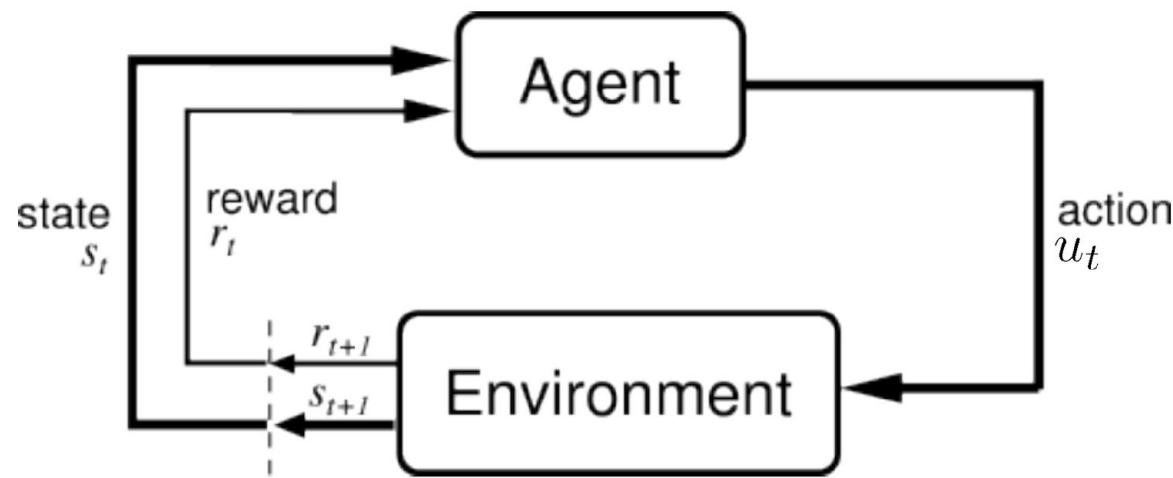


Lecture 7

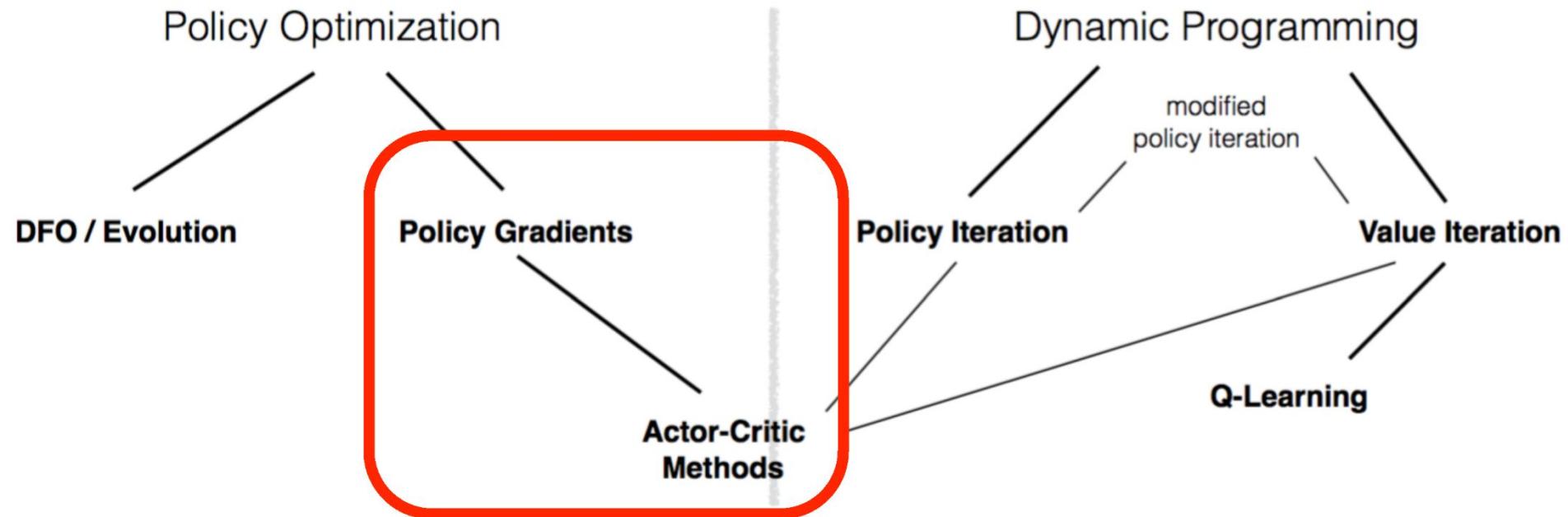
Policy Gradients II

Reinforcement Learning

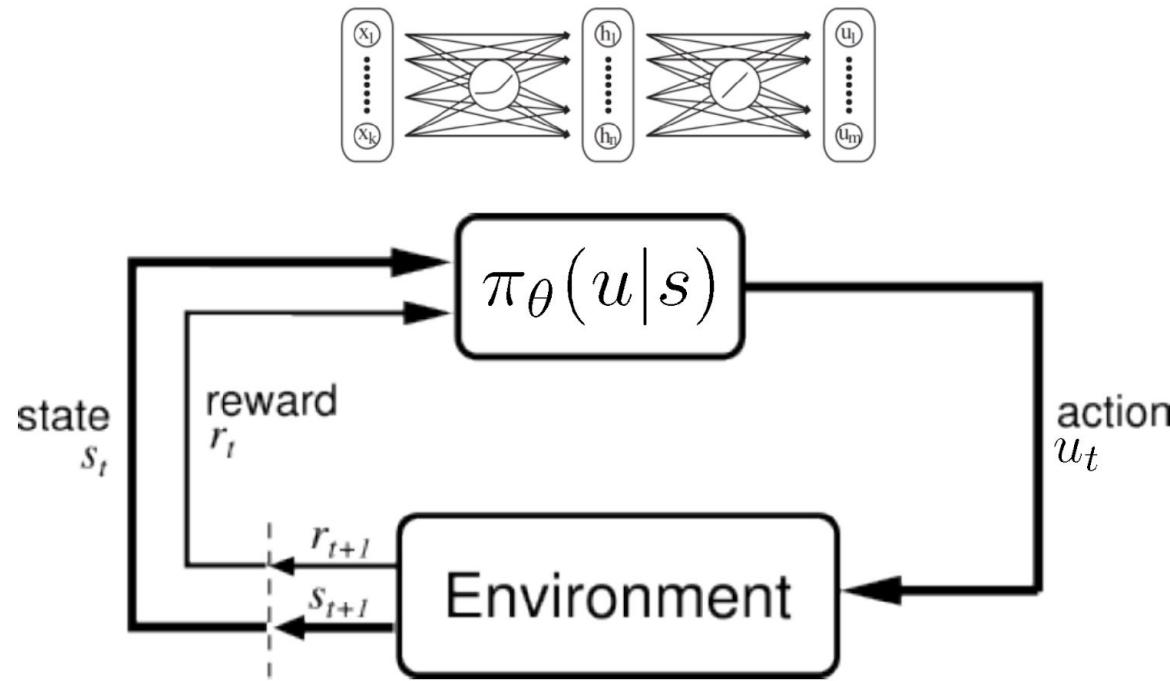


[Figure source: Sutton & Barto, 1998]

Policy Optimization in the RL Landscape



Policy Optimization

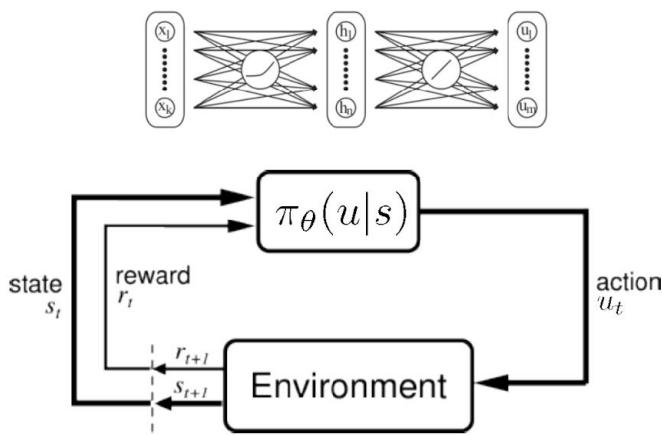


[Figure source: Sutton & Barto, 1998]

Policy Optimization

- Consider control policy parameterized by parameter vector θ

$$\max_{\theta} \mathbb{E}\left[\sum_{t=0}^H R(s_t) | \pi_{\theta}\right]$$



- Stochastic policy class (smooths out the problem):

$\pi_{\theta}(u|s)$: probability of action u in state s

Why Policy Optimization

- Often π can be simpler than Q or V
 - E.g., robotic grasp
- V: doesn't prescribe actions
 - Would need dynamics model (+ compute 1 Bellman back-up)
- Q: need to be able to efficiently solve $\arg \max_u Q_\theta(s, u)$
 - Challenge for continuous / high-dimensional action spaces*

*some recent work (partially) addressing this:

NAF: Gu, Lillicrap, Sutskever, Levine ICML 2016

Input Convex NNs: Amos, Xu, Kolter arXiv 2016

Deep Energy Q: Haarnoja, Tang, Abbeel, Levine, ICML 2017

- Conceptually:
- Empirically:

Policy Optimization

Optimize what you care about

More compatible with rich architectures (including recurrence)

More versatile

More compatible with auxiliary objectives

Dynamic Programming

Indirect, exploit the problem structure, self-consistency

More compatible with exploration and off-policy learning

More sample-efficient when they work

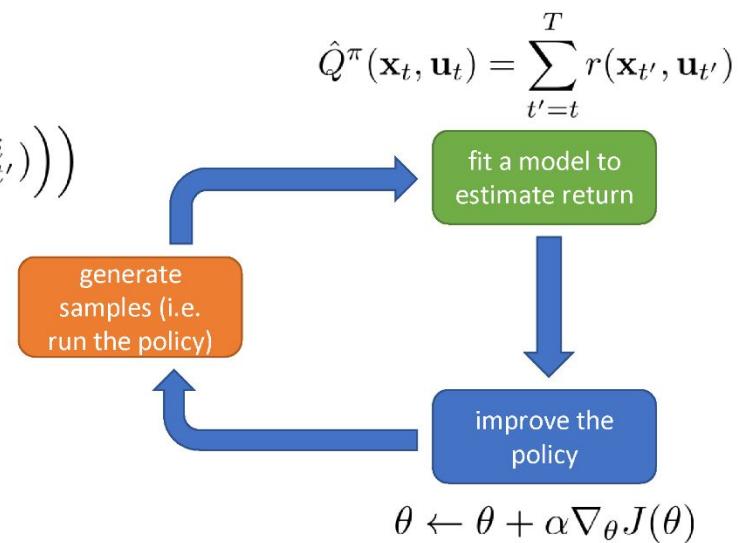
Recap: policy gradients

REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i | \mathbf{s}_t^i) \left(\sum_{t'=t}^T r(\mathbf{s}_{t'}^i, \mathbf{a}_{t'}^i) \right) \right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \hat{Q}_{i,t}^\pi$$

“reward to go”



$$\hat{Q}^\pi(\mathbf{x}_t, \mathbf{u}_t) = \sum_{t'=t}^T r(\mathbf{x}_{t'}, \mathbf{u}_{t'})$$

Improving the policy gradient

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \underbrace{\left(\sum_{t'=1}^T r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)}_{\text{"reward to go"}}$$

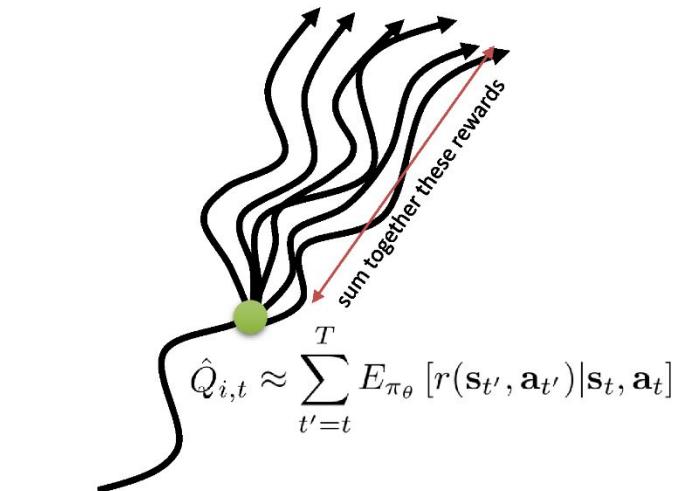
$\hat{Q}_{i,t}$

$\hat{Q}_{i,t}$: estimate of expected reward if we take action $\mathbf{a}_{i,t}$ in state $\mathbf{s}_{i,t}$

can we get a better estimate?

$Q(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^T E_{\pi_{\theta}} [r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t, \mathbf{a}_t]$: true *expected* reward-to-go

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) (Q(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) - V(\mathbf{s}_{i,t}))$$



$$V(\mathbf{s}_t) = E_{\mathbf{a}_t \sim \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t)} [Q(\mathbf{s}_t, \mathbf{a}_t)]$$

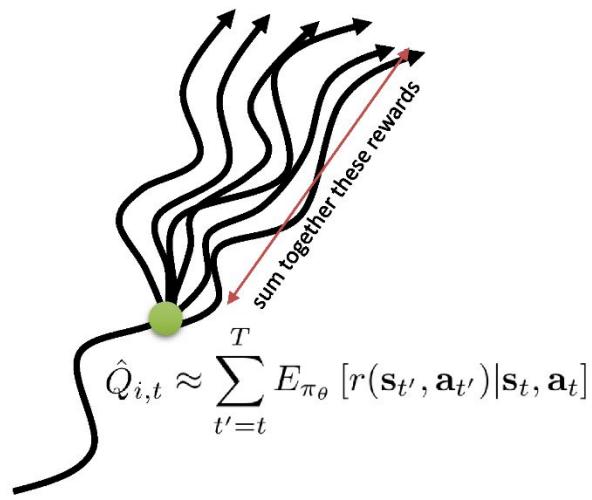
What about the baseline?

$Q(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^T E_{\pi_\theta} [r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t, \mathbf{a}_t]$: true expected reward-to-go

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) (Q(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) - V(\mathbf{s}_{i,t}))$$

$$b_t = \frac{1}{N} \sum_i Q(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \quad \text{average what?}$$

$$V(\mathbf{s}_t) = E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} [Q(\mathbf{s}_t, \mathbf{a}_t)]$$



State & state-action value functions

$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^T E_{\pi_\theta}[r(\mathbf{s}_{t'}, \mathbf{a}_{t'})|\mathbf{s}_t, \mathbf{a}_t]: \text{total reward from taking } \mathbf{a}_t \text{ in } \mathbf{s}_t$$

$$V^\pi(\mathbf{s}_t) = E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}[Q^\pi(\mathbf{s}_t, \mathbf{a}_t)]: \text{total reward from } \mathbf{s}_t$$

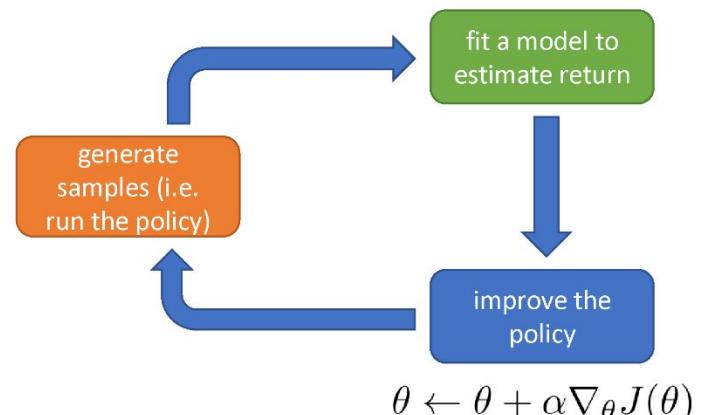
$$A^\pi(\mathbf{s}_t, \mathbf{a}_t) = Q^\pi(\mathbf{s}_t, \mathbf{a}_t) - V^\pi(\mathbf{s}_t): \text{how much better } \mathbf{a}_t \text{ is}$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) A^\pi(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$



$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \underbrace{\left(\sum_{t'=1}^T r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) - b \right)}$$

unbiased, but high variance single-sample estimate



fit Q^π , V^π , or A^π

fit a model to estimate return

generate samples (i.e. run the policy)

improve the policy

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

the better this estimate, the lower the variance

Value function fitting

$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^T E_{\pi_\theta} [r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t, \mathbf{a}_t]$$

$$V^\pi(\mathbf{s}_t) = E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} [Q^\pi(\mathbf{s}_t, \mathbf{a}_t)]$$

$$A^\pi(\mathbf{s}_t, \mathbf{a}_t) = Q^\pi(\mathbf{s}_t, \mathbf{a}_t) - V^\pi(\mathbf{s}_t)$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) A^\pi(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$

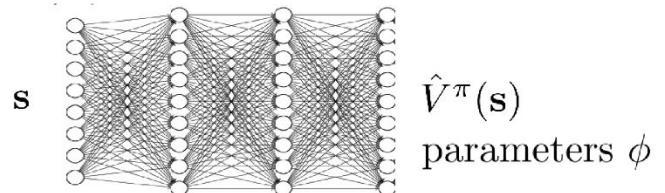
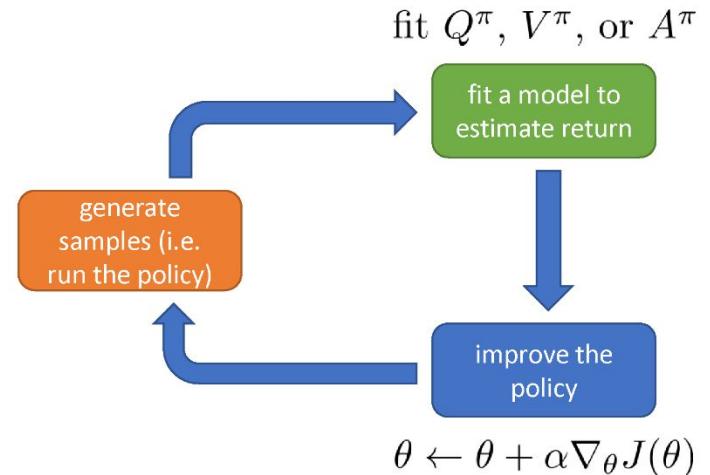
fit what to what?

Q^π, V^π, A^π ?

$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + E_{\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)} [V^\pi(\mathbf{s}_{t+1})]$$

$$A^\pi(\mathbf{s}_t, \mathbf{a}_t) \approx r(\mathbf{s}_t, \mathbf{a}_t) + V^\pi(\mathbf{s}_{t+1}) - \hat{V}^\pi(\mathbf{s}_t)$$

let's just fit $V^\pi(\mathbf{s})$!



Policy evaluation

$$V^\pi(\mathbf{s}_t) = \sum_{t'=t}^T E_{\pi_\theta} [r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t]$$

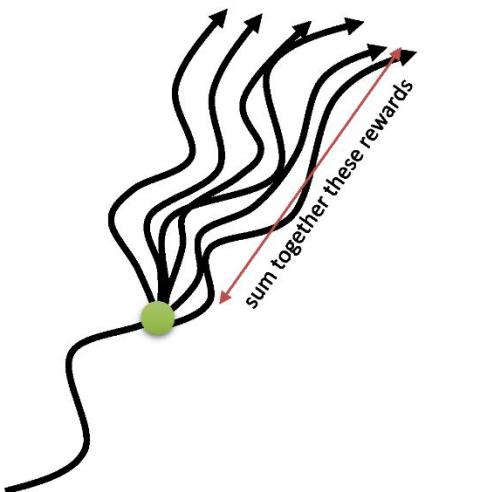
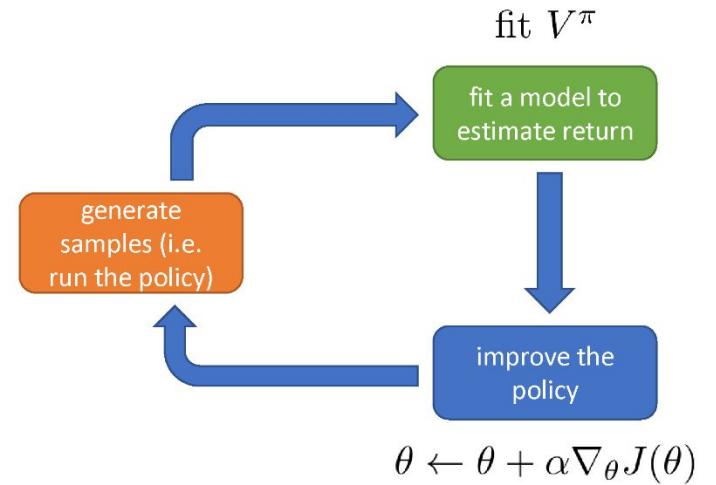
$$J(\theta) = E_{\mathbf{s}_1 \sim p(\mathbf{s}_1)} [V^\pi(\mathbf{s}_1)]$$

how can we perform policy evaluation?

Monte Carlo policy evaluation (this is what policy gradient does)

$$V^\pi(\mathbf{s}_t) \approx \sum_{t'=t}^T r(\mathbf{s}_{t'}, \mathbf{a}_{t'})$$

$$V^\pi(\mathbf{s}_t) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t'=t}^T r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \quad (\text{requires us to reset the simulator})$$



Monte Carlo evaluation with function approximation

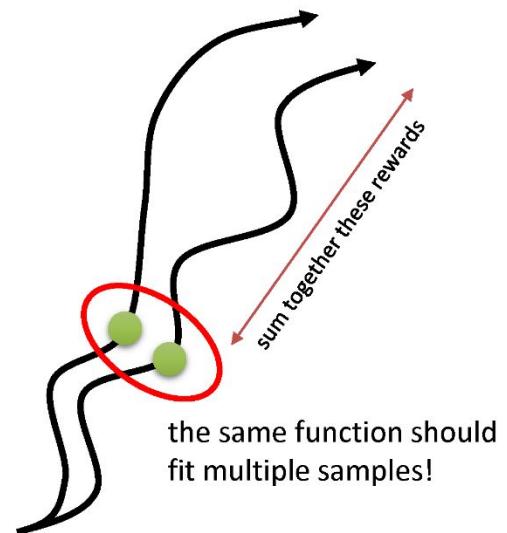
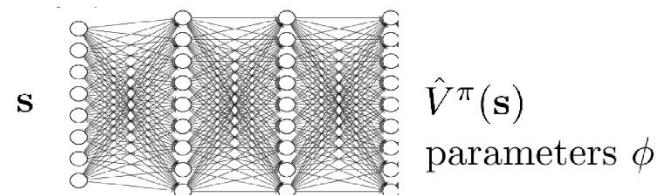
$$V^\pi(\mathbf{s}_t) \approx \sum_{t'=t}^T r(\mathbf{s}_{t'}, \mathbf{a}_{t'})$$

not as good as this: $V^\pi(\mathbf{s}_t) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t'=t}^T r(\mathbf{s}_{t'}, \mathbf{a}_{t'})$

but still pretty good!

training data: $\left\{ \left(\mathbf{s}_{i,t}, \underbrace{\sum_{t'=t}^T r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'})}_{y_{i,t}} \right) \right\}$

supervised regression: $\mathcal{L}(\phi) = \frac{1}{2} \sum_i \left\| \hat{V}_\phi^\pi(\mathbf{s}_i) - y_i \right\|^2$



Can we do better?

ideal target: $y_{i,t} = \sum_{t'=t}^T E_{\pi_\theta} [r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_{i,t}] \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + V^\pi(\mathbf{s}_{i,t+1}) \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \hat{V}_\phi^\pi(\mathbf{s}_{i,t+1})$

Monte Carlo target: $y_{i,t} = \sum_{t'=t}^T r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'})$

directly use previous fitted value function!

training data: $\left\{ \underbrace{\left(\mathbf{s}_{i,t}, r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \hat{V}_\phi^\pi(\mathbf{s}_{i,t+1}) \right)}_{y_{i,t}} \right\}$

supervised regression: $\mathcal{L}(\phi) = \frac{1}{2} \sum_i \left\| \hat{V}_\phi^\pi(\mathbf{s}_i) - y_i \right\|^2$

sometimes referred to as a “bootstrapped” estimate

Policy evaluation examples

TD-Gammon, Gerald Tesauro 1992

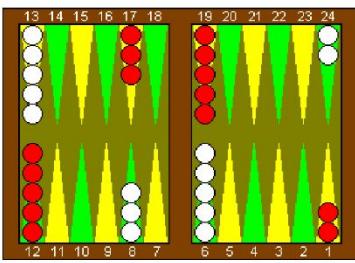


Figure 2. An illustration of the normal opening position in backgammon. TD-Gammon has sparked a near-universal conversion in the way experts play certain opening rolls. For example, with an opening roll of 4-1, most players have now switched from the traditional move of 13-9, 6-5, to TD-Gammon's preference, 13-9, 24-23. TD-Gammon's analysis is given in Table 2.

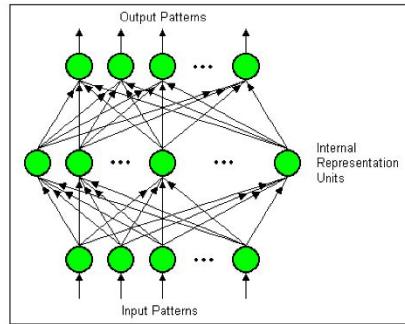


Figure 1. An illustration of the multilayer perception architecture used in TD-Gammon's neural network. This architecture is also used in the popular backpropagation learning procedure. Figure reproduced from [9].

AlphaGo, Silver et al. 2016



reward: game outcome

value function $\hat{V}_\phi^\pi(\mathbf{s}_t)$:

expected outcome given board state

reward: game outcome

value function $\hat{V}_\phi^\pi(\mathbf{s}_t)$:

expected outcome given board state

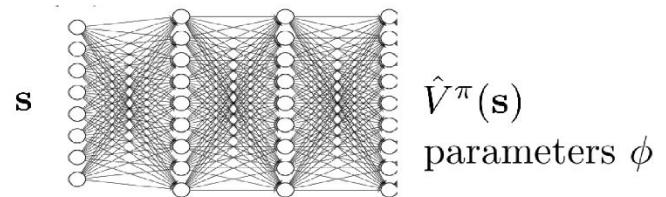
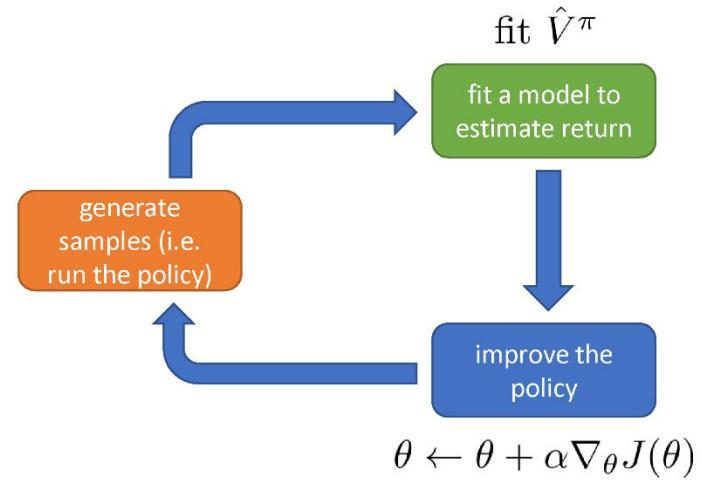
An actor-critic algorithm

batch actor-critic algorithm:

1. sample $\{\mathbf{s}_i, \mathbf{a}_i\}$ from $\pi_\theta(\mathbf{a}|\mathbf{s})$ (run it on the robot)
2. fit $\hat{V}_\phi^\pi(\mathbf{s})$ to sampled reward sums
3. evaluate $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \hat{V}_\phi^\pi(\mathbf{s}'_i) - \hat{V}_\phi^\pi(\mathbf{s}_i)$
4. $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i) \hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i)$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

$$y_{i,t} \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \hat{V}_\phi^\pi(\mathbf{s}_{i,t+1})$$

$$\mathcal{L}(\phi) = \frac{1}{2} \sum_i \left\| \hat{V}_\phi^\pi(\mathbf{s}_i) - y_i \right\|^2$$



$$V^\pi(\mathbf{s}_t) = \sum_{t'=t}^T E_{\pi_\theta} [r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t]$$

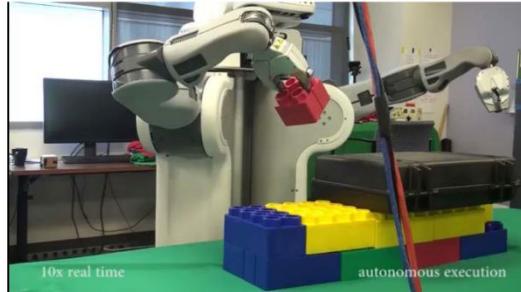
Aside: discount factors

$$y_{i,t} \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \hat{V}_\phi^\pi(\mathbf{s}_{i,t+1})$$

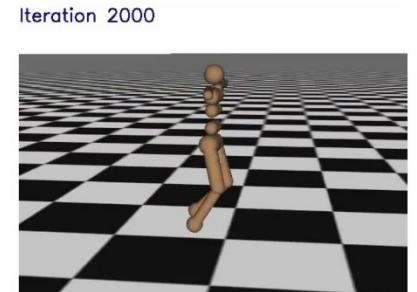
$$\mathcal{L}(\phi) = \frac{1}{2} \sum_i \left\| \hat{V}_\phi^\pi(\mathbf{s}_i) - y_i \right\|^2$$

what if T (episode length) is ∞ ?

\hat{V}_ϕ^π can get infinitely large in many cases



episodic tasks



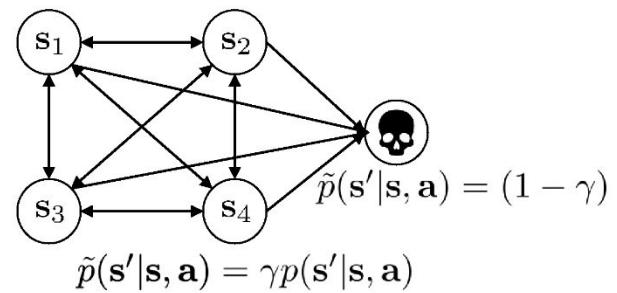
continuous/cyclical tasks

simple trick: better to get rewards sooner than later

$$y_{i,t} \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \gamma \hat{V}_\phi^\pi(\mathbf{s}_{i,t+1})$$

↑
discount factor $\gamma \in [0, 1]$ (0.99 works well)

γ changes the MDP:



Aside: discount factors for policy gradients

$$y_{i,t} \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \gamma \hat{V}_\phi^\pi(\mathbf{s}_{i,t+1})$$

$$\mathcal{L}(\phi) = \frac{1}{2} \sum_i \left\| \hat{V}_\phi^\pi(\mathbf{s}_i) - y_i \right\|^2$$

with critic:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \underbrace{\left(r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \gamma \hat{V}_\phi^\pi(\mathbf{s}_{i,t+1}) - \hat{V}_\phi^\pi(\mathbf{s}_{i,t}) \right)}_{\hat{A}^\pi(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})}$$

what about (Monte Carlo) policy gradients?

option 1: $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \left(\sum_{t'=t}^T \gamma^{t'-t} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)$

option 2: $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left(\sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^T \gamma^{t-1} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$

$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \left(\sum_{t'=t}^T \gamma^{t'-1} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)$

$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \gamma^{t-1} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \left(\sum_{t'=t}^T \gamma^{t'-t} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)$ (later steps matter less)

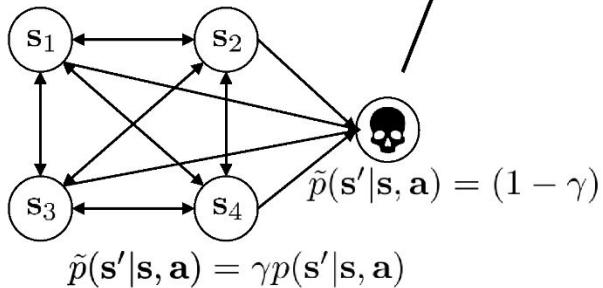
not the same!

Which version is the right one?

$$\text{option 1: } \nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \left(\sum_{t'=t}^T \gamma^{t'-t} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)$$

$$\text{option 2: } \nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \gamma^{t-1} \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \left(\sum_{t'=t}^T \gamma^{t'-t} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)$$

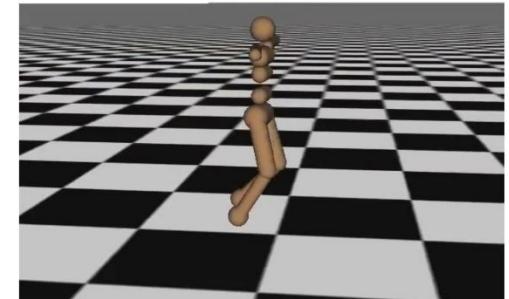
later steps don't matter if you're dead!



this is what we actually use...

why?

Iteration 2000



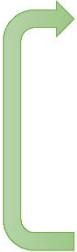
Further reading: Philip Thomas, Bias in natural actor-critic algorithms. ICML 2014

Actor-critic algorithms (with discount)

batch actor-critic algorithm:

- 
1. sample $\{\mathbf{s}_i, \mathbf{a}_i\}$ from $\pi_\theta(\mathbf{a}|\mathbf{s})$ (run it on the robot)
 2. fit $\hat{V}_\phi^\pi(\mathbf{s})$ to sampled reward sums
 3. evaluate $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \hat{V}_\phi^\pi(\mathbf{s}'_i) - \hat{V}_\phi^\pi(\mathbf{s}_i)$
 4. $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i) \hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i)$
 5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

online actor-critic algorithm:

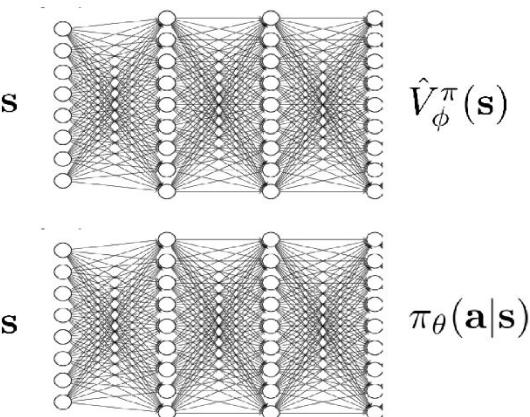
- 
1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
 2. update \hat{V}_ϕ^π using target $r + \gamma \hat{V}_\phi^\pi(\mathbf{s}')$
 3. evaluate $\hat{A}^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \hat{V}_\phi^\pi(\mathbf{s}') - \hat{V}_\phi^\pi(\mathbf{s})$
 4. $\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) \hat{A}^\pi(\mathbf{s}, \mathbf{a})$
 5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Architecture design

online actor-critic algorithm:

1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
2. update \hat{V}_ϕ^π using target $r + \gamma \hat{V}_\phi^\pi(\mathbf{s}')$
3. evaluate $\hat{A}^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \hat{V}_\phi^\pi(\mathbf{s}') - \hat{V}_\phi^\pi(\mathbf{s})$
4. $\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) \hat{A}^\pi(\mathbf{s}, \mathbf{a})$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

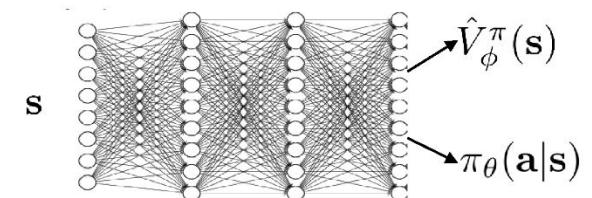
two network design



+ simple & stable

- no shared features between actor & critic

shared network design

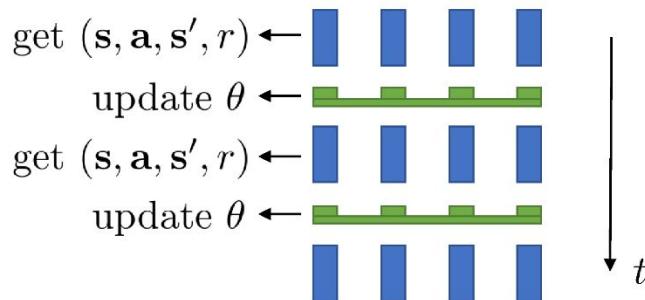


Online actor-critic in practice

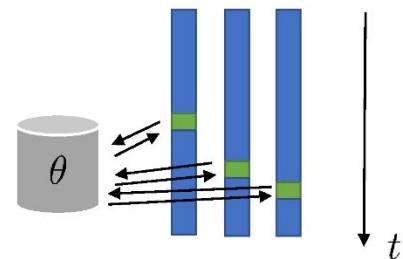
online actor-critic algorithm:

1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
2. update \hat{V}_ϕ^π using target $r + \gamma \hat{V}_\phi^\pi(\mathbf{s}') \leftarrow$ works best with a batch (e.g., parallel workers)
3. evaluate $\hat{A}^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \hat{V}_\phi^\pi(\mathbf{s}') - \hat{V}_\phi^\pi(\mathbf{s})$
4. $\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) \hat{A}^\pi(\mathbf{s}, \mathbf{a}) \leftarrow$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

synchronized parallel actor-critic



asynchronous parallel actor-critic



Critics as state-dependent baselines

Actor-critic:

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \left(r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \gamma \hat{V}_{\phi}^{\pi}(\mathbf{s}_{i,t+1}) - \hat{V}_{\phi}^{\pi}(\mathbf{s}_{i,t}) \right)$$

+ lower variance (due to critic)
- not unbiased (if the critic is not perfect)

Policy gradient:

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \left(\left(\sum_{t'=t}^T \gamma^{t'-t} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right) - b \right)$$

+ no bias
- higher variance (because single-sample estimate)

can we use \hat{V}_{ϕ}^{π} and still keep the estimator unbiased?

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \left(\left(\sum_{t'=t}^T \gamma^{t'-t} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right) - \hat{V}_{\phi}^{\pi}(\mathbf{s}_{i,t}) \right)$$

+ no bias
+ lower variance (baseline is closer to rewards)

Control variates: action-dependent baselines

$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^T E_{\pi_\theta} [r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t, \mathbf{a}_t]$$

$$V^\pi(\mathbf{s}_t) = E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} [Q^\pi(\mathbf{s}_t, \mathbf{a}_t)]$$

$$A^\pi(\mathbf{s}_t, \mathbf{a}_t) = Q^\pi(\mathbf{s}_t, \mathbf{a}_t) - V^\pi(\mathbf{s}_t)$$

$$\hat{A}^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - V_\phi^\pi(\mathbf{s}_t)$$

+ no bias

- higher variance (because single-sample estimate)

$$\hat{A}^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - Q_\phi^\pi(\mathbf{s}_t, \mathbf{a}_t)$$

+ goes to zero in expectation if critic is correct!
- not correct

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \left(\hat{Q}_{i,t} - Q_\phi^\pi(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right) + \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_\theta E_{\mathbf{a} \sim \pi_\theta(\mathbf{a}_t | \mathbf{s}_{i,t})} [Q_\phi^\pi(\mathbf{s}_{i,t}, \mathbf{a}_t)]$$

use a critic *without* the bias (still unbiased), provided second term can be evaluated

Gu et al. 2016 (Q-Prop) – we'll talk more about variance reduction later

Eligibility traces & n-step returns

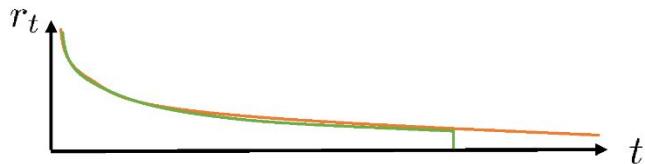
$$\hat{A}_C^\pi(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \hat{V}_\phi^\pi(\mathbf{s}_{t+1}) - \hat{V}_\phi^\pi(\mathbf{s}_t)$$

- + lower variance
- higher bias if value is wrong (it always is)

$$\hat{A}_{MC}^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - \hat{V}_\phi^\pi(\mathbf{s}_t)$$

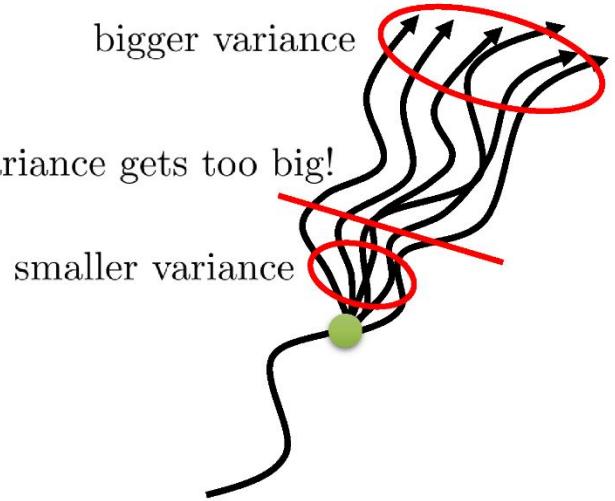
- + no bias
- higher variance (because single-sample estimate)

Can we combine these two, to control bias/variance tradeoff?



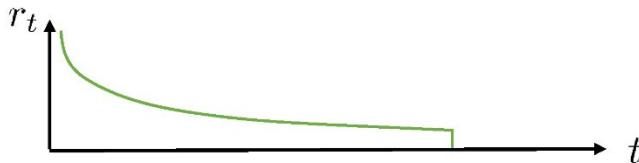
cut here before variance gets too big!

$$\hat{A}_n^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{t+n} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - \hat{V}_\phi^\pi(\mathbf{s}_t) + \gamma^n \hat{V}_\phi^\pi(\mathbf{s}_{t+n})$$



choosing $n > 1$ often works better!

Generalized advantage estimation



Do we have to choose just one n?

Cut everywhere all at once!

$$\hat{A}_n^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{t+n} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - \hat{V}_\phi^\pi(\mathbf{s}_t) + \gamma^n \hat{V}_\phi^\pi(\mathbf{s}_{t+n})$$

$$\hat{A}_{\text{GAE}}^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{n=1}^{\infty} w_n \hat{A}_n^\pi(\mathbf{s}_t, \mathbf{a}_t)$$

Weighted combination of n-step returns

How to weight?

Mostly prefer cutting earlier (less variance)

$$w_n \propto \lambda^{n-1}$$

exponential falloff

$$\hat{A}_{\text{GAE}}^\pi(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma((1-\lambda)\hat{V}_\phi^\pi(\mathbf{s}_{t+1}) + \lambda(r(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) + \gamma((1-\lambda)\hat{V}_\phi^\pi(\mathbf{s}_{t+2}) + \lambda r(\mathbf{s}_{t+2}, \mathbf{a}_{t+2}) + \dots))$$

$$\hat{A}_{\text{GAE}}^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{\infty} (\gamma \lambda)^{t'-t} \delta_{t'}$$

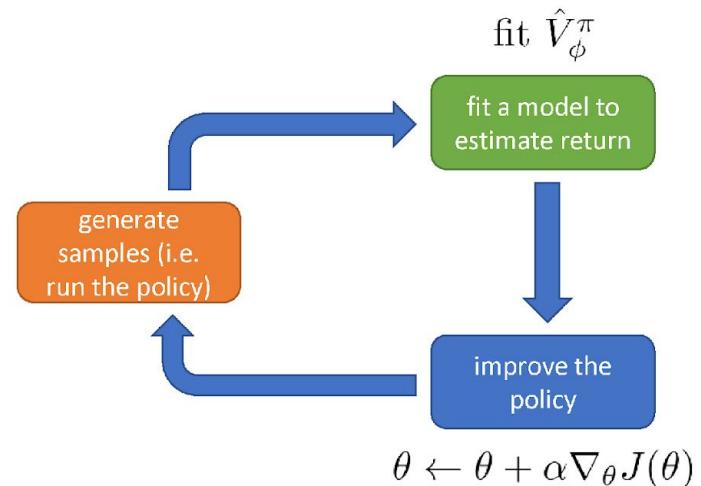
similar effect as discount!

option 1: $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \left(\sum_{t'=t}^T \gamma^{t'-t} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)$

remember this?
discount = variance reduction!
Schulman, Moritz, Levine, Jordan, Abbeel '16

Review

- Actor-critic algorithms:
 - Actor: the policy
 - Critic: value function
 - Reduce variance of policy gradient
- Policy evaluation
 - Fitting value function to policy
- Discount factors
 - Carpe diem Mr. Robot 
 - ...but also a variance reduction trick
- Actor-critic algorithm design
 - One network (with two heads) or two networks
 - Batch-mode, or online (+ parallel)
- State-dependent baselines
 - Another way to use the critic
 - Can combine: n-step returns or GAE



Policy Gradients Review

Policy gradient algorithms try to solve the optimization problem

$$\max_{\theta} J(\pi_{\theta}) \doteq \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]$$

by taking stochastic gradient ascent on the policy parameters θ , using the *policy gradient*

$$g = \nabla_{\theta} J(\pi_{\theta}) = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^{\infty} \gamma^t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) A^{\pi_{\theta}}(s_t, a_t) \right].$$

Limitations of policy gradients:

- Sample efficiency is poor
 - Because recycling old data to estimate policy gradients is hard
- Distance in parameter space \neq distance in policy space!
 - What is policy space? For tabular case, set of matrices

$$\Pi = \left\{ \pi : \pi \in \mathbb{R}^{|S| \times |A|}, \sum_a \pi_{sa} = 1, \pi_{sa} \geq 0 \right\}$$

- Policy gradients take steps in parameter space
- Step size is hard to get right as a result

Sample Efficiency in Policy Gradients

- Sample efficiency for policy gradient methods is pretty poor
- We throw out each batch of data immediately after **just one gradient step**
- Why? PG is an **on-policy expectation**. There are two main ways of estimating it:¹
 - Run policy in environment and collect sample trajectories, then form sample estimate. (More stable)
 - Use trajectories from other policies with **importance sampling**. (Less stable)

¹In an unbiased way.

Importance Sampling Review

Importance sampling is a technique for estimating expectations using samples drawn from a different distribution.

$$\mathbb{E}_{x \sim P}[f(x)] = \mathbb{E}_{x \sim Q}\left[\frac{P(x)}{Q(x)}f(x)\right] \approx \frac{1}{|D|} \sum_{x \in D} \frac{P(x)}{Q(x)}f(x), \quad D \sim Q$$

The ratio $P(x)/Q(x)$ is the **importance sampling weight** for x .

What is the variance of an importance sampling estimator?

$$\begin{aligned}\text{var}(\hat{\mu}_Q) &= \frac{1}{N} \text{var}\left(\frac{P(x)}{Q(x)}f(x)\right) \\ &= \frac{1}{N} \left(\mathbb{E}_{x \sim Q}\left[\left(\frac{P(x)}{Q(x)}f(x)\right)^2\right] - \mathbb{E}_{x \sim Q}\left[\frac{P(x)}{Q(x)}f(x)\right]^2 \right) \\ &= \frac{1}{N} \left(\mathbb{E}_{x \sim P}\left[\frac{P(x)}{Q(x)}f(x)^2\right] - \mathbb{E}_{x \sim P}[f(x)]^2 \right)\end{aligned}$$

The term in red is problematic—if $P(x)/Q(x)$ is large in the wrong places, the variance of the estimator explodes.

Importance Sampling for Policy Gradients

Here, we compress the notation π_θ down to θ in some places for compactness.

$$\begin{aligned} g = \nabla_\theta J(\theta) &= \mathbb{E}_{\tau \sim \theta} \left[\sum_{t=0}^{\infty} \gamma^t \nabla_\theta \log \pi_\theta(a_t | s_t) A^\theta(s_t, a_t) \right] \\ &= \mathbb{E}_{\tau \sim \theta'} \left[\sum_{t=0}^{\infty} \frac{P(\tau_t | \theta)}{P(\tau_t | \theta')} \gamma^t \nabla_\theta \log \pi_\theta(a_t | s_t) A^\theta(s_t, a_t) \right] \end{aligned}$$

Looks useful—what's the issue? **Exploding or vanishing importance sampling weights.**

$$\frac{P(\tau_t | \theta)}{P(\tau_t | \theta')} = \frac{\mu(s_0) \prod_{t'=0}^t P(s_{t'+1} | s_{t'}, a_{t'}) \pi_\theta(a_{t'} | s_{t'})}{\mu(s_0) \prod_{t'=0}^t P(s_{t'+1} | s_{t'}, a_{t'}) \pi_{\theta'}(a_{t'} | s_{t'})} = \prod_{t'=0}^t \frac{\pi_\theta(a_{t'} | s_{t'})}{\pi_{\theta'}(a_{t'} | s_{t'})}$$

Even for policies only slightly different from each other, **many small differences multiply to become a big difference.**

Big question: how can we make efficient use of the data we already have from the old policy, while avoiding the challenges posed by importance sampling?

Choosing a Step Size for Policy Gradients

Policy gradient algorithms are stochastic gradient ascent:

$$\theta_{k+1} = \theta_k + \alpha_k \hat{g}_k$$

with step $\Delta_k = \alpha_k \hat{g}_k$.

- If the step is too large, **performance collapse** is possible
- If the step is too small, progress is unacceptably slow
- “Right” step size changes based on θ

Automatic learning rate adjustment like advantage normalization, or Adam-style optimizers, can help. But does this solve the problem?

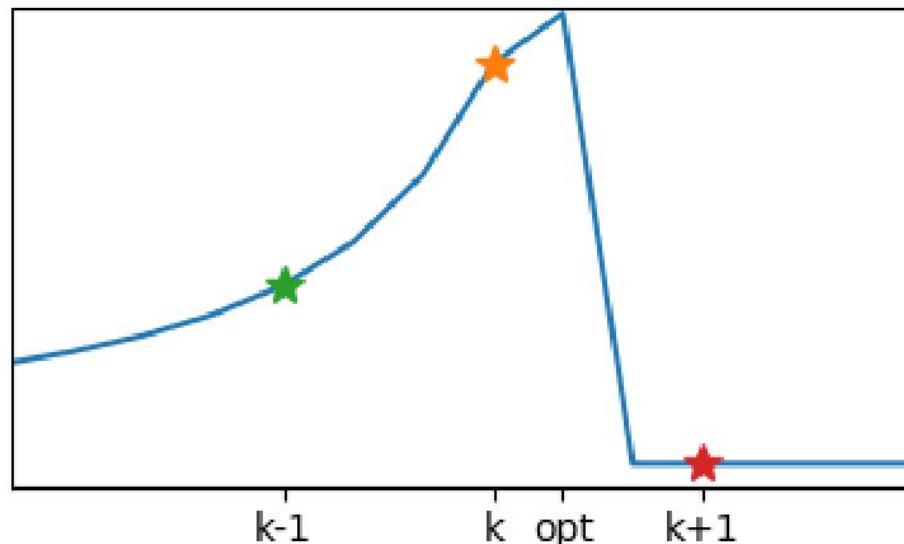


Figure: Policy parameters on x-axis and performance on y-axis. A bad step can lead to performance collapse, which may be hard to recover from.

The Problem is More Than Step Size

Consider a family of policies with parametrization:

$$\pi_\theta(a) = \begin{cases} \sigma(\theta) & a = 1 \\ 1 - \sigma(\theta) & a = 2 \end{cases}$$

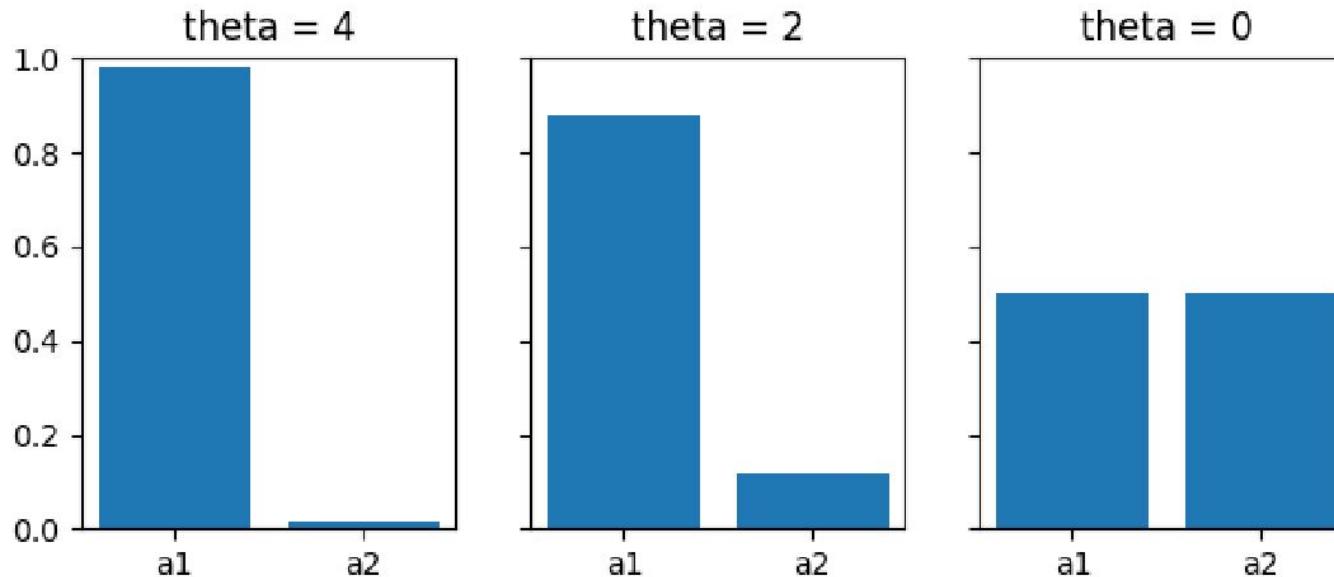


Figure: Small changes in the policy parameters can unexpectedly lead to **big** changes in the policy.

Big question: how do we come up with an update rule that doesn't ever change the policy more than we meant to?

Relative Performance of Two Policies

In a policy optimization algorithm, we want an update step that

- uses rollouts collected from the most recent policy as efficiently as possible,
- and takes steps that respect **distance in policy space** as opposed to distance in parameter space.

To figure out the right update rule, we need to exploit relationships between the performance of two policies.

Relative policy performance identity: for any policies π, π'

$$J(\pi') - J(\pi) = \mathbb{E}_{\tau \sim \pi'} \left[\sum_{t=0}^{\infty} \gamma^t A^\pi(s_t, a_t) \right] \quad (1)$$

Proof of Relative Policy Performance Identity

$$\begin{aligned} J(\pi') - J(\pi) &= \mathbb{E}_{\tau \sim \pi'} \left[\sum_{t=0}^{\infty} \gamma^t A^\pi(s_t, a_t) \right] \\ &= \mathbb{E}_{\tau \sim \pi'} \left[\sum_{t=0}^{\infty} \gamma^t (R(s_t, a_t, s_{t+1}) + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)) \right] \\ &= J(\pi') + \mathbb{E}_{\tau \sim \pi'} \left[\sum_{t=0}^{\infty} \gamma^{t+1} V^\pi(s_{t+1}) - \sum_{t=0}^{\infty} \gamma^t V^\pi(s_t) \right] \\ &= J(\pi') + \mathbb{E}_{\tau \sim \pi'} \left[\sum_{t=1}^{\infty} \gamma^t V^\pi(s_t) - \sum_{t=0}^{\infty} \gamma^t V^\pi(s_t) \right] \\ &= J(\pi') - \mathbb{E}_{\tau \sim \pi'} [V^\pi(s_0)] \\ &= J(\pi') - J(\pi) \end{aligned}$$

What is it good for?

Can we use this for policy improvement, where π' represents the new policy and π represents the old one?

$$\begin{aligned}\max_{\pi'} J(\pi') &= \max_{\pi'} J(\pi') - J(\pi) \\ &= \max_{\pi'} \mathbb{E}_{\tau \sim \pi'} \left[\sum_{t=0}^{\infty} \gamma^t A^\pi(s_t, a_t) \right]\end{aligned}$$

This is suggestive, but not useful yet.

Nice feature of this optimization problem: defines the performance of π' in terms of the advantages from π !

But, problematic feature: still requires trajectories sampled from π' ...

Looking at it from another angle...

In terms of the **discounted future state distribution** d^π , defined by

$$d^\pi(s) = (1 - \gamma) \sum_{t=0}^{\infty} \gamma^t P(s_t = s | \pi),$$

we can rewrite the relative policy performance identity:

$$\begin{aligned} J(\pi') - J(\pi) &= \mathbb{E}_{\tau \sim \pi'} \left[\sum_{t=0}^{\infty} \gamma^t A^\pi(s_t, a_t) \right] \\ &= \frac{1}{1 - \gamma} \mathbb{E}_{\substack{s \sim d^{\pi'} \\ a \sim \pi'}} [A^\pi(s, a)] \\ &= \frac{1}{1 - \gamma} \mathbb{E}_{\substack{s \sim d^{\pi'} \\ a \sim \pi}} \left[\frac{\pi'(a|s)}{\pi(a|s)} A^\pi(s, a) \right] \end{aligned}$$

...almost there! Only problem is $s \sim d^{\pi'}$.

A Useful Approximation

What if we just said $d^{\pi'} \approx d^\pi$ and didn't worry about it?

$$\begin{aligned} J(\pi') - J(\pi) &\approx \frac{1}{1 - \gamma} \mathbb{E}_{\substack{s \sim d^\pi \\ a \sim \pi}} \left[\frac{\pi'(a|s)}{\pi(a|s)} A^\pi(s, a) \right] \\ &\doteq \mathcal{L}_\pi(\pi') \end{aligned}$$

Turns out: this approximation is pretty good when π' and π are close! But why, and how close do they have to be?

Relative policy performance bounds: ²

$$|J(\pi') - (J(\pi) + \mathcal{L}_\pi(\pi'))| \leq C \sqrt{\mathbb{E}_{s \sim d^\pi} [D_{KL}(\pi' || \pi)[s]]} \quad (2)$$

If policies are close in KL-divergence—the approximation is good!

²Achiam, Held, Tamar, Abbeel, 2017

What is KL-divergence?

For probability distributions P and Q over a discrete random variable,

$$D_{KL}(P||Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}$$

Properties:

- $D_{KL}(P||P) = 0$
- $D_{KL}(P||Q) \geq 0$
- $D_{KL}(P||Q) \neq D_{KL}(Q||P)$ — Non-symmetric!

What is KL-divergence between policies?

$$D_{KL}(\pi'||\pi)[s] = \sum_{a \in \mathcal{A}} \pi'(a|s) \log \frac{\pi'(a|s)}{\pi(a|s)}$$

A Useful Approximation

What did we gain from making that approximation?

$$J(\pi') - J(\pi) \approx \mathcal{L}_\pi(\pi')$$

$$\begin{aligned}\mathcal{L}_\pi(\pi') &= \frac{1}{1-\gamma} \underset{\substack{s \sim d^\pi \\ a \sim \pi}}{\mathbb{E}} \left[\frac{\pi'(a|s)}{\pi(a|s)} A^\pi(s, a) \right] \\ &= \underset{\tau \sim \pi}{\mathbb{E}} \left[\sum_{t=0}^{\infty} \gamma^t \frac{\pi'(a_t|s_t)}{\pi(a_t|s_t)} A^\pi(s_t, a_t) \right]\end{aligned}$$

- This is something we can optimize using trajectories sampled from the old policy π !
- Similar to using importance sampling, but because weights only depend on current timestep (and not preceding history), they don't vanish or explode.

Something else cool—the approximation matches $J(\pi_\theta) - J(\pi_{\theta_k})$ to first order in policy parameters! That is, $\nabla_\theta \mathcal{L}_{\theta_k}(\theta)|_{\theta_k}$ is equal to policy gradient:

$$\begin{aligned}\nabla_\theta \mathcal{L}_{\theta_k}(\theta)|_{\theta_k} &= \underset{\tau \sim \pi_{\theta_k}}{\mathbb{E}} \left[\sum_{t=0}^{\infty} \gamma^t \frac{\nabla_\theta \pi_\theta(a_t|s_t)|_{\theta_k}}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t) \right] \\ &= \underset{\tau \sim \pi_{\theta_k}}{\mathbb{E}} \left[\sum_{t=0}^{\infty} \gamma^t \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} A^{\pi_{\theta_k}}(s_t, a_t) \right]\end{aligned}$$

Monotonic Improvement Theory

From the bound on the previous slide, we get

$$J(\pi') - J(\pi) \geq \mathcal{L}_\pi(\pi') - C \sqrt{\mathbb{E}_{s \sim d^\pi} [D_{KL}(\pi' || \pi)[s]]}.$$

- Cool: If we maximize the RHS with respect to π' , we are **guaranteed to improve over π** .
 - This is a *majorize-maximize* algorithm w.r.t. the true objective, the LHS.
- Cooler: $\mathcal{L}_\pi(\pi')$ and the KL-divergence term *can both be estimated with samples from π !*

Monotonic Improvement Theory

Proof of improvement guarantee: Suppose π_{k+1} and π_k are related by

$$\pi_{k+1} = \arg \max_{\pi'} \mathcal{L}_{\pi_k}(\pi') - C \sqrt{\mathbb{E}_{s \sim d^{\pi_k}} [D_{KL}(\pi' || \pi_k)[s]]}.$$

- π_k is a feasible point, and the objective at π_k is equal to 0.
 - $\mathcal{L}_{\pi_k}(\pi_k) \propto \mathbb{E}_{s, a \sim d^{\pi_k}, \pi_k} [A^{\pi_k}(s, a)] = 0$
 - $D_{KL}(\pi_k || \pi_k)[s] = 0$
- \implies optimal value ≥ 0
- \implies by the performance bound, $J(\pi_{k+1}) - J(\pi_k) \geq 0$

Approximate Monotonic Improvement

$$\pi_{k+1} = \arg \max_{\pi'} \mathcal{L}_{\pi_k}(\pi') - C \sqrt{\mathbb{E}_{s \sim d^{\pi_k}} [D_{KL}(\pi' || \pi_k)[s]]}. \quad (3)$$

Problem:

- C provided by theory is quite high when γ is near 1
- \Rightarrow steps from (3) are too small.

Solution:

- Instead of KL penalty, use KL constraint (called **trust region**).
- Can control worst-case error through constraint upper limit!

$$\begin{aligned} \pi_{k+1} &= \arg \max_{\pi'} \mathcal{L}_{\pi_k}(\pi') \\ \text{s.t. } &\mathbb{E}_{s \sim d^{\pi_k}} [D_{KL}(\pi' || \pi_k)[s]] \leq \delta \end{aligned} \quad (4)$$

Approximate Monotonic Improvement

$$\begin{aligned}\pi_{k+1} = \arg \max_{\pi'} \mathcal{L}_{\pi_k}(\pi') \\ \text{s.t. } \mathbb{E}_{s \sim d^{\pi_k}} [D_{KL}(\pi' || \pi_k)[s]] \leq \delta\end{aligned}\tag{4}$$

This policy optimization step satisfies the two conditions we wanted:

- The objective and constraint can be estimated using rollouts from the most recent policy—efficient!
- From the constraint, **steps respect (a notion of) distance in policy space!**
Update is parametrization-invariant.

As a result: the basis of many algorithms!

Monotonic Improvement Theory

From the bound on the previous slide, we get

$$J(\pi') - J(\pi) \geq \mathcal{L}_\pi(\pi') - C \sqrt{\mathbb{E}_{s \sim d^\pi} [D_{KL}(\pi' || \pi)[s]]}.$$

- Cool: If we maximize the RHS with respect to π' , we are **guaranteed to improve over π** .
 - This is a *majorize-maximize* algorithm w.r.t. the true objective, the LHS.
- Cooler: $\mathcal{L}_\pi(\pi')$ and the KL-divergence term *can both be estimated with samples from π !*

Natural Policy Gradient

So we have this nice optimization problem:

$$\begin{aligned}\pi_{k+1} &= \arg \max_{\pi'} \mathcal{L}_{\pi_k}(\pi') \\ \text{s.t. } \bar{D}_{KL}(\pi' || \pi_k) &\leq \delta\end{aligned}\tag{4}$$

but how do we solve it? Solution: approximately!

$$\begin{aligned}\mathcal{L}_{\theta_k}(\theta) &\approx \mathcal{L}_{\theta_k}(\theta_k) + g^T (\theta - \theta_k) & g &\doteq \nabla_{\theta} \mathcal{L}_{\theta_k}(\theta) |_{\theta_k} \\ \bar{D}_{KL}(\theta || \theta_k) &\approx \frac{1}{2} (\theta - \theta_k)^T H (\theta - \theta_k) & H &\doteq \nabla_{\theta}^2 \bar{D}_{KL}(\theta || \theta_k) |_{\theta_k}\end{aligned}$$

Note: zeroth and first-order terms for \bar{D}_{KL} are zero at θ_k .

$$\begin{aligned}\theta_{k+1} &= \arg \max_{\theta} g^T (\theta - \theta_k) \\ \text{s.t. } \frac{1}{2} (\theta - \theta_k)^T H (\theta - \theta_k) &\leq \delta\end{aligned}$$

Solution to approximate problem:

$$\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g$$

Properties of Natural Policy Gradient

- Recall that $\nabla_{\theta} \mathcal{L}_{\theta_k}(\theta)|_{\theta_k}$ is equal to the policy gradient—so this update gives a policy gradient algorithm where we pre-multiply by H^{-1} .
- The KL-divergence Hessian H is equal to a special matrix called the **Fisher information matrix**, which comes up in a few other places:

$$H = \underset{s, a \sim \theta^k}{\text{E}} \left[\nabla_{\theta} \log \pi_{\theta}(a|s)|_{\theta_k} \nabla_{\theta} \log \pi_{\theta}(a|s)|_{\theta_k}^T \right]$$

- The NPG direction $H^{-1}g$ is **covariant**—that is, **it points in the same direction regardless of the parametrization used to compute it.**

Covariance of Natural Policy Gradient

What does it mean for something to be **covariant**?

- In a Riemannian space, the distance between points v and $v + \delta v$ is given by

$$\text{dist}^2(v, v + \delta v) = \delta v^T G(v) \delta v$$

where G is the **metric tensor**. (Note: G depends on where in the space you are!)

- A (true, mathematical) tensor is **more than just a matrix**. It has components (like a matrix) but they depend on the coordinates in which you express the space.
- Example:
 - Euclidean 2-space \mathbb{R}^2 can be expressed in Cartesian (x, y) , or polar coordinates (r, θ) .
 - For Cartesian coordinates, the metric tensor is just the identity.
 - For polar coordinates, the metric tensor is $\text{diag}(1, r^2)$:

$$x = r \cos \theta \implies \delta x = \cos \theta \delta r - r \sin \theta \delta \theta$$

$$y = r \sin \theta \implies \delta y = \sin \theta \delta r + r \cos \theta \delta \theta$$

$$\begin{aligned}\text{dist}^2(v, v + \delta v) &= \delta x^2 + \delta y^2 \\ &= (\cos^2 \theta \delta r^2 + r^2 \sin^2 \theta \delta \theta^2 - 2r \sin \theta \cos \theta \delta r \delta \theta) \\ &\quad + (\sin^2 \theta \delta r^2 + r^2 \cos^2 \theta \delta \theta^2 + 2r \sin \theta \cos \theta \delta r \delta \theta) \\ &= \delta r^2 + r^2 \delta \theta^2 \\ &= (\delta r, \delta \theta)^T \text{diag}(1, r^2) (\delta r, \delta \theta)\end{aligned}$$

Covariance of Natural Policy Gradient

Consider the same vector and vector difference in two coordinate systems:

- In system 1 (v), we write $(v, \delta v)$, and the metric tensor is written as G_v
- In system 2 (w), we write $(w, \delta w)$, and the metric tensor is written as G_w

Note: $v = w$, but we are just **writing the same vector with different parametrization**. Because the deltas are also equal ($\delta v = \delta w$), their components are related by:

$$\delta v_i = \sum_j \frac{\partial v_i}{\partial w_j} \delta w_j \implies \delta v = A^T \delta w, \text{ where } A_{ji} = \frac{\partial v_i}{\partial w_j}$$

The distances must be the same in both, so metrics are related as follows:

$$\begin{aligned} \text{dist}^2(v, v + \delta v) &= \text{dist}^2(w, w + \delta w) \\ \text{dist}^2(v, v + \delta v) &= \delta v^T G_v \delta v = \delta w^T A G_v A^T \delta w \\ \text{dist}^2(w, w + \delta w) &= \delta w^T G_w \delta w \\ &\implies G_w = A G_v A^T \end{aligned}$$

Gradients are related by chain rule:

$$[g_w]_j = \frac{\partial f}{\partial w_j} = \sum_i \frac{\partial v_i}{\partial w_j} \frac{\partial f}{\partial v_i} \implies g_w = A g_v$$

Covariance of Natural Policy Gradient

Consider $\Delta_v = G_v^{-1}g_v$, and $\Delta_w = G_w^{-1}g_w$. Are these the same vector in different coordinates?

If they are, from $A^T \delta w = \delta v$, they will satisfy $A^T \Delta_w = \Delta_v$.

$$\begin{aligned}\Delta_w &= G_w^{-1}g_w \\ &= (AG_v A^T)^{-1}Ag_v \\ &= (A^T)^{-1}G_v^{-1}A^{-1}Ag_v \\ &= (A^T)^{-1}G_v^{-1}g_v \\ \therefore A^T \Delta_w &= \Delta_v\end{aligned}$$

They are indeed the same vector!

The punchline: the FIM, H , in the natural policy gradient, **is the metric tensor for policy space.**⁶

Thus the natural policy gradient $H^{-1}g$ is **invariant to parametrization**, as shown above.

⁶Peters, Vijayakumar, Schaal, 2005.

Natural Policy Gradient

Algorithm 1 Natural Policy Gradient

Input: initial policy parameters θ_0

for $k = 0, 1, 2, \dots$ **do**

 Collect set of trajectories \mathcal{D}_k on policy $\pi_k = \pi(\theta_k)$

 Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm

 Form sample estimates for

- policy gradient \hat{g}_k (using advantage estimates)
- and KL-divergence Hessian / Fisher Information Matrix \hat{H}_k

 Compute Natural Policy Gradient update:

$$\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{\hat{g}_k^T \hat{H}_k^{-1} \hat{g}_k}} \hat{H}_k^{-1} \hat{g}_k$$

end for

Truncated Natural Policy Gradient

Problem: for neural networks, number of parameters N is **large—thousands or millions**. Hessian has size N^2 (expensive to store) and matrix inversion complexity is $\mathcal{O}(N^3)$

Solution: use the **conjugate gradient (CG) algorithm** to compute $H^{-1}g$ without inverting H .

- With j iterations, CG solves systems of linear equations $Hx = g$ for x by finding projection onto Krylov subspace, $\text{span}\{g, Hg, H^2g, \dots, H^{j-1}g\}$
- For CG, only matrix-vector product function $f(v) = Hv$ is necessary—and this, we can do:⁷

```
kl = ... # define KL divergence as function of vars theta
v = tf.placeholder(dtype=tf.float32, shape=[N])
kl_gradient = tf.gradients(kl, theta)
kl_gradient_vector_product = tf.sum( kl_gradient * v )
kl_hessian_vector_product = tf.gradients(kl_gradient_vector_product, theta)
```

Natural Policy Gradient with fixed-iteration CG as inner loop is called **Truncated Natural Policy Gradient (TNPG)**

See Wu et al. 2017 (ACKTR algorithm) for an alternate solution to this problem

⁷ Wright and Nocedal, Numerical Optimization, 1999

Trust Region Policy Optimization

Small problems with NPG update:

- Might not be robust to trust region size δ ; at some iterations δ may be too large and performance can degrade
- Because of quadratic approximation, KL-divergence constraint may be violated

Solution:

- Require improvement in surrogate (make sure that $\mathcal{L}_{\theta_k}(\theta_{k+1}) \geq 0$)
- Enforce KL-constraint

How? Backtracking line search with exponential decay (decay coeff $\alpha \in (0, 1)$, budget L)

Algorithm 2 Line Search for TRPO

Compute proposed policy step $\Delta_k = \sqrt{\frac{2\delta}{\hat{g}_k^T \hat{H}_k^{-1} \hat{g}_k}} \hat{H}_k^{-1} \hat{g}_k$

for $j = 0, 1, 2, \dots, L$ **do**

 Compute proposed update $\theta = \theta_k + \alpha^j \Delta_k$

if $\mathcal{L}_{\theta_k}(\theta) \geq 0$ and $\bar{D}_{KL}(\theta || \theta_k) \leq \delta$ **then**

 accept the update and set $\theta_{k+1} = \theta_k + \alpha^j \Delta_k$

 break

end if

end for

Trust Region Policy Optimization

Trust Region Policy Optimization is implemented as TNPG plus a line search. Putting it all together:

Algorithm 3 Trust Region Policy Optimization

Input: initial policy parameters θ_0

for $k = 0, 1, 2, \dots$ **do**

 Collect set of trajectories \mathcal{D}_k on policy $\pi_k = \pi(\theta_k)$

 Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm

 Form sample estimates for

- policy gradient \hat{g}_k (using advantage estimates)
- and KL-divergence Hessian-vector product function $f(v) = \hat{H}_k v$

 Use CG with n_{cg} iterations to obtain $x_k \approx \hat{H}_k^{-1} \hat{g}_k$

 Estimate proposed step $\Delta_k \approx \sqrt{\frac{2\delta}{x_k^T \hat{H}_k x_k}} x_k$

 Perform backtracking line search with exponential decay to obtain final update

$$\theta_{k+1} = \theta_k + \alpha^j \Delta_k$$

end for

Performance for TNPG/TRPO

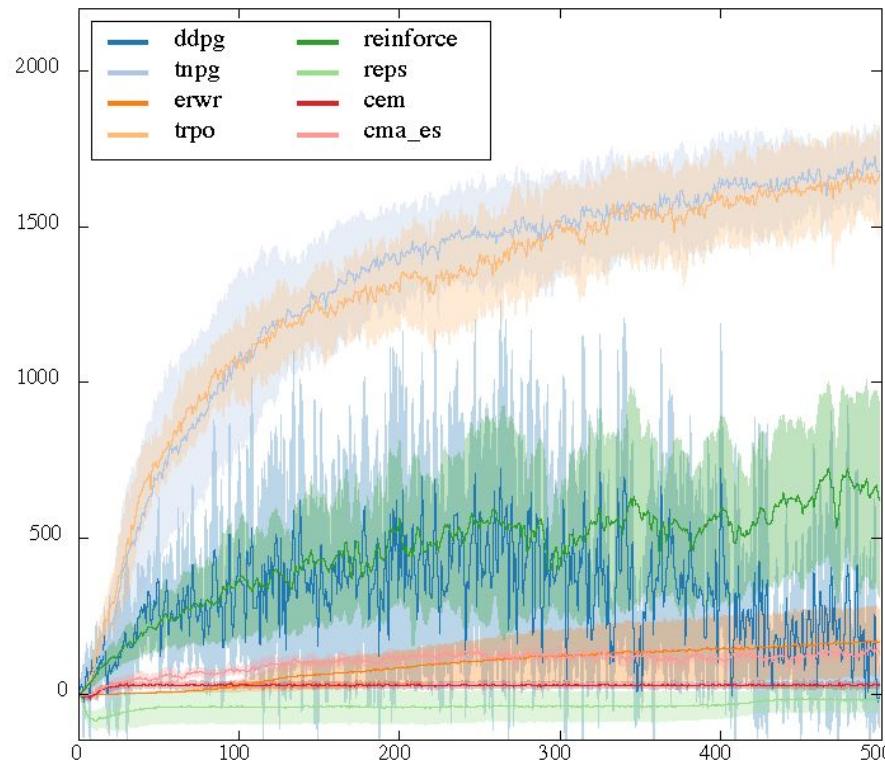


Figure: Comparison between various methods for deep RL including TNPG and TRPO on Walker-2d task. Showing average scores over 5 seeds for each method.⁸

⁸Duan, Chen, Houthooft, Schulman, Abbeel, 2016

Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a family of methods that approximately enforce KL constraint **without computing natural gradients**. Two variants:

- Adaptive KL Penalty

- Policy update solves unconstrained optimization problem

$$\theta_{k+1} = \arg \max_{\theta} \mathcal{L}_{\theta_k}(\theta) - \beta_k \bar{D}_{KL}(\theta || \theta_k)$$

- Penalty coefficient β_k changes between iterations to approximately enforce KL-divergence constraint

- Clipped Objective

- New objective function: let $r_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_k}(a_t|s_t)$. Then

$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \mathbb{E}_{\tau \sim \pi_k} \left[\sum_{t=0}^T \left[\min(r_t(\theta) \hat{A}_t^{\pi_k}, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) \hat{A}_t^{\pi_k}) \right] \right]$$

- where ϵ is a hyperparameter (maybe $\epsilon = 0.2$)

- Policy update is $\theta_{k+1} = \arg \max_{\theta} \mathcal{L}_{\theta_k}^{CLIP}(\theta)$

Proximal Policy Optimization with Adaptive KL-penalty

Algorithm 4 PPO with Adaptive KL Penalty

Input: initial policy parameters θ_0 , initial KL penalty β_0 , target KL-divergence δ

for $k = 0, 1, 2, \dots$ **do**

 Collect set of partial trajectories \mathcal{D}_k on policy $\pi_k = \pi(\theta_k)$

 Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm

 Compute policy update

$$\theta_{k+1} = \arg \max_{\theta} \mathcal{L}_{\theta_k}(\theta) - \beta_k \bar{D}_{KL}(\theta || \theta_k)$$

 by taking K steps of minibatch SGD (via Adam)

if $\bar{D}_{KL}(\theta_{k+1} || \theta_k) \geq 1.5\delta$ **then**

$$\beta_{k+1} = 2\beta_k$$

else if $\bar{D}_{KL}(\theta_{k+1} || \theta_k) \leq \delta/1.5$ **then**

$$\beta_{k+1} = \beta_k/2$$

end if

end for

- Initial KL penalty not that important—it adapts quickly
- Some iterations may violate KL constraint, but most don't

Proximal Policy Optimization with Clipped Objective

Algorithm 5 PPO with Clipped Objective

Input: initial policy parameters θ_0 , clipping threshold ϵ

for $k = 0, 1, 2, \dots$ **do**

 Collect set of partial trajectories \mathcal{D}_k on policy $\pi_k = \pi(\theta_k)$

 Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm

 Compute policy update

$$\theta_{k+1} = \arg \max_{\theta} \mathcal{L}_{\theta_k}^{CLIP}(\theta)$$

 by taking K steps of minibatch SGD (via Adam), where

$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \mathbb{E}_{\tau \sim \pi_k} \left[\sum_{t=0}^T \left[\min(r_t(\theta) \hat{A}_t^{\pi_k}, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t^{\pi_k}) \right] \right]$$

end for

- Clipping prevents policy from having incentive to go far away from θ_{k+1}
- Clipping seems to work at least as well as PPO with KL penalty, but is simpler to implement

Proximal Policy Optimization with Clipped Objective

But *how* does clipping keep policy close? By making objective as pessimistic as possible about performance far away from θ_k :

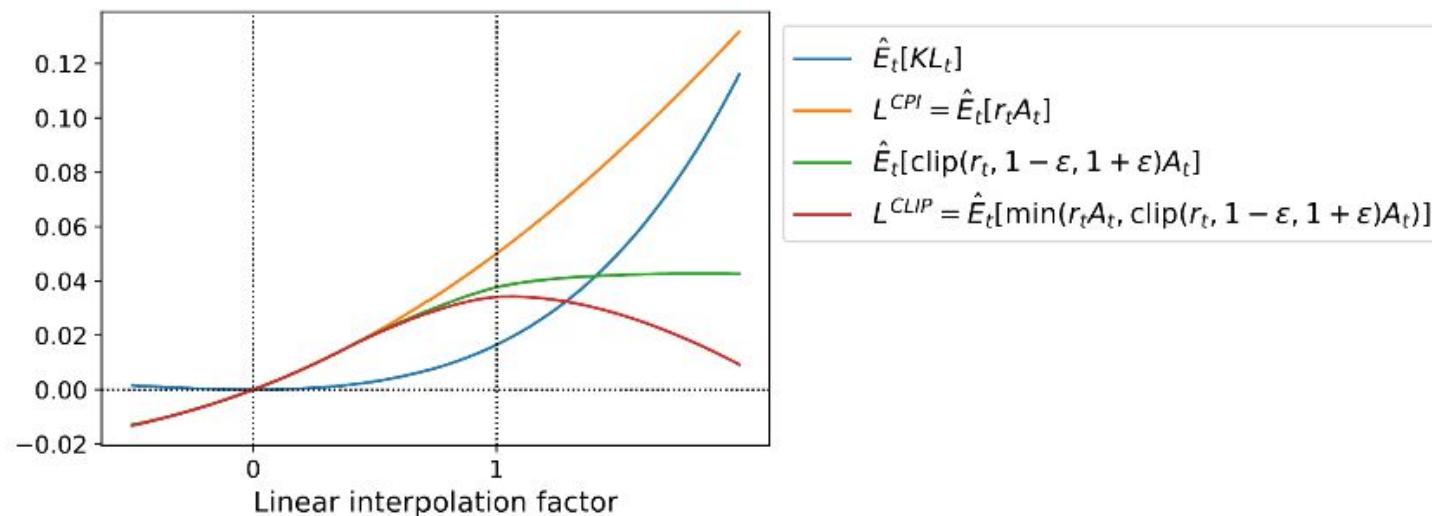


Figure: Various objectives as a function of interpolation factor α between θ_{k+1} and θ_k after one update of PPO-Clip⁹

Performance for PPO

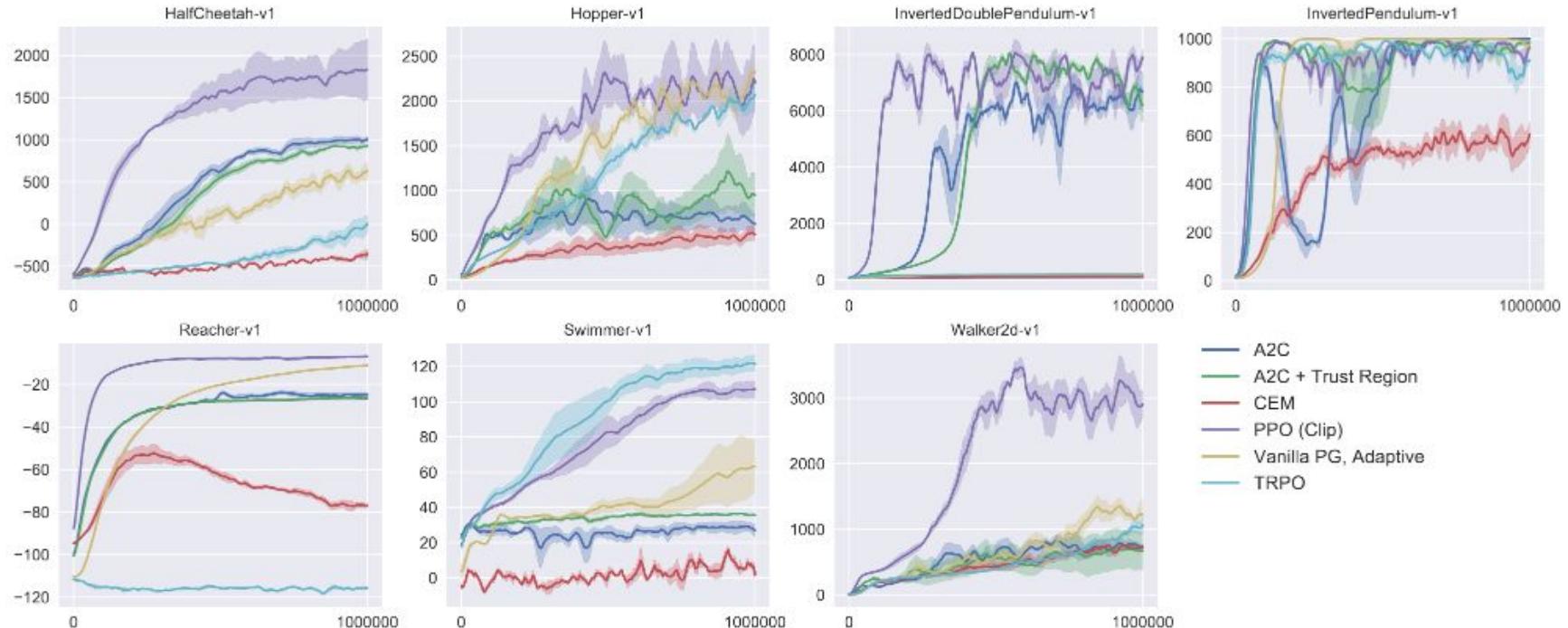


Figure: Performance comparison between PPO with clipped objective and various other deep RL methods on a slate of MuJoCo tasks.¹⁰

¹⁰Schulman, Wolski, Dhariwal, Radford, Klimov, 2017