Advanced Policy Gradient Methods

Joshua Achiam

UC Berkeley, OpenAI

October 11, 2017

Outline

Theory:

1. Problems with Policy Gradient Methods
2. Policy Performance Bounds
3. Monotonic Improvement Theory

Algorithms:

1. Natural Policy Gradients
2. Trust Region Policy Optimization
3. Proximal Policy Optimization

The Problems with Policy Gradients

Policy Gradients Review

Policy gradient algorithms try to solve the optimization problem

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

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

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

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)

[1] In an unbiased way.

Importance Sampling Review

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

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

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

What is the variance of an importance sampling estimator?

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

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.

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

$$= \mathop{\mathrm{E}}_{\tau\sim\theta'}\left[\sum_{t=0}^{\infty} \frac{P(\tau_t|\theta)}{P(\tau_t|\theta')} \gamma^t \nabla_\theta \log \pi_\theta(a_t|s_t) A^\theta(s_t, a_t)\right]$$

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?

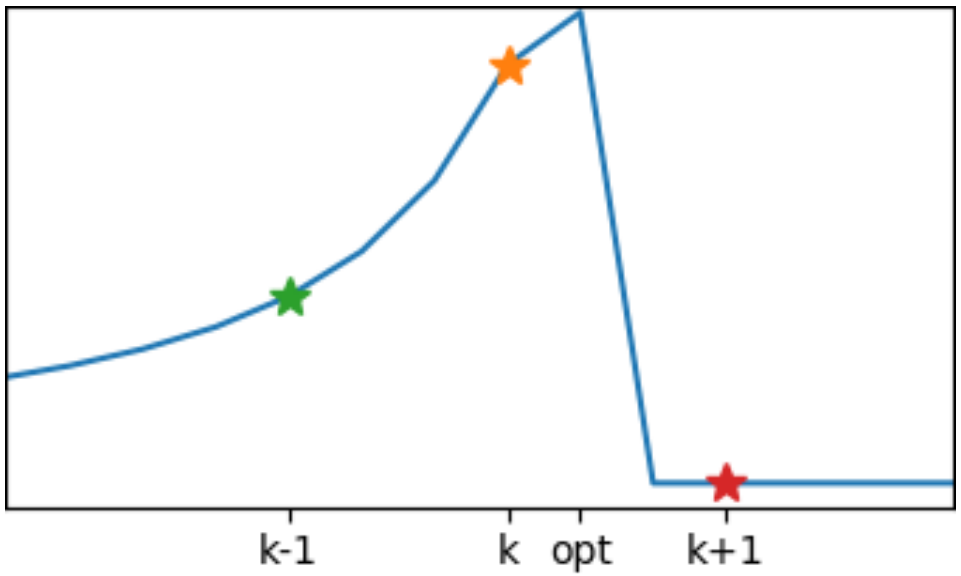

Figure: Policy parameters on x-axis and performance on y-axis. A bad step can lead to performance collapse, which may be hard to recover from.

The Problem is More Than Step Size

Consider a family of policies with parametrization:

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

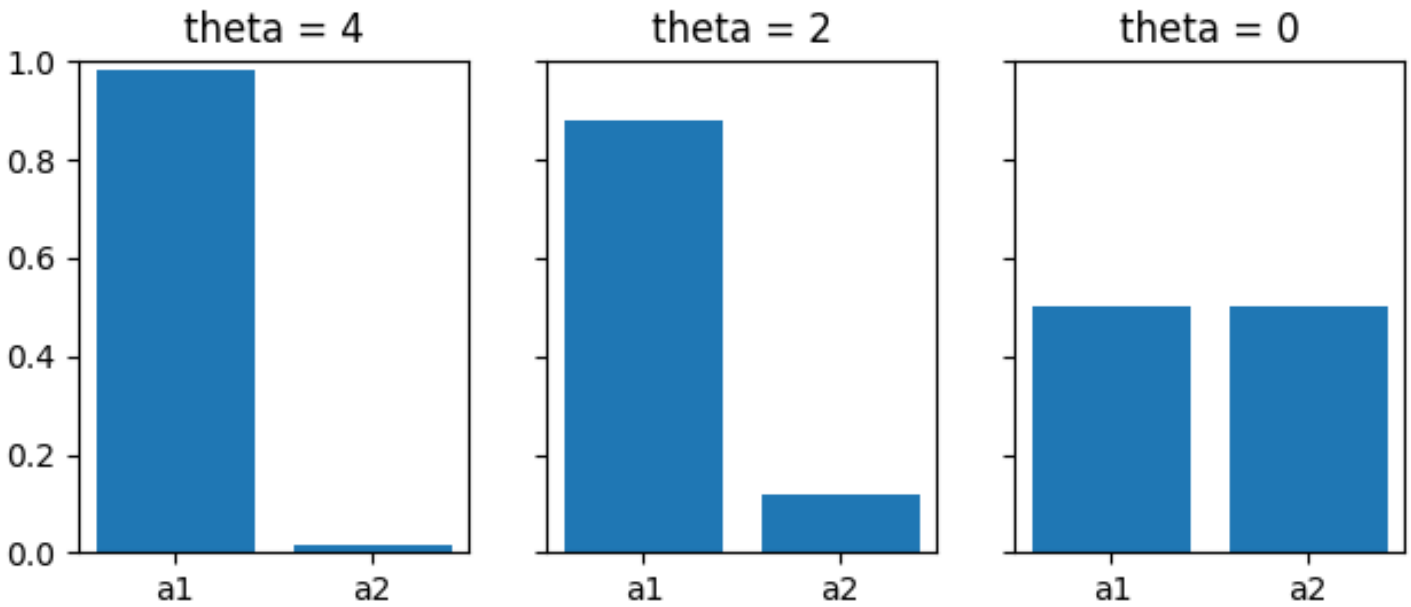

Figure: Small changes in the policy parameters can unexpectedly lead to **big** changes in the policy.

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

Policy Performance Bounds

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'

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

Proof of Relative Policy Performance Identity

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

What is it good for?

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

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

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:

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

...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?

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

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]

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

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

[2] Achiam, 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?

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

$$\mathcal{L}_\pi(\pi') = \frac{1}{1-\gamma} \mathop{\mathrm{E}}_{\substack{s\sim d^\pi \\ a\sim\pi}} \left[\frac{\pi'(a|s)}{\pi(a|s)} A^\pi(s,a) \right]$$

$$= \mathop{\mathrm{E}}_{\tau\sim\pi} \left[\sum_{t=0}^{\infty} \gamma^t \frac{\pi'(a_t|s_t)}{\pi(a_t|s_t)} A^\pi(s_t,a_t) \right]$$

- 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:

$$\nabla_\theta \mathcal{L}_{\theta_k}(\theta)|_{\theta_k} = \mathop{\mathrm{E}}_{\tau\sim\pi_{\theta_k}} \left[\sum_{t=0}^{\infty} \gamma^t \frac{\nabla_\theta \pi_\theta(a_t|s_t)|_{\theta_k}}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t,a_t) \right]$$

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

Recommended Reading

- "Approximately Optimal Approximate Reinforcement Learning," Kakade and Langford, 2002 [3]
- "Trust Region Policy Optimization," Schulman et al. 2015 [4]
- "Constrained Policy Optimization," Achiam et al. 2017 [5]

[3] https://people.eecs.berkeley.edu/ pabbeel/cs287-fa09/readings/KakadeLangford-icml2002.pdf
[4] https://arxiv.org/pdf/1502.05477.pdf
[5] https://arxiv.org/pdf/1705.10528.pdf

Monotonic Improvement Theory

Monotonic Improvement Theory

From the bound on the previous slide, we get

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

- 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*!

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

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

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

Approximate Monotonic Improvement

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

Problem:

- C provided by theory is quite high when γ is near 1
- $\Longrightarrow$ 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!

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

Approximate Monotonic Improvement

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

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!

Algorithms

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!

$$\mathcal{L}_{\theta_k}(\theta) \approx \mathcal{L}_{\theta_k}(\theta_k) + g^T(\theta - \theta_k) \qquad g \doteq \nabla_\theta \mathcal{L}_{\theta_k}(\theta) \,|_{\theta_k}$$

$$\bar{D}_{KL}(\theta || \theta_k) \approx \frac{1}{2}(\theta - \theta_k)^T H (\theta - \theta_k) \qquad H \doteq \nabla^2_\theta \bar{D}_{KL}(\theta || \theta_k) \,|_{\theta_k}$$

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 the 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:

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

- 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 the Natural Policy Gradient (Can Skip)

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

- In a Reimannian 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 $\text{diag}\left(1, r^2\right)$:

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

Covariance of the Natural Policy Gradient (Can Skip)

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

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

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

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

Gradients are related by chain rule:

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

Covariance of the Natural Policy Gradient (Can Skip)

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

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

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

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.

[6] Peters, Vijayakumar, Schaal, 2005.

Algorithm 1 Natural Policy Gradient

Input: initial policy parameters θ_0
for $k = 0, 1, 2, \ldots$ **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, $\text{span}\{g, Hg, H^2g, ..., H^{j-1}g\}$
- 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 $j = 0, 1, 2, ..., L$ **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 $k = 0, 1, 2, \ldots$ **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

Empirical Performance for TNPG / TRPO

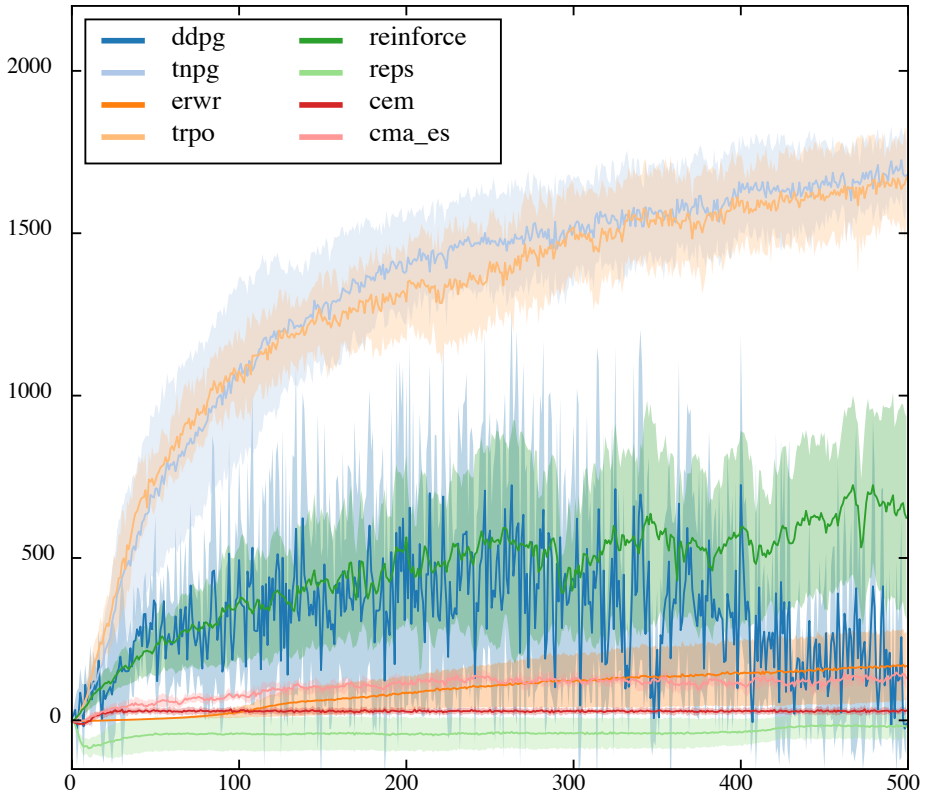

Figure: Comparison between various methods for deep RL including TNPG and TRPO on Walker-2d task. Showing average scores over 5 seeds for each method. [8]

[8] Duan, 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

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

    where ϵ is a hyperparameter (maybe $\epsilon = 0.2$)
  - Policy update is $\theta_{k+1} = \arg\max_\theta \mathcal{L}^{CLIP}_{\theta_k}(\theta)$

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 $k = 0, 1, 2, \ldots$ **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 $k = 0, 1, 2, \ldots$ **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

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

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:

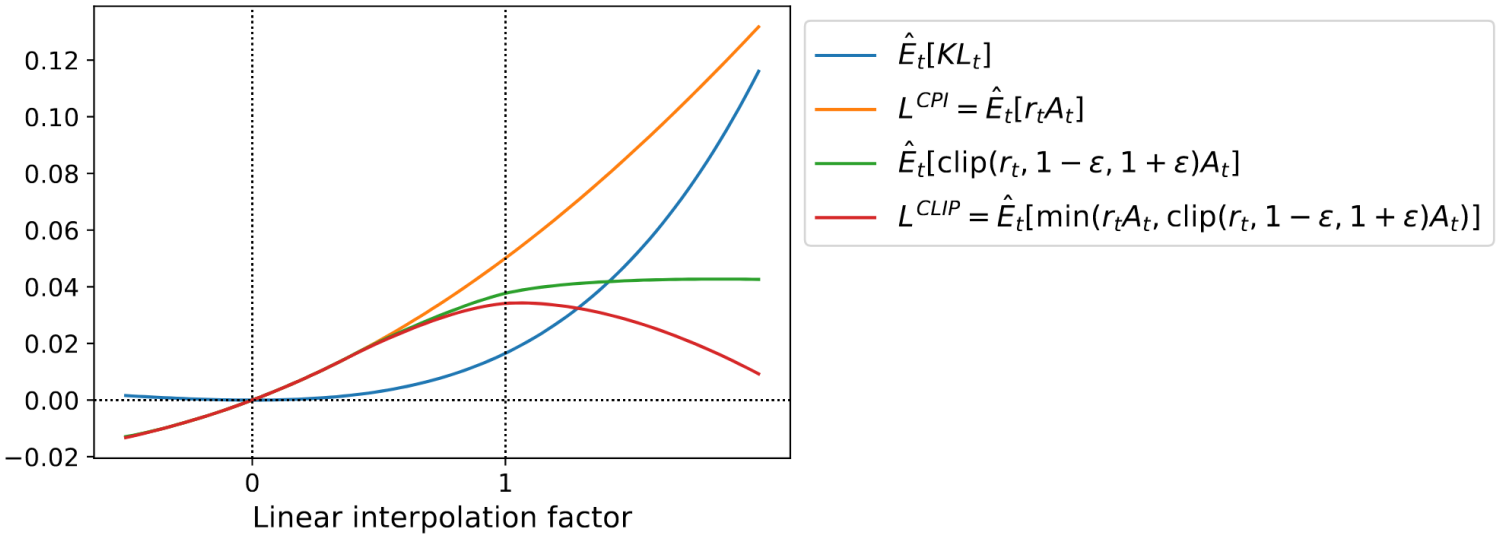

Figure: Various objectives as a function of interpolation factor α between θ_{k+1} and θ_k after one update of PPO-Clip [9]

[9] Schulman, Wolski, Dhariwal, Radford, Klimov, 2017

Empirical Performance of PPO

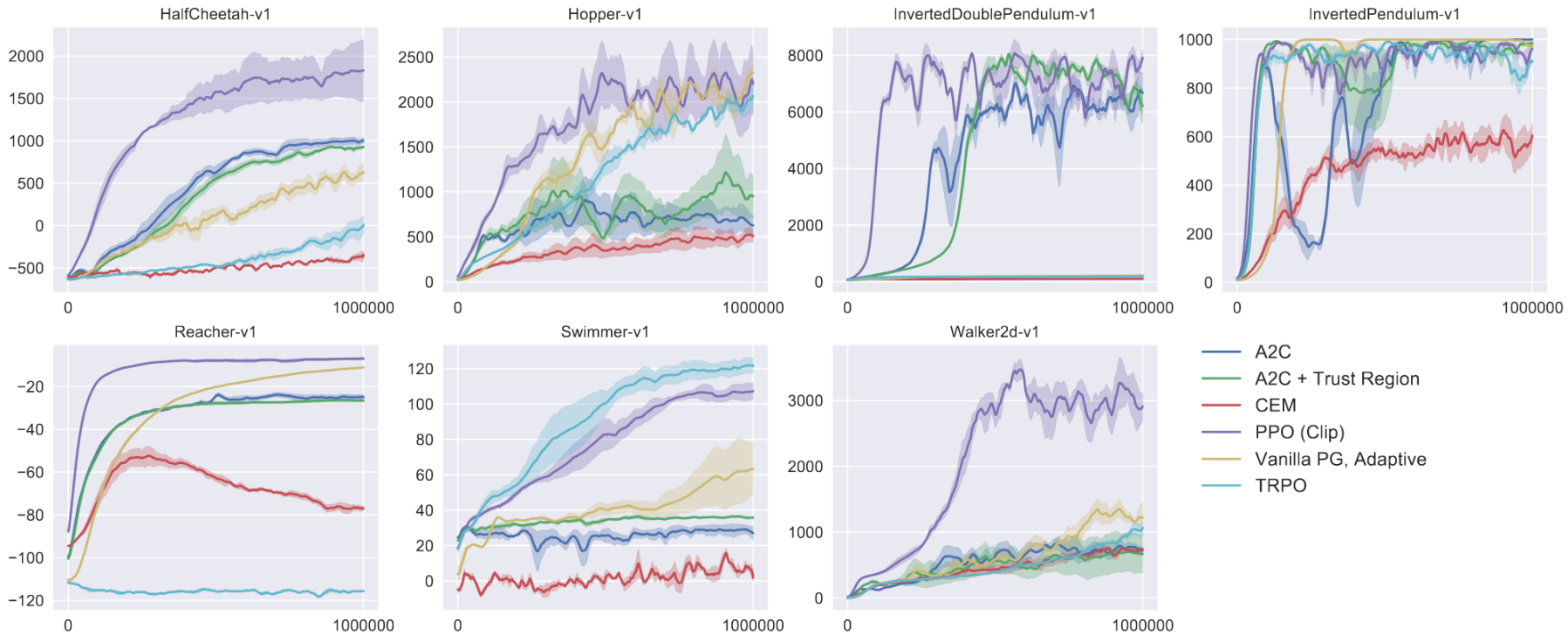

Figure: Performance comparison between PPO with clipped objective and various other deep RL methods on a slate of MuJoCo tasks. [10]

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

Recommended Reading

Natural Gradients / Natural Policy Gradients

- "Why Natural Gradient?" S. Amari and S. C. Douglas, 1998 [11]
- "A Natural Policy Gradient," Sham Kakade, 2001 [12]
- "Reinforcement Learning of Motor Skills with Policy Gradients," Jan Peters and Stefan Schaal, 2008 [13]

[11] http://www.yaroslavvb.com/papers/amari-why.pdf
[12] https://papers.nips.cc/paper/2073-a-natural-policy-gradient.pdf
[13] http://www.kyb.mpg.de/fileadmin/user_upload/files/publications/attachments/Neural-Netw-2008-21-682_4867%5b0%5d.pdf

Recommended Reading

TRPO / PPO

- "Trust Region Policy Optimization," Schulman et al. 2015 [14]
- "Benchmarking Deep Reinforcement Learning for Continuous control," Duan et al. 2016 [15]
- "Proximal Policy Optimization Algorithms," Schulman et al. 2017 [16]
- OpenAI blog post on PPO, 2017 [17]
- "Emergence of Locomotion Behaviours in Rich Environments," Heess et al. 2017 [18]
- "Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation," Wu et al. 2017 [19]

[14] https://arxiv.org/pdf/1502.05477.pdf
[15] https://arxiv.org/pdf/1604.06778.pdf
[16] https://arxiv.org/pdf/1707.06347.pdf
[17] https://blog.openai.com/openai-baselines-ppo/
[18] https://arxiv.org/pdf/1707.02286.pdf
[19] https://arxiv.org/pdf/1708.05144.pdf