Policy Gradient II

Introduction to Reinforcement Learning

Bogdan Ivanyuk-Skulskyi, Dmytro Kuzmenko

Department of Mathematics,
National University of Kyiv-Mohyla Academy

March 20, 2023

"Vanilla" Policy Gradient Algorithm

Initialize policy parameter θ, baseline b
for iteration=$1, 2, \cdots$ **do**
  Collect a set of trajectories by executing the current policy
  At each timestep t in each trajectory τ^i, compute
    Advantage estimate $\hat{A}_{it}^n$
  Update the policy, using a policy gradient estimate $\hat{g}$,
    Which is a sum of terms $\nabla_\theta \log \pi(a_t|s_t, \theta)\hat{A}_{it}^n$.
    (**Plug $\hat{g}$ into SGD or ADAM**)
endfor

Likelihood Ratio / Score Function Policy Gradient

- Policy gradient:

$$\nabla_\theta \mathbb{E}[R] \approx (1/m) \sum_{i=1}^{m} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t, s_t)(G_t^{(i)} - b(s_t))$$

- Fixes that improve simplest estimator
  - Temporal structure (shown in above equation)
  - Baseline (shown in above equation)
  - Alternatives to using Monte Carlo returns G_t^i as estimate of expected discounted sum of returns for the policy parameterized by θ?

Choosing the Target

- G_t^i is an estimation of the value function at st from a single roll out
- Unbiased but high variance
- Reduce variance by introducing bias using bootstrapping and function approximation
  - Just like in we saw for TD vs MC, and value function approximation

Actor-critic Methods

- Estimate of V/Q is done by a critic
- Actor-critic methods maintain an explicit representation of policy and the value function, and update both
- A3C (Mnih et al. ICML 2016) is a very popular actor-critic method

Policy Gradient Formulas with Value Functions

- Recall:

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

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

- Letting the baseline be an estimate of the value V , we can represent the gradient in terms of the state-action advantage function

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

- where the advantage function $A^\pi(s_t,a_t) = Q^\pi(s,a) - V^\pi(s)$

Advanced Policy Gradients: Table of contents

Policy Gradients Review

Policy gradient algorithms try to solve the optimization problem

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

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

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

Limitations of policy gradients:

- Sample efficiency is poor
- Distance in parameter space $\neq$ distance in policy space!
  - What is policy space? For tabular case, set of matrices

$$\Pi = \{\pi : \pi \in \Re^{|S|x|A|}, \sum_a \pi_{sa} = 1, \pi_{sa} \geq 0\}$$

  - 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 vanilla policy gradient methods is poor
- Discard each batch of data immediately after just one gradient step
- Why? PG is an on-policy expectation.
- Two main approaches to obtaining an unbiased estimate of the policy gradient
  - Run policy in environment and collect sample trajectories, then form sample estimate. (More stable)
  - Use trajectories from other policies with importance sampling. (Less stable)

Importance Sampling

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

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

The ratio $P(x)/Q(x)$ is the importance sampling weight for x The variance of an importance sampling estimator

$$var(\hat{\mu}_Q) = \frac{1}{N}var\left(\frac{P(x)}{Q(x)}f(x)\right)$$

$$= \frac{1}{N}\left(E_{x\sim Q}\left[\left(\frac{P(x)}{Q(x)}f(x)\right)^2\right] - E_{x\sim Q}\left[\frac{P(x)}{Q(x)}f(x)\right]^2\right)$$

$$= \frac{1}{N}\left(E_{x\sim P}\left[\frac{P(x)}{Q(x)}f(x)^2\right] - E_{x\sim P}\left[f(x)\right]^2\right)$$

The term in 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.

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

$$= \sum_\tau \sum_{t=0}^{\infty} \gamma^t P(\tau_t|\theta)\nabla_\theta \log \pi_\theta(a_t|s_t) A^\theta(s_t,a_t)$$

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

Exploding or vanishing importance sampling weights.

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

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

Choosing a Step Size for Policy Gradients

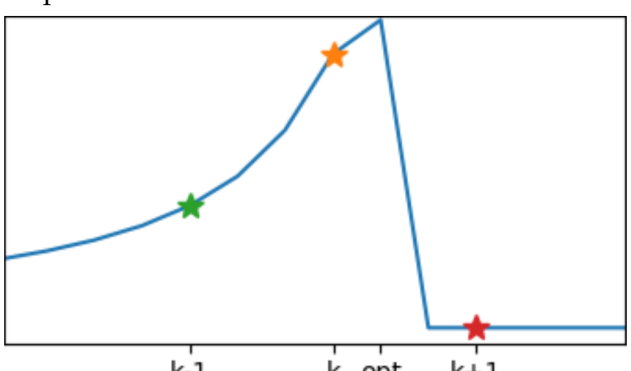
Policy gradient algorithms are stochastic gradient ascent:

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

with step $\nabla_k = \alpha_k \hat{g}_k$

- If the step is too large, performance collapse is possible (Why?)
- 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?

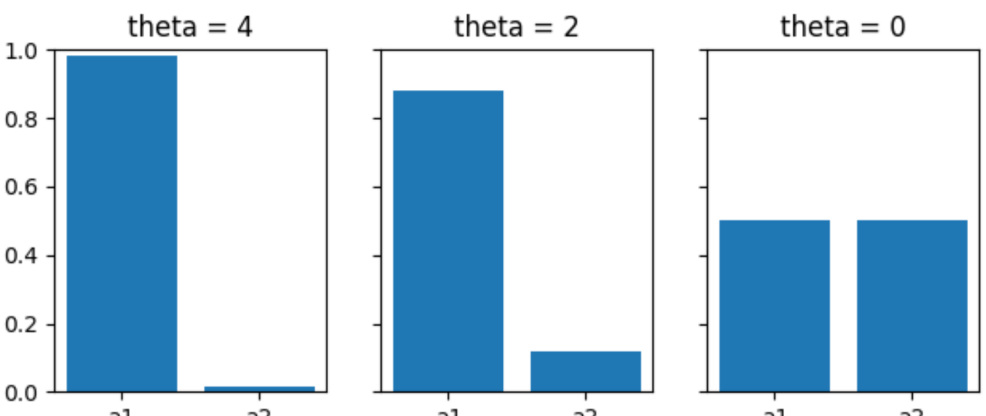


Policy parameters on x-axis and performance on y-axis. A bad step can lead to performance collapse, which may be hard to recover from.

The Problem is More Than Step Size

Consider a family of policies with parametrization:

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



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

Advanced Policy Gradients: Table of contents

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.
Performance difference lemma:

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

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

where

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

WHAT IS IT GOOD FOR?

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

$$\max_{\pi'} J(\pi') = \max_{\pi'} J(\pi') - J(\pi)$$

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

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:

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

$$= \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^{\pi'}_{a\sim\pi'}}\left[A^\pi(s,a)\right]$$

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

A Useful Approximation

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

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

$$= \mathcal{L}_{\pi}(\pi')$$

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:

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

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

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?

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

A Useful Approximation

What did we gain from making that approximation?

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

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

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

- 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.

Recommended Reading

- "Approximately Optimal Approximate Reinforcement Learning," Kakade and Langford, 2002[2]
- "Trust Region Policy Optimization," Schulman et al. 2015[3]
- "Constrained Policy Optimization," Achiam et al. 2017[4]

Advanced Policy Gradients: Table of contents

Monotonic Improvement Theory

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

- 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.
- And $\mathcal{L}_\pi(\pi')$ 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

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

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

This proof works even if we restrict the domain of optimization to an arbitrary class of parametrized policies Π_θ, as long as $\pi_k \in \Pi_\theta$.

Approximate Monotonic Improvement

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

Problem:

- C provided by theory is quite high when γ is near 1
- steps are too small

Potential Solution:

- Tune the KL penalty
- Use KL constraint (called trust region).

Advanced Policy Gradients: Table of contents

Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a family of methods that approximately penalize policies from changing too much between steps. Two variants:

- Adaptive KL Penalty
  - Policy update solves unconstrained optimization problem

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

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

Proximal Policy Optimization with Adaptive KL Penalty

Algorithm 1 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

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

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

  - 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

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

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

Proximal Policy Optimization with Clipped Objective

Algorithm 2 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

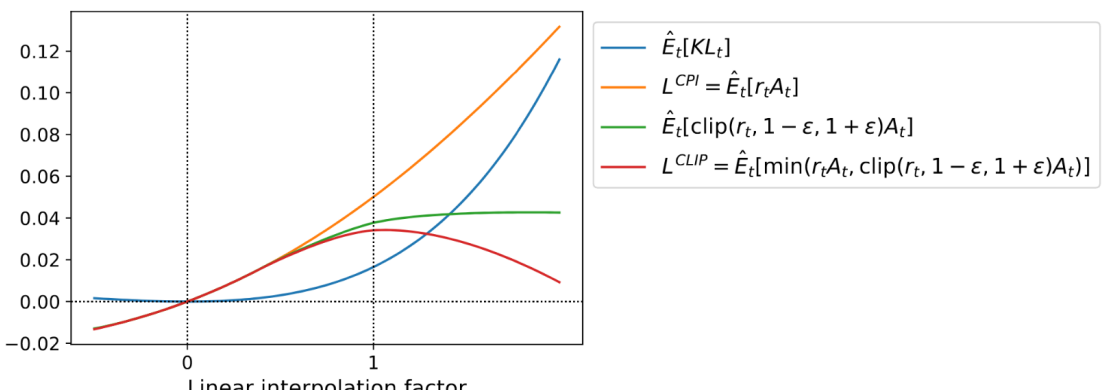
$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_k} \left[\sum_{t=0}^{T} \left[\min(r_t(\theta)\hat{A}_t^{\pi_k}, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t^{\pi_k}) \right] \right]$$

end for

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

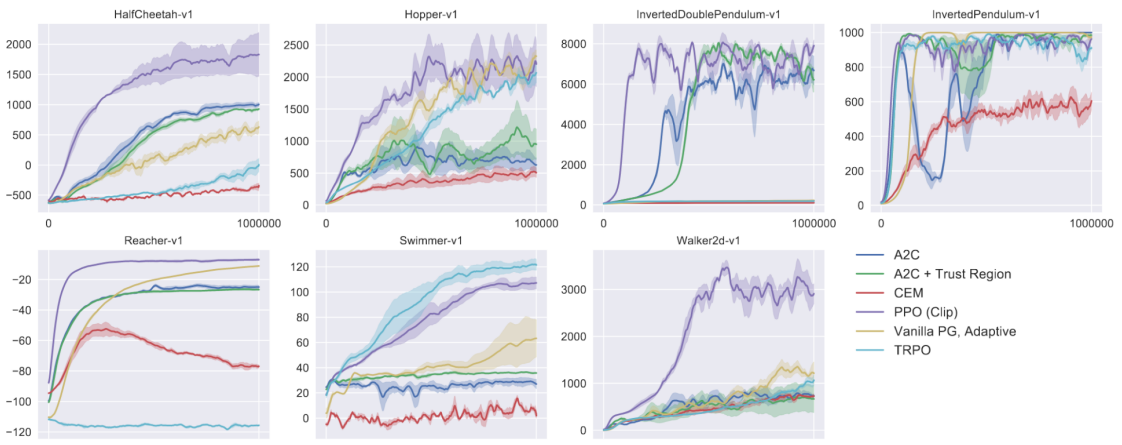
Proximal Policy Optimization with Clipped Objective

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



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

Empirical Performance of PPO



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

Wildly popular, and key component of ChatGPT

Recommended Reading

- "Proximal Policy Optimization Algorithms," Schulman et al. 2017
- OpenAI blog post on PPO, 2017