CSC415: Introduction to Reinforcement Learning

Lecture 6: Advanced Policy Gradients

Dr. Amey Pore

Winter 2026

February 11, 2026

Structure adapted from Chelsea Finn's course on Deep RL and Emma Brunskill's course on Introduction to RL.

Course Logistics

- **Project Teams:**
  - Still many unassigned teams on Quercus.
  - Students not in a team will be assigned randomly soon.
  - **Team Building:** put a note of the topic you want to work on the sticky note.

Think pair wise

Which of the following are true about REINFORCE? In the following options, PG stands for policy gradient.

- (A) Adding a baseline term can help to reduce the variance of the PG updates
- (B) It will converge to a global optima
- (C) It can be initialized with a sub-optimal, deterministic policy and still converge to a local optima, given the appropriate step sizes
- (D) If we take one step of PG, it is possible that the resulting policy gets worse (in terms of achieved returns) than our initial policy

Outline

REINFORCE Algorithm

- Using policy gradient theorem
- Using return G_t as an unbiased estimate of $Q^{\pi_\theta}(s_t, a_t)$
- Stochastic gradient ascent update:

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) G_t$$

1: Initialize policy parameters θ arbitrarily
2: **for** each episode $\{s_1, a_1, r_2, \cdots, s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
3: **for** $t = 1$ to $T - 1$ **do**
4: $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t|a_t) G_t$
5: **end for**
6: **end for**
7: **return** θ

Very high variance!

Action-Value Actor-Critic

- Simple actor-critic algorithm based on action-value critic
- Using linear value fn approx. $Q_w(s,a) = \phi(s,a)^\top w$
  - Critic Updates w by linear TD(0)
  - Actor Updates θ by policy gradient

---

function Q-Actor Critic
Initialize s, θ
Sample action $a \sim \pi_\theta(\cdot|s)$
for each step **do**
    Sample reward $r = R(s,a)$; sample transition $s' \sim P(\cdot|s,a)$
    Sample action $a' \sim \pi_\theta(\cdot|s')$
    $\delta \leftarrow r + \gamma Q_w(s',a') - Q_w(s,a)$
    $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a|s) Q_w(s,a)$
    $w \leftarrow w + \beta \delta \phi(s,a)$
    $a \leftarrow a', s \leftarrow s'$
end for
end function

"Vanilla" Policy Gradient Algorithm

1: Initialize policy parameter θ, baseline b
2: **for** iteration $= 1, 2, \cdots$ **do**
3: Collect a set of trajectories by executing the current policy
4: At each timestep t in each trajectory τ^i, compute:
5: Return $G_t^i = \sum_{t'=t}^{T-1} r_{t'}^i$
6: Advantage estimate $\hat{A}_t^i = G_t^i - b(s_t)$
7: Re-fit the baseline, by minimizing $\sum_i \sum_t \|b(s_t) - G_t^i\|^2$
8: Update the policy, using a policy gradient estimate $\hat{g}$:
9: $\hat{g} = \sum_i \sum_t \nabla_\theta \log \pi(a_t|s_t, \theta) \hat{A}_t^i$
10: (Plug $\hat{g}$ into SGD or ADAM)
11: **end for**

- Other Baseline: $\underbrace{A^\pi(s,a) = Q_w^\pi(s,a) - V_w^\pi(s)}_{\text{Advantage Actor Critic}}$

Asynchronous Advantage Actor-Critic (A3C)

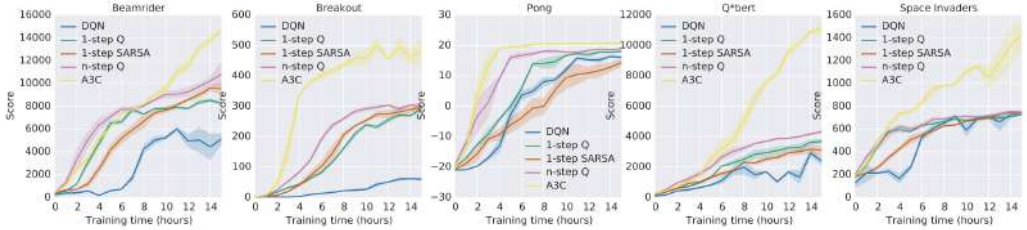


Figure: A3C: Multiple workers interact with their own environments and update a global network asynchronously.

Mnih, Badia, Mirza, Graves, Lillicrap, Harley, Silver, Kavukcuoglu. *Asynchronous Methods for Deep Reinforcement Learning.* ICML 2016.

Outline

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
- Distance in parameter space $\neq$ distance in policy space!
  - What is policy space? For tabular case, set of matrices $\Pi = \left\{ \pi : \pi \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{A}|}, \; \sum_a \pi_{sa} = 1, \; \pi_{sa} \geq 0 \right\}$
  - 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 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

- Collect sample trajectories from policy, then form sample estimate. (More stable)
- Use trajectories from other policies (Less stable)

- **Opportunity:** use old data to take **multiple gradient** steps before using the resulting new policy to gather more data
- **Challenge:** even if this is possible to use old data to estimate multiple gradients, how many steps should be taken?

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 (Why?)

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 (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?

Choosing a Step Size for Policy Gradients

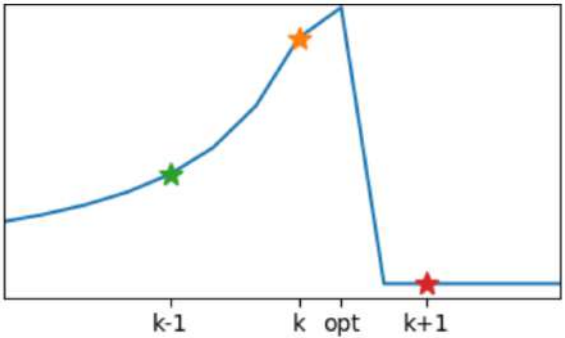
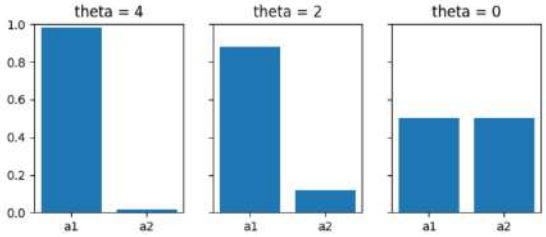


Figure: Policy parameters on x-axis and performance on y -axis. A bad step can lead to performance collapse, which may be hard to recover from.

The Problem is More Than Step Size

Consider a family of policies with parametrization:

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



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

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

Outline

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: For any policies π, π'

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

$$= \frac{1}{1-\gamma} \mathbb{E}_{\substack{s \sim d^{\pi'} \\ a \sim \pi'}} [A^{\pi}(s, a)] \tag{2}$$

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 π' represents the new policy and π 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 $\pi' \ldots$

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:

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

Note: Instance of Importance Sampling

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:

$$J(\pi') - J(\pi) = \mathbb{E}_{\tau\sim\pi'}\left[\sum_{t=0}^{\infty}\gamma^t A^\pi(s_t, a_t)\right] \tag{3}$$

$$= \frac{1}{1-\gamma}\,\mathbb{E}_{\substack{s\sim d^{\pi'}\\ a\sim\pi'}}[A^\pi(s,a)] \tag{4}$$

$$= \frac{1}{1-\gamma}\,\mathbb{E}_{\substack{s\sim d^{\pi'}\\ a\sim\pi}}\left[\frac{\pi'(a|s)}{\pi(a|s)}A^\pi(s,a)\right] \tag{5}$$

Last step is an instance of importance sampling. 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?

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

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:[1]

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

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

[1] Achiam, Held, Tamar, Abbeel, 2017

What is KL-divergence?

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

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

Properties:

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

What is KL-divergence between policies?

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

A Useful Approximation

What did we gain from making that approximation?

$$J(\pi') - J(\pi) \approx L_\pi(\pi')$$

$$L_\pi(\pi') = \frac{1}{1-\gamma} \mathbb{E}_{\substack{s\sim d^\pi \\ a\sim\pi}} \left[\frac{\pi'(a|s)}{\pi(a|s)} A^\pi(s,a) \right] \tag{6}$$

$$= \mathbb{E}_{\tau\sim\pi} \left[\sum_{t=0}^{\infty} \gamma^t \frac{\pi'(a_t|s_t)}{\pi(a_t|s_t)} A^\pi(s_t,a_t) \right] \tag{7}$$

- 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
- "Trust Region Policy Optimization," Schulman et al. 2015
- "Constrained Policy Optimization," Achiam et al. 2017

Outline

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

$$\theta_{k+1} = \arg\max_{\theta} \; L_{\theta_k}(\theta) - \beta_k \, \bar{D}_{\mathrm{KL}}(\theta \| \theta_k) \tag{8}$$

$$\bar{D}_{\mathrm{KL}}(\theta \| \theta_k) = \mathbb{E}_{s \sim d^{\pi_k}} \left[D_{\mathrm{KL}}\big(\pi_{\theta_k}(\cdot|s), \; \pi_{\theta}(\cdot|s)\big) \right] \tag{9}$$

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

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

$$\theta_{k+1} = \arg\max_{\theta} \; L_{\theta_k}(\theta) - \beta_k \, \bar{D}_{\mathrm{KL}}(\theta \| \theta_k)$$

    by taking K steps of minibatch SGD (via Adam)
    if $\bar{D}_{\mathrm{KL}}(\theta_{k+1} \| \theta_k) \geq 1.5\delta$ **then**
        $\beta_{k+1} = 2\beta_k$
    else if $\bar{D}_{\mathrm{KL}}(\theta_{k+1} \| \theta_k) \leq \delta / 1.5$ **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

PPO with Adaptive KL Penalty: Multiple Gradient Steps

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

$$\theta_{k+1} = \arg\max_{\theta} \; L_{\theta_k}(\theta) - \beta_k \, \bar{D}_{\mathrm{KL}}(\theta \| \theta_k)$$

by taking K **steps of minibatch SGD (via Adam)**
    if $\bar{D}_{\mathrm{KL}}(\theta_{k+1} \| \theta_k) \geq 1.5\delta$ **then**
        $\beta_{k+1} = 2\beta_k$
    else if $\bar{D}_{\mathrm{KL}}(\theta_{k+1} \| \theta_k) \leq \delta / 1.5$ **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 $\theta_{k+1} = \arg\max_\theta \; L_{\theta_k}(\theta) - \beta_k \, \bar{D}_{\mathrm{KL}}(\theta \| \theta_k)$
- 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

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

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

Think Pair wise: Proximal Policy Optimization

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

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

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

Consider the figure. Select all that are true. $\epsilon \in (0, 1)$.

- The left graph shows the L^{CLIP} objective when the advantage function $A > 0$ and the right graph shows when $A < 0$
- The right graph shows the L^{CLIP} objective when the advantage function $A > 0$ and the left graph shows when $A < 0$
- It depends on the value of ϵ
- Not sure

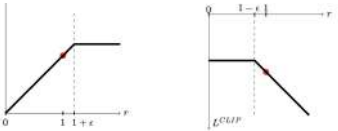


Figure: Schulman, Wolski, Dhariwal, Radford, Klimov, 2017

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:

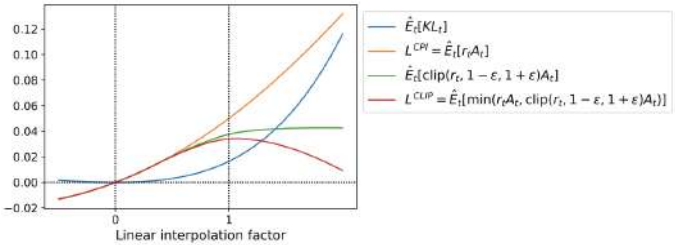


Figure: Figure from Schulman et al., 2017: Various objectives as a function of interpolation factor α between θ_{k+1} and θ_k after one update of PPO-Clip

Proximal Policy Optimization with Clipped Objective

1: **Input:** initial policy parameters θ_0, clipping threshold ϵ
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
4: Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
5: Compute policy update

$$\theta_{k+1} = \arg\max_{\theta} \; L_{\theta_k}^{\text{CLIP}}(\theta)$$

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

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

6: **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

Empirical Performance of PPO

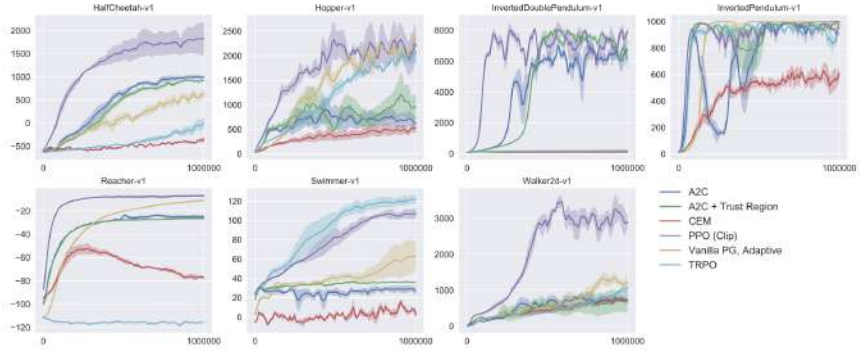


Figure: Figure: Performance comparison between PPO with clipped objective and various other deep RL methods on a slate of MuJoCo tasks.

- Wildly popular, and key component of ChatGPT

Recommended Reading

PPO

- "Proximal Policy Optimization Algorithms," Schulman et al. 2017
  `https://arxiv.org/pdf/1707.06347.pdf`
- OpenAI blog post on PPO, 2017
  `https://blog.openai.com/openai-baselines-ppo/`

PPO: Algorithm and Code Implementation Details

Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. *Implementation Matters in Deep RL: A Case Study on PPO and TRPO.* ICLR 2020

`https://openreview.net/forum?id=r1etN1rtPB`

- Reward scaling, learning rate annealing, etc. can make a significant difference

Course Logistics

- **Mid-term Exam:** will be distributed on Feb 25th (in class).
- **Project Teams:**
  - Still many unassigned teams on Quercus.
  - Students not in a team will be assigned randomly soon.
  - **Team Building:** put a note of the topic you want to work on the sticky note.
- **A1 Code Implementations Clarifications:**
  - Goal: Reproduce the exact results reported in the paper.
  - Form a hypothesis on how the behaviour changes with variations in specific parameters.
- **Project Proposal (Per Team):**
  - Due date pushed to **March 27th**.
  - Expectation: Reading papers, proposing a new idea, and providing some preliminary results.
- **Logistics for A2:**
  - Still checking if peer review is possible for A2.
  - If not, we will have another quiz. (10%)

Break: 10 minutes

Paste your ideas on the sticky note and find your team.

Outline

Recall Proximal Policy Optimization

PPO is a family of methods that approximately enforce KL constraint

- 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

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

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

How do we estimate the advantage function inside the policy update?

Recall N-step estimators

$$\nabla_\theta V(\theta) \approx (1/m) \sum_{i=1}^{m} \sum_{t=0}^{T-1} A_{ti} \nabla_\theta \log \pi_\theta(a_{ti}|s_{ti})$$

- Recall the N-step advantage estimators

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

- Define $\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$. Then

$$\hat{A}_t^{(1)} = \delta_t^V$$
$$\hat{A}_t^{(2)} = \delta_t^V + \gamma \delta_{t+1}^V$$
$$\hat{A}_t^{(k)} = \sum_{l=0}^{k-1} \gamma^l \delta_{t+l}^V$$

Generalized Advantage Estimator (GAE)

$$\hat{A}_t^{(k)} = \sum_{l=0}^{k-1} \gamma^l r_{t+l} + \gamma^k V(s_{t+k}) - V(s_t) \tag{1}$$

- GAE is an exponentially-weighted average of k-step estimators

$$\begin{aligned}
\hat{A}_t^{GAE(\gamma,\lambda)} &= (1-\lambda)(\hat{A}_t^{(1)} + \lambda \hat{A}_t^{(2)} + \lambda^2 \hat{A}_t^{(3)} + \dots) \\
&= (1-\lambda)(\delta_t^V + \lambda(\delta_t^V + \gamma\delta_{t+1}^V) + \lambda^2(\delta_t^V + \gamma\delta_{t+1}^V + \gamma^2\delta_{t+2}^V) + \dots) \\
&= (1-\lambda)(\delta_t^V(1 + \lambda + \lambda^2 + \dots) + \gamma\delta_{t+1}^V(\lambda + \lambda^2 + \dots) \\
&\quad + \gamma^2\delta_{t+2}^V(\lambda^2 + \lambda^3 + \dots) + \dots) \\
&= (1-\lambda)(\delta_t^V \frac{1}{1-\lambda} + \gamma\lambda\delta_{t+1}^V \frac{1}{1-\lambda} + \gamma^2\lambda^2\delta_{t+2}^V \frac{1}{1-\lambda} + \dots) \\
&= \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V
\end{aligned}$$

Generalized Advantage Estimator (GAE) in PPO

- GAE is an exponentially-weighted average of k-step estimators

$$\begin{aligned}
\hat{A}_t^{(k)} &= \sum_{l=0}^{k-1} \gamma^l r_{t+l} + \gamma^k V(s_{t+k}) - V(s_t) \\
\delta_t^V &= r_t + \gamma V(s_{t+1}) - V(s_t) \\
\hat{A}_t^{GAE(\gamma,\lambda)} &= (1-\lambda)(\hat{A}_t^{(1)} + \lambda \hat{A}_t^{(2)} + \lambda^2 \hat{A}_t^{(3)} + \dots) \\
&= \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V
\end{aligned}$$

- PPO uses a truncated version of a GAE

$$\hat{A}_t = \sum_{l=0}^{T-t-1} (\gamma\lambda)^l \delta_{t+l}^V$$

- Benefits: Only have to run policy in environment for T timesteps before updating,

Final Proximal Policy Optimization

1: **Input:** initial policy parameters θ_0, clipping threshold ϵ
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
4: Estimate advantages $\hat{A}_t^{\pi_k}$ using Generalized Advantage Estimation (GAE)
5: Compute policy update

$$\theta_{k+1} = \arg\max_{\theta} \; L_{\theta_k}^{\text{CLIP}}(\theta)$$

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

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

6: **end for**

Some example hyperparameters:
$\sim$2000 timesteps in batch of data
$\sim$10 epochs when updating policy
($M \approx 300$ gradient steps with batch size 64)

Outline

Off-Policy Actor-Critic Methods

So far:

- use one batch of policy data for one gradient step (fully on-policy)
- use one batch of policy data for multiple gradient steps (starting to be off-policy)

Can we be even more off-policy?

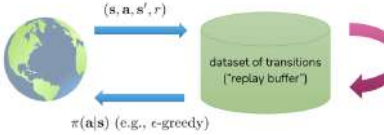
Idea: Can we reuse data from previous batches, i.e. all of the past trial-and-error data?

Two key ideas

1. Maintain a **replay buffer** of all past data
2. Adjust equations to remove on-policy assumptions

DQN Pseudocode

Input: $E, \alpha, s, a, r, s' \sim \pi$; **Initialize** $\mathcal{D} = \emptyset$, $\mathbf{w} = 0$
Set other state $\mathbf{w}_0$
for episode $= 1, \ldots, E$ **do do**
    Initialize s_1
    for $t = 1, \ldots, T$ **do do**
        Observe reward r_t and next state s_{t+1}
        Store transition (s_t, a_t, r_t, s_{t+1}) in replay buffer $\mathcal{D}$
        for $i = 1, \ldots, K$ **do do**
            Sample random minibatch of transitions (s, a, r, s') from $\mathcal{D}$
            if s_{t+1} is terminal at step $t + 1$ **then then**
                Set $y_t = r_t$
            else
                Set $y_t = r_t + \gamma \max_{a'} \hat{Q}(s_{t+1}, a'; \mathbf{w}^-)$
            end if
            Perform gradient descent step on $(y_t - \hat{Q}(s_t, a_t; \mathbf{w}))^2$ w.r.t. $\mathbf{w}$
        end for
        Every C steps: $\mathbf{w}^- = \mathbf{w}$
    end for
end for



Fixing the Value Function

Solution: Fit $Q_w(s, a)$ instead of $V_v(s)$.

The datapoints we have: $(s, a, r, s') +$ future reward.

- If we fit $V_v(s)$, we assume the next action comes from π (but a in buffer is from old policy).
- If we fit $Q_w(s, a)$, we pass the action as input!
- It is okay if a is different from what the current policy would have done.

Q-Learning Update Strategy:

$$Q_w(s, a) \leftarrow r(s, a) + \gamma \mathbb{E}_{s' \sim p(\cdot|s,a), a' \sim \pi(\cdot|s')}[Q_w(s', a')]$$

We use samples (s, a, r, s') from the replay buffer, but sample the *next action* a' from the *current policy*.

Off Policy Actor-Critic

1: **Input:** initial policy parameters θ, Q-function parameters w, Replay Buffer $\mathcal{D}$
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of partial trajectories on policy π_k and add to $\mathcal{D}$
4: Sample batch of transitions $B \sim \mathcal{D}$
5: Update critic Q_w using Q-learning update strategy
6: Compute policy update

- Original Policy Gradient: $\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi}[\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t)\hat{A}_t]$
- Adapted for SAC (Q-function):

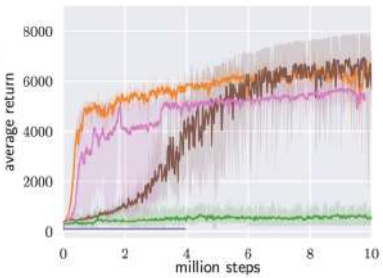
$$\nabla_\theta J(\theta) \approx \frac{1}{|B|} \sum_{s \in B} \nabla_\theta \log \pi_\theta(a|s) Q_w(s, a)$$

7: **end for**

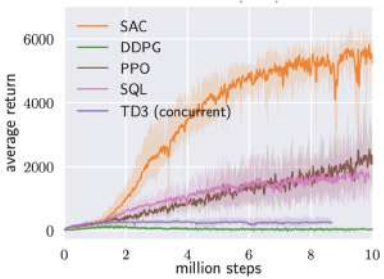
Haarnoja, Zhou, Abbeel, Levine. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor.* 2018.

`https://sites.google.com/view/sac-and-applications/`

Soft actor critic



(e) Humanoid-v1



(f) Humanoid (rllab)

<- more off-policy
(i.e. with replay buffer)

<- less off-policy
(i.e. no replay buffer)

- + Off-policy with replay buffer (e.g. soft actor-critic) can be far more data efficient
- - They can also generally be a lot harder to tune hyperparameters, less stable (than e.g. PPO)

When to use one online RL algorithm vs. another?

- **PPO & variants**
  - When you care about stability, ease-of-use
  - When you don't care about data efficiency
- **DQN & variants**
  - When you have discrete actions or low-dimensional continuous actions
- **SAC & variants**
  - When you care most about data efficiency
  - When you are okay with tuning hyperparameters, less stability

Outline

Monotonic Improvement Theory

From the bound on the previous slide, we get

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

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 $L_\pi(\pi')$ 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

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

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

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

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

Problem:

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

Potential Solution:

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