ECE7121 Learning-based control – 2025 Fall

Actor-critic

INHA UNIVERSITY

Overview

- Advantage
- Actor-critic structure
- PPO
- Off-policy AC

Recap

> Policy gradients

$$\nabla_\phi J(\phi) \approx \underbrace{\frac{1}{N}\sum_{i=1}^{N}}_{\text{samples from policy}} \sum_t \nabla_\phi \underbrace{\log \pi_\phi(a_{i,t}|s_{i,t})}_{\text{policy likelihood}} \Big(\underbrace{\sum_{t'=t} r(s_{i,t}, a_{i,t})}_{\text{rewards to go}} - \underbrace{b}_{\text{baseline}}\Big)$$

- do more of the above average stuff, less of the below average stuff

> Importance weights for off-policy PG

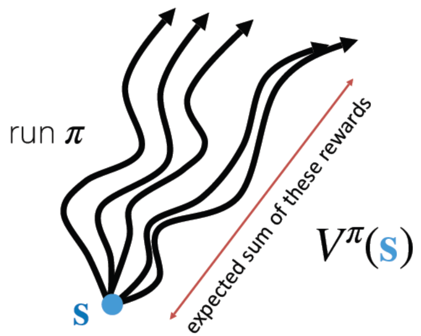
$$\nabla_{\phi'} J(\phi') \approx \frac{1}{N}\sum_i \sum_t \Big(\underbrace{\frac{\pi_{\phi'}(a_{i,t}|s_{i,t})}{\pi_\phi(a_{i,t}|s_{i,t})}}\Big)\big(\nabla_{\phi'} \log \pi_{\phi'}(a_{i,t}|s_{i,t})\big)\Big(\big(\sum_{t'=t} r(s_{i,t}, a_{i,t}\big) - b\Big)$$

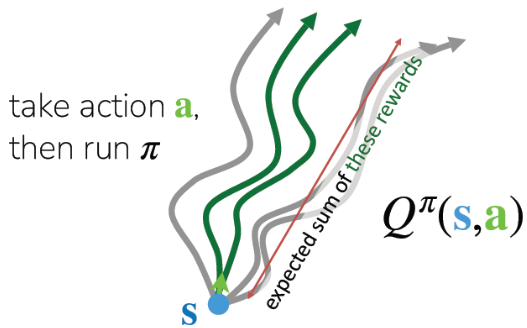
- only suitable when policies are very similar

Advantage

> Value function $V^{\pi}(s)$
  - future expected rewards starting at s and following π

> State-action value function $Q^{\pi}(s, a)$
  - future expected rewards starting at s, taking a, then following π





$$V^{\pi}(s) = \mathbb{E}_{a \sim \pi(\cdot|s)}[Q^{\pi}(s, a)]$$

> Advantage $A^{\pi}(s, a)$
  - how much better it is to take a than to follow π at state s
  - $A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s)$

Advantage

- $\nabla_\phi J(\phi) \approx \frac{1}{N}\sum_{i=1}^{N} \sum_t \nabla_\phi \log \pi_\phi(a_{i,t}|s_{i,t}) \left(\sum_{t'=t} r(s_{i,t}, a_{i,t}) - b\right)$

$$\rightarrow \frac{1}{N}\sum_{i=1}^{N} \sum_t \nabla_\phi \log \pi_\phi(a_{i,t}|s_{i,t}) \left(Q(s_{i,t}, a_{i,t}) - b\right)$$

$$\rightarrow \frac{1}{N}\sum_{i=1}^{N} \sum_t \nabla_\phi \log \pi_\phi(a_{i,t}|s_{i,t}) \left(Q(s_{i,t}, a_{i,t}) - V^\pi(s_t)\right)$$

$$= \frac{1}{N}\sum_{i=1}^{N} \sum_t \nabla_\phi \log \pi_\phi(a_{i,t}|s_{i,t})\, A^\pi(s_{i,t}, a_{i,t})$$

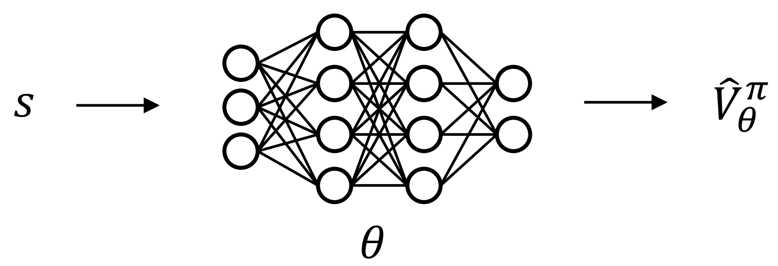
- Better estimates of A^π lead to less noisy gradients

Advantage

- Estimating expected return
- Should we fit V^π, Q^π or A^π?

- $$\begin{aligned} A^\pi(s_t, a_t) &= \mathrm{Q}^\pi(\mathrm{s_t}, \mathrm{a_t}) - \mathrm{V}^\pi(\mathrm{s_t}) \\ &= \sum_{t'=t}^{T} \mathbb{E}_\pi[r(s_{t'}, a_{t'}) | s_t, a_t] - V^\pi(s_t) \\ &= r(s_t, a_t) + \sum_{t'=t+1}^{T} \mathbb{E}_\pi[r(s_{t'}, a_{t'}) | s_t, a_t] - V^\pi(s_t) \\ &= r(s_t, a_t) + \mathbb{E}_{s_{t+1} \sim p(\cdot | s_t, a_t)}[V^\pi(s_{t+1})] - V^\pi(s_t) \\ &\approx r(s_t, a_t) + V^\pi(s_{t+1}) - V^\pi(s_t) \end{aligned}$$

- Let's just fit V^π



Estimating the value function

> In policy evaluation,
  - Monte-Carlo: $V(s) \leftarrow V(s) + \alpha\big(G_t - V(s)\big)$
  - TD(0): $V(s) \leftarrow V(s) + \alpha\big(r + \gamma V(s') - V(s)\big)$
  - n-step TD: $V(s) \leftarrow V(s) + \alpha(G_t^{(n)} - V(s))$
    - $G_t^{(n)} = r_t + \gamma r_{t+1} + \cdots + \gamma^{n-1} r_{t+n-1} + \gamma^n V(s_{t+n})$

> Likewise, fitting can be done in a similar way
  - Monte-Carlo: $\hat{V}_\theta^\pi(s_i) \leftarrow \sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'})$
  - TD(0): $\hat{V}_\theta^\pi(s_i) \leftarrow r\big(s_{i,t'}, a_{i,t'}\big) + \gamma \hat{V}_\theta^\pi\big(s_{i,t+1}\big)$
  - n-step TD: $\hat{V}_\theta^\pi(s_i) \leftarrow \sum_{t'=t}^{t+n-1} \gamma^{t'-t} r(s_{i,t'}, a_{i,t'}) + \gamma^{\mathrm{n}}\, \hat{V}_\theta^\pi\big(s_{i,t+n}\big)$

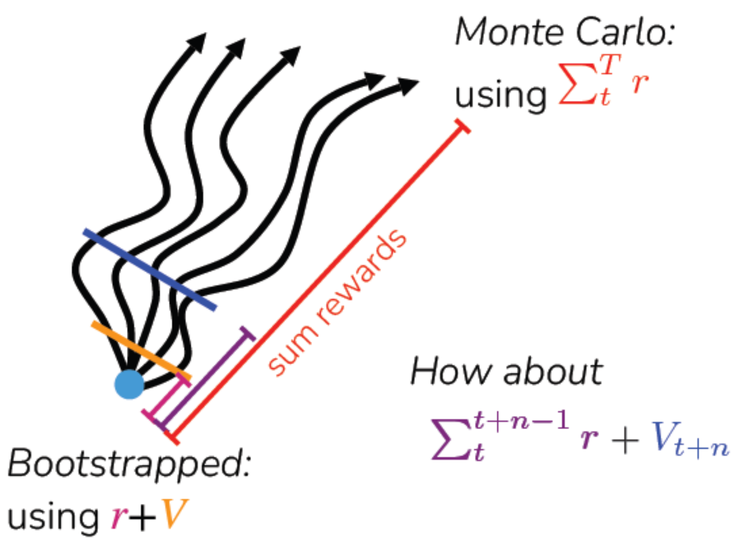
Estimating the value function

> Monte-Carlo: supervise with roll-out's summed rewards

> Bootstrap: supervise using reward and current value estimate

> N-step returns: lies between MC and Bootstrapped

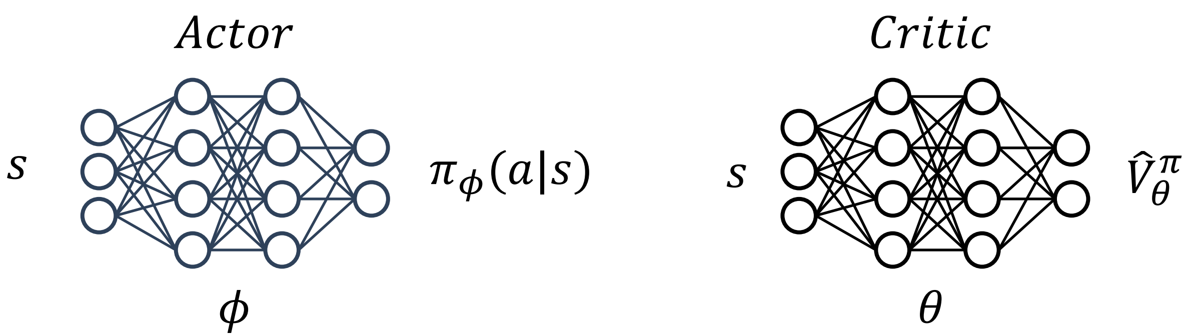
- less variance than MC
- lower bias than 1-step bootstrap
- often works the best in practice



A full actor-critic algorithm

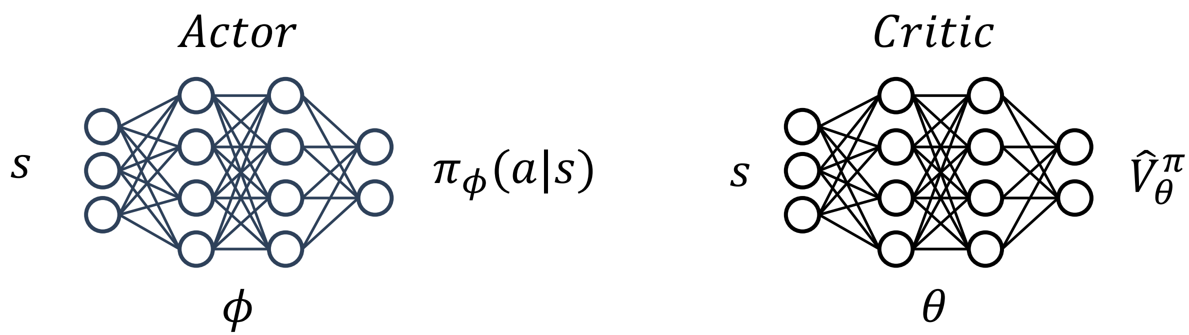
- Actor-critic: learn to estimate what is good vs. bad, then do more of the good stuff
  - get a better PG by using NN to estimate value function

- Algorithm
  1. sample a batch of data $\{\tau^i\}$ from $\pi_\phi(a_t|s_t)$ (run the policy)
  2. Fit $\hat{V}_\theta^{\pi_\phi}$ to summed rewards in data (n-step TD)
  3. Evaluate $\hat{A}^{\pi_\phi}(s_{i,t}, a_{i,t}) = r(s_{i,t}, a_{i,t}) + \gamma \hat{V}_\theta^{\pi_\phi}(s_{i,t+1}) - \hat{V}_\theta^{\pi_\phi}(s_{i,t})$
  4. Evaluate $\nabla_\phi J(\phi) \approx \sum_{i=1}^{N} \sum_t \nabla_\phi \log \pi_\phi(a_{i,t}|s_{i,t}) \hat{A}^{\pi_\phi}(s_{i,t}, a_{i,t})$
  5. $\phi \leftarrow \phi + \alpha \nabla_\phi J(\phi)$ and go to 1



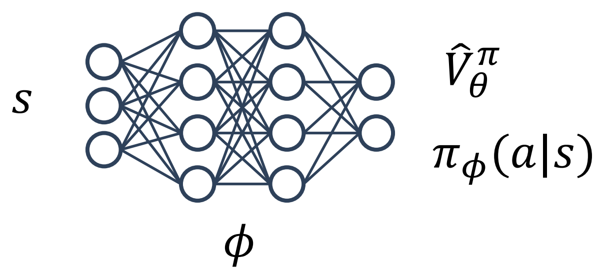
Architecture design

- Two network design



  - Simple and stable, but no shared features between actor & critic

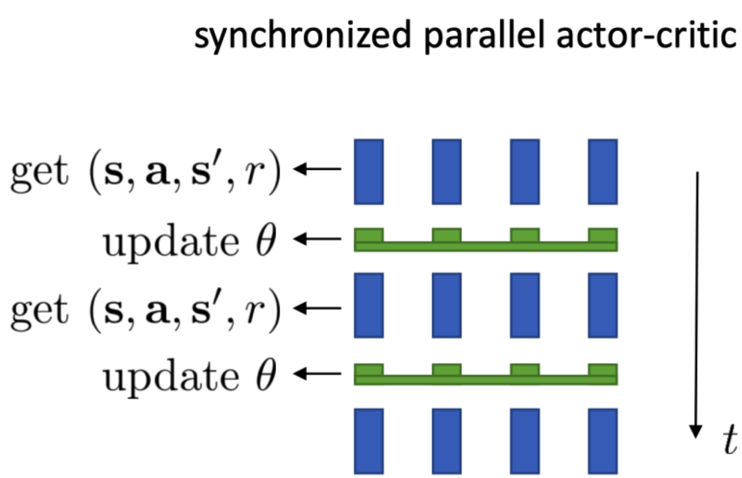
- Shared network design

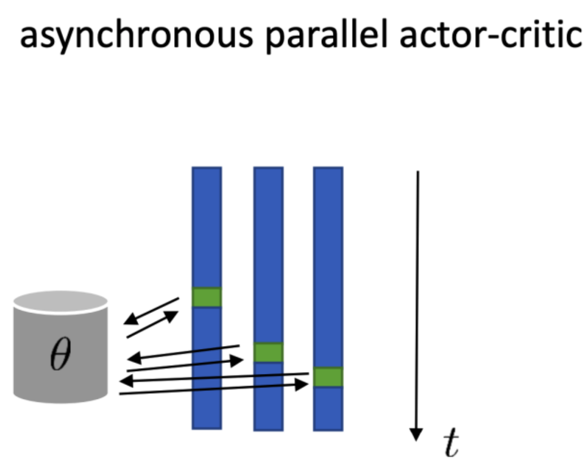


  - computational efficiency, shared representation learning, good empirical performance

Online actor-critic in practice

- Online actor-critic works best with multiple parallel environments
  - consecutive observations are temporally correlated, violating i.i.d.
  - diversity of data allows the agent to generalize better and avoid overfitting
  - improve the signal-to-noise ratio of policy updates
- Updates can be done either synchronously or asynchronously





Off-policy AC

> Version 1. Use multiple gradient steps using IS
>
> 1. sample a batch of data $\{\tau^i\}$ from $\pi_\phi(a_t|s_t)$ (run the policy)
> 2. fit $\hat{V}_\theta^{\pi_\phi}$ to summed rewards in data (n-step TD)
> 3. evaluate $\hat{A}^{\pi_\phi}(s_{i,t}, a_{i,t}) = r(s_{i,t}, a_{i,t}) + \gamma \hat{V}_\theta^{\pi_\phi}(s_{i,t+1}) - \hat{V}_\theta^{\pi_\phi}(s_{i,t})$
> 4. evaluate $\nabla_{\phi'} J(\phi') \approx \sum_{i=1}^{N} \sum_t \left(\frac{\pi_{\phi'}(a_{i,t}|s_{i,t})}{\pi_\phi(a_{i,t}|s_{i,t})}\right) \nabla_{\phi'} \log \pi_{\phi'}(a_{i,t}|s_{i,t})\, \hat{A}^{\pi_\phi}(s_{i,t}, a_{i,t})$
> 5. $\phi \leftarrow \phi + \alpha \nabla_\phi J(\phi)$

Advantages based on old policy
> will get out-dated

can take multiple gradient steps

> So far, we used one batch of policy data for multiple gradient steps

> Can we reuse data from previous batches?
>
> - all the past trial-and-error data

Off-policy AC

> Version 1. Use multiple gradient steps using IS

1. sample a batch of data $\{\tau^i\}$ from $\pi_\phi(a_t|s_t)$ (run the policy)
2. fit $\hat{V}_\theta^{\pi_\phi}$ to summed rewards in data (n-step TD)
3. evaluate $\hat{A}^{\pi_\phi}(s_{i,t}, a_{i,t}) = r(s_{i,t}, a_{i,t}) + \gamma \hat{V}_\theta^{\pi_\phi}(s_{i,t+1}) - \hat{V}_\theta^{\pi_\phi}(s_{i,t})$
4. evaluate $\nabla_{\phi'} J(\phi') \approx \sum_{i=1}^{N} \sum_t \left(\frac{\pi_{\phi'}(a_{i,t}|s_{i,t})}{\pi_\phi(a_{i,t}|s_{i,t})}\right) \nabla_{\phi'} \log \pi_{\phi'}(a_{i,t}|s_{i,t})\ \hat{A}^{\pi_\phi}(s_{i,t}, a_{i,t})$
5. $\phi \leftarrow \phi + \alpha \nabla_\phi J(\phi)$

Advantages based on old policy
> will get out-dated

can take multiple gradient steps

> Idea1: Use KL constraint on policy

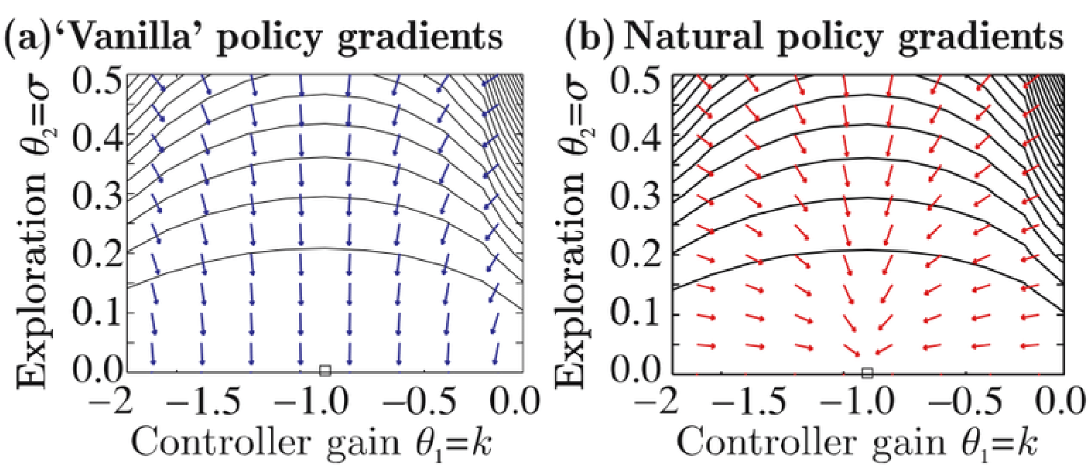
- $\mathbb{E}_{s \sim \pi_\phi}\left[D_{KL}\left(\pi_{\phi'}(\cdot\,|s) \parallel \pi_\phi(\cdot\,|s)\right)\right] \le \delta$
- Trust region policy optimization (TRPO)
- very common. used in LLM preference optimization

Natural policy gradient

> Reformulate to a constrained optimization problem
  - $\max_{\phi'} (\phi' - \phi)^\top \nabla_\phi J(\phi)$
    $s.t. \|\phi - \phi'\|^2 \leq \delta$
  - $\max_{\phi'} (\phi' - \phi)^\top \nabla_\phi J(\phi)$
    $s.t. (\phi' - \phi)^\top F(\phi' - \phi) \leq \delta$

$$F = \mathbb{E}_{s\sim\pi_\phi}\left[\nabla_\phi \log \pi_\phi(a|s) \nabla_\phi \log \pi_\phi(a|s)^\top\right]$$

  - Update becomes $\phi' = \phi + \alpha F^{-1} \nabla_\phi J(\phi)$



Trust region policy optimization (TRPO)

- Constrained optimization problem
  - $\max_{\phi'} (\phi' - \phi)^\top \nabla_\phi J(\phi)$

    $\text{s.t.}\ \mathbb{E}_{s\sim\pi_\phi}\left[D_{KL}\left(\pi_{\phi'}(\cdot\,|s) \parallel \pi_\phi(\cdot\,|s)\right)\right] \leq \delta$

    $D_{KL}\left(\pi_{\phi'}(\cdot\,|s) \parallel \pi_\phi(\cdot\,|s)\right) = \frac{1}{2}\Delta\phi^\top \underbrace{\nabla^2_{\phi'} \mathrm{D}_{KL}(\pi_{\phi'}(\cdot\,|s) \parallel \pi_\phi(\cdot\,|s))}_{F} \Delta\phi$

  - Update $\phi' = \phi + \eta F^{-1} \nabla_\phi J(\phi), \quad \eta = \sqrt{\frac{2\delta}{\nabla_\phi J^\top F^{-1} \nabla_\phi J}}$ (to meet the constraint)

  - it is a constrained non-convex optimization problem
  - complicated and hard/slow to solve

  - PPO designs a surrogate objective function

Off-policy AC

> Version 1. Use multiple gradient steps using IS

1. sample a batch of data $\{\tau^i\}$ from $\pi_\phi(a_t|s_t)$ (run the policy)
2. fit $\hat{V}_\theta^{\pi_\phi}$ to summed rewards in data (n-step TD)
3. evaluate $\hat{A}^{\pi_\phi}(s_{i,t}, a_{i,t}) = r(s_{i,t}, a_{i,t}) + \gamma \hat{V}_\theta^{\pi_\phi}(s_{i,t+1}) - \hat{V}_\theta^{\pi_\phi}(s_{i,t})$
4. evaluate $\nabla_{\phi'} J(\phi') \approx \sum_{i=1}^{N} \sum_t \left(\frac{\pi_{\phi'}(a_{i,t}|s_{i,t})}{\pi_\phi(a_{i,t}|s_{i,t})}\right) \nabla_{\phi'} \log \pi_{\phi'}(a_{i,t}|s_{i,t}) \, \hat{A}^{\pi_\phi}(s_{i,t}, a_{i,t})$
5. $\phi \leftarrow \phi + \alpha \nabla_\phi J(\phi)$

Advantages based on old policy
> will get out-dated

can take multiple gradient steps

> Idea2: Can we bound the importance weights?

- Doesn't directly constrain policy, but removes incentives
- A key idea behind proximal policy optimization (PPO)

Proximal policy optimization (PPO)

> $\nabla_{\phi'} J(\phi') \approx \sum_{i=1}^{N} \sum_t \left(\frac{\pi_{\phi'}(a_{i,t}|s_{i,t})}{\pi_\phi(a_{i,t}|s_{i,t})}\right) \nabla_{\phi'} \log \pi_{\phi'}(a_{i,t}|s_{i,t})\, \hat{A}^{\pi_\phi}(s_{i,t}, a_{i,t})$

$$J(\phi') \approx \sum_{t,i} \underbrace{\frac{\pi_{\phi'}(a_{i,t}|s_{i,t})}{\pi_\phi(a_{i,t}|s_{i,t})} \hat{A}^{\pi_\phi}(s_{i,t}, a_{i,t})}_{[1]}$$

$$p_\theta(x)\nabla_\theta(\log p_\theta(x)) = \nabla_\theta p_\theta(x)$$

> Trick 1: clip the importance weights:

- $J(\phi') \approx \sum_{t,i} \underbrace{clip\left(\frac{\pi_{\phi'}(a_{i,t}|s_{i,t})}{\pi_\phi(a_{i,t}|s_{i,t})}, 1-\epsilon, 1+\epsilon\right) \hat{A}^{\pi_\phi}(s_{i,t}, a_{i,t})}_{[2]}$
- policy no longer incentivized to deviate significantly

> Trick 2: take minimum w.r.t. original objective

- in rare event where clipping makes objective better
- $J(\phi') \approx \sum_{t,i} \min([1], [2])$

Proximal policy optimization (PPO)

> $J(\phi') \approx \sum_{t,i} \min([1],[2])$

> Trick 3: generalized advantage estimation (GAE)
  - Fit V^{π} with Monte-Carlo or bootstrapping
    e.g. n-step TD: $\hat{V}_{\theta}^{\pi}(s_i) \leftarrow \sum_{t'=t}^{t+n-1} \gamma^{t'-t} r(s_{i,t'}, a_{i,t'}) + \gamma^{\mathrm{n}} \hat{V}_{\theta}^{\pi}(s_{i,t+n})$

  - Then, use varying horizon to estimate advantage:
    $\hat{A}^{\pi_\phi}(s_{i,t}, a_{i,t}) = r(s_{i,t}, a_{i,t}) + \gamma \hat{V}_{\theta}^{\pi_\phi}(s_{i,t+1}) - \hat{V}_{\theta}^{\pi_\phi}(s_{i,t})$
    $\rightarrow \hat{A}_n^{\pi_\phi}(s_{i,t}, a_{i,t}) = \sum_{t'=t}^{t+n} \gamma^{t'-t} r(s_{i,t'}, a_{i,t'}) + \gamma^{\mathrm{n}} \hat{V}_{\theta}^{\pi_\phi}(s_{i,t+1}) - \hat{V}_{\theta}^{\pi_\phi}(s_{i,t})$

    GAE: $\hat{A}_{GAE}^{\pi_\phi}(s_{i,t}, a_{i,t}) = \sum_{n=1}^{\infty} \omega_n \hat{A}_n^{\pi_\phi}(s_{i,t}, a_{i,t})$

$\omega_n \propto \lambda^{\mathrm{n}-1}, \lambda \in (0,1)$

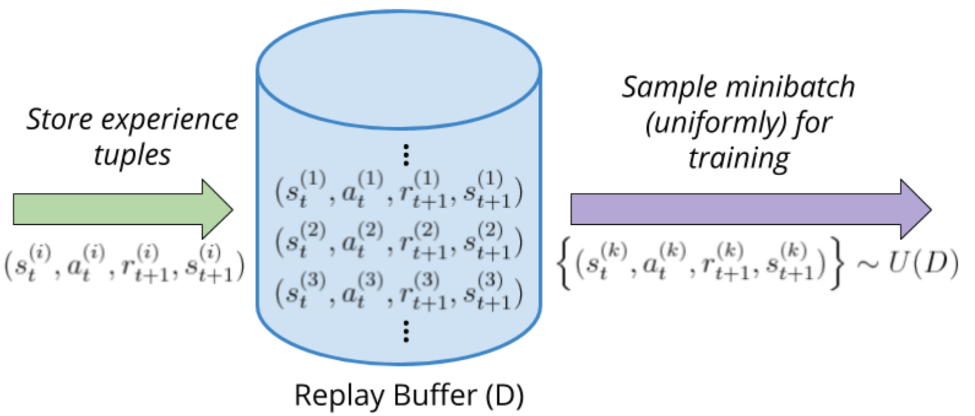
Proximal policy optimization (PPO)

> Algorithm:
1. sample a batch of data $\{\tau^i\}$ from $\pi_\phi(a_t|s_t)$ (run the policy)
2. fit $\hat{V}_\theta^{\pi_\phi}$ to summed rewards in data (n-step TD)
3. **evaluate $\hat{A}_{GAE}^{\pi_\phi}(s_{i,t}, a_{i,t})$**
4. **compute the clipped surrogate objective gradient $\nabla_{\phi'} J(\phi')$**
5. update policy with M gradient steps $\phi \leftarrow \phi + \alpha \nabla_\phi J(\phi)$

Off-policy AC

> Version 2. Replay buffer

1. collect experience $\{s_i, a_i\}$ from π_ϕ and add to replay buffer
2. sample a batch $\{s_i, a_i, r_i, s_i{}'\}$ from buffer $\mathcal{R}$
3. update $\hat{V}_\theta^{\pi_\phi}$ using targets $r(s_i\,, a_i\,) + \gamma \hat{V}_\theta^{\pi}(s_i')$ for each s_i
4. evaluate $\hat{A}^{\pi_\phi}(s_i, a_i) = r(s_i, a_i) + \gamma \hat{V}_\theta^{\pi_\phi}(s_i) - \hat{V}_\theta^{\pi_\phi}(s_i)$
5. $\nabla_\phi J(\phi) \approx \sum_{i=1}^{N} \nabla_\phi \log \pi_\phi(a_i|s_i) \hat{A}^{\pi_\phi}(s_i, a_i)$
6. $\phi \leftarrow \phi + \alpha \nabla_\phi J(\phi)$



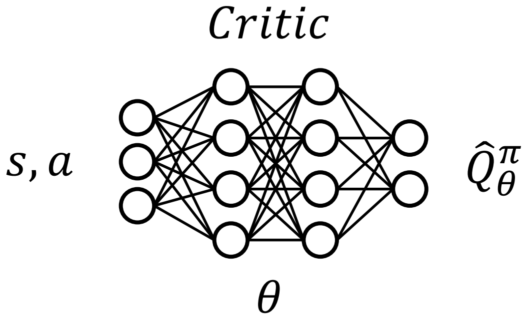
Off-policy AC

> Version 2. Replay buffer

1. collect experience $\{s_i, a_i\}$ from π_ϕ and add to replay buffer
2. sample a batch $\{s_i, a_i, r_i, s_i{}'\}$ from buffer $\mathcal{R}$
3. update $\hat{Q}^{\pi}_{\theta}$ using targets $r(s_i\,, a_i\,) + \gamma \hat{Q}^{\pi}_{\theta}(s_i', a_i')$ for each s_i, a_i
4. evaluate $\hat{A}^{\pi_\phi}(s_i, a_i) = \hat{Q}^{\pi}_{\theta}(s_i, a_i) - \hat{V}^{\pi_\phi}_{\theta}(s_i)$
5. $\nabla_\phi J(\phi) \approx \sum_{i=1}^{N} \nabla_\phi \log \pi_\phi(a_i|s_i) \hat{A}^{\pi_\phi}(s_i, a_i)$
6. $\phi \leftarrow \phi + \alpha \nabla_\phi J(\phi)$

$\hat{V}^{\pi}_{\theta}(s_i') = \hat{Q}^{\pi}_{\theta}(s_i', a_i'),\ \ a_i{}' \sim \pi_\phi$

not from replay buffer $\mathcal{R}$



Off-policy AC

> Version 2. Replay buffer

1. collect experience $\{s_i, a_i\}$ from π_ϕ and add to replay buffer
2. sample a batch $\{s_i, a_i, r_i, s_i{}'\}$ from buffer $\mathcal{R}$
3. update $\hat{Q}_\theta^\pi$ using targets $r(s_i\,, a_i\,) + \gamma \hat{Q}_\theta^\pi(s_i', a_i')$ for each s_i, a_i
4. evaluate $\hat{A}^{\pi_\phi}(s_i, a_i) = \hat{Q}_\theta^\pi(s_i, a_i) - \hat{V}_\theta^{\pi_\phi}(s_i)$
5. $\nabla_\phi J(\phi) \approx \sum_{i=1}^N \nabla_\phi \log \pi_\phi(a_i|s_i)\hat{A}^{\pi_\phi}(s_i, a_i)$
6. $\phi \leftarrow \phi + \alpha \nabla_\phi J(\phi)$

$\nabla_\phi \log \pi_\phi(a_i|s_i)\hat{A}^{\pi_\phi}(s_i, a_i)$

a_i is not from π_ϕ

sample $a_i^\pi \sim \pi_\phi(a|s_i)$

$\nabla_\phi \log \pi_\phi(a_i^\pi|s_i)\hat{A}^{\pi_\phi}(s_i, a_i^\pi)$

in practice: use without baseline

$\nabla_\phi \log \pi_\phi(a_i^\pi|s_i)\hat{Q}^{\pi_\phi}(s_i, a_i^\pi)$

(higher variance, but convenient)

Off-policy AC

> Version 2. Replay buffer

1. collect experience $\{s_i, a_i\}$ from π_ϕ and add to replay buffer
2. sample a batch $\{s_i, a_i, r_i, s_i{}'\}$ from buffer $\mathcal{R}$
3. update $\hat{Q}_\theta^\pi$ using targets $r(s_i\,, a_i\,) + \gamma \hat{Q}_\theta^\pi(s_i', a_i')$ for each s_i, a_i
4. $\nabla_\phi J(\phi) \approx \sum_{i=1}^N \nabla_\phi \log \pi_\phi(a_i|s_i) \hat{Q}_\theta^{\pi_\phi}(s_i, a_i)$ where $a_i \sim \pi_\phi(a|s_i)$
5. $\phi \leftarrow \phi + \alpha \nabla_\phi J(\phi)$

> Any remaining problems?

- s_i didn't come from $p_\phi(s)$
- nothing we can do here, just accept it
- intuition: we want optimal policy on $p_\phi(s)$, but we get optimal policy on a broader distribution

Off-policy AC

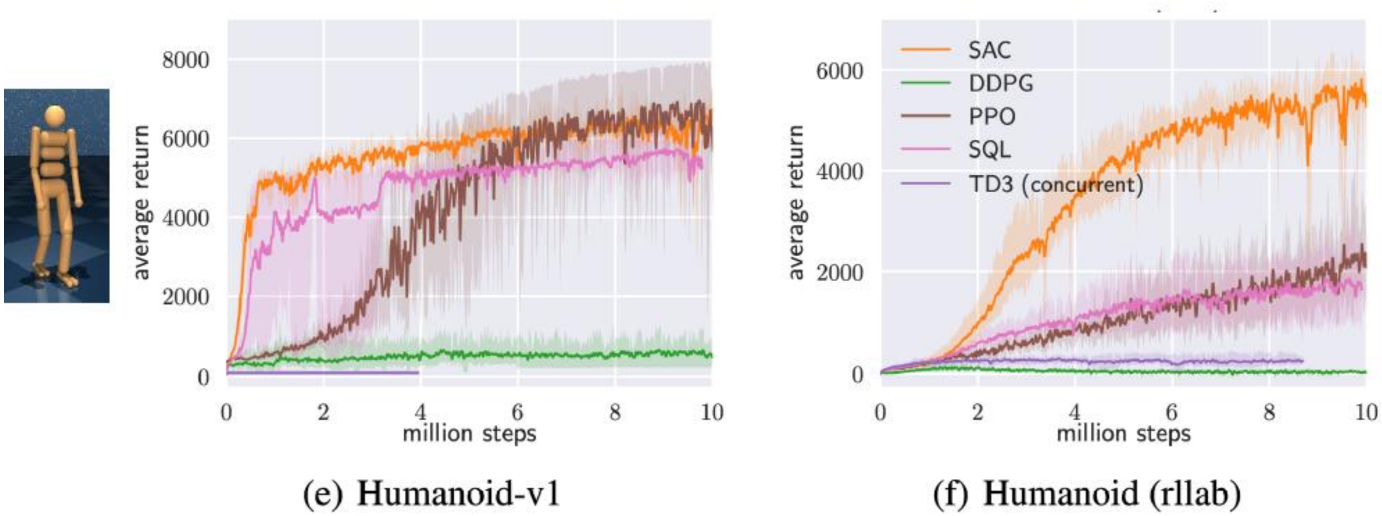
- Version 2. Replay buffer
  1. collect experience $\{s_i, a_i\}$ from π_ϕ and add to replay buffer
  2. sample a batch $\{s_i, a_i, r_i, s_i{}'\}$ from buffer $\mathcal{R}$
  3. update $\hat{Q}^{\pi}_{\theta}$ using targets $r(s_i\,, a_i\,) + \gamma \hat{Q}^{\pi}_{\theta}(s_i', a_i')$ for each s_i, a_i
  4. $\nabla_\phi J(\phi) \approx \sum_{i=1}^{N} \nabla_\phi \log \pi_\phi(a_i|s_i) \hat{Q}^{\pi_\phi}_{\theta}(s_i, a_i)$ where $a_i \sim \pi_\phi(a|s_i)$
  5. $\phi \leftarrow \phi + \alpha \nabla_\phi J(\phi)$

- Implementation details
  - How to fit Q-functions better?
  - To estimate the gradient better, use reparameterization trick

- Example practical algorithm:
  - Soft Actor-critic: off-policy maximum entropy deep RL with a stochastic actor (Haarnoja, et al., 2018)

Off-policy AC

- Off-policy with replay buffer (SAC) can be far more data efficient
- They can also generally be a lot harder to tune hyperparameters, less stable (than PPO)

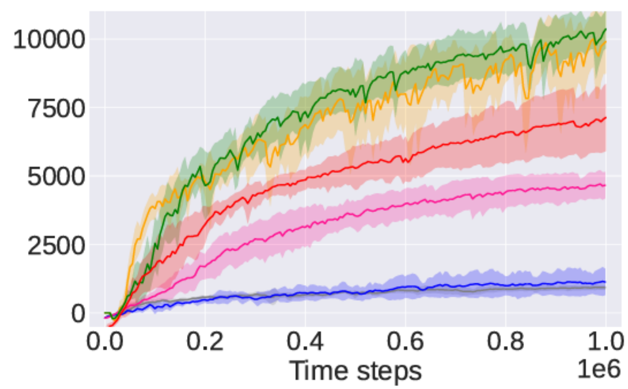


<- more off-policy
(i.e. with replay buffer)

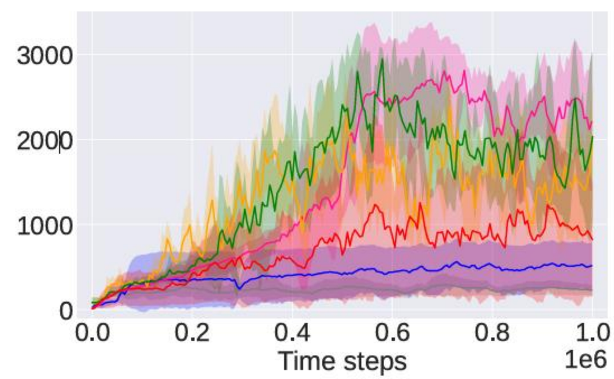
<- less off-policy
(i.e. no replay buffer)

Off-policy AC

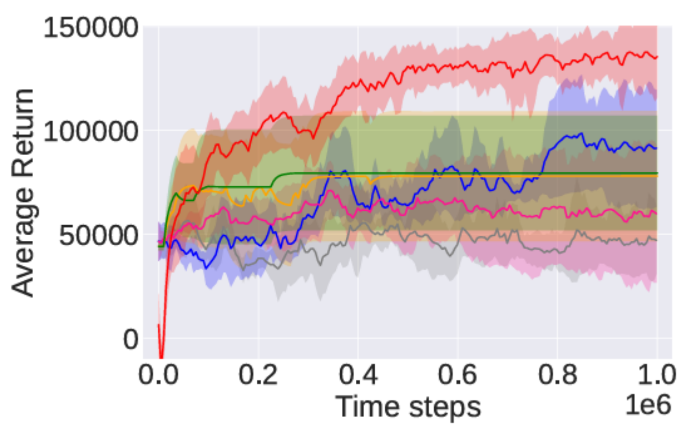
> Learning curves of algorithms (no free lunch theorem)



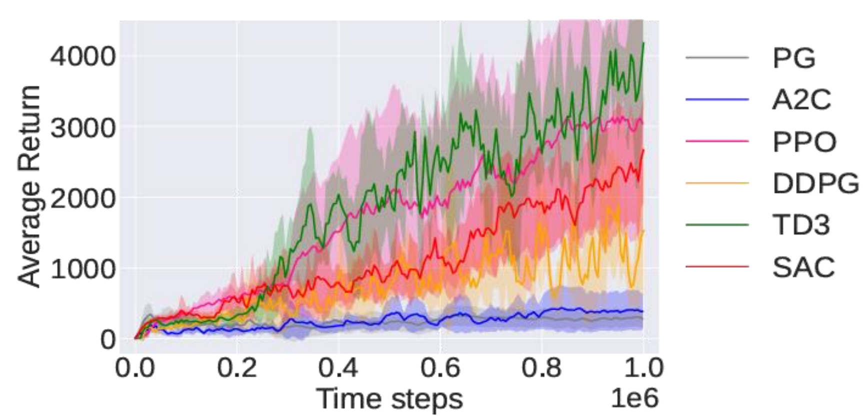
(b) HalfCheetah



(c) Hopper



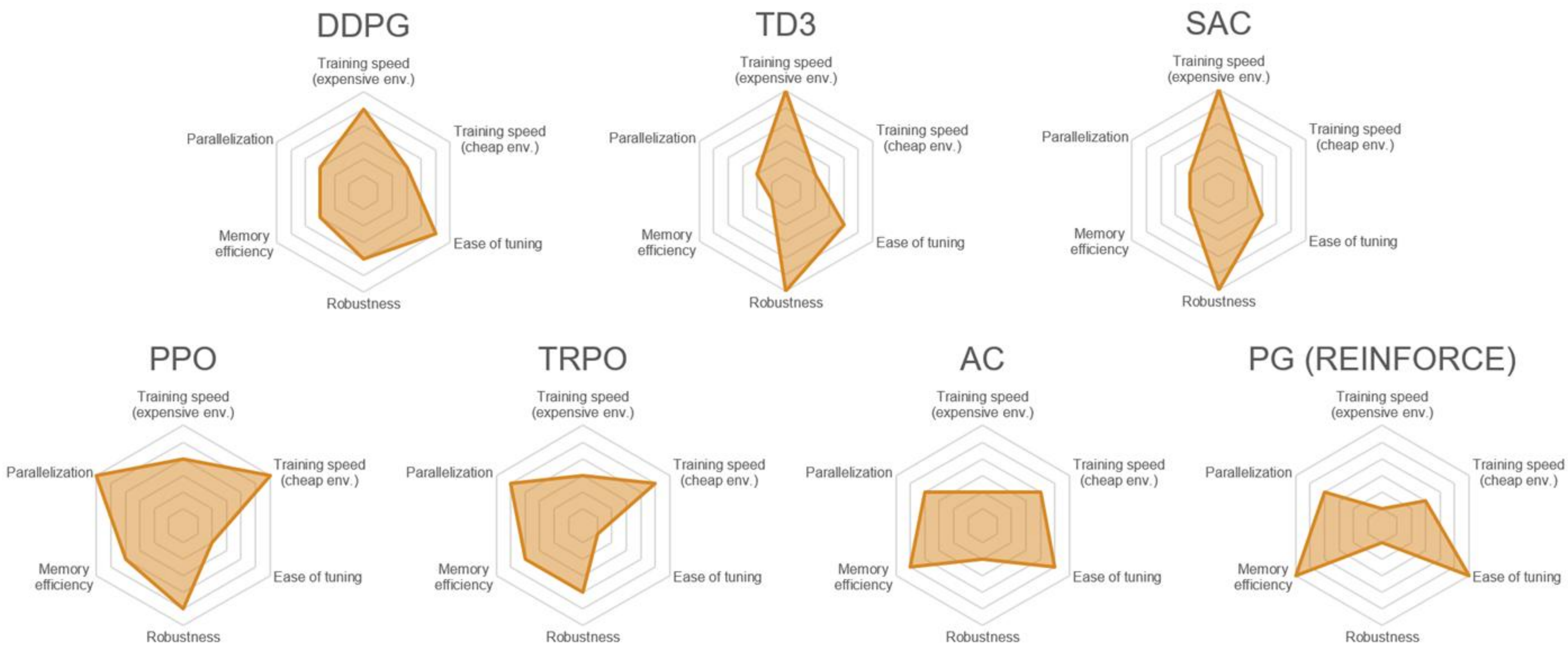
(d) HumanoidStandup



(j) Walker2d

Off-policy AC

> Continuous state space



General RL tips

- Start with the simple and fast algorithms
- DQN, DDPG and PPO exhibit good performance overall
  - DQN, DDPG are easy to tune
  - PPO is harder to tune, but stable
  - can perform well with smaller model
  - good choice for cheap envs.
- TD3 and SAC are good choices for expensive envs.
  - SAC generates stochastic policy that can be useful for exploration
  - harder to tune but can be more memory efficient
  - both benefit from larger network architectures
  - sensitive to input normalization

- Even with the same algorithm, performance may vary significantly depending on the implementation details! (Use the verified libraries)

Trade-off between algorithms

- Sampling efficiency
  - this is only a general trend
  - more efficient doesn't necessarily mean shorter wall clock time