

Lecture 14: Policy Optimization IV: State of the Arts

Bolei Zhou

The Chinese University of Hong Kong

bzhou@ie.cuhk.edu.hk

March 15, 2020

This Week's Plan: Walking through the State of the Art RL Methods

Two lines of works on policy optimization:

- ① Policy Gradient→TRPO→ACKTR→PPO
- ② Q-learning→DDPG→TD3→SAC

State of the Art (SOTA) for Policy Optimization

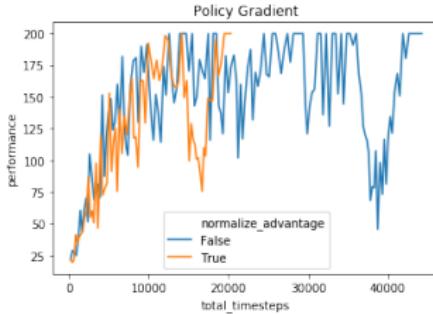
- ① Policy Gradient→TRPO→ACKTR→PPO
 - ① **TRPO**: Trust region policy optimization. Schulman, L., Moritz, Jordan, Abbeel. 2015
 - ② **ACKTR**: Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. Y. Wu, E. Mansimov, S. Liao, R. Grosse, and J. Ba. 2017
 - ③ **PPO**: Proximal policy optimization algorithms. Schulman, Wolski, Dhariwal, Radford, Klimov. 2017
- ② Q-learning→DDPG→TD3→SAC
 - ① **DDPG**: Deterministic Policy Gradient Algorithms, Silver et al. 2014
 - ② **TD3**: Addressing Function Approximation Error in Actor-Critic Methods, Fujimoto et al. 2018
 - ③ **SAC**: Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, Haarnoja et al. 2018

Problems with Policy Gradient

- ① Poor sample efficiency as PG is on-policy learning,

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(\tau) r(\tau)]$$

- ② Large policy update or improper step size destroy the training
 - ① This is different from supervised learning where the learning and data are independent
 - ② In RL, step too far \rightarrow bad policy \rightarrow bad data collection
 - ③ May not be able to recover from a bad policy, which collapses the overall performance



Extending Policy Gradient with Importance Sampling

- ① We can modify PG into off-policy learning using importance sampling
- ② **Importance sampling:** IS calculates the expected value of $f(x)$ where x has a data distribution p
 - ① we can sample data from another distribution q and use the probability ratio between p and q to re-calibrate the result

$$\mathbb{E}_{x \sim p}[f(x)] = \mathbb{E}_{x \sim q}\left[\frac{p(x)}{q(x)}f(x)\right]$$

- ② Code example of IS: link
- ③ Using important sampling in policy objective

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[r(\tau)] = \mathbb{E}_{\tau \sim \hat{\pi}}\left[\frac{\pi_\theta(\tau)}{\hat{\pi}(\tau)}r(\tau)\right]$$

Increasing the Robustness with Trust Regions

- ① The behavior policy could be the old policy directly, thus we can have a surrogate objective function

$$\theta = \arg \max_{\theta} J_{\theta_{old}}(\theta) = \arg \max_{\theta} \mathbb{E}_t \left[\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} A_t \right]$$

- ② The estimate might be excessively large when $\pi_{\theta}/\pi_{\theta_{old}}$ is too large
- ③ Solution: to limit the difference between subsequent policies
- ④ For instance, use the Kullbeck-Leibler (KL) divergence to measure the distance between two policies

$$KL(\pi_{\theta_{old}} || \pi_{\theta}) = \sum_a \pi_{\theta_{old}}(a|s) \log \frac{\pi_{\theta}(a|s)}{\pi_{\theta_{old}}(a|s)}$$

Increasing the Robustness with Trust Regions

- ① Thus our objective with trust region becomes, to maximize

$$J_{\theta_{old}}(\theta) = \mathbb{E}_t \left[\frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{old}}(a_t | s_t)} A_t \right]$$

subject to $KL(\pi_{\theta_{old}}(.|s_t) || \pi_\theta(.|s_t)) \leq \delta$

- ② In the trust region, we limit our parameter search within a region controlled by δ . This is the intuition behind algorithms TRPO and PPO



Gradient ascend



Trust region

Trust Region Optimization

- ① Following Taylor's series expansion on both terms above up to the second-order
- ② After some derivations we can have

$$J_{\theta_t}(\theta) \approx g^T(\theta - \theta_t)$$
$$KL(\theta || \theta_t) \approx \frac{1}{2}(\theta - \theta_t)^T H(\theta - \theta_t)$$

where $g = \nabla_{\theta} J_{\theta_t}(\theta)$ and $H = \nabla_{\theta}^2 KL(\theta || \theta_t)$ and θ_t is the old policy parameter

- ③ Then the objective turns to:

$$\theta_{t+1} = \arg \max_{\theta} g^T(\theta - \theta_t) \text{ s.t. } \frac{1}{2}(\theta - \theta_t)^T H(\theta - \theta_t) \leq \delta$$

- ④ This is a quadratic equation and can be solved analytically:

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

Natural Policy Gradient

- ① Natural gradient is the steepest ascent direction with respect to the Fisher information

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

- ② H is the Fisher Information Matrix (FIM) which can be computed explicitly as

$$H = \nabla_\theta^2 KL(\pi_{\theta_t} || \pi_\theta) = E_{a,s \sim \pi_{\theta_t}} \left[\nabla_\theta \log \pi_\theta(a, s) \nabla_\theta \log \pi_\theta(a, s)^T \right]$$

- ③ Learning rate (δ) can be thought of as choosing a step size that is normalized **with respect to the change in the policy**
- ④ This is beneficial because it means that we don't do any parameter updates that will significantly change the output of the policy network.

Natural Policy Gradient

Algorithm 1 Natural Policy Gradient

Input: initial policy parameters θ_0

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

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

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

 Form sample estimates for

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

 Compute Natural Policy Gradient update:

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

end for

- ➊ Sham Kakade. "A Natural Policy Gradient." NIPS 2001
- ➋ A nice read on natural gradient: <https://wiseodd.github.io/techblog/2018/03/14/natural-gradient/>

Trust Region Policy Optimization (TRPO)

- ① FIM and its inverse are very expensive to compute
- ② TRPO estimates the term $x = H^{-1}g$ by solving the following linear equation $Hx = g$
- ③ Consider the optimization for a quadratic equation

Solving $Ax = b$ is equivalent to

$$x = \arg \max_x f(x) = \frac{1}{2}x^T Ax - b^T x$$

$$\text{since } f'(x) = Ax - b = 0$$

- ④ Thus we can optimize the quadratic equation as

$$\min_x \frac{1}{2}x^T Hx - g^T x$$

- ⑤ Use conjugate gradient method to solve it. It is very similar to the gradient ascent but can be done in fewer iterations

Trust Region Policy Optimization (TRPO)

- ① Resulting algorithm is a refined version of natural policy gradient

Algorithm 3 Trust Region Policy Optimization

Input: initial policy parameters θ_0

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

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

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

 Form sample estimates for

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

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

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

 Perform backtracking line search with exponential decay to obtain final update

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

end for

Trust Region Policy Optimization (TRPO)

- ① Schulman, et al. ICML 2015: a lot of proofs
- ② The appendix A of the TRPO paper provides a 2-page proof that establishes the guaranteed monotonic improvement that **the policy update in each TRPO iteration creates a better policy**

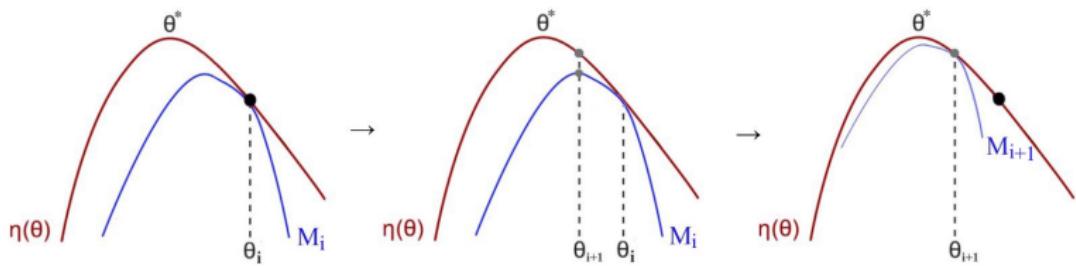
$$J(\pi_{t+1}) - J(\pi_t) \geq M_t(\pi_{t+1}) - M(\pi_t)$$

where $M_t(\pi) = L_{\pi_t}(\pi) - CD_{KL}(\pi_t, \pi)$

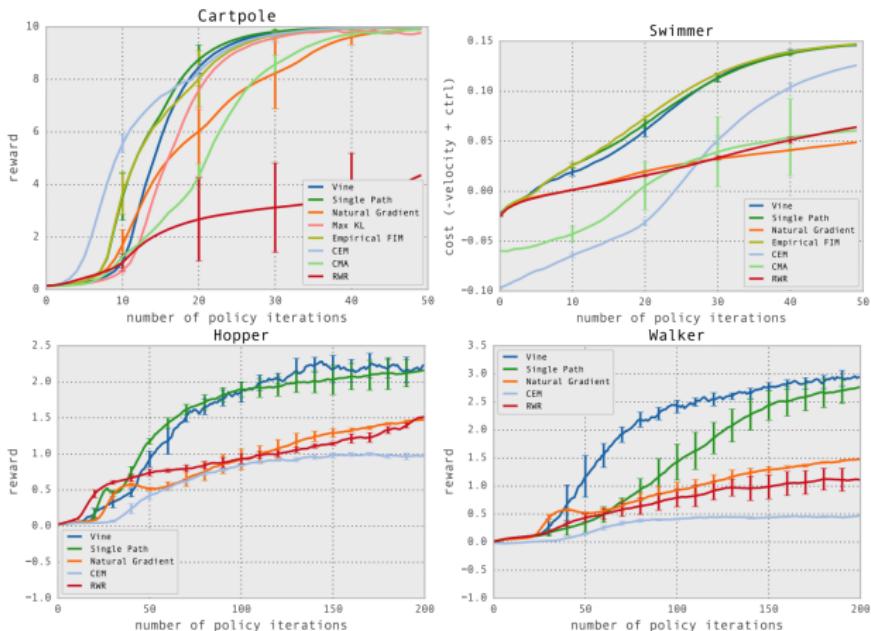
- ③ Thus by maximizing M_t at each iteration, we guarantee that the true objective J is non-decreasing

Trust Region Policy Optimization (TRPO)

- ① It is a type of Minorize-Maximization (MM) algorithm which is a class of methods that includes expectation maximization
- ② The MM algorithm achieves this iteratively by maximizing a lower bound function (the blue line below) approximating the expected reward locally.



Result and Demo of TRPO



① Demo video is at

<https://www.youtube.com/watch?v=KJ15iGGJFvQ>

Limitations of TRPO

① Scalability issue for TRPO

- ① Computing H every time for the current policy model is expensive
- ② It requires a large batch of rollouts to approximate H .

$$H = E_{x \sim \pi_{\theta_t}} \left[(\nabla_{\theta} \log \pi_{\theta}(x))^T (\nabla_{\theta} \log \pi_{\theta}(x)) \right]$$

- ③ Conjugate Gradient(CG) makes implementation more complicated
- ② TRPO does not work well for deep networks

| | <i>B. Rider</i> | <i>Breakout</i> | <i>Enduro</i> | <i>Pong</i> | <i>Q*bert</i> | <i>Seaquest</i> | <i>S. Invaders</i> |
|-------------------------------------|-----------------|-----------------|---------------|-------------|---------------|-----------------|--------------------|
| Random | 354 | 1.2 | 0 | -20.4 | 157 | 110 | 179 |
| Human (Mnih et al., 2013) | 7456 | 31.0 | 368 | -3.0 | 18900 | 28010 | 3690 |
| Deep Q Learning (Mnih et al., 2013) | 4092 | 168.0 | 470 | 20.0 | 1952 | 1705 | 581 |
| UCC-I (Guo et al., 2014) | 5702 | 380 | 741 | 21 | 20025 | 2995 | 692 |
| TRPO - single path | 1425.2 | 10.8 | 534.6 | 20.9 | 1973.5 | 1908.6 | 568.4 |
| TRPO - vine | 859.5 | 34.2 | 430.8 | 20.9 | 7732.5 | 788.4 | 450.2 |

ACKTR: Calculating Natural Gradient with KFAC

- ① Y. Wu, et al. "Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation". NIPS 2017.
- ② ACKTR speeds up the optimization by reducing the complexity of calculating the inverse of the $H(\text{FIM})$ using the Kronecker-factored approximation curvature (K-FAC).

$$F = E_{x \sim \pi_{\theta_t}} \left[(\nabla_{\theta} \log \pi_{\theta}(x))^T (\nabla_{\theta} \log \pi_{\theta}(x)) \right]$$

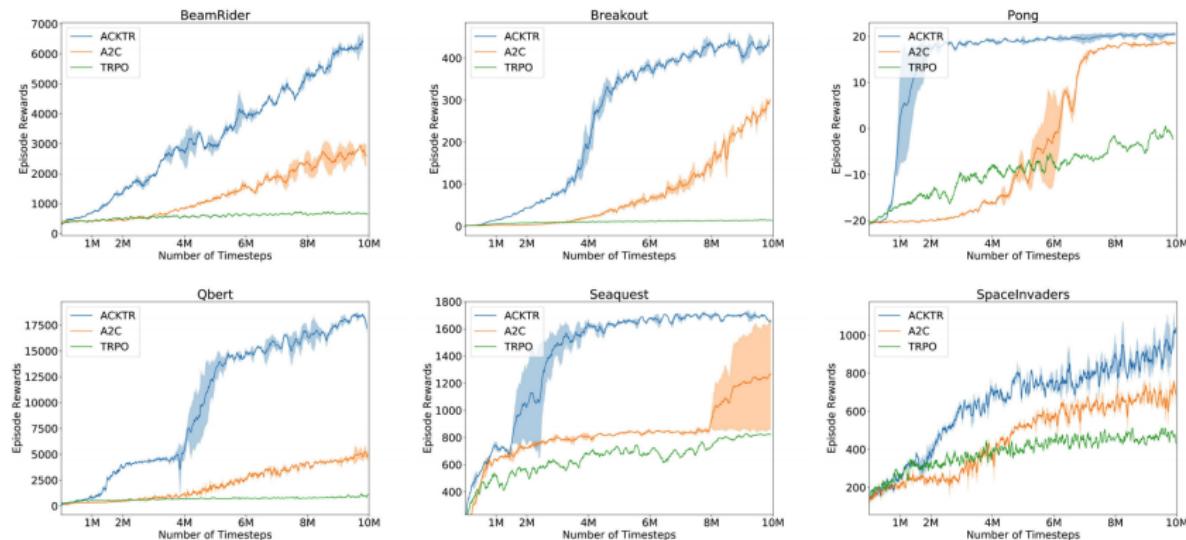
- ③ It is replaced as layer-wise calculation

$$\begin{aligned} F_{\ell} &= \mathbb{E}[\text{vec}\{\nabla_W L\} \text{vec}\{\nabla_W L\}^T] = \mathbb{E}[aa^T \otimes \nabla_s L (\nabla_s L)^T] \\ &\approx \mathbb{E}[aa^T] \otimes \mathbb{E}[\nabla_s L (\nabla_s L)^T] := A \otimes S := \hat{F}_{\ell}, \end{aligned}$$

where A denotes $\mathbb{E}[aa^T]$ and S denotes $\mathbb{E}[\nabla_s L (\nabla_s L)^T]$.

Performance of ACKTR

① Performance of ACKTR



② Introductory link:

<https://blog.openai.com/baselines-acktr-a2c/>

Proximal Policy Optimization (PPO)

- ① The loss function in the Natural Gradient and TRPO

$$\text{maximize}_{\theta} \mathbb{E}_t \left[\frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{old}}(a_t | s_t)} A_t \right]$$

subject to $\mathbb{E}_t [KL[\pi_{\theta_{old}}(\cdot | s_t), \pi_{\theta}(\cdot | s_t)]] \leq \delta$

- ② It also can be written into an unconstrained form,

$$\text{maximize}_{\theta} \mathbb{E}_t \left[\frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{old}}(a_t | s_t)} A_t \right] - \beta \mathbb{E}_t [KL[\pi_{\theta_{old}}(\cdot | s_t), \pi_{\theta}(\cdot | s_t)]]$$

- ③ PPO: include the adaptive KL Penalty, so the optimization will have better insurance that we are optimizing within a trust region

Proximal Policy Optimization (PPO)

Algorithm 4 PPO with Adaptive KL Penalty

Input: initial policy parameters θ_0 , initial KL penalty β_0 , target KL-divergence δ
for $k = 0, 1, 2, \dots$ **do**

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

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

 Compute policy update

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

 by taking K steps of minibatch SGD (via Adam)

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

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

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

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

end if

end for

- ① Same performance as TRPO, but solved much faster by first-order optimization (SGD)

PPO with clipping

- ① Let $r_t(\theta)$ denote the probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ so different surrogate objectives:
 - ① PG without trust region: $L_t(\theta) = r_t(\theta)\hat{A}_t$
 - ② KL constraint: $L_t(\theta) = r_t(\theta)\hat{A}_t$ s.t. $KL[\pi_{\theta_{old}}, \pi_\theta] \leq \delta$
 - ③ KL penalty: $L_t(\theta) = r_t(\theta)\hat{A}_t - \beta KL[\pi_{\theta_{old}}, \pi_\theta]$
- ② A new objective function to clip the estimated advantage function if the new policy is far away from the old policy (r_t is too large)

$$L_t(\theta) = \min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t \right)$$

- ① If the probability ratio between the new policy and the old policy falls outside the range $(1 - \epsilon)$ and $(1 + \epsilon)$, the advantage function will be clipped.
- ② ϵ is set to 0.2 for the experiments in the PPO paper.

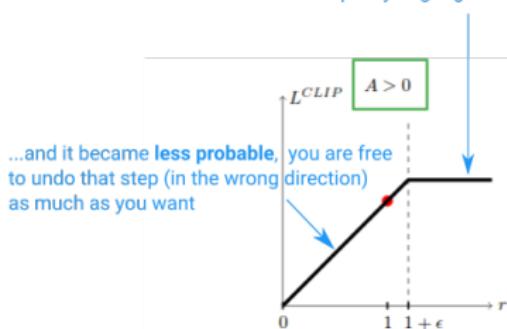
Understanding the clipping

$$L^{CLIP}(\theta) = \min \left(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right)$$

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$$

If the action was **good**....

...and it became **more probable** the last time you took a gradient step, don't keep updating it too far or else the policy might get worse



If the action was **bad**....

...and it became **less probable**, don't keep making it too much less probable or else the policy might get worse (i.e., don't step too far)

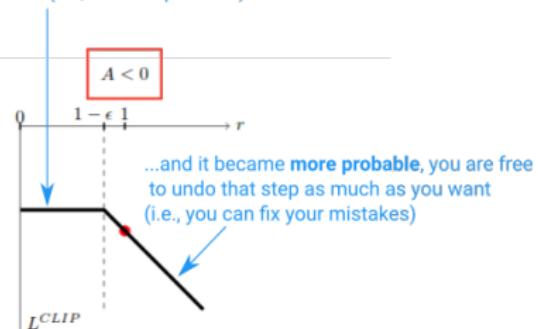


Figure 1: Plots showing one term (i.e., a single timestep) of the surrogate function L^{CLIP} as a function of the probability ratio r , for positive advantages (left) and negative advantages (right). The red circle on each plot shows the direction for the gradient update for the clipping function $\text{clip}(r, 1 - \epsilon, 1 + \epsilon)$.

PPO with clipping

Algorithm 5 PPO with Clipped Objective

Input: initial policy parameters θ_0 , clipping threshold ϵ

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

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

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

 Compute policy update

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

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

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

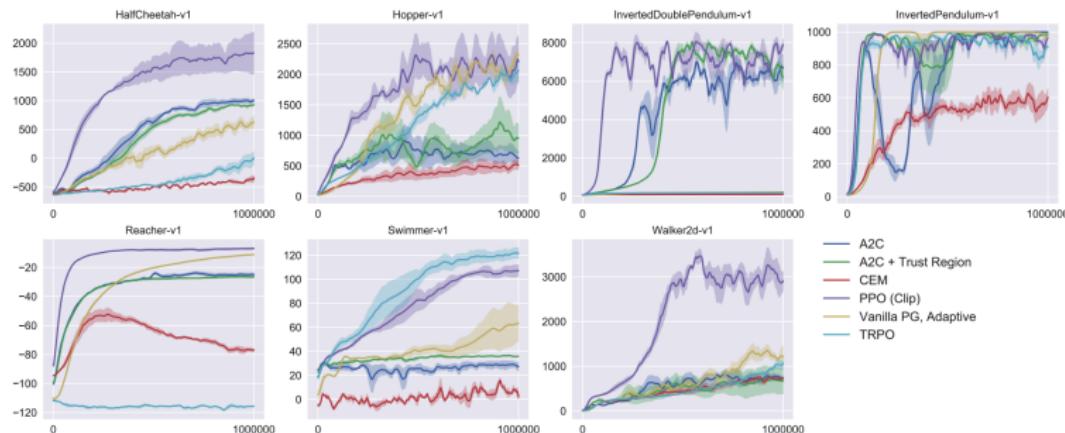
end for

- ① PPOs have the stability and reliability of trust-region methods but are much simpler to implement, requiring only few lines of code change to a vanilla policy gradient implementation

Result of PPO

① Result on the continuous control tasks (MuJuCo)

<https://gym.openai.com/envs/mujoco>



② Demo of PPO at

<https://blog.openai.com/openai-baselines-ppo/>

③ Emergence of Locomotion Behaviours in Rich Environments by DeepMind (Distributed PPO)

① https://www.youtube.com/watch?v=hx_bgoTF7bs

Code of PPO

- ① Paper link of PPO: <https://arxiv.org/abs/1707.06347>
- ② Code example: <https://github.com/cuhkrlcourse/DeepRL-Tutorials/blob/master/14.PPO.ipynb>

State of the Art on Policy Optimization

State-of-the-art RL methods are almost all policy-based

- ① **TRPO**: Schulman, L., Moritz, Jordan, Abbeel (2015). Trust region policy optimization
 - ① comment: Solid math proofs and guarantee, but hard to follow
- ② **ACKTR**: Y. Wu, E. Mansimov, S. Liao, R. Grosse, and J. Ba (2017). Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation.
 - ① comment: numeric optimization-based improvement, scalable to real-problems
- ③ **PPO**: Schulman, Wolski, Dhariwal, Radford, Klimov (2017). Proximal policy optimization algorithms
 - ① comment: Easy to read, elegant design of loss function, easy to implement, widely used

Q-learning → DDPG → TD3 → SAC

- ① **DDPG**: Deterministic Policy Gradient Algorithms, Silver et al. 2014
- ② **TD3**: Addressing Function Approximation Error in Actor-Critic Methods, Fujimoto et al. 2018
- ③ **SAC**: Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, Haarnoja et al. 2018

Deep Deterministic Policy Gradient (DDPG)

- ① Motivation: how to extend DQN to the environment with continuous action space?
- ② DDPG is very similar to DQN, which can be considered as a continuous action version of DQN

$$\textbf{DQN} : a^*(s) = \arg \max_a Q^*(s, a)$$

$$\textbf{DDPG} : a^*(s) = \arg \max_a Q^*(s, a) \approx Q_\phi(s, \mu_\theta(s))$$

- ① a deterministic policy $\mu_\theta(s)$ directly gives the action that maximizes $Q_\phi(s, a)$
- ② as action a is continuous we assume Q-function $Q_\phi(s, a)$ is differentiable with respect to a

From Discrete Action Space to Continuous Action Space

Why we care about the continuous action space

Deep Deterministic Policy Gradient (DDPG)

- ① Thus the DDPG has the following objective:

Q-function: $\min E_{\mathcal{D}}[(Q_{\phi}(s, a) - (r + \gamma(1 - d)Q_{\phi_{targ}}(s', \mu_{\theta_{targ}}(s'))))]$

policy: $\max_{\theta} E_{s \sim \mathcal{D}}[Q_{\phi}(s, \mu_{\theta}(s))]$

- ② Reply buffer and target networks for both value network and policy network are the same as DQN
- ③ DDPG example code: <https://github.com/MoritzTaylor/ddpg-pytorch/blob/master/ddpg.py>

Twin Delayed DDPG (TD3)

- ① One drawback of DDPG is that the learned Q-function begins to dramatically overestimate Q-values, which then leads to the policy breaking, and it is sensitive to hyperparameters
- ② TD3 introduces three critical tricks
 - ① **Clipped Double-Q Learning.** TD3 learns two Q-functions instead of one (hence “twin”), and uses the smaller of the two Q-values to form the targets in the Bellman error loss functions.
 - ② **“Delayed” Policy Updates.** TD3 updates the policy (and target networks) less frequently than the Q-function. The paper recommends one policy update for every two Q-function updates.
 - ③ **Target Policy Smoothing.** TD3 adds noise to the target action, to make it harder for the policy to exploit Q-function errors by smoothing out Q along changes in action.

Twin Delayed DDPG (TD3)

- ① TD3 concurrently learns two Q-functions, Q_{ϕ_1} and Q_{ϕ_2} , by mean square Bellman error minimization
- ② Both Q-functions use a single target, calculated using whichever of the two Q-functions gives a smaller target value:

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{i,targ}}(s', \color{red}{a'(s')})$$

- ③ Target policy smoothing works as follows

$$\color{red}{a'(s')} = \text{clip}(\mu_{\theta_{targ}}(s') + \text{clip}(\epsilon, -c, c), a_{low}, a_{high}), \epsilon \sim \mathcal{N}(0, \sigma)$$

- ① It serves as a regularizer: to avoid the incorrect sharp peak for some action output by the policy

Soft Actor-Critic (SAC)

- ① SAC optimizes a stochastic policy in an off-policy way, which unifies stochastic policy optimization and DDPG-style approaches
- ② SAC incorporates **entropy regularization**
- ③ Entropy is a quantity which measures how random a random variable is, $H(P) = E_{x \sim P}[-\log P(x)]$
- ④ Entropy-regularized RL: the policy is trained to maximize a trade-off between expected return and entropy, a measure of randomness in the policy

$$\pi^* = \arg \max E_{\tau \sim \pi} \left[\sum_t \gamma^t (R(s_t, a_t, s_{t+1}) + \alpha H(\pi(\cdot | s_t))) \right]$$

Soft Actor-Critic (SAC)

- ① Value function is changed to include the entropy bonus from every timestep

$$V^\pi(s) = E_{\tau \sim \pi} \left[\sum_t \gamma^t (R(s_t, a_t, s_{t+1}) + \alpha H(\pi(\cdot | s_t))) \mid s_0 = s \right]$$

- ② The recursive Bellman equation for the entropy-regularized Q^π

$$\begin{aligned} Q^\pi(s, a) &= E_{s' \sim P, a' \sim \pi} [R(s, a, s') + \gamma(Q^\pi(s', a') + \alpha H(\pi(\cdot | s')))] \\ &= E_{s' \sim P, a' \sim \pi} [R(s, a, s') + \gamma(Q^\pi(s', a') - \alpha \log \pi(a' | s'))] \end{aligned}$$

- ③ So the sample update for Q function is

$$Q^\pi(s, a) \approx r + \gamma(Q^\pi(s', \hat{a}') - \alpha \log \pi(\hat{a}' | s')), \hat{a}' \sim \pi(\cdot | s').$$

Soft Actor-Critic (SAC)

- ① SAC concurrently learns a policy π_θ and two Q-functions Q_{ϕ_1} and Q_{ϕ_2}
- ② Like in TD3, the shared target makes use of the clipped double-Q trick and both Q-functions are learned with MSBE minimization

$$L(\phi_i, \mathcal{D}) = E[(Q_\phi(s, a) - y(r, s', d))^2] \quad (1)$$

$$y(r, s', d) = r + \gamma(1 - d) \left(\min_{j=1,2} Q_{\phi_{targ,j}}(s', \hat{a}') - \alpha \log \pi_\theta(\hat{a}'|s') \right), \quad (2)$$

$$\hat{a}' \sim \pi_\theta(\cdot|s'). \quad (3)$$

- ③ Policy is learned through maximizing the expected future return plus expected future entropy, thus maximize $V^\pi(s)$

$$\begin{aligned} V^\pi(s) &= E_{a \sim \pi}[Q^\pi(s, a)] + \alpha H(\pi(\cdot|s)) \\ &= E_{a \sim \pi}[Q^\pi(s, a) - \alpha \log \pi(a|s)]. \end{aligned}$$

Soft Actor-Critic (SAC)

- ① We reparameterize a as sample from a squashed Gaussian policy

$$\hat{a}_\theta(s, \epsilon) = \tanh(\mu_\theta(s) + \sigma_\theta(s) \odot \xi), \epsilon \sim \mathcal{N}(0, I).$$

- ② Reparameterization trick allows us to rewrite the expectation over actions (which contains a pain point: the distribution depends on the policy parameters) into an expectation over noise (which removes the pain point: the distribution now has no dependence on parameters):

$$\begin{aligned} & E_{a \sim \pi_\theta}[Q^{\pi_\theta}(s, a) - \alpha \log \pi_\theta(a|s)] \\ &= E_{\epsilon \sim \mathcal{N}}[Q^{\pi_\theta}(s, \hat{a}_\theta(s, \epsilon)) - \alpha \log \pi_\theta(\hat{a}_\theta(s, \epsilon)|s)] \end{aligned}$$

- ③ The policy is thus optimized as

$$\max_{\theta} E_{s \sim \mathcal{D}, \epsilon \sim \mathcal{N}} \left[\min_{j=1,2} Q_{\phi_j}(s, \hat{a}_\theta(s, \epsilon)) - \alpha \log \pi_\theta(\hat{a}_\theta(s, \epsilon)|s) \right] \quad (4)$$

Brief Intro on Reparameterization Trick

- ① Let's say we want to compute the gradient of the expected value of the function $\nabla_{\theta} E_{x \sim p_{\theta}(x)}[f(x)]$, the difficulty is that x depends on the distribution with parameter θ
- ② We can rewrite the samples of the distribution p_{θ} in terms of a noise variable ϵ which is independent of θ

$$\epsilon \sim q(\epsilon) \quad (5)$$

$$x = g_{\theta}(\epsilon) \quad (6)$$

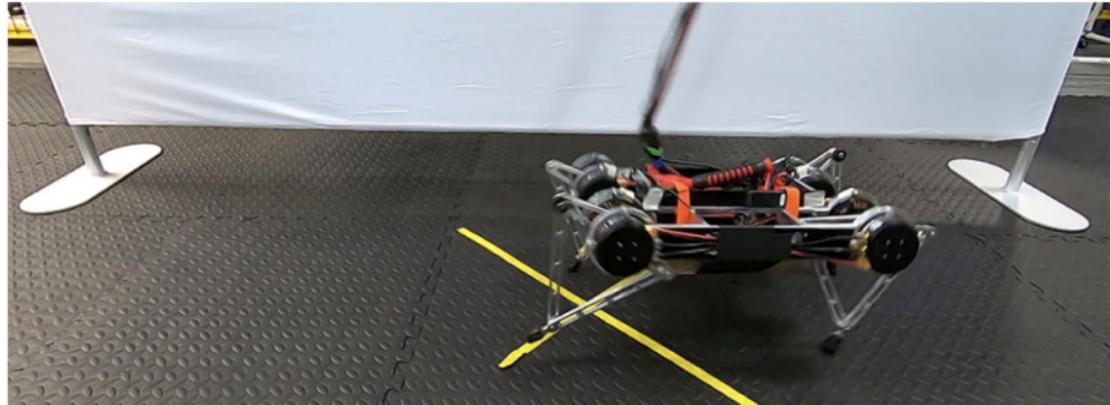
$$\nabla_{\theta} E_{x \sim p_{\theta}(x)}[f(x)] = \nabla_{\theta} E_{\epsilon \sim q(\epsilon)}[f(g_{\theta}(\epsilon))] \quad (7)$$

$$= E_{\epsilon \sim q(\epsilon)}[\nabla_{\theta} f(g_{\theta}(\epsilon))] \quad (8)$$

- ① where x is reparameterized as a function of ϵ and the stochasticity of p_{θ} is replaced by the distribution $q(\epsilon)$, which could be just a standard Gaussian $\mathcal{N}(0, 1)$, thus $g_{\theta}(\epsilon) = \mu_{\theta} + \epsilon \sigma_{\theta}$ where $\epsilon \sim \mathcal{N}(0, 1)$
- ③ On REINFORCE and Reparameterization: <http://stillbreeze.github.io/REINFORCE-vs-Reparameterization-trick/>

Soft Actor-Critic (SAC)

- ① Sample code on SAC: <https://github.com/pranz24/pytorch-soft-actor-critic/blob/master/sac.py>
- ② SAC is known as SOTA for robot learning:
 - ① Learning to Walk in the Real World with Minimal Human Effort.
<https://arxiv.org/pdf/2002.08550.pdf>
 - ② <https://www.youtube.com/watch?v=cwyiq6dCg0c>



Tutorial on SOTA RL algorithms

- ① SpinningUp: Nice implementations and summary of the algorithms from OpenAI: <https://spinningup.openai.com/>



- ② Stable-baseline: <https://stable-baselines.readthedocs.io/>
 - ① Currently in TensorFlow
 - ② PyTorch version is being actively developed:
<https://github.com/hill-a/stable-baselines/issues/733>

