Lecture 14: Policy Optimization IV: State of the Arts

Bolei Zhou

The Chinese University of Hong Kong

bzhou@ie.cuhk.edu.hk

March 9, 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

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

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

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

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

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

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

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

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

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

1. Sham Kakade. "A Natural Policy Gradient." NIPS 2001
2. 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

$$\text{Solving } Ax = b \text{ is equivalent to}$$
$$x = \arg\max_x f(x) = \frac{1}{2}x^T Ax - b^T x$$
$$\text{since } f'(x) = Ax - b = 0$$

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

Trust Region Policy Optimization (TRPO)

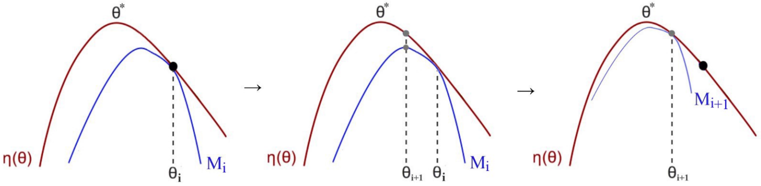
1. Schulman, et al. ICML 2015: a lot of proofs
2. The appendix A of the TRPO paper provides a 2-page proof that establishes the guaranteed monotonic improvement that **the policy update in each TRPO iteration creates a better policy**

$$
\begin{aligned}
J(\pi_{t+1}) - J(\pi_t) \geq & M_t(\pi_{t+1}) - M(\pi_t) \\
\text{where } M_t(\pi) = & L_{\pi_t}(\pi) - C D_{KL}(\pi_t, \pi)
\end{aligned}
$$

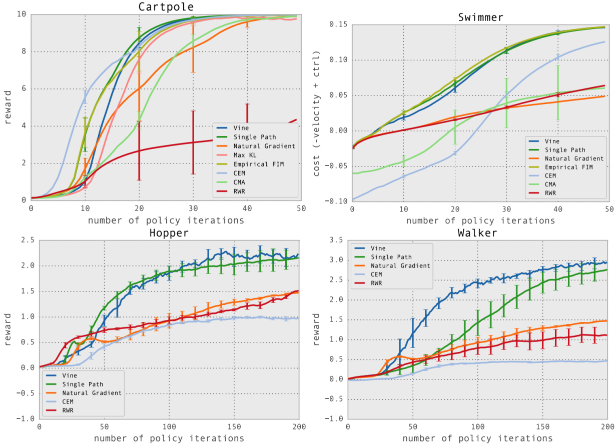
3. Thus by maximizing M_t at each iteration, we guarantee that the true objective J is non-decreasing

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

	B. Rider	*Breakout*	*Enduro*	*Pong*	*Q*bert*	*Seaquest*	*S. Invaders*
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(FIM) using the Kronecker-factored approximation curvature (K-FAC).

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

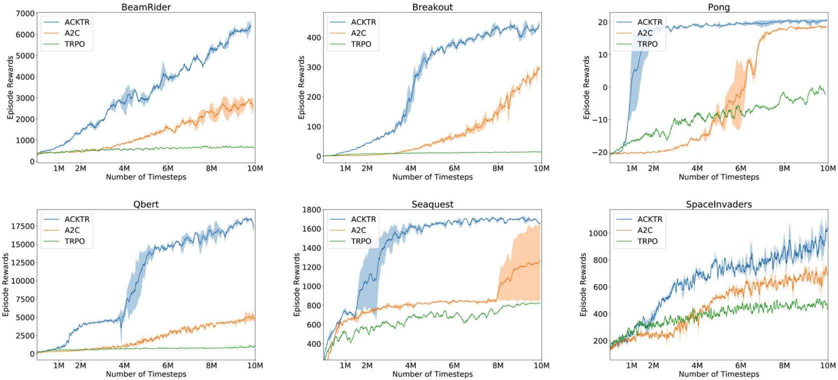
3. It is replaced as layer-wise calculation

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

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

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

$$\text{maximize}_\theta \mathbb{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}A_t\right]$$
$$\text{subject to } \mathbb{E}_t[KL[\pi_{\theta_{old}}(.|s_t), \pi_\theta(.|s_t)]] \leq \delta$$

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

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

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

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)

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

    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

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

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

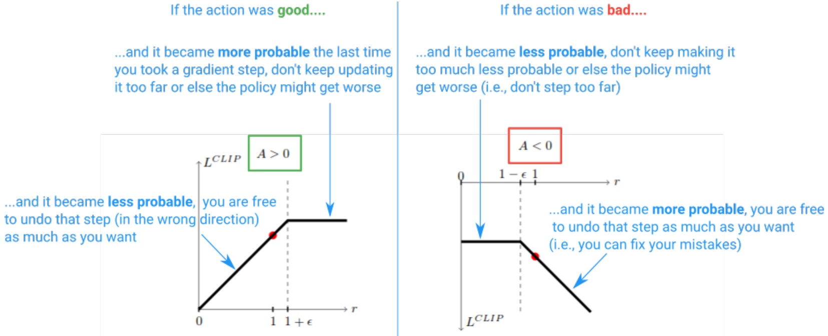


Figure 1: Plots showing one term (i.e., a single timestep) of the surrogate function L^{CLIP} as a function of the probability ratio r, for positive advantages (left) and negative advantages (right). The red circle on each plot shows the starting point for the optimization, i.e., $r = 1$. Note that L^{CLIP} sums many of these terms

PPO with clipping

Algorithm 5 PPO with Clipped Objective

Input: initial policy parameters θ_0, clipping threshold ϵ
for $k = 0, 1, 2, ...$ **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

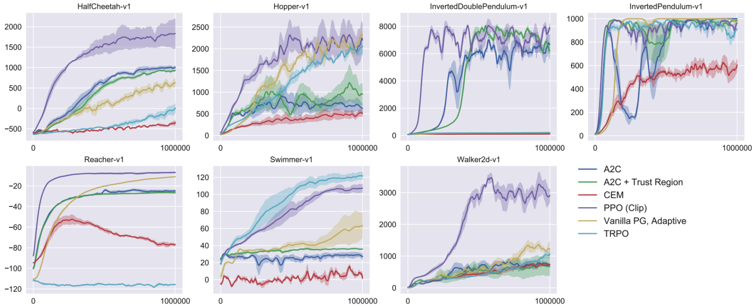
$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_k} \left[\sum_{t=0}^{T} \left[\min(r_t(\theta) \hat{A}_t^{\pi_k}, \mathrm{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right) \hat{A}_t^{\pi_k}) \right] \right]$$

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

DDPG

1. DDPG: `https://github.com/MoritzTaylor/ddpg-pytorch/blob/master/ddpg.py`

TD3

SAC

Tutorial on SOTA RL algorithms

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

OpenAI Spinning Up

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`