Week 6: Policy Optimization II

Bolei Zhou

UCLA

February 10, 2026

Announcement

1. Assignment 2 due by last week, and Assignment 3 is out
    1. `https://github.com/ucla-rlcourse/cs260r-assignment-2026winter/blob/main/assignment3/assignment3.ipynb`
    2. Implement TD3 and PPO

Review of Policy Gradient

1. Softmax policy: weight actions using a linear combination of features $\phi(s,a)^T\theta$

$$\pi_\theta(s,a) = \frac{\exp^{\phi(s,a)^T\theta}}{\sum_{a'} \exp^{\phi(s,a')^T\theta}}$$

2. Gaussian policy:
   1. Mean is a linear combination of state features $\mu(s) = \phi(s)^T\theta$
   2. Variance may be fixed σ^2 or can also be parameterized
   3. Policy is Gaussian, the continuous $a \sim \mathcal{N}(\mu(s), \sigma^2)$
3. Stochastic policy can be considered as a probability (discrete action) or a probability density function (continuous action)
4. Above are the most common stochastic policy parameterizations to use

Review of Policy Gradient

1. In policy optimization, for the policy function π_θ the objective is to maximize

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

   1. $R(\tau)$ could be any reasonable reward function on τ
2. So the gradient is

$$\begin{aligned}\nabla_\theta J(\theta) =& \nabla_\theta \sum_\tau P(\tau;\theta) R(\tau) \\ =& \mathbb{E}_\tau[R(\tau) \nabla_\theta \log P(\tau;\theta)]\end{aligned}$$

   1. **log-derivative trick**: $\log \nabla_\theta f_\theta(x) = f_\theta(x) \nabla_\theta \log f_\theta(x)$
   2. Trajectory distribution: $P(\tau;\theta) = \mu(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t)$
   3. **canceling trick**: $\nabla_\theta \log P(\tau;\theta) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)$

Review of Policy Gradient

1. After decomposing τ we derived that

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \Big[\Big(\sum_{t=0}^{T-1} r_t \Big) \Big(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \Big) \Big]$$

2. After applying the causality, we have

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \Big[\sum_{t=0}^{T-1} G_t \cdot \nabla_\theta \log \pi_\theta(a_t|s_t) \Big]$$

   1. $G_t = \sum_{t'=t}^{T-1} r_{t'}$ is the return for a trajectory at step t

Summary of Policy Gradient

1. The policy gradient has many forms

actual return

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) G_t] \text{ — REINFORCE}$$

q function approximator (boot strapped estimate)

$$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q^{\mathbf{w}}(s,a)] \text{ — Q Actor-Critic}$$

these use MC/TD leraning to estimate Q/A

q - v (more than average)

$$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A^{\mathbf{w}}(s,a)] \text{— Advantage Actor-Critic}$$

$$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) \delta] \text{— TD Actor-Critic}$$

2. Critic uses policy evaluation (e.g. MC or TD learning) to estimate $Q^\pi(s,a)$, $A^\pi(s,a)$, or $V^\pi(s,a)$

Two Approaches to Reinforcement Learning

1. Value-based RL: solve RL through dynamic programming
    1. Classic RL and control theory
    2. Representative algorithms: Deep Q-learning and its variant
    3. Representative researchers: Richard Sutton (no more than 20 pages on PG out of the 500-page textbook), David Silver, from DeepMind
2. Policy-based RL: solve RL mainly through learning
    1. Machine learning and deep learning
    2. Representative algorithms: PG, and its variants TRPO, PPO
    3. Representative researchers: Pieter Abbeel, Sergey Levine, John Schulman, from OpenAI, Berkelely

This Week's Plan: More Recent Advances

Two lines of work on policy optimization:

1. Stochastic policy-based approaches: Policy Gradient$\rightarrow$TRPO$\rightarrow$ACKTR$\rightarrow$PPO
2. Deterministic policy-based (Value-based) approaches: Q-learning$\rightarrow$DDPG$\rightarrow$TD3
3. SAC: an algorithm that optimizes a stochastic policy in an off-policy way, forming a bridge between stochastic policy optimization and DDPG-style approaches

More Recent Advances for Policy Optimization

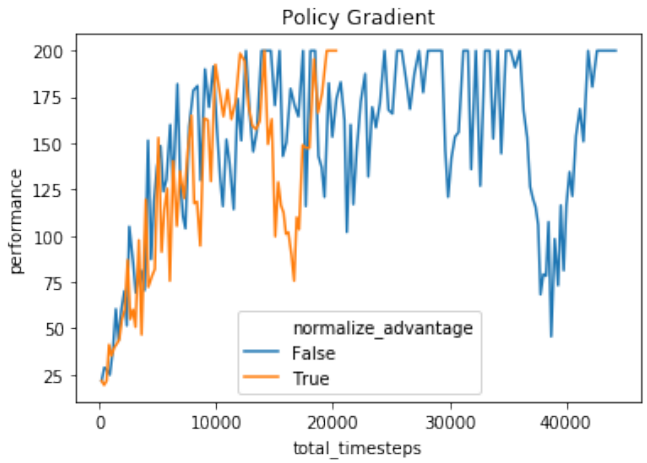
1. Policy Gradient→TRPO→ACKTR→PPO→SAC
    1. **TRPO**: Trust region policy optimization. Schulman, L., Moritz, Jordan, Abbeel. 2015
    2. **ACKTR**: Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. Y. Wu, E. Mansimov, S. Liao, R. Grosse, and J. Ba.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

Problems with Policy Gradient

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

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

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. We may not be able to recover from a bad policy, which collapses the training



Policy Gradient with Importance Sampling

1. We can turn 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 we want to reuse old data and transitions
   1. we can sample data from another distribution q and use the probability ratio between p and q to re-calibrate the result

p(x)/q(x) we know the value, its just p is hard to sample from?

f(x) is like a rewrad function

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

sample from q(x) which is easier to sample from

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

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

this is our current policy

pi hat is a behavior policy

Increasing the Robustness with Trust Regions

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

expectation can be estimated with samples

$$\theta = \arg\max_{\theta} J(\theta; \theta_{old}) = \arg\max_{\theta} \mathbb{E}_{\pi_{\theta_{old}}} \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} R_t \right]$$

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

works because policies are distributions over actions

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

this is like cross entropy loss

Increasing the Robustness with Trust Regions

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

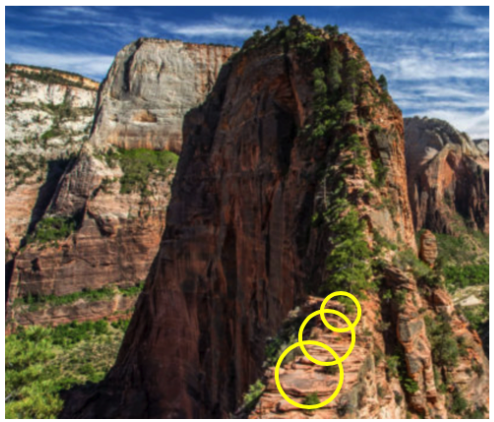
$$J(\theta;\theta_{old}) = \mathbb{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}R_t\right]$$

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

limit updates so that the KL divergence doesn't change too muc

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

Prior Work: Natural Policy Gradient

1. Policy gradient is the steepest ascend in **parameter space** with Euclidean metric:

constrain updates to parameters by \eps

$$d^* = \nabla_\theta J(\theta) = \lim_{\epsilon \to 0} \frac{1}{\epsilon} \arg\max J(\theta + d), s.t. ||d|| \leq \epsilon$$

this is on the parameters

this may change agent behavior a lot

   1. Drawback: it is sensitive to the parameterization of the policy function

2. Natural (policy) gradient is the steepest ascend in **distribution space (policy output)** with KL-divergence as constraint

$$d^* = \arg\max J(\theta + d), s.t. KL[\pi_\theta || \pi_{\theta+d}] = c \tag{1}$$

this is on the output of the distribution

   1. Fixing KL-divergence as a constant c makes sure we move along the distribution space with constant speed, regardless the curvature (thus robust to the reparametrization of the model as we only care about the distribution induced by the parameter)

KL-Divergence as the Distance between Two Distributions

1. KL-divergence measures the "closeness" of two distributions.

$$KL[\pi_\theta || \pi_{\theta'}] = E_{\pi_\theta}[\log \pi_\theta] - E_{\pi_\theta}[\log \pi_{\theta'}]$$

2. Although the KL-divergence is non-symmetric and thus not a true metric, we can use it anyway. This is because as d goes to zero, KL-divergence is asymptotically symmetric. So, within a local neighbourhood, KL-divergence is approximately symmetric
3. We can prove that the second-order Taylor expansion of KL-divergence is

F is the fischer information metric (second derivative of KL?)

$$KL[\pi_\theta || \pi_{\theta+d}] \approx \frac{1}{2} d^T F d$$

where F is the **Fisher Information Matrix** as second order derivative of KL divergence $E_{\pi_\theta}[\nabla \log \pi_\theta \nabla \log \pi_\theta^T]$

Natural Policy Gradient

$$d^* = \arg\max J(\theta + d), s.t. KL[\pi_\theta || \pi_{\theta+d}] = c$$

1. Write the above as Lagrangian form, with the $J(\theta + d)$ approximated by its first order Tyler expansion and the constraint KL-divergence approximated by its second order Tyler expansion

$$\begin{aligned} d^* &= \arg\max_d J(\theta + d) - \lambda(KL[\pi_\theta || \pi_{\theta+d}] - c) \\ &\approx \arg\max_d J(\theta) + \nabla_\theta J(\theta)^T d - \frac{1}{2}\lambda d^T F d + \lambda c \end{aligned}$$

its because KL(P||P) = 0, so we need to take the second derivative so that we get the curivature

2. To solve this maximization we set its derivative wrt. d to zero, then we have the natural policy gradient: $d = \frac{1}{\lambda} F^{-1} \nabla_\theta J(\theta)$

Natural Policy Gradient

1. Natural policy gradient is a **second-order** optimization that is more accurate and works regardless of how the model is parameterized (model invariant).

$$\theta_{t+1} = \theta_t + F^{-1} \nabla_\theta J(\theta)$$

we have to calculate KL divergence (second order is much slower)
moreover inverse is terrible for compute

   1. where $F = E_{\pi_\theta(s,a)}[\nabla \log \pi_\theta(s,a) \nabla \log \pi_\theta(s,a)^T]$ is the Fisher information matrix, also as the second-order derivative of the KL-divergence
   2. F measures the curvature of the policy(distribution) relative to the model parameter θ

2. Natural policy gradient produces the same policy change **regardless of the model parameterization**.
3. Idea behind TRPO. A detailed read on natural gradient: `https://agustinus.kristia.de/blog/natural-gradient/`

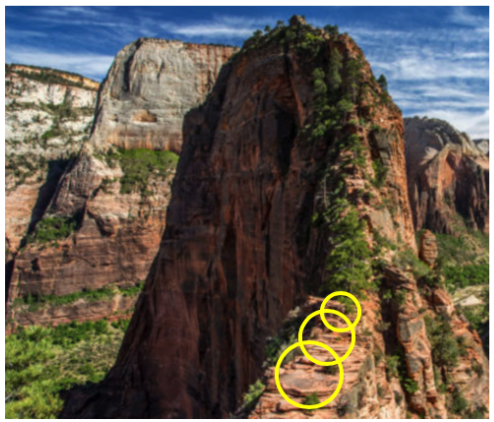
Back to TRPO

1. The objective of TRPO is to maximize

$$J(\theta;\theta_{old}) = \mathbb{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}R_t\right]$$
$$\text{subject to } KL(\pi_{\theta_{old}}(.|s_t)||\pi_\theta(.|s_t)) \leq \delta$$

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

Natural Policy Gradient part of TRPO

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

$$J(\theta;\theta_t) \approx g^T(\theta-\theta_t)$$

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

   where $g = \nabla_\theta J_{\theta_t}(\theta)$ and $H = \nabla^2_\theta KL(\theta||\theta_t)$ and θ_t is the old policy parameter
3. Then the objective becomes:

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

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 part of TRPO

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 any parameter update won't significantly change the output of the policy network

Natural Policy Gradient part of TRPO

"update policy in a way that is invariant to how parameters are represented"

Algorithm 1 Natural Policy Gradient

Input: initial policy parameters θ_0

for $k = 0, 1, 2, \ldots$ **do** mc like policy -> get the rollouts

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 this is the critic neural network (to calculate the value function, which we then use to calculate the q-value function)

- policy gradient $\hat{g}_k$ (using advantage estimates) steepest ascent in the parameter space
- and KL-divergence Hessian / Fisher Information Matrix $\hat{H}_k$

Compute Natural Policy Gradient update:

This is the Natural Gradient

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

This is the Weight using the KL Divergence

end for

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

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 problem for a quadratic equation

$$\text{Solving } Ax = b \text{ is equivalent to}$$

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

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

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

get an approximation of Hhat_k^{-1}ghat_k with Conjugate gradient method becuase its faster

Appendix of 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**

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

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

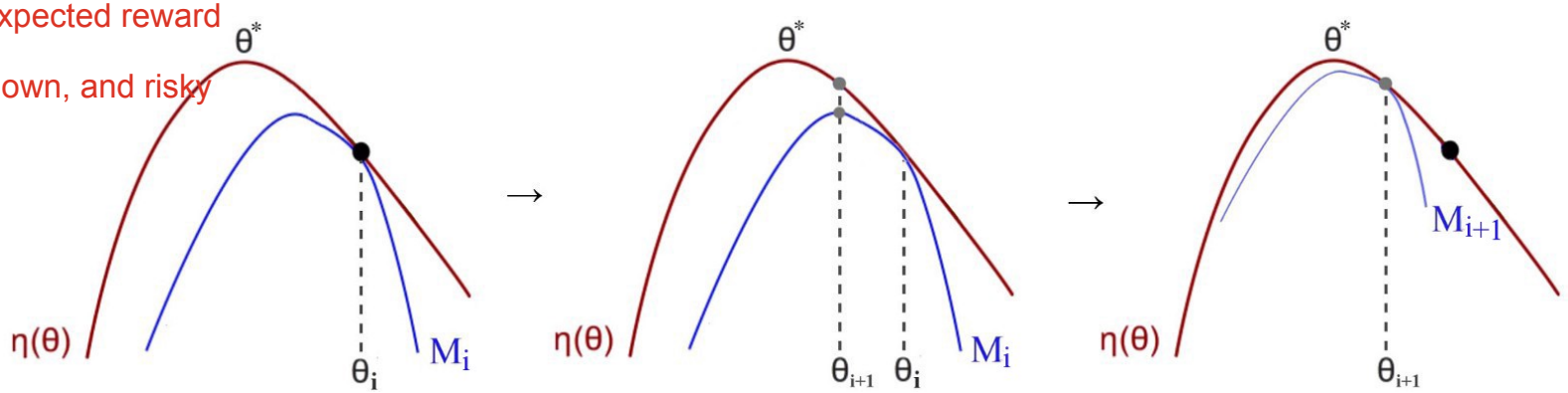
Appendix of Trust Region Policy Optimization (TRPO)

maximize a surrogate function?

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 surrogate function (the blue line below) approximating the expected reward locally.

nu(\theta) is expected reward

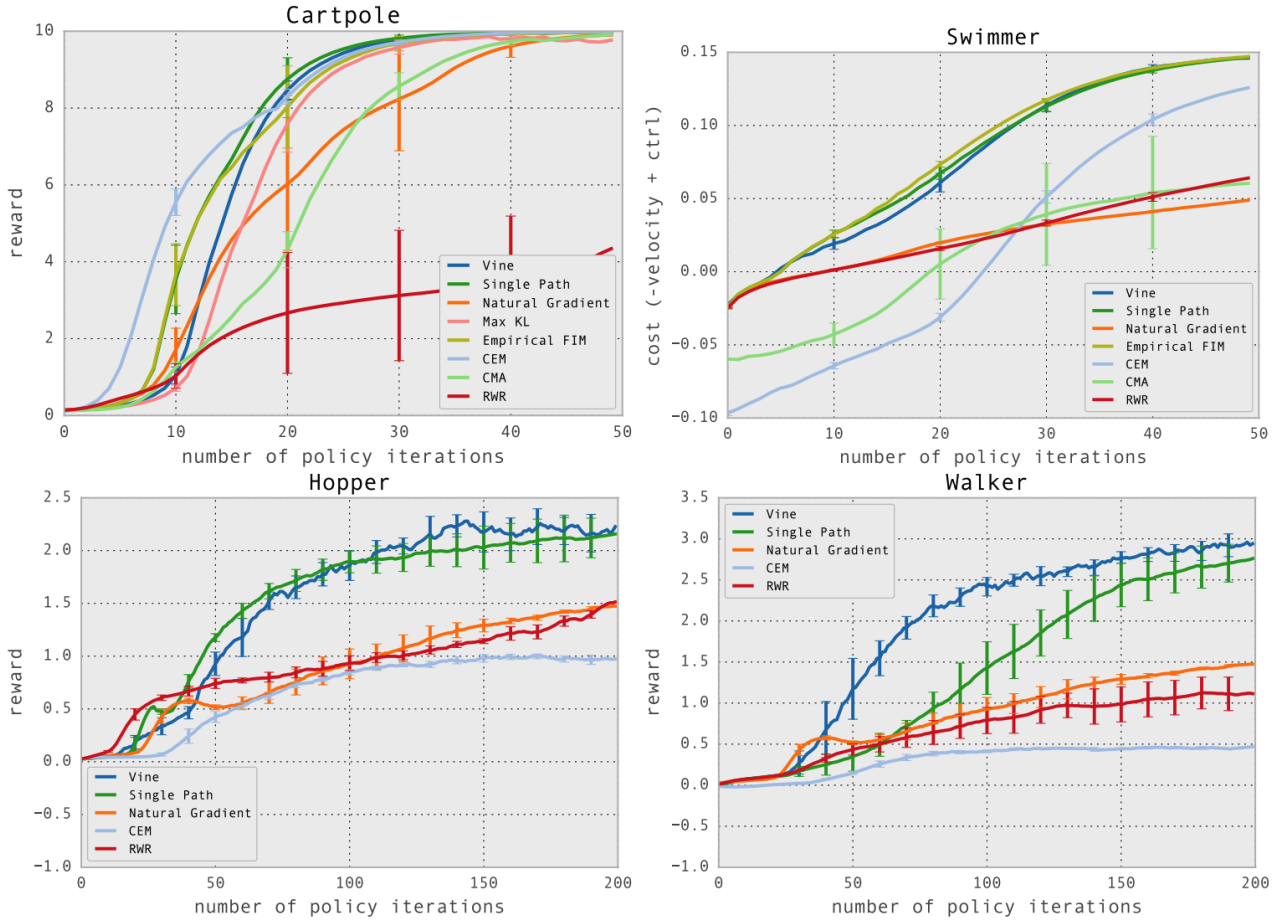
but complex, unknown, and risky to climb up itself



instead we build M_i (blue curve),the surrgate function such that its always leq than nu

M = L(\theta) =- C KL(\theta_old || \theta_new)

Result and Demo of TRPO



1. Demo video is at `https://www.youtube.com/embed/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.

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

   3. Conjugate Gradient(CG) makes implementation more complicated
2. TRPO does not work well on some of the tasks compared to DQN

	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 Fisher Information Matrix (FIM) using the Kronecker-factored approximation curvature (K-FAC).

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

calculate Natural Gradient with a different approximation method

1. It is replaced as layer-wise calculation

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

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

Optimizing Neural Networks with Kronecker-factorized Approximate Curvature.

1. Martens et al.ICML'15: `https://arxiv.org/pdf/1503.05671.pdf`
2. Stochastic Gradient Descend (SGD) is the first-order optimization, which is widely used for network optimization; Natural Gradient Descend converges faster in terms of iterations than first-order method, by taking into consideration the curvature of the loss function. However it involves computing the inverse of the Fisher Information Matrix $\theta_{t+1} = \theta_t + \alpha F^{-1} \nabla_\theta J(\theta)$, where $F = E_{\pi_\theta(s,a)}[\nabla \log \pi_\theta(s,a) \nabla \log \pi_\theta(s,a)^T]$
3. K-FAC (Kronecker-factorized Approximate Curvatue) approximates Fisher Information Matrix for large-scale neural network optimization

Optimizing Neural Networks with Kronecker-factorized Approximate Curvature.

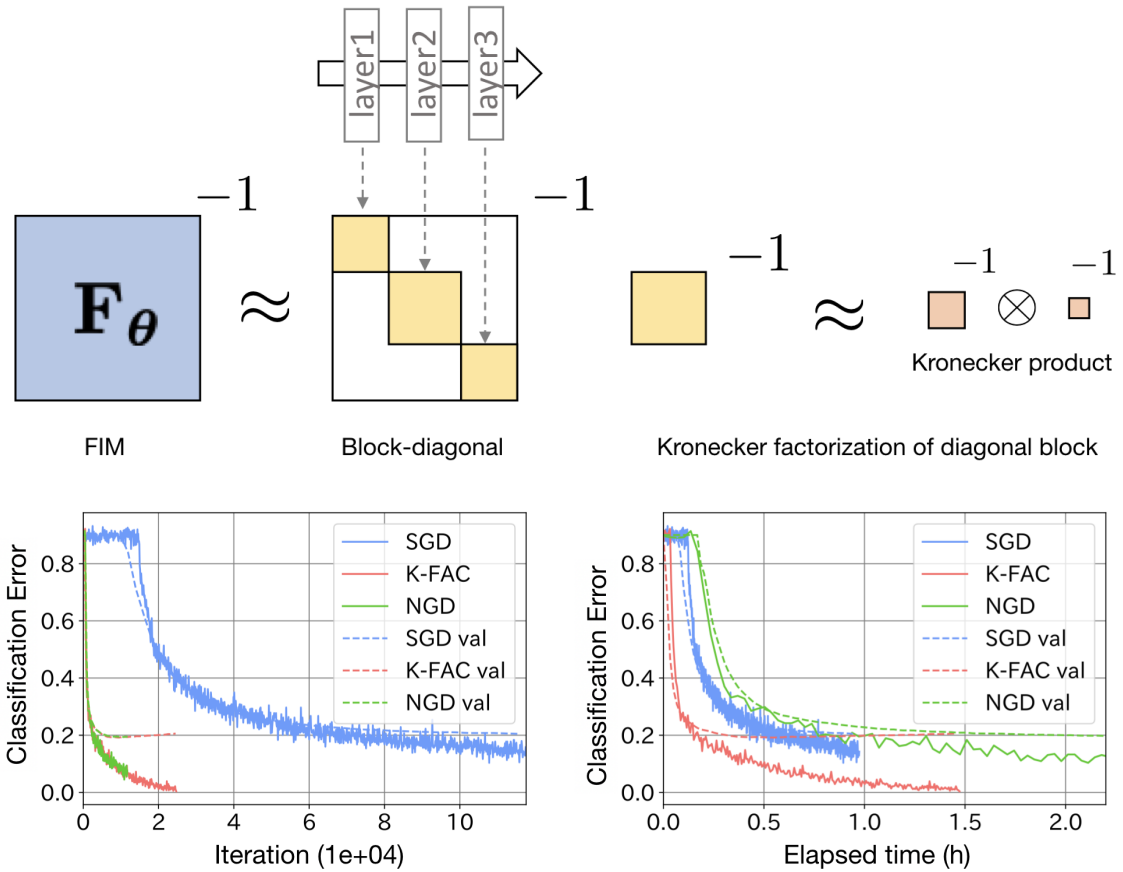
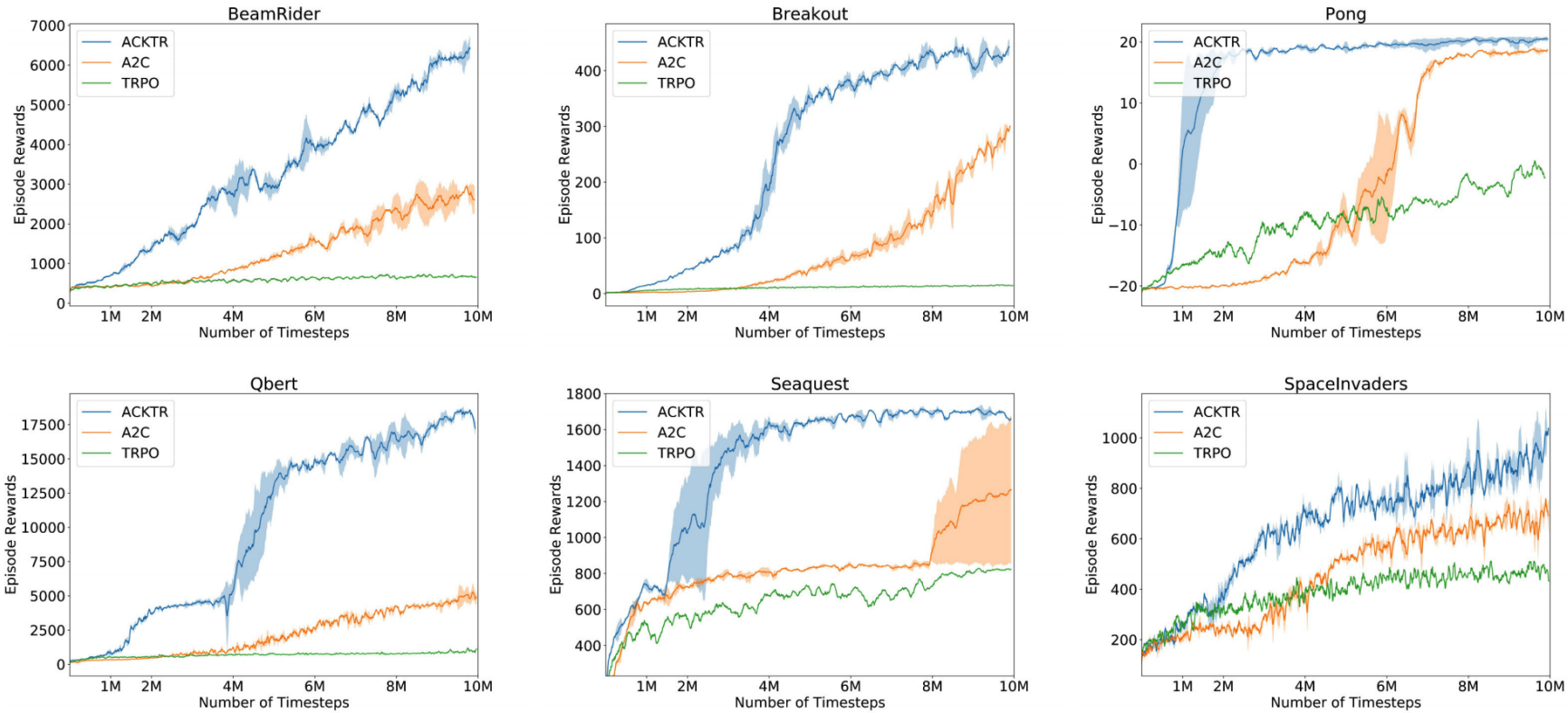


Figure: The comparison of training of ConvNet for CIFAR-10 dataset. Solid line: train, dashed line: validation. (data iteration vs. time) . Read more on K-FAC tutorial

Performance of ACKTR

1. Performance of ACKTR



2. Introductory link:
   `https://openai.com/index/openai-baselines-acktr-a2c/`

Proximal Policy Optimization (PPO)

1. The loss function in the Natural Policy Gradient and TRPO

how good the new policy is while following data from the old policy

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

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

Hard constraint, hard to implement in practice

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

use this penalty instead to inforce the hard constraint

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

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

PPO with adaptive KL penalty

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 as or better performance than TRPO, but can be 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 the different surrogate objectives are:
   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)

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

   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.

How the clipping works

1. Clipping serves as a regularizer by removing incentives for the policy to change dramatically

how much a ratio has changed

$$L^{CLIP}(\theta) = \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t, \text{clip}\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)$$

must stay within this interval

2. When the advantage is positive, we encourage the action thus $\pi_\theta(a|s)$ increases,

we don't want to increase the the Probability of taking the action (L) by too mcuh

$$L(\theta; \theta_{old}) = \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, (1+\epsilon)\right)\hat{A}_t$$

3. When the advantage is negative, we discourage the action thus $\pi_\theta(a|s)$ decreases,

$$L(\theta; \theta_{old}) = \max\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, (1-\epsilon)\right)\hat{A}_t$$

PPO with clipping

Algorithm 5 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

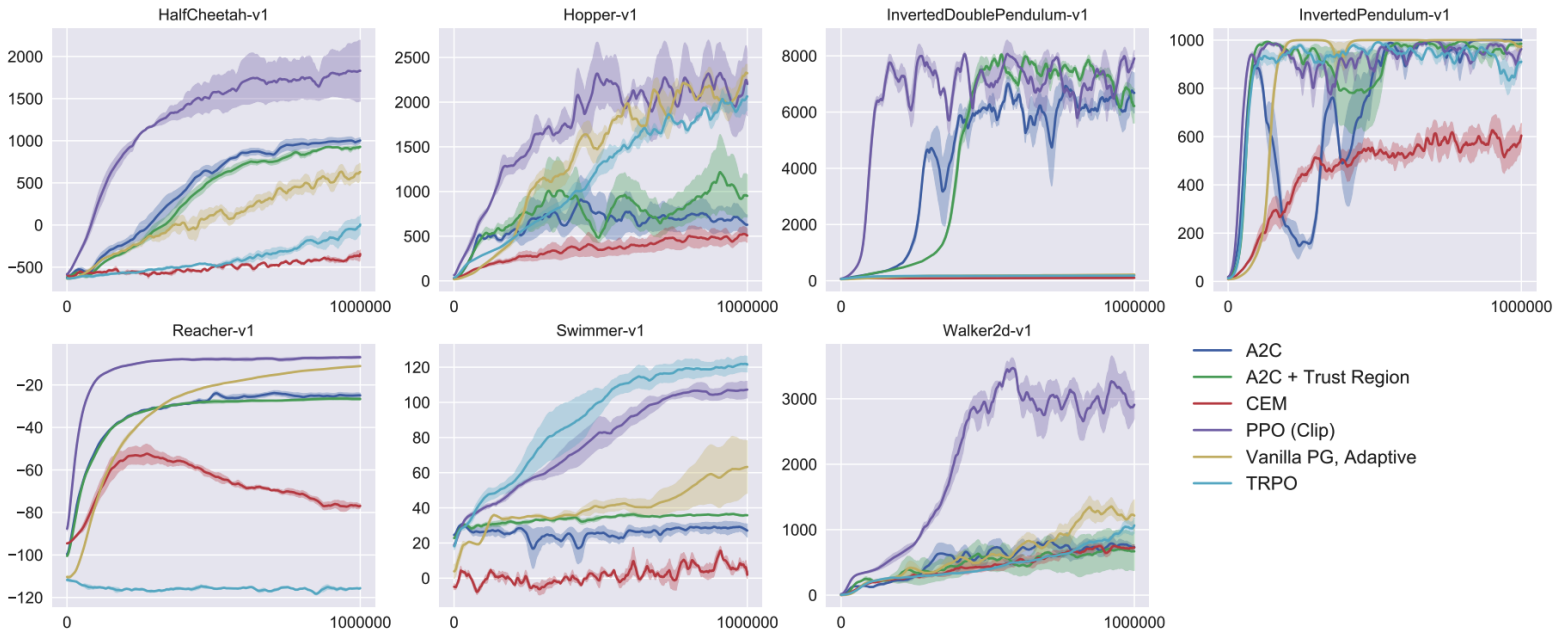
$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_k} \left[\sum_{t=0}^{T} \left[\min(r_t(\theta)\hat{A}_t^{\pi_k}, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t^{\pi_k}) \right] \right]$$

end for

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 (MuJoCo)
   `https://gym.openai.com/envs/mujoco`



2. Demo of PPO at
   `https://openai.com/index/openai-baselines-ppo/`
3. Emergence of Locomotion Behaviours in Rich Environments by DeepMind (Distributed PPO):
   `https://www.youtube.com/embed/hx_bgoTF7bs`

Code of PPO

1. Paper link of PPO: `https://arxiv.org/abs/1707.06347`
2. Code example: `https://github.com/ucla-rlcourse/DeepRL-Tutorials/blob/master/14.PPO.ipynb`
3. Very interesting one: **The 37 Implementation Details of Proximal Policy Optimization (ICLR'22 blog track)**
   1. `https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/`
   2. Devil is in the details

State of the Art on Policy Optimization

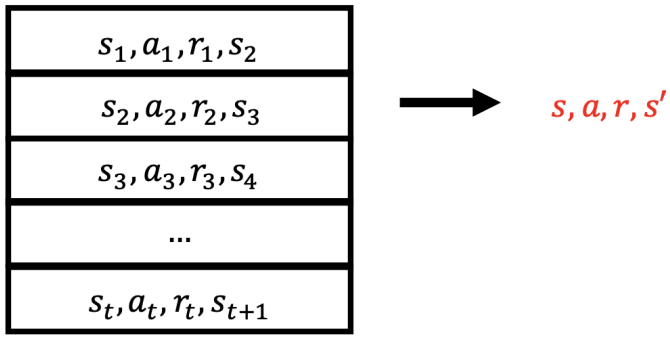
1. **TRPO**: Schulman, L., Moritz, Jordan, Abbeel (2015). Trust region policy optimization
    1. comment: Solid math proofs and guarantee, but hard to follow
2. **ACKTR**: Y. Wu, E. Mansimov, S. Liao, R. Grosse, and J. Ba (2017). Scalable trust-region method for deep reinforcement learning using Kronecker-factored 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, on-policy method

Q-learning→DQN→DDPG→TD3

1. **DQN**: Human-level control through deep reinforcement learning, V Mnih et al., Nature 2015
2. **DDPG**: Deterministic Policy Gradient Algorithms, Silver et al. ICML 2014
3. **TD3**: Addressing Function Approximation Error in Actor-Critic Methods, Fujimoto et al. ICML 2018

Review of DQN: Experience Replay

1. To reduce the correlations among samples, store transition (s_t, a_t, r_t, s_{t+1}) in replay memory $\mathcal{D}$



2. To perform experience replay, repeat the following
   1. sample an experience tuple from the dataset: $(s, a, r, s') \sim \mathcal{D}$
   2. compute the target value for the sampled tuple: $r + \gamma \max_{a'} \hat{Q}(s', a', \mathbf{w})$
   3. use stochastic gradient descent to update the network weights

$$\Delta \mathbf{w} = \alpha \Big(r + \gamma \max_{a'} \hat{Q}(s', a', \mathbf{w}) - Q(s, a, \mathbf{w}) \Big) \nabla_{\mathbf{w}} \hat{Q}(s, a, \mathbf{w})$$

Review of DQN: Fixed Targets

1. To help improve stability, fix the target weights used in the target calculation for multiple updates
2. Let a different set of parameter $\mathbf{w}^-$ be the set of weights used in the target, and $\mathbf{w}$ be the weights that are being updated
3. To perform experience replay with fixed target, repeat the following
   1. sample an experience tuple from the dataset: $(s, a, r, s') \sim \mathcal{D}$
   2. compute the target value for the sampled tuple: $r + \gamma \max_{a'} \hat{Q}(s', a', \mathbf{w}^-)$
   3. use stochastic gradient descent to update the network weights

$$\Delta \mathbf{w} = \alpha \Big(r + \gamma \max_{a'} \hat{Q}(s', a', \mathbf{w}^-) - \hat{Q}(s, a, \mathbf{w}) \Big) \nabla_{\mathbf{w}} \hat{Q}(s, a, \mathbf{w})$$

Deep Deterministic Policy Gradient (DDPG)

this is determinstic

1. Motivation: how to extend DQN to the environment with continuous action space?

think driving a car for action space

2. DDPG is very similar to DQN, which can be considered as a continuous action space version of DQN

check all actions and choose the one with the highest value

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

\mu_\theta is the action - outputs a specific action

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

so this is like a regression?

Q_\phi is the critic (computes how good an action is)

   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

1. input state S
2. pass it to policy (\mu_\theta) to get action
3. pass it to state S and action a into the Q-function (Q_\phi)
4. out[ut the estimate value

Deep Deterministic Policy Gradient (DDPG)

1. Thus the DDPG has the following objective:

$$\textbf{Q-target:} \quad y(r, s') = r + \gamma Q_{\phi_{targ}}(s', \mu_{\theta_{targ}}(s')))$$

$$\textbf{Q-function update:} \quad \min E_{s,r,s',d\sim\mathcal{D}}[(y(r, s') - Q_\phi(s, a))^2]$$

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

2. Reply buffer and target networks for both value network and policy network are the same as DQN
3. DDPG example code (using the sample codebase for TD3): `https://github.com/sfujim/TD3/blob/master/DDPG.py`

Twin Delayed DDPG (TD3)

1. One drawback of DDPG is that the learned Q-function sometimes dramatically overestimates Q-values, which then leads to the breaking in training

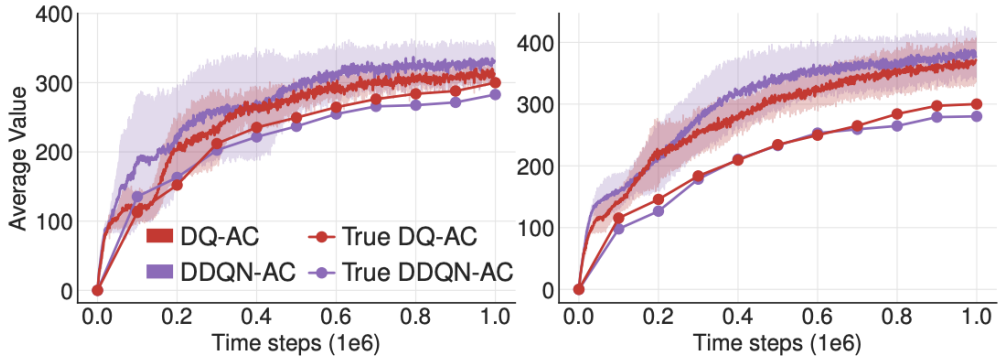


Figure 2. Measuring overestimation bias in the value estimates of actor critic variants of Double DQN (DDQN-AC) and Double Q-learning (DQ-AC) on MuJoCo environments over 1 million time steps.

1. true value is estimated using the average discounted return over 1000 episodes following the current policy

Twin Delayed DDPG (TD3)

1. TD3 introduces three critical designs
   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.

prevents overfitting

Twin Delayed DDPG (TD3)

aimed to fix over estimation bias (agent thinks actions are better than they actually are)

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 value as the following **Q-target**:

$$y(r, s') = r + \gamma \min_{i=1,2} Q_{\phi_{i,targ}}(s', a_{TD3}(s'))$$

standard Q-learning typically overestimates a Q-value

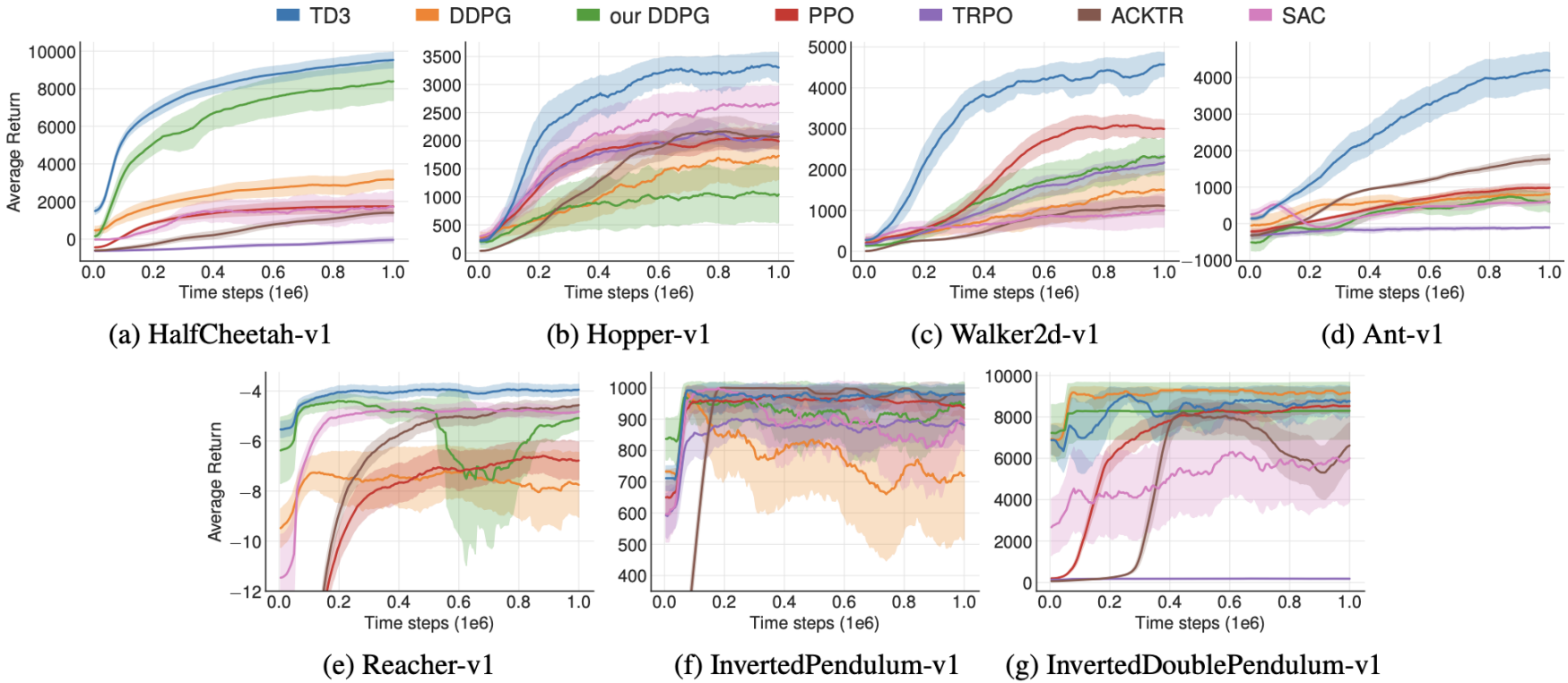
3. Target policy smoothing works as follows

$$a_{TD3}(s') = clip(\mu_{\theta,targ}(s') + clip(\epsilon, -c, c), a_{low}, a_{high}), \epsilon \sim \mathcal{N}(0, \sigma)$$

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

Twin Delayed DDPG (TD3)

1. Result and comparison



Twin Delayed DDPG (TD3)

1. TD3 paper: Fujimoto, et al. Addressing Function Approximation Error in Actor-Critic Methods. ICML'18: `https://arxiv.org/pdf/1802.09477.pdf`
2. Author's Pytorch implementation (very clean implementation!): `https://github.com/sfujim/TD3/`
3. Have a comparison to see the difference between DDPG and TD3
    1. `https://github.com/sfujim/TD3/blob/master/DDPG.py`
    2. `https://github.com/sfujim/TD3/blob/master/TD3.py`
4. Authors are from neither Berkelely/OpenAI nor DeepMind!
    1. Scott Fujimoto, Herke van Hoof, David Meger from McGill University

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

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

policy gradient

entropy regularization

Soft Actor-Critic (SAC)

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

$$V^{\pi}(s) = E_{\tau\sim\pi}[\sum_{t} \gamma^{t}\big(R(s_t, a_t) + \alpha H(\pi(.|s_t))\big)|s_0 = s]$$

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

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

3. So the sample update for Q function is

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

Soft Actor-Critic (SAC)

we want to be able to find all paths that are good to the goal by adding the entropy term

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

$$L(\phi_i, \mathcal{D}) = E[(Q_\phi(s,a) - y(r,s',d)^2] \tag{3}$$

$$y(r,s') = r + \gamma\left(\min_{j=1,2} Q_{\phi_{targ,j}}(s', \hat{a}') - \alpha \log \pi_\theta(\hat{a}'|s')\right), \tag{4}$$

bonus for acting more randomly
\alpha controls randomness temp

$$\hat{a}' \sim \pi_\theta(.|s'). \tag{5}$$

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

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

Soft Actor-Critic (SAC)

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

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

mean and standard deviation as predicted by the network

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):

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

now expectation is indpenedent of the parameters?

sample many times and take the average

3. The policy is thus optimized as 1/N \sum_k Q_{\pi_0}(s, abar, s_1, \eps_k)??

$$\max_\theta E_{s\sim\mathcal{D}, \epsilon\sim\mathcal{N}}[\min_{j=1,2} Q_{\phi_j}(s, \hat{a}_\theta(s, \epsilon)) - \alpha \log \pi_\theta(\hat{a}_\theta(s, \epsilon)|s)] \quad (6)$$

Brief Intro on Reparameterization Trick

used in variational autoencoders

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 the distribution with parameter θ
2. We can rewrite the samples of the distribution p_θ in terms of a random variable ϵ sampled from distribution q which is independent of θ

context: we have a neural net that outputs the parameters of a distribution, which we sample to get a value f(x), but we can't back propagate because we are sampling

$$\epsilon \sim q \quad \text{q is some fixed dist indep from N} \tag{7}$$

$$x = g_\theta(\epsilon) \quad \text{this uses our parameters to calculate x} \tag{8}$$

$$\nabla_\theta E_{x\sim p_\theta(x)}[f(x)] = \nabla_\theta E_{\epsilon\sim q}[f(g_\theta(\epsilon))] \tag{9}$$

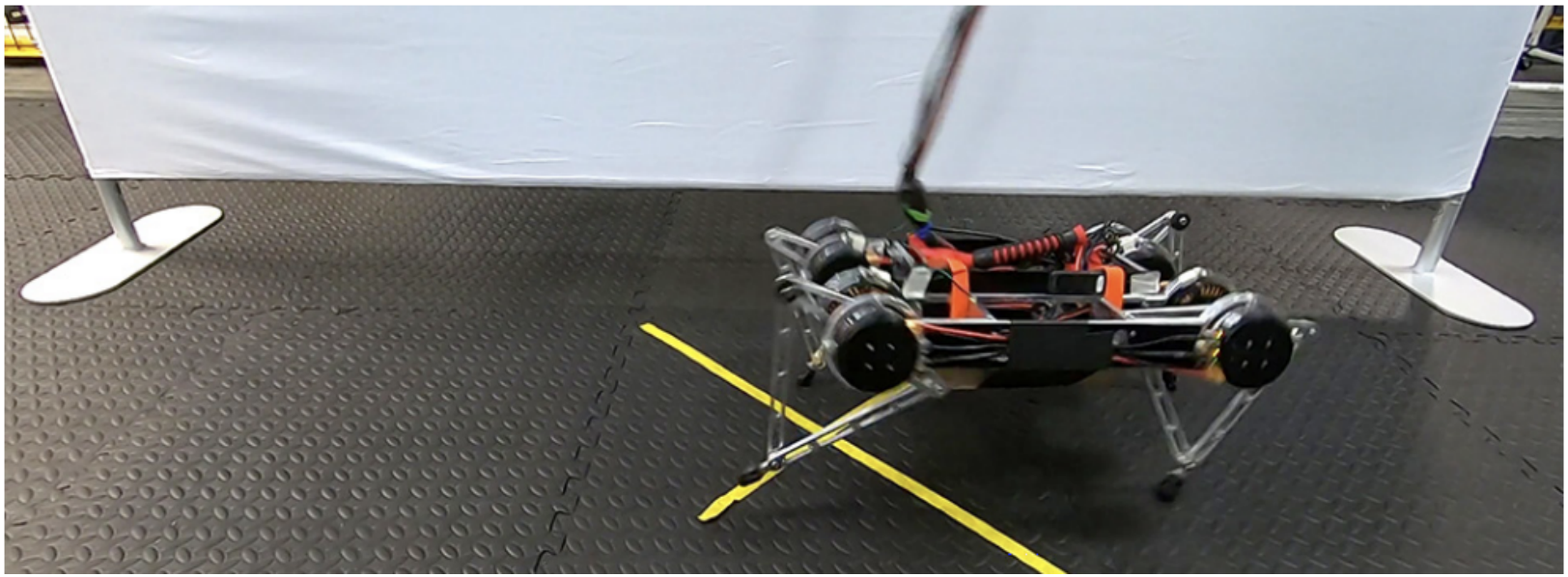
$$= E_{\epsilon\sim q}[\nabla_\theta f(g_\theta(\epsilon))] \tag{10}$$

since x no longer depends on sampling from p_\theta, we can move the gradient inside because of linearity and no dependence on q and \theta

   1. where x is reparameterized as a function of ϵ and the stochasticity of p_θ is replaced by the distribution q, 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/embed/cwyiq6dCgOc`



Summary of the SOTAs

1. Policy Gradient→TRPO→ACKTR→PPO
   1. Stochastic policy thus output probability over discrete actions
   2. Start with policy gradient and importance sampling for off-policy learning
2. Q-learning→DDPG→TD3
   1. Deterministic policy thus output continuous action spaces.
   2. Start with Bellman equation, which doesn't care which transition tuples are used, or how the actions were selected, or what happens after a given transition
   3. Optimal Q-function should satisfy the Bellman equation for all possible transitions, so very esay for off-policy learning
3. SAC
   1. SAC optimizes a stochastic policy in an off-policy way, which unifies stochastic policy optimization and DDPG-style approaches
   2. off-policy method, high sample efficiency, it incorporates the clipped double-Q trick like TD3, and due to the inherent stochasticity of the policy in SAC, it also winds up benefiting from something like target policy smoothing.

Tutorial and Toolkit of SOTA RL algorithms

1. They become handy for your research project
2. OpenAI Spinning Up: Nice implementations and summary of the algorithms from OpenAI
    1. `https://spinningup.openai.com/`

3. Stable-baseline3 in PyTorch:
    1. `https://github.com/DLR-RM/stable-baselines3`
4. CleanRL:
    1. `https://github.com/vwxyzjn/cleanrl`
    2. High-quality single file implementation of deep RL algorithms

Great tutorial video on how to make deep RL work

1. John Schulman (OpenAI research scientist): Nuts and Bolts of Deep RL Experimentation, August 26, 2017
    1. Slide: `http://joschu.net/docs/nuts-and-bolts.pdf`
    2. Video: `https://www.youtube.com/watch?v=8EcdaCk9KaQ`
2. His more recent talk on Reinforcement Learning from Human Feedback: video:
   `https://www.youtube.com/watch?v=hhiLw5Q_UFg`