535514: Reinforcement Learning
Lecture 15 — DDPG, TD3, and TRPO

Ping-Chun Hsieh

April 15, 2024

Question: Critic Design in OPDAC?

- Off-Policy Deterministic Actor-Critic (OPDAC):

Step 1: Initialize θ_0, w_0 and step sizes α_θ, α_w

Step 2: Sample a trajectory $\tau = (s_0, a_0, r_1, \cdots) \sim P_\mu^\beta$

For each step of the current trajectory $t = 0,1,2,\cdots$

$$\Delta w_k \leftarrow \Delta w_k + \alpha_w \left(r_t + \gamma Q_{w_k}(s_{t+1}, \pi_\theta(s_{t+1})) - Q_{w_k}(s_t, a_t) \right) \nabla_w Q_w(s_t, a_t)|_{w=w_k}$$

$$\Delta \theta_k \leftarrow \Delta \theta_k + \alpha_\theta \gamma^t \left(\underline{\nabla_\theta \pi_\theta(s_t) \nabla_a Q_{w_k}(s_t, a)|_{a=\pi_\theta(s_t)}} \right)$$

$$\theta_{k+1} \leftarrow \theta_k + \Delta\theta_k, w_{k+1} \leftarrow w_k + \Delta w_k \qquad = \nabla_\theta Q_{w_k}(s_t, \pi_\theta(s_t))|_{\theta=\theta_k}$$

- Question: Why no change on the critic in Off-Policy DAC?

Learning a Critic for OPDAC under VFA

- Goal: find $\mathbf{w}$ that minimizes MSE between $Q^{\pi}(s,a)$ and $\hat{Q}(s,a;\mathbf{w})$

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \underbrace{\mathbb{E}_{s\sim\rho(s)}\left[\left(Q^{\pi}(s,a) - \hat{Q}(s,a;\mathbf{w})\right)^2\right]}_{=:F(\mathbf{w})}$$

- Find $\mathbf{w}^*$ by iterative GD updates

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \mathbb{E}_{s\sim\rho(s)}\left[\left(Q^{\pi}(s,a) - \hat{Q}(s,a;\mathbf{w})\right)\nabla_{\mathbf{w}}\hat{Q}(s,a;\mathbf{w})\right]$$

- Since true Q^{π} is unknown, let's use bootstrapping (e.g.TD(0))

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \mathbb{E}_{s\sim\rho(s)}\left[\left(r + \gamma\hat{Q}_{\mathbf{w}}(s',\pi(s')) - \hat{Q}(s,a;\mathbf{w})\right)\nabla_{\mathbf{w}}\hat{Q}(s,a;\mathbf{w})\right]$$

- Question: To learn a critic for OPDAC, what distribution $\rho(s)$ shall we choose?

On-Policy vs Off-Policy Methods

<table>
<tr><th></th><th>Policy Optimization</th><th>Value-Based</th><th>Model-Based</th><th>Imitation-Based</th></tr>
<tr><td>On-policy</td><td>Exact PG
REINFORCE (w/i baseline)
A2C
On-policy DAC
TRPO
Natural PG (NPG)
PPO-KL & PPO-Clip</td><td>Epsilon-Greedy MC
Sarsa
Expected Sarsa</td><td rowspan="2">Model-Predictive Control (MPC)
PETS</td><td rowspan="2">IRL
GAIL
IQ-Learn
RLHF</td></tr>
<tr><td>Off-policy</td><td>Off-policy DPG & DDPG
Twin Delayed DDPG (TD3)</td><td>Q-learning
Double Q-learning
DQN & DDQN
C51 / QR-DQN / IQN
Soft Actor-Critic (SAC)</td></tr>
</table>

Deep Deterministic Policy Gradient (DDPG)

(= OPDAC with Deep Neural Nets)

Lillicrap et al., “Continuous control with deep reinforcement learning”, ICLR 2016

What is DDPG?

- DDPG: Combine OPDAC with NN nonlinear VFA
  - Off-policy: Exploration
  - Nonlinear VFA: Convergence issue

- To tackle the above issues, DDPG applies several techniques:

(T1) Experience replay (for data-efficient off-policy learning)

(T2) Ornstein-Uhlenbeck process for exploration (optional)

(T3) Target networks

(T1) Experience Replay

- Main idea:

  1. Store the previous experiences (s, a, s', r) into a buffer
  2. Sample a mini-batch from the buffer at each step

  (similar to mini-batch SGD in supervised learning)

- Purposes:

1. Better estimate of DPG: Break correlations between successive steps in a trajectory (“more stable learning”, as stated in many papers)

2. Better data efficiency: Fewer interactions with environment needed for convergence

(T2) Ornstein-Uhlenbeck Process for Exploration

- Issue with Gaussian noise exploration $a_t = \pi_\theta(s_t) + N(0, \sigma^2)$?

- Ornstein-Uhlenbeck (OU) process: Similar to Gaussian policies, but with temporal correlation

Brownian motion

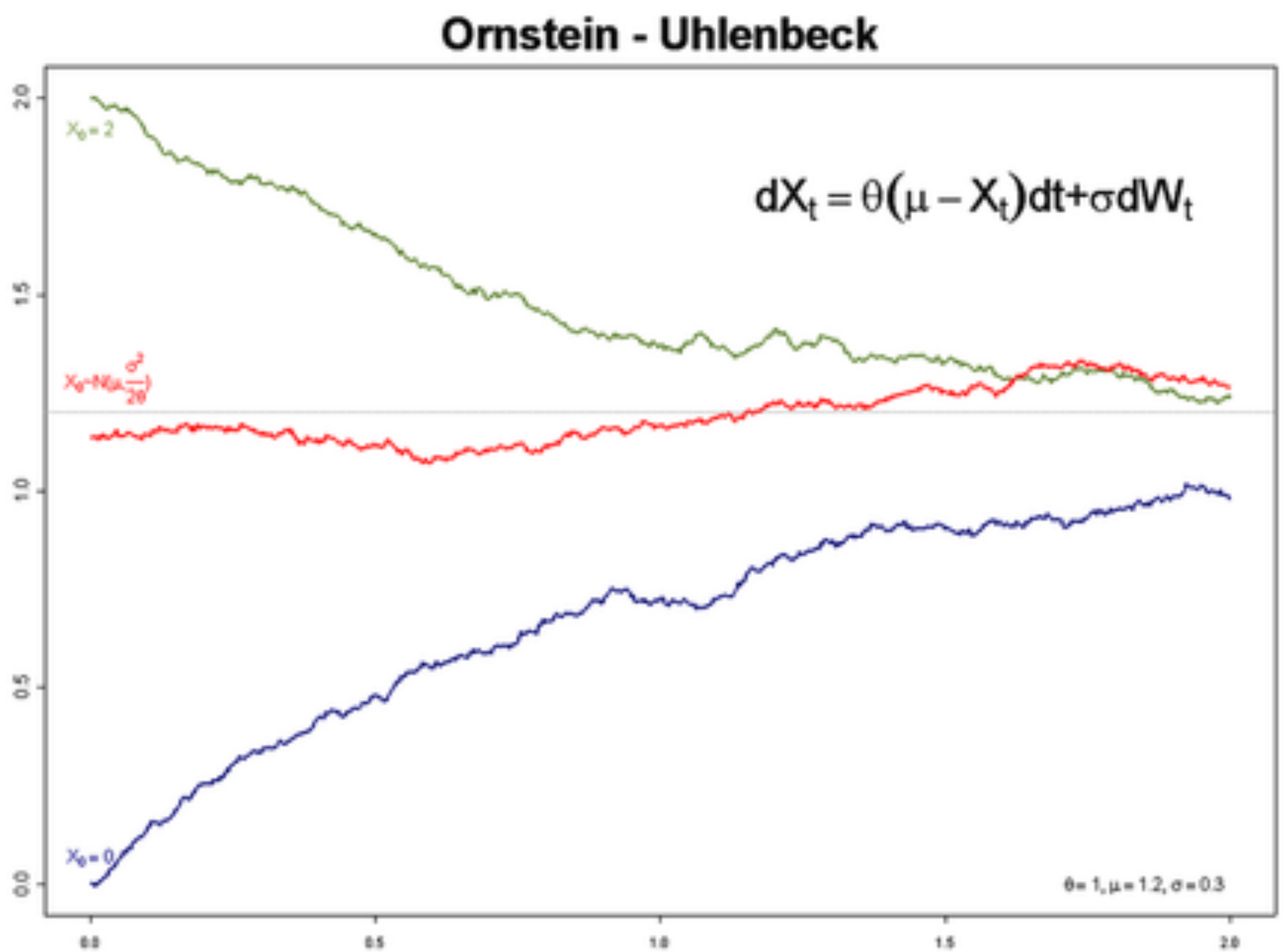
$$dx_t = \theta(\mu - x_t)dt + \sigma \cdot dW_t$$

- Discrete-time approximation of OU:

$$X_{t+1} - X_t = \theta(\mu - X_t)\Delta t + \sigma \cdot \Delta W_t$$

i.i.d. normal random variables $\sim \mathcal{N}(0, \Delta_t)$

Example of OU Process



(Same OU process with 3 different initial conditions)

How about a sequence of i.i.d. Gaussian random variables?

(Figure Source: Wikipedia)

(T3) Target Networks

- Idea: Use separate *target networks* ($\bar{\pi}_\theta$ for actor, $\bar{Q}_w$ for critic) that are updated only periodically

- For DDPG, the critic update with target networks

$$\Delta w_k \leftarrow \Delta w_k + \alpha_w \Big(r_t + \gamma \underbrace{\bar{Q}_{w_k}(s_{t+1}, \bar{\pi}_\theta(s_{t+1}))}_{\text{target}} - \underbrace{Q_{w_k}(s_t, a_t)}_{\text{update}} \Big) \nabla_w Q_w(s_t, a_t)|_{w=w_k}$$

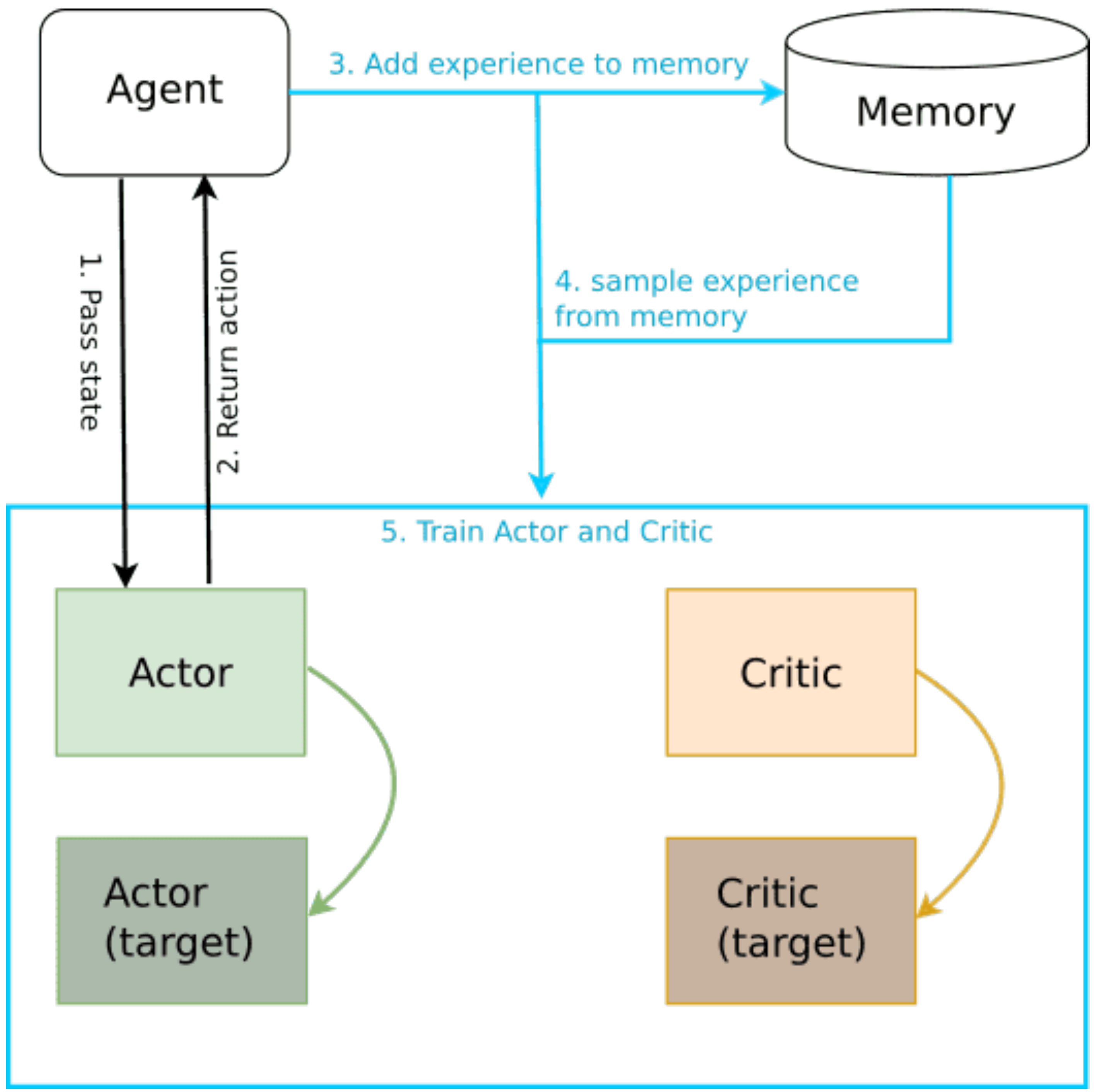
- Similar to value iteration:

$$\underbrace{V(s)}_{\text{update}} \leftarrow \max_a R(s,a) + \gamma \sum_{s'} P(s'|s,a) \underbrace{\bar{V}(s)}_{\text{target}}$$

- Purpose: Mitigate divergence

(Slide Credit: Pascal Poupart)

DDPG Architecture



 (Figure Credit: https://www.katnoria.com/ddpg-reacher/)

Pseudo Code of DDPG Algorithm

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$

2 evaluation networks and 2 target networks

Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**

action drawn from a deterministic policy with exploration

   Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
   Execute action a_t and observe reward r_t and observe new state s_{t+1}
   Store transition (s_t, a_t, r_t, s_{t+1}) in R
   Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R

experience replay

   Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
   Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
   Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

Update actor and critic

This can be viewed as the gradient of Q w.r.t. θ

   Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1 - \tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1 - \tau)\theta^{\mu'}$$

Update target networks (small τ for stability)

  end for
end for

Twin Delayed DDPG (TD3)

Fujimoto et al., “Addressing Function Approximation Error in Actor-Critic Methods,” ICML 2018

Overestimation Bias in DDPG

- Overestimation bias: Estimated Q values are larger than the true ones

$$\mathbb{E}_s[Q_w(s, \pi_\theta(s))] \geq \mathbb{E}_s[Q^{\pi_\theta}(s, \pi_\theta(s))]$$

- Question: How to empirically measure such bias?
- Empirical evidence:

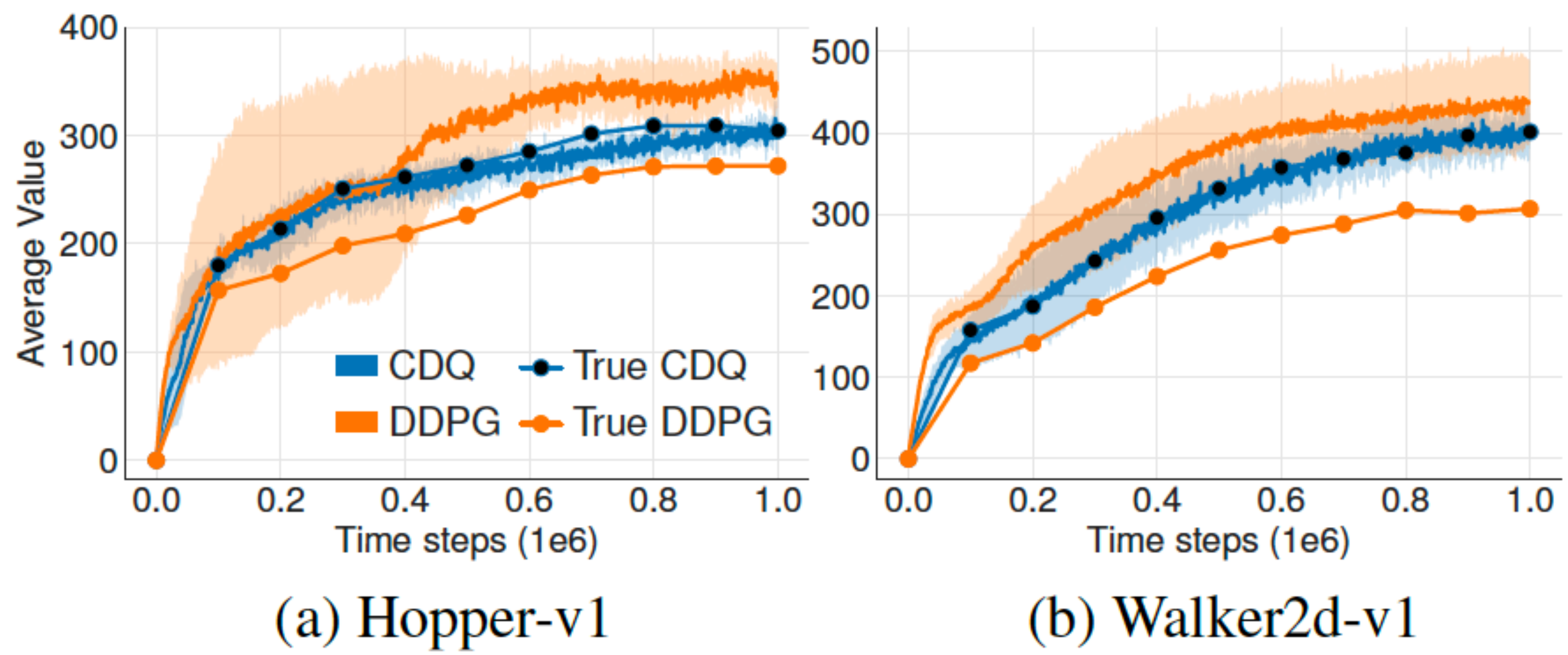
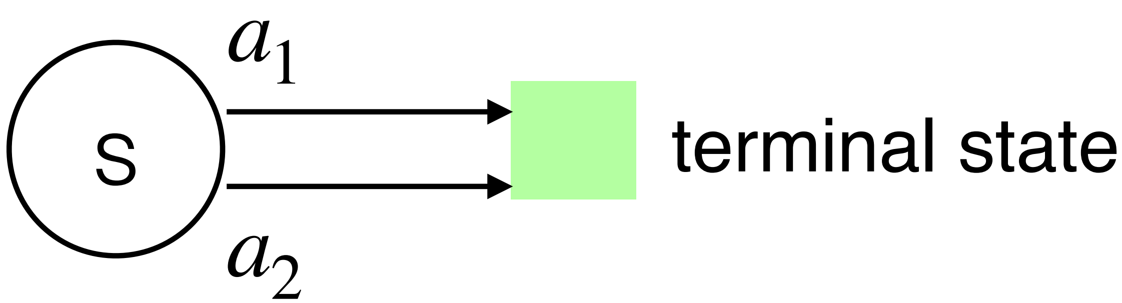


Figure 1. Measuring overestimation bias in the value estimates of DDPG and our proposed method, Clipped Double Q-learning (CDQ), on MuJoCo environments over 1 million time steps.

Fujimoto et al., Addressing Function Approximation Error in Actor-Critic Methods, ICML 2018

Overestimation Bias: A Motivating Example

- Example: Consider a 1-state MDP with 2 actions



$$r(s, a_1) \sim \text{Bernoulli}(0.5)$$
$$r(s, a_2) \sim \text{Bernoulli}(0.5)$$

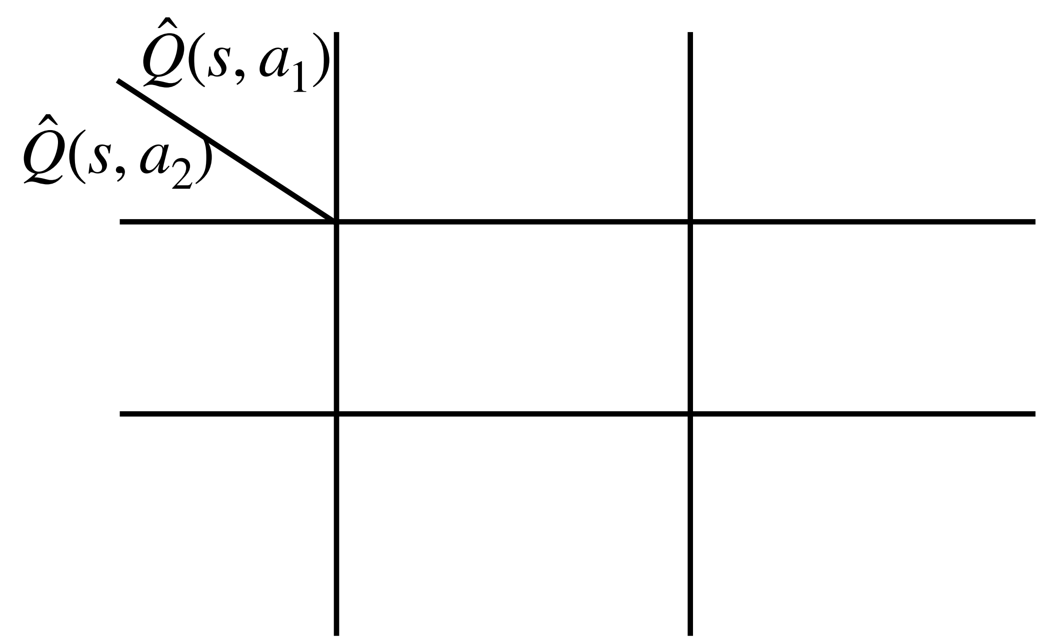
- Let $\hat{Q}(s, a_1)$, $\hat{Q}(s, a_2)$ be unbiased estimators (each based on 1 sample)

- An Interesting Fact:

$$\mathbb{E}\Big[\underbrace{\max\{\hat{Q}(s,a_1), \hat{Q}(s,a_2)\}}_{=:M}\Big] > \max\Big\{\mathbb{E}[\hat{Q}(s,a_1)], \mathbb{E}[\hat{Q}(s,a_2)]\Big\}$$

$$\max\Big\{\mathbb{E}[\hat{Q}(s,a_1)], \mathbb{E}[\hat{Q}(s,a_2)]\Big\} =$$

$$\mathbb{E}\Big[\max\{\hat{Q}(s,a_1), \hat{Q}(s,a_2)\}\Big] =$$



"Clipped Double-Q" in Twin Delayed DDPG (TD3)

- Heuristic 1: Use two Q networks and take the minimum

DDPG's TD-target: $$y = r + \gamma Q_w(s', \pi_\theta(s'))$$

TD3's TD-target: $$y = r + \gamma \min_{i=1,2} Q_{w_i}(s', \pi_\theta(s'))$$

- Heuristic 2: Adding noise for a smooth Q

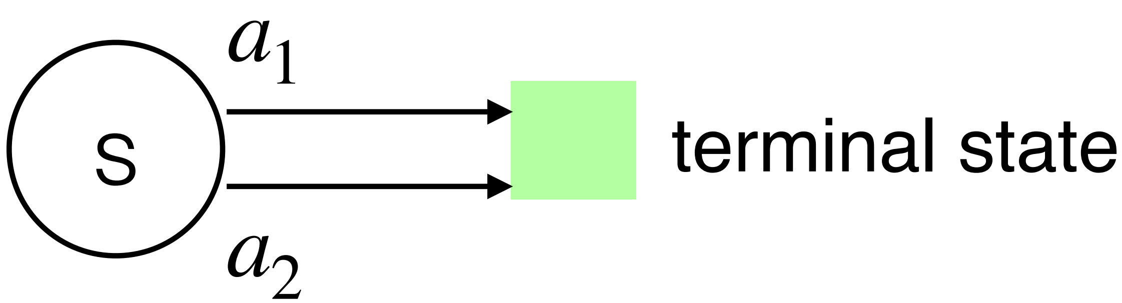
TD3's clipped noise: $$y = r + \gamma \min_{i=1,2} Q_{w_i}(s', \pi_\theta(s') + \epsilon)$$

$$\epsilon \sim \text{Clip}(N(0,\sigma^2), \epsilon_{\min}, \epsilon_{\max})$$

(Fujimoto et al., ICML 2018)

Mitigating Overestimation Bias via CDQ: An Example

- Example: Consider a 1-state MDP with 2 actions



$r(s, a_1) \sim$ Bernoulli(0.5)

$r(s, a_2) \sim$ Bernoulli(0.5)

CDQ uses twin Q functions:

- Let $\hat{Q}_a(s, a_1)$, $\hat{Q}_a(s, a_2)$ be the first unbiased Q estimate
- Let $\hat{Q}_b(s, a_1)$, $\hat{Q}_b(s, a_2)$ be the second unbiased Q estimate

---

$$\max\left\{\mathbb{E}[\hat{Q}(s, a_1)], \mathbb{E}[\hat{Q}(s, a_2)]\right\} =$$

$$\mathbb{E}\left[\max\left\{\underbrace{\min\left\{\hat{Q}_a(s, a_1), \hat{Q}_b(s, a_1)\right\}}_{=:\,X}, \underbrace{\min\left\{\hat{Q}_a(s, a_1), \hat{Q}_b(s, a_2)\right\}}_{=:\,Y}\right\}\right] =$$

Y \ X		

A2C/DPG/DDPG/TD3 ≈ Policy Iteration, But With a Slight Difference

Policy iteration:

Step 1: Policy evaluation for the current policy (i.e., find $Q^{\pi}(s,a)$)

Step 2: Policy improvement based on Bellman optimality equations

A2C/DPG/DDPG/TD3:

Step 1: Estimate $Q^{\pi_\theta}(s,a) \approx Q_w(s,a)$ for the current policy by TD

Step 2: Use $Q_w(s,a)$ to improve the policy by PG (deterministic or stochastic)

- Question: Is $V^{\pi_\theta}(\mu)$ guaranteed to be improved in Step 2?

Trust Region Policy Optimization (TRPO)

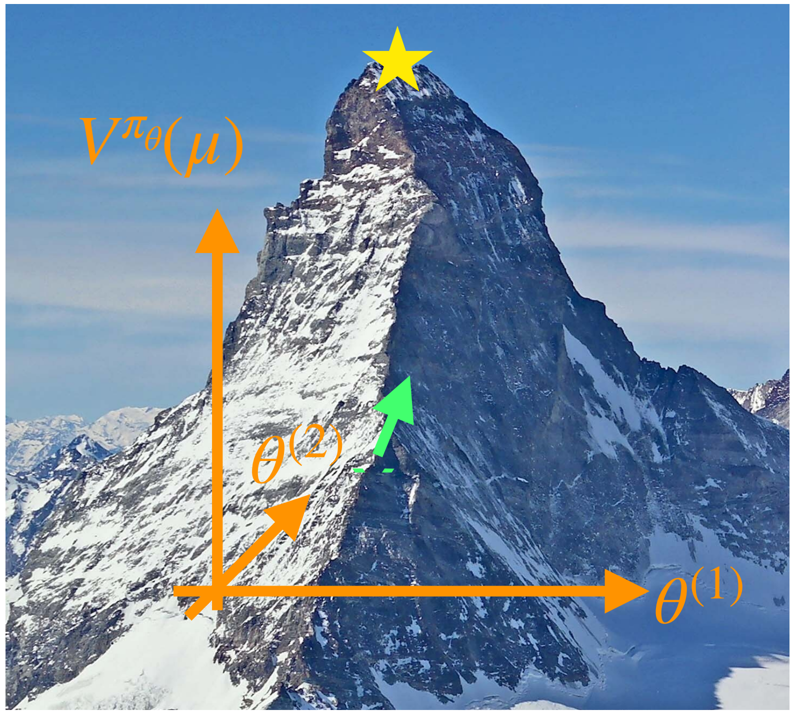
Schulman et al., “Trust Region Policy Optimization”, ICML 2015

Strict Policy Improvement

- Learning rate is critical for policy improvement

“Gradient ascent” uses first-order approximation

First-order approximation is accurate only in a small neighborhood



- Question: How to guarantee strict improvement?

TRPO: 1-Slide Summary

Idea: TRPO iteratively updates the policy by solving the following:

In each iteration k, the policy is updated from π_{θ_k} to $\pi_{\theta_{k+1}}$

$$\pi_{\theta_{k+1}} = \arg\max_{\theta} \mathbb{E}_{s \sim d_{\mu}^{\pi_{\theta_k}}, a \sim \pi_{\theta_k}(\cdot|s)} \left[\frac{\pi_\theta(a\,|\,s)}{\pi_{\theta_k}(a\,|\,s)} A^{\pi_{\theta_k}}(s, a;) \right]$$

subject to $D(\pi_{\theta_k}, \pi_\theta) \leq \delta$ ⟵ "trust region"

TRPO wants to achieve policy improvement via constrained optimization

State-Wise Performance Difference Lemma

- **State-Wise Performance Difference** [Kakade & Langford, 2002]

  For any two policies π_{old}, π_{new} and any initial state s, we have

$$V^{\pi_{new}}(s) - V^{\pi_{old}}(s) = \frac{1}{1-\gamma}\mathbb{E}_{s' \sim d_s^{\pi_{new}}}\mathbb{E}_{a' \sim \pi_{new}(\cdot|s')}\left[A^{\pi_{old}}(s', a')\right]$$

- Question 1: How to interpret this result?

- Question 2: Under tabular policies, what will we have if π_{old} is obtained from π_{new} by "*one-step policy improvement*"?

- Question 3: How about neural policies? Could we still possibly achieve monotonic policy improvement?

Kakade and Langford, "Approximately Optimal Approximate Reinforcement Learning," ICML 2002

Average Performance Difference Lemma

- Idea: Though we could NOT achieve monotonic policy improvement, could we resort to improvement in average performance?

- Question: Could we find the "average difference" $\sum_{s} \mu(s)\left(V^{\pi_{new}}(s) - V^{\pi_{old}}(s)\right)$ by state-wise performance difference lemma?

- **"Average" Performance Difference Lemma**:

$$V^{\pi_{new}}(\mu) - V^{\pi_{old}}(\mu) = \frac{1}{1-\gamma}\mathbb{E}_{s'\sim d_{\mu}^{\pi_{new}}}\mathbb{E}_{a'\sim\pi_{new}(\cdot|s')}\left[A^{\pi_{old}}(s',a')\right]$$

To simplify notations, let's use

$$\eta(\pi_{new}) \equiv (1-\gamma)V_{\mu}(\pi_{new})$$

$$\eta(\pi_{old}) \equiv (1-\gamma)V_{\mu}(\pi_{old})$$

Question: How about directly optimizing

$$\sum_s d_\mu^{\pi_{new}}(s) \sum_a \pi_{new}(a \mid s) A^{\pi_{old}}(s, a) = (1-\gamma)\big(V_\mu(\pi_{new}) - V_\mu(\pi_{old})\big)$$

usually difficult to get (why?)

Idea: Approximate $d_\mu^{\pi_{new}}(s)$ by $d_\mu^{\pi_{old}}(s)$

$$(1-\gamma)\big(V_\mu(\pi_{new}) - V_\mu(\pi_{old})\big) \approx \sum_s d_\mu^{\pi_{old}}(s) \sum_a \pi_{new}(a \mid s) A^{\pi_{old}}(s, a)$$

(called a "surrogate function")

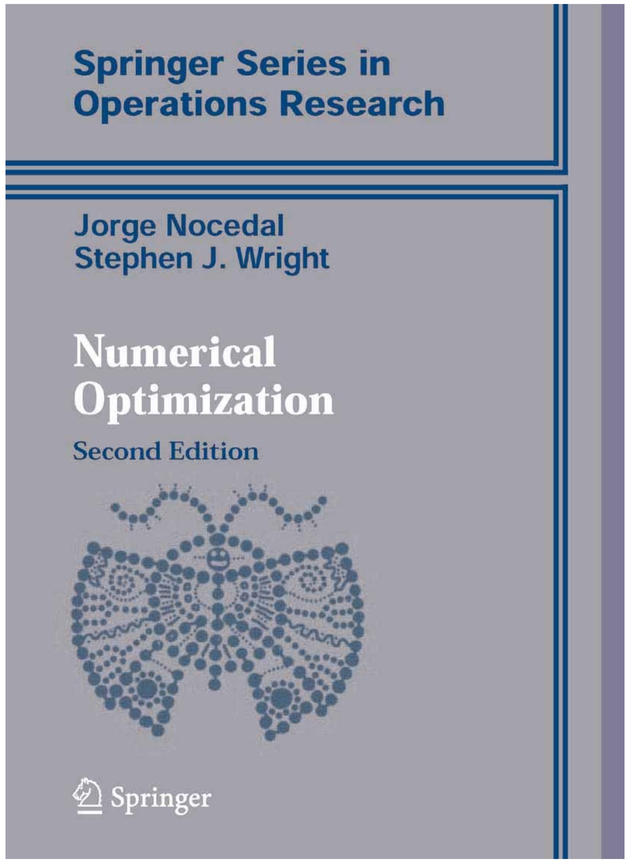
Question: How to ensure that $d_\mu^{\pi_{new}}(s)$ and $d_\mu^{\pi_{old}}(s)$ are close?

π_{new} and π_{old} need to be quite close

Trust Region Methods for Optimization!

Trust Region Methods for Optimization

Jorge Nocedal and Stephen Wright, “Numerical Optimization”, 2006

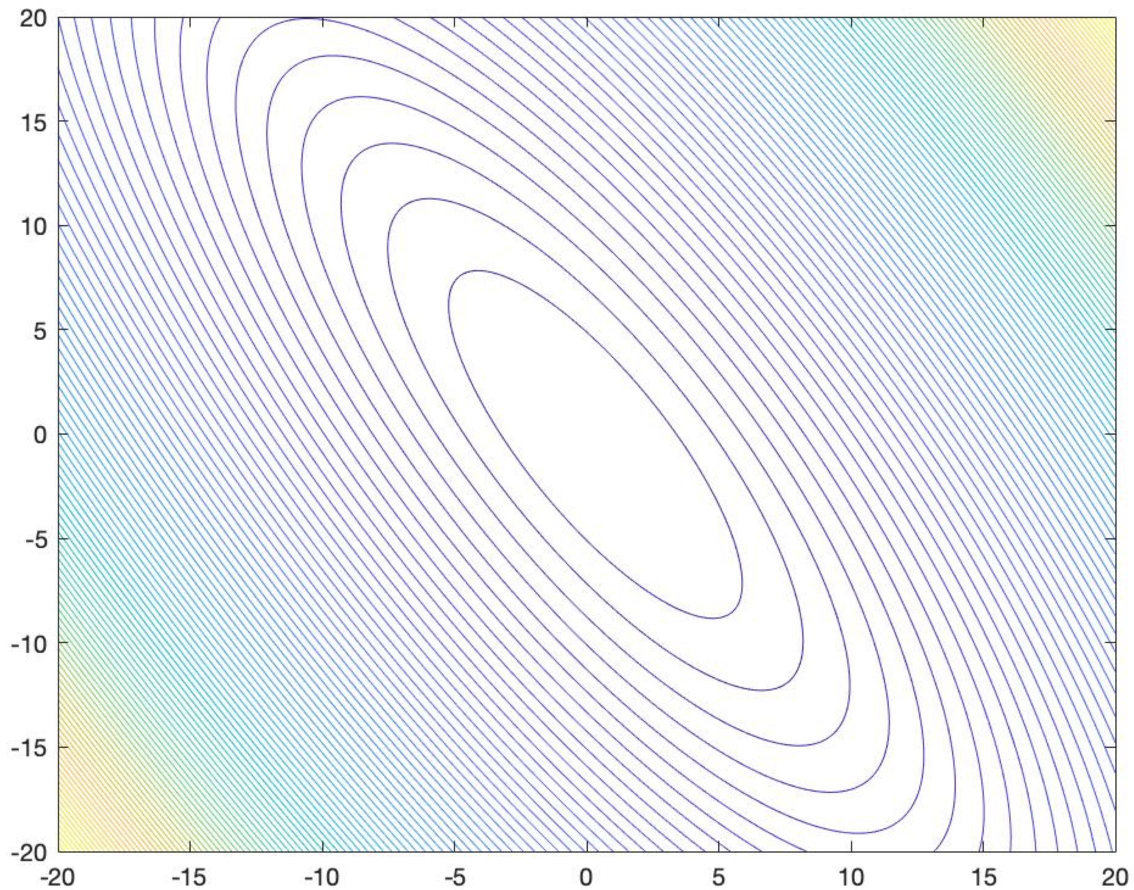


Trust Region vs Line Search

- 2 major types of iterative algorithms for numerical optimization:

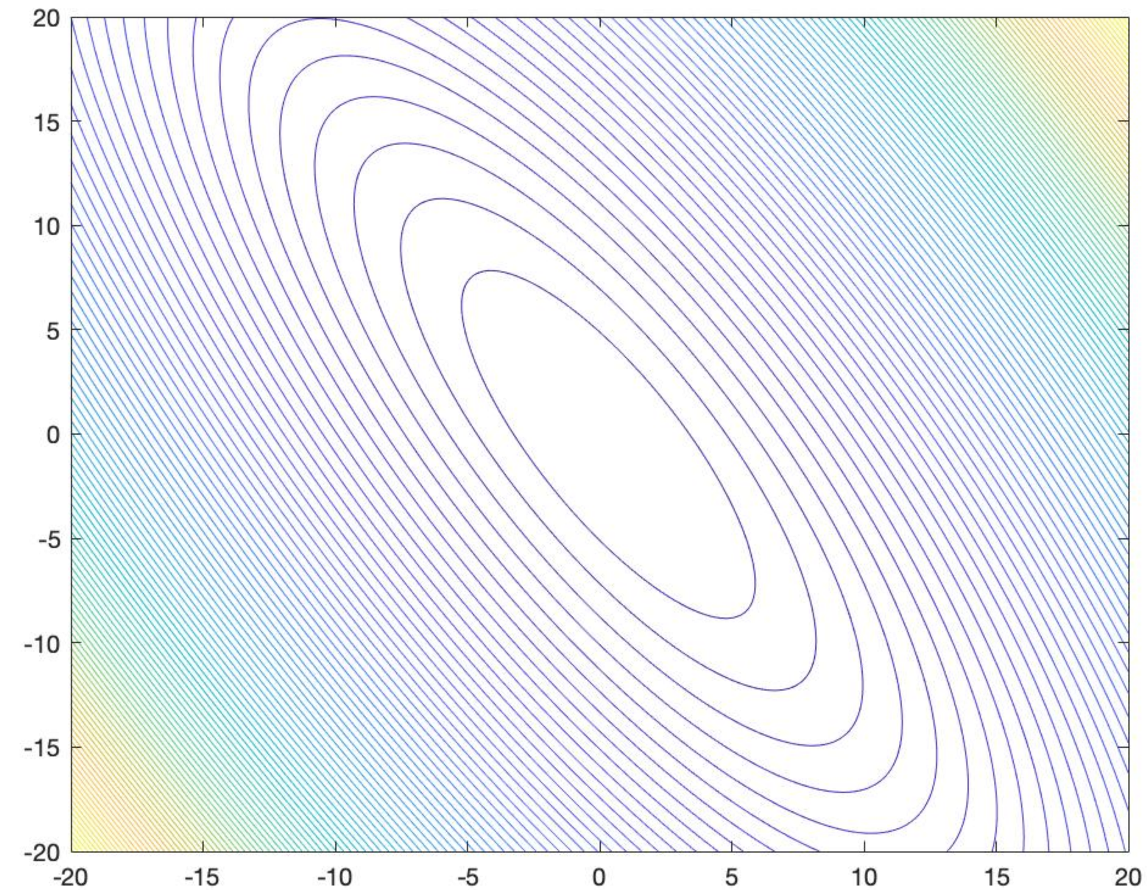
Contour of objective function $f(x)$

Line Search



$$x_{k+1} = x_k + \alpha_k \delta_k$$

Trust Region

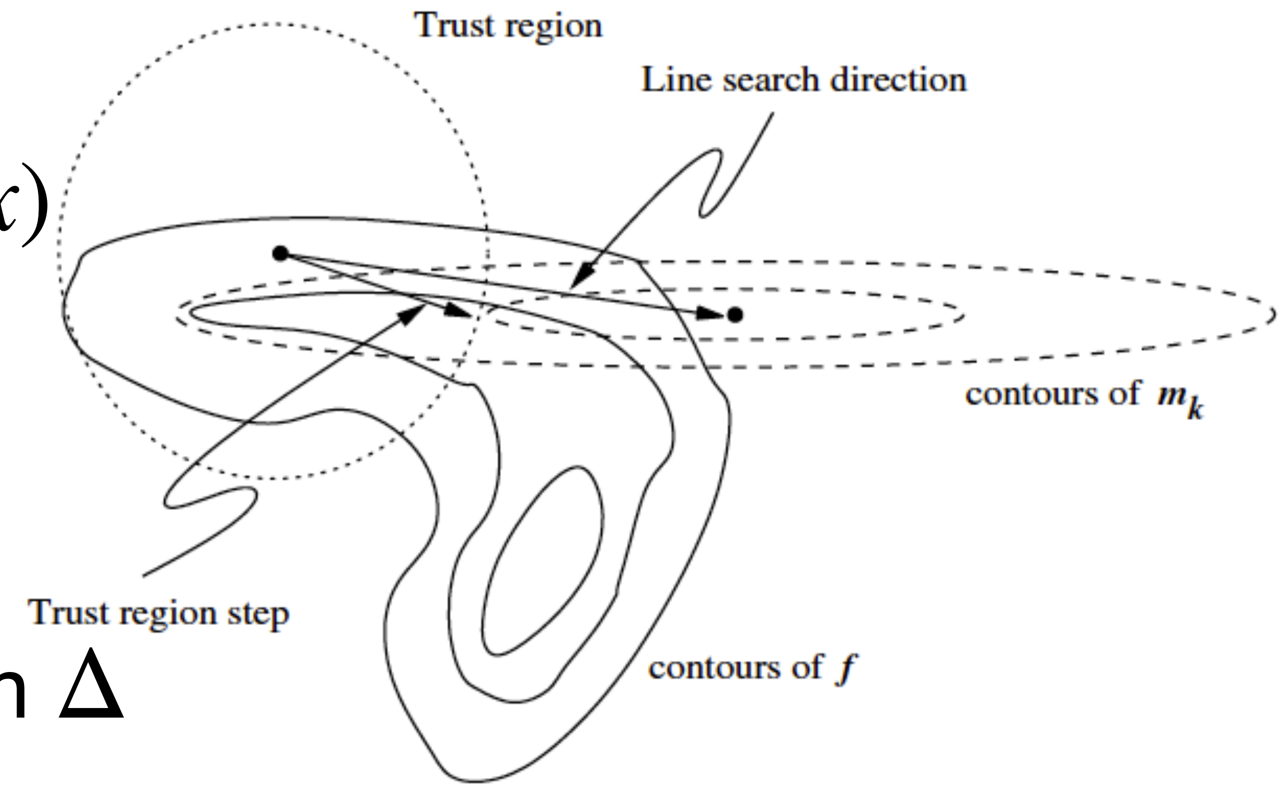


$$\min_{\delta \in \Delta_k} \hat{f}_k(x_k + \delta)$$

- Remark: GD is a line search method

Trust Region Methods (Formally)

- 3 major steps of a trust region method:



(A1) Choose a surrogate function $m_k(x)$

(A2) Specify a trust region Δ

(A3) Find an approximate optimizer in Δ

- Question: How to choose the surrogate function?
- Question: How to specify a good trust region?

(A1) Surrogate Functions

- **Surrogate functions** are required to be easy to optimize or evaluate
- Examples:

Linear approx. $$f(x_k + p) \approx m_k(p) := f(x_k) + \nabla f(x_k)^T p$$

Quadratic approx. $$f(x_k + p) \approx m_k(p) := f(x_k) + \nabla f(x_k)^T p + \frac{1}{2} p^T \nabla^2 f(x_k) p$$

- Example: Trust region optimization (TRO) subproblem with a quadratic surrogate function

$$\min \; m_k(p) = f(x_k) + \nabla f(x_k)^T p + \frac{1}{2} p^T \nabla^2 f(x_k) p$$

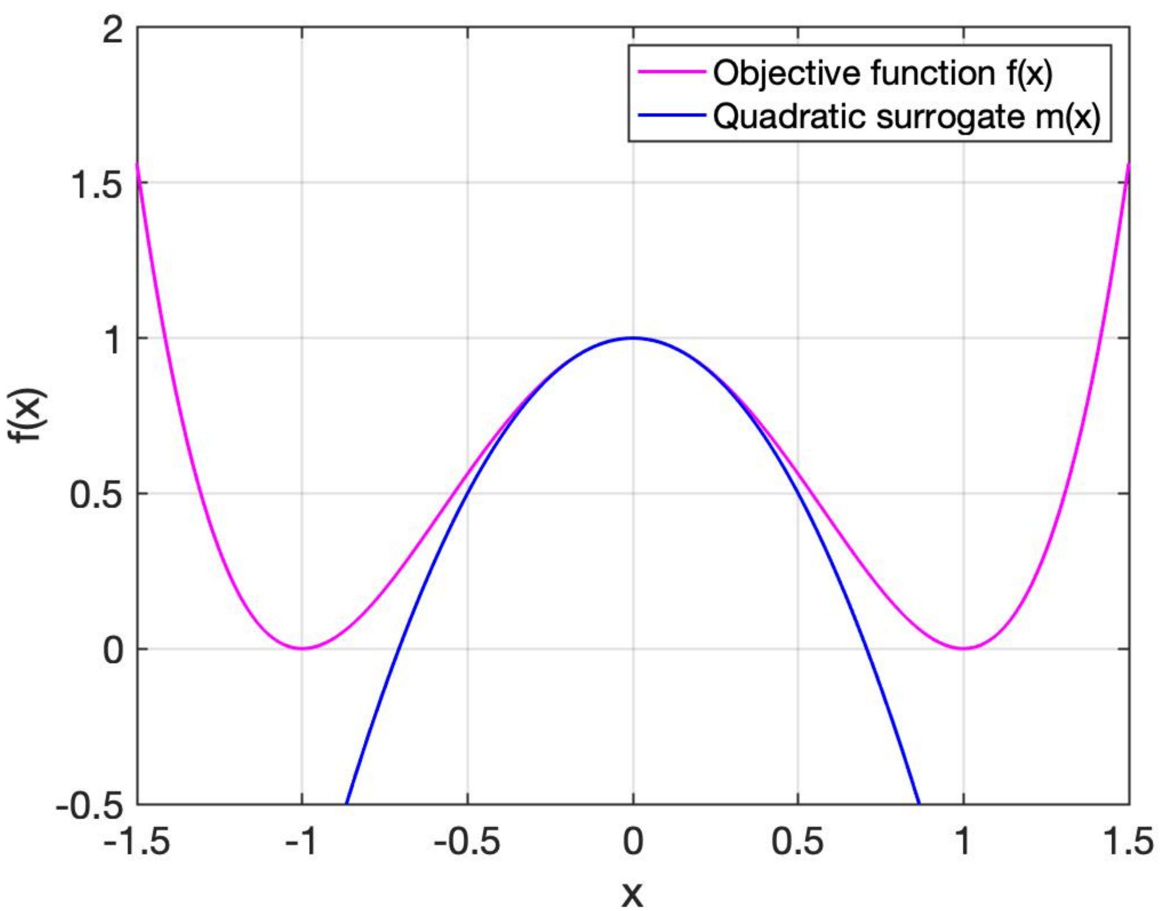
$$\text{s.t. } \|p\|_2 \leq \Delta_k$$

(If $\nabla^2 f(x_k)$ is positive semi-definite, then the problem is a QCQP and is convex)

Example: Quadratic Surrogate Functions

- Example: Minimize $f(x) = (x^2 - 1)^2, x \in \mathbb{R}$

$$\min \ m_k(p) = f(x_k) + \nabla f(x_k)^T p + \frac{1}{2} p^T \nabla^2 f(x_k) p$$
$$\text{s.t. } \|p\|_2 \leq \Delta_k$$



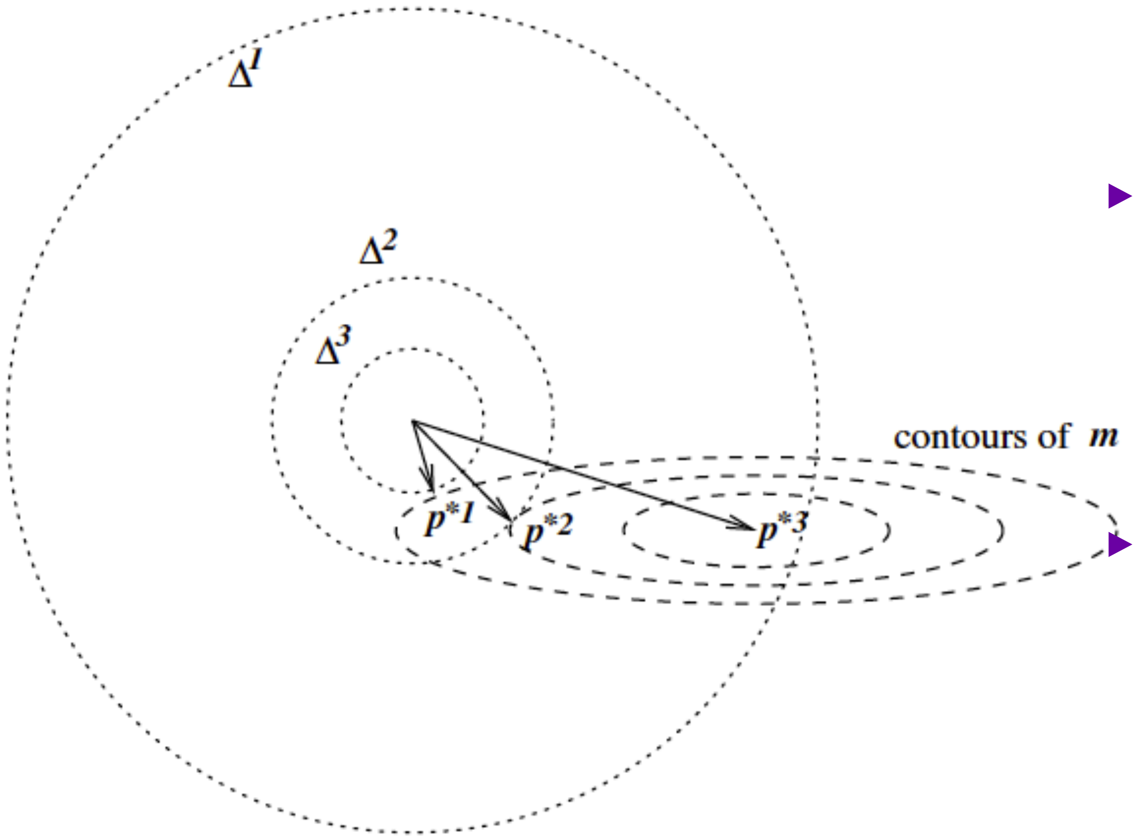
- Question: What is the quadratic surrogate function at $x_k = 0$? Any benefit of TR compared to GD?

(Example Credit: Mihai Anitescu)

(A2) Specify the Size of a Trust Region

- Trust region size is usually captured by the radius Δ_k
- Question: How to tune Δ_k?

Define $\rho_k := \dfrac{f(x_k) - f(x_k + p)}{m_k(0) - m_k(p)}$



- If ρ_k is close to 0 or negative:

  Decrease Δ

- If ρ_k is close to 1 and $\|p_k\| = \Delta_k$:

  Increase Δ

- Otherwise: Δ *unchanged*

Let’s apply TR method and build TRPO step by step!

Goal: The Ultimate TRPO Algorithm

- Trust-Region Policy Optimization (TRPO) Algorithm:

Step 1: Initialize θ_0

Step 2: For iteration $k = 0,1,2,\cdots$

Step 2-1: Collect trajectories by running the current policy π_{θ_k}

Step 2-2: Obtain advantage $A^{\theta_k}(s,a)$ for the current policy π_{θ_k}

Step 2-3: Update the policy by solving

$$\theta_{k+1} = \arg\max_{\theta} \mathbb{E}_{s \sim d_\mu^{\pi_{\theta_k}}, a \sim \pi_{\theta_k}(\cdot|s)} \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\theta_k}(s,a) \right]$$

subject to $\bar{D}_{KL}(\pi_{\theta_k} \| \pi_\theta) \leq \delta$

Recall: Trust Region Methods

- Recall: 3 major steps of a trust region method:

(A1) Choose a surrogate function $m_k(x)$

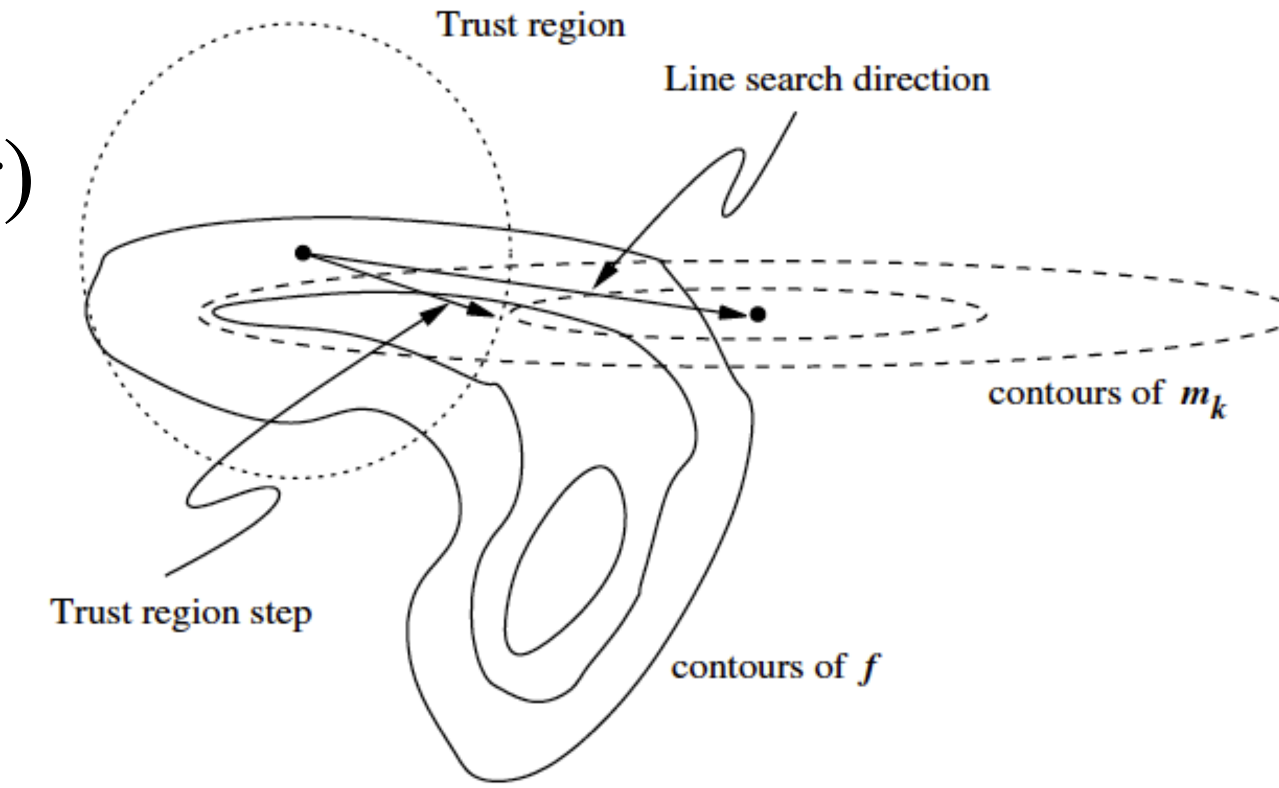
Approximate $d_\mu^{\pi_{new}}(s)$ by $d_\mu^{\pi_{old}}(s)$

(A2) Specify a trust region Δ

KL divergence between π_{old}, π_{new}

(A3) Find an approximate optimizer in Δ

Convexify the problem by 1st-order and 2nd-order approximation



(A1) Surrogate Function in TRPO

- Recall: Average performance difference lemma

$$\eta(\pi_{new}) = \eta(\pi_{old}) + \mathbb{E}_{s \sim d_\mu^{\pi_{new}}, a \sim \pi_{new}(\cdot|s)}\left[A^{\pi_{old}}(s_t, a_t)\right]$$
$$= \eta(\pi_{old}) + \sum_s d_\mu^{\pi_{new}}(s) \sum_a \pi_{new}(a|s) A^{\pi_{old}}(s, a)$$

Approximate $d_\mu^{\pi_{new}}(s)$ by $d_\mu^{\pi_{old}}(s)$:

$$\eta(\pi_{new}) - \eta(\pi_{old}) \approx \sum_s d_\mu^{\pi_{old}}(s) \sum_a \pi_{new}(a|s) A^{\pi_{old}}(s, a)$$

- Define: Surrogate function $L_{\pi_{old}}(\pi_{new})$ in TRPO

$$L_{\pi_{old}}(\pi_{new}) = \eta(\pi_{old}) + \sum_s d_\mu^{\pi_{old}}(s) \sum_a \pi_{new}(a|s) A^{\pi_{old}}(s, a)$$

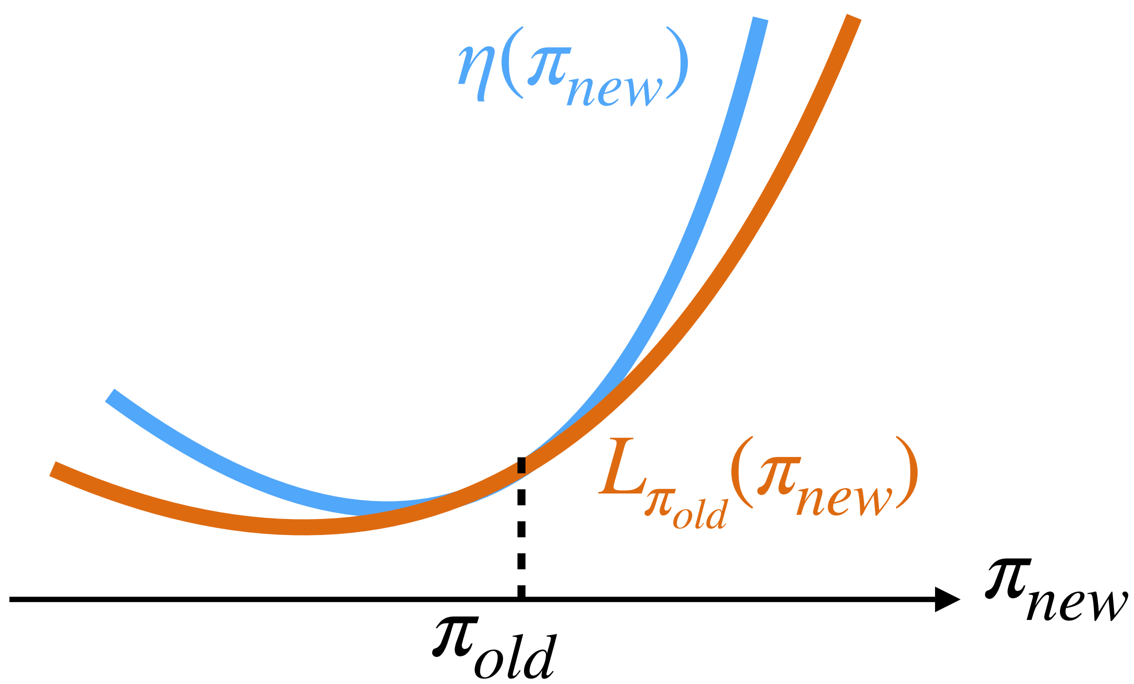
(A1) Why is $L_{\pi_{old}}(\pi_{new})$ a Good Surrogate Function?

$$L_{\pi_{old}}(\pi_{new}) = \eta(\pi_{old}) + \sum_s d_\mu^{\pi_{old}}(s) \sum_a \pi_{new}(a\,|\,s) A^{\pi_{old}}(s,a)$$

- $L_{\pi_{old}}(\pi_{new})$ satisfy two properties: $\pi_{old} \equiv \pi_{\theta_1}, \pi_{new} \equiv \pi_\theta$

1. $L_{\pi_{\theta_1}}(\pi_{\theta_1}) = \eta(\pi_{\theta_1})$
2. $\nabla_\theta L_{\pi_{\theta_1}}(\pi_\theta)|_{\theta=\theta_1} = \nabla_\theta \eta(\pi_\theta)|_{\theta=\theta_1}$

(HW2 problem)



- Intuition: If π_{old}, π_{new} are close, then improvement in $L_{\pi_{old}}(\pi_{new})$ implies improvement in $\eta(\pi_{new})$

Kullback-Leibler divergence Between Policies

- Notation: Kullback-Leibler (KL) divergence

$$D_{KL}(\pi(\cdot \mid s) \| \tilde{\pi}(\cdot \mid s)) := \sum_a \pi(a \mid s) \log\left(\frac{\pi(a \mid s)}{\tilde{\pi}(a \mid s)}\right)$$

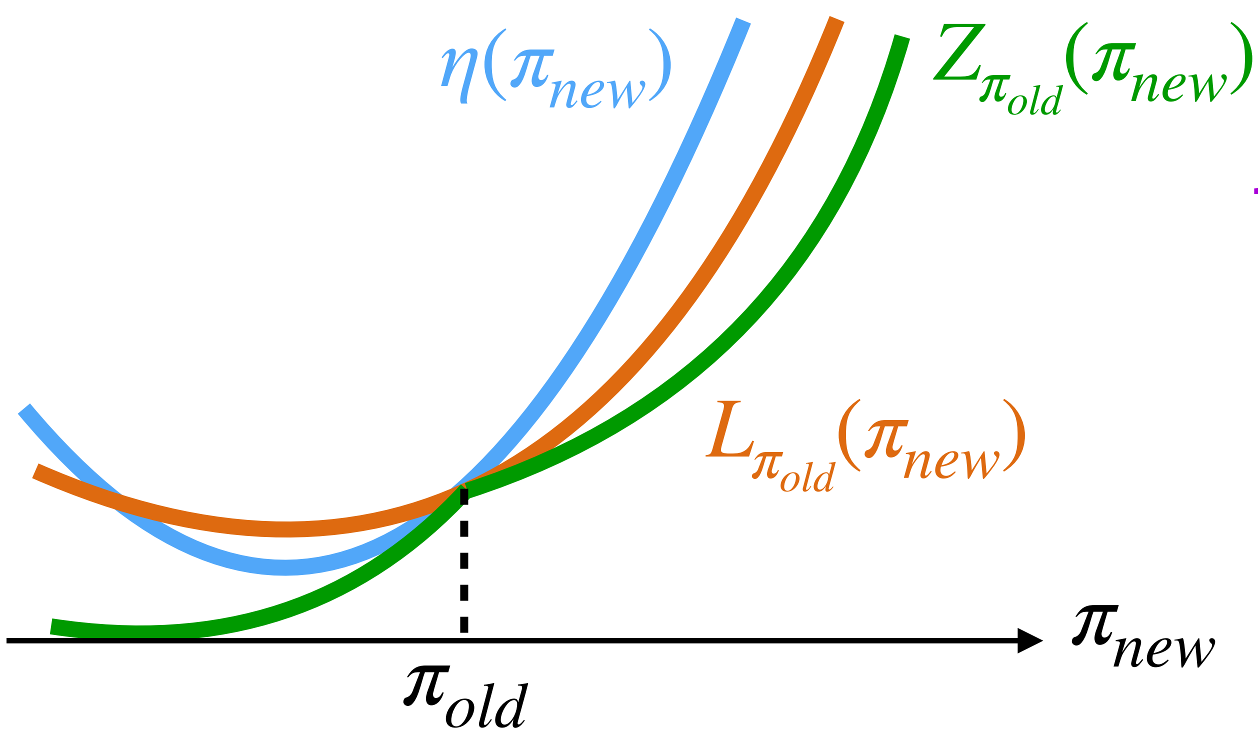
$$D_{KL}^{\max}(\pi \| \tilde{\pi}) := \max_s D_{KL}(\pi(\cdot \mid s) \| \tilde{\pi}(\cdot \mid s))$$

(A2) How to Specify a Trust Region Δ?

- **Recall**: $L_{\pi_{old}}(\pi_{new})$ and $\eta(\pi_{new})$ are close if π_{old}, π_{new} are close
- **Policy Improvement Bound (PIB)**: Let $\varepsilon := \max_{s,a} |A^{\pi_{old}}(s,a)|$

$$\eta(\pi_{new}) \geq \underbrace{L_{\pi_{old}}(\pi_{new}) - \frac{4\varepsilon\gamma}{(1-\gamma)^2} D_{KL}^{\max}(\pi_{old} \| \pi_{new})}_{=:Z_{\pi_{old}}(\pi_{new})}$$

- **Question**: How to specify a trust region?



1st attempt: $D_{KL}^{\max}(\pi_{old} \| \pi_{new}) \leq \delta$

Is $D_{KL}^{\max}(\pi_{old} \| \pi_{new})$ easy to evaluate?

(A2) How to Specify a Trust Region Δ?

- **Policy Improvement Bound (PIB)**: Let $\varepsilon := \max_{s,a} |A^{\pi_{old}}(s,a)|$

$$\eta(\pi_{new}) \geq \underbrace{L_{\pi_{old}}(\pi_{new}) - \frac{4\varepsilon\gamma}{(1-\gamma)^2} D_{KL}^{\max}(\pi_{old} \| \pi_{new})}_{=:Z_{\pi_{old}}(\pi_{new})}$$

2nd attempt: Use $\bar{D}_{KL}(\pi_{old} \| \pi_{new})$ instead of $D_{KL}^{\max}(\pi_{old} \| \pi_{new})$

$$\bar{D}_{KL}(\pi_{old} \| \pi_{new}) := \mathbb{E}_{s \sim \pi_{old}}[D_{KL}(\pi_{old}(\cdot \mid s) \,||\, \pi_{new}(\cdot \mid s))]$$

Trust region in TRPO: $\bar{D}_{KL}(\pi_{old} \| \pi_{new}) \leq \delta$

Put Everything Together

- Trust-Region Policy Optimization (TRPO) Algorithm:

Step 1: Initialize θ_0

Step 2: For iteration $k = 0,1,2,\cdots$

Step 2-1: Collect trajectories by running the current policy π_{θ_k}

Step 2-2: Obtain advantage $A^{\theta_k}(s,a)$ for the current policy π_{θ_k}

Step 2-3: Update the policy by solving

$$\theta_{k+1} = \arg\max_{\theta} L_{\pi_{\theta_k}}(\pi_\theta) \quad \Big(\equiv \arg\max_{\theta} \mathbb{E}_{s\sim d_\mu^{\pi_{\theta_k}}, a\sim\pi_{\theta_k}(\cdot|s)}\Big[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\theta_k}(s,a)\Big]\Big)$$

subject to $\bar{D}_{KL}(\pi_{\theta_k} \| \pi_\theta) \le \delta$

One remaining practical issue with TRPO…

$$\theta_{k+1} = \arg\max_{\theta} L_{\pi_{\theta_k}}(\pi_\theta) \quad \left(\equiv \arg\max_{\theta} \mathbb{E}_{s \sim d_\mu^{\pi_{\theta_k}}, a \sim \pi_{\theta_k}(\cdot|s)} \left[\frac{\pi_\theta(a\,|\,s)}{\pi_{\theta_k}(a\,|\,s)} A^{\theta_k}(s,a) \right] \right.$$

subject to $\bar{D}_{KL}(\pi_{\theta_k} \| \pi_\theta) \leq \delta$

How to efficiently solve this constrained problem?

(A3) Find an approximate optimizer in Δ

$$\theta_{k+1} = \arg\max_{\theta} L_{\pi_{\theta_k}}(\pi_\theta) \quad \left(\equiv \arg\max_{\theta} \mathbb{E}_{s \sim d_\mu^{\pi_{\theta_k}}, a \sim \pi_{\theta_k}(\cdot|s)} \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\theta_k}(s,a) \right] \right)$$

subject to $\bar{D}_{KL}(\pi_{\theta_k} \| \pi_\theta) \leq \delta$

- Idea: Approximation
    1. Linear approximation to the objective $L_{\theta_k}(\theta)$
    2. Quadratic approximation to the KL constraint
- The problem under approximation:

Hessian of $\bar{D}_{KL}(\pi_{\theta_k} \| \pi_\theta)$

Maximize	$(\theta - \theta_k)^\top \nabla_\theta L_{\theta_k}(\theta)\|_{\theta=\theta_k}$
subject to	$\frac{1}{2}(\theta - \theta_k)^\top H (\theta - \theta_k) \leq \delta$

- Question: Why is this approximation helpful?

- The problem under approximation:

Hessian of $\bar{D}_{KL}(\pi_{\theta_k} \| \pi_\theta)$

$$\begin{aligned} &\text{Maximize} \quad (\theta - \theta_k)^\top \nabla_\theta L_{\theta_k}(\theta)|_{\theta=\theta_k} \\ &\text{subject to} \quad \frac{1}{2}(\theta - \theta_k)^\top H (\theta - \theta_k) \le \delta \end{aligned}$$

- The Hessian of $\bar{D}_{KL}(\pi_{\theta_k} \| \pi_\theta)$ is positive semi-definite

- The constraint is therefore convex (and can be easily analyzed)

- The solution is:

usually called "natural policy gradient"

$$\theta = \theta_k + \alpha H^{-1} \nabla_\theta L_{\theta_k}(\theta)|_{\theta=\theta_k} \qquad \text{(HW2 problem)}$$

Sham Kakade, "A Natural Policy Gradient", NIPS 2002

Quick Summary: What is TRPO?

TRPO = TR Method on RL

With 3 key steps…

1. Approximate $d_\mu^{\pi_{new}}(s)$ by $d_\mu^{\pi_{old}}(s)$

2. Trust region by KL divergence between π_{old}, π_{new}

3. Simplify the problem by linear and quadratic approximation

Assignment for this lecture:

- Spend 30 minutes going through the idea of TRPO again
- Spend 30 minutes reading the code of TRPO
    - https://github.com/ikostrikov/pytorch-trpo

```python
def trpo_step(model, get_loss, get_kl, max_kl, damping):
    loss = get_loss()
    grads = torch.autograd.grad(loss, model.parameters())
    loss_grad = torch.cat([grad.view(-1) for grad in grads]).data

    def Fvp(v):
        kl = get_kl()
        kl = kl.mean()

        grads = torch.autograd.grad(kl, model.parameters(), create_graph=True)
        flat_grad_kl = torch.cat([grad.view(-1) for grad in grads])

        kl_v = (flat_grad_kl * Variable(v)).sum()
        grads = torch.autograd.grad(kl_v, model.parameters())
        flat_grad_grad_kl = torch.cat([grad.contiguous().view(-1) for grad in grads]).data

        return flat_grad_grad_kl + v * damping

    stepdir = conjugate_gradients(Fvp, -loss_grad, 10)

    shs = 0.5 * (stepdir * Fvp(stepdir)).sum(0, keepdim=True)

    lm = torch.sqrt(shs / max_kl)
    fullstep = stepdir / lm[0]

    neggdotstepdir = (-loss_grad * stepdir).sum(0, keepdim=True)
    print(("lagrange multiplier:", lm[0], "grad_norm:", loss_grad.norm()))

    prev_params = get_flat_params_from(model)
    success, new_params = linesearch(model, get_loss, prev_params, fullstep,
                                     neggdotstepdir / lm[0])
    set_flat_params_to(model, new_params)

    return loss
```

- Could you explain the purpose of each line?
- Could you find any part of the code that we have not discussed in this lecture?

We will discuss this next time!