535514: Reinforcement Learning
Lecture 16 — TRPO and NPG

Ping-Chun Hsieh

April 18, 2024

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>

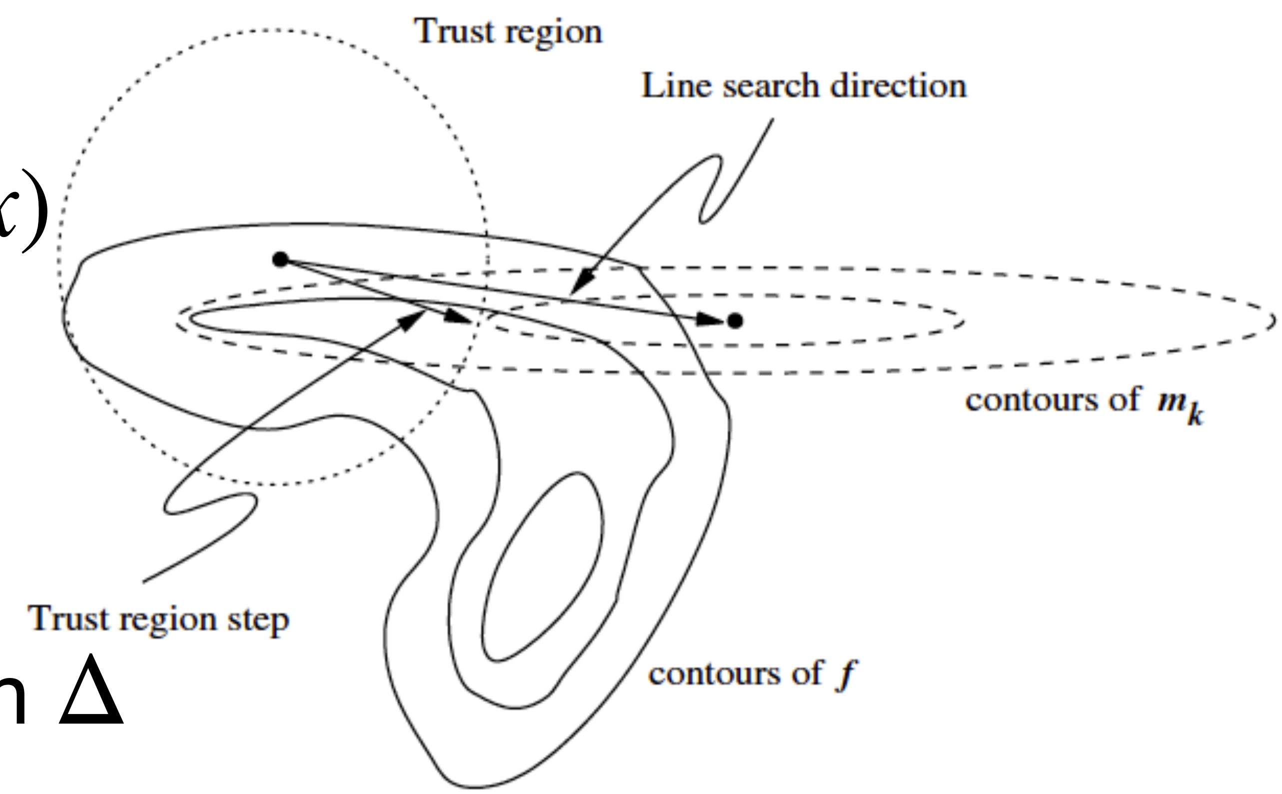
Recall: Trust Region Methods

- 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?

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

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

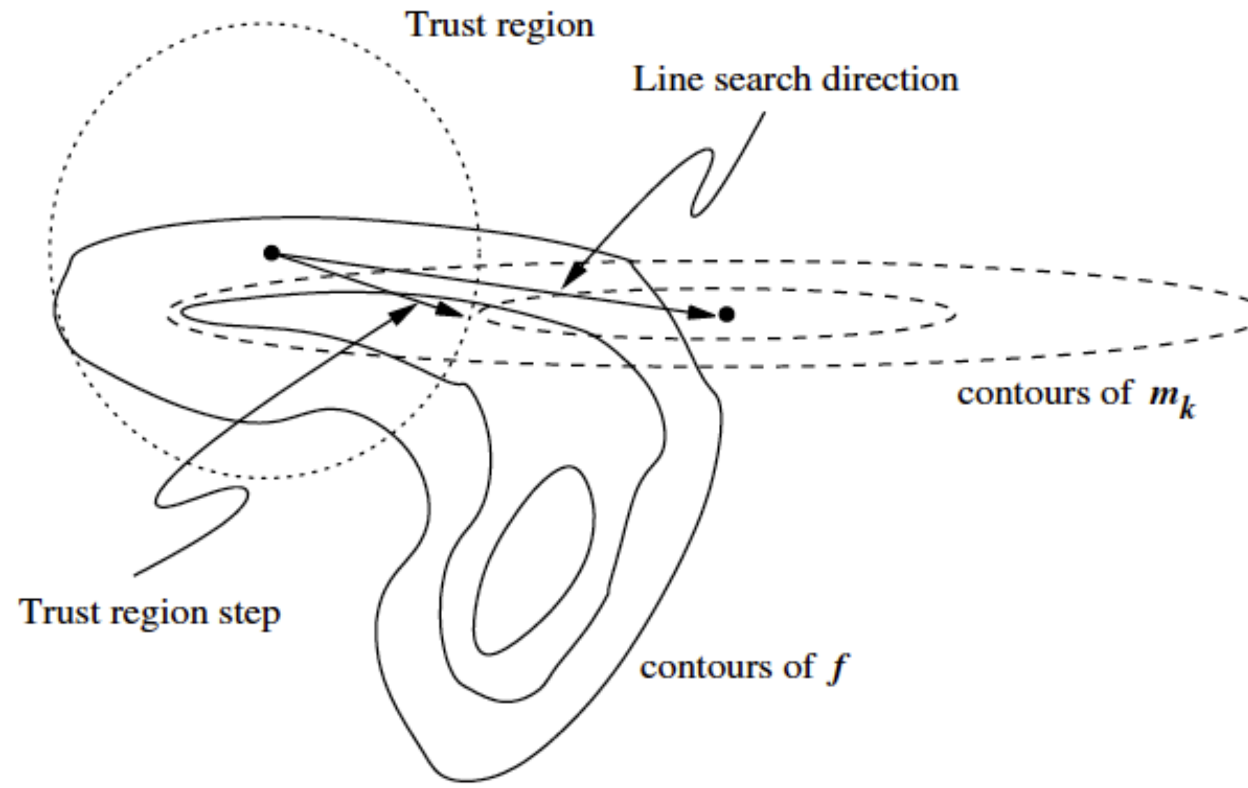
$$\text{subject to } \bar{D}_{KL}(\pi_{\theta_k} \| \pi_\theta) \leq \delta$$

Trust Region Methods for RL

- 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

Average Performance Difference Lemma

- **Performance Difference Lemma**:

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

- 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*"?

To simplify notations, let's use

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

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

(A1) Surrogate Function in TRPO

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?)

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

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

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

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

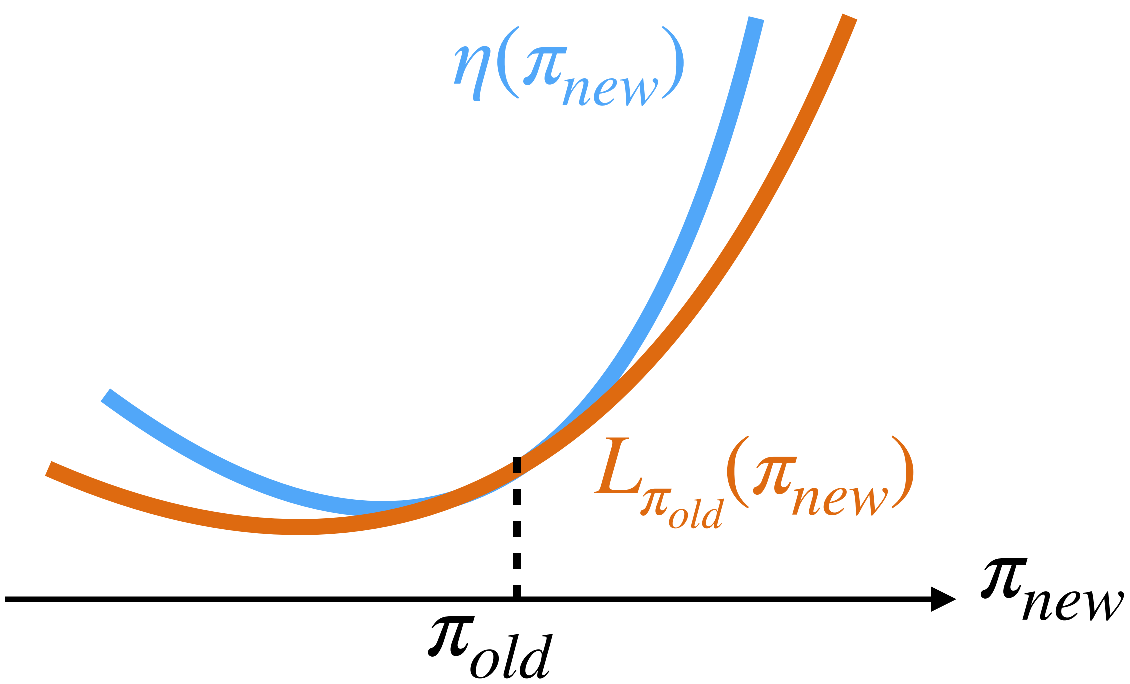
(A1) Why is $L_{\pi_{old}}(\pi_{new})$ a Good Surrogate Function?

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

- $L_{\pi_{old}}(\pi_{new})$ 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

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

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

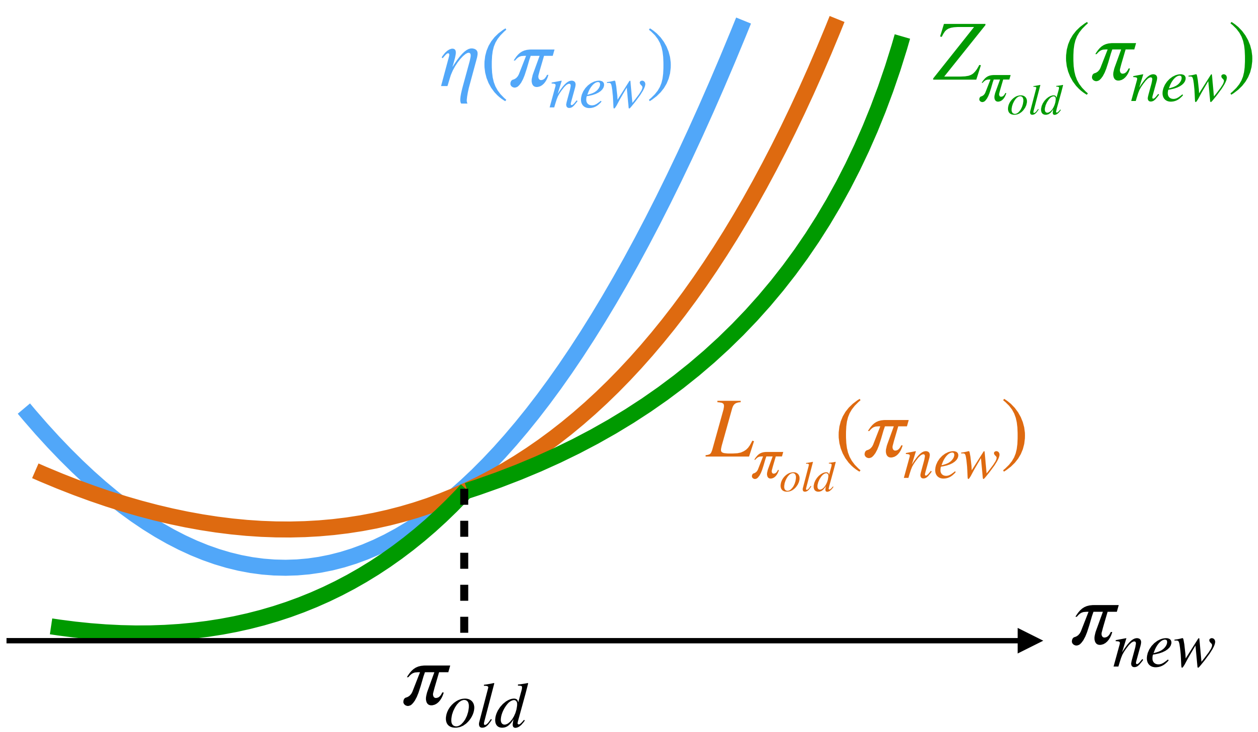
(A2) How to Specify a Trust Region Δ?

- 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…

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

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

- 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	$(\theta - \theta_k)^\top \nabla_\theta L_{\theta_k}(\theta)\vert_{\theta=\theta_k}$
subject to	$\frac{1}{2}(\theta - \theta_k)^\top H (\theta - \theta_k) \le \delta$

- Question: Why is this approximation helpful?

- The problem under approximation:

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

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

- 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 (NPG)"

$$\theta = \theta_k + \alpha H_{\theta_k}^{-1} \nabla_\theta L_{\theta_k}(\theta)|_{\theta=\theta_k}$$

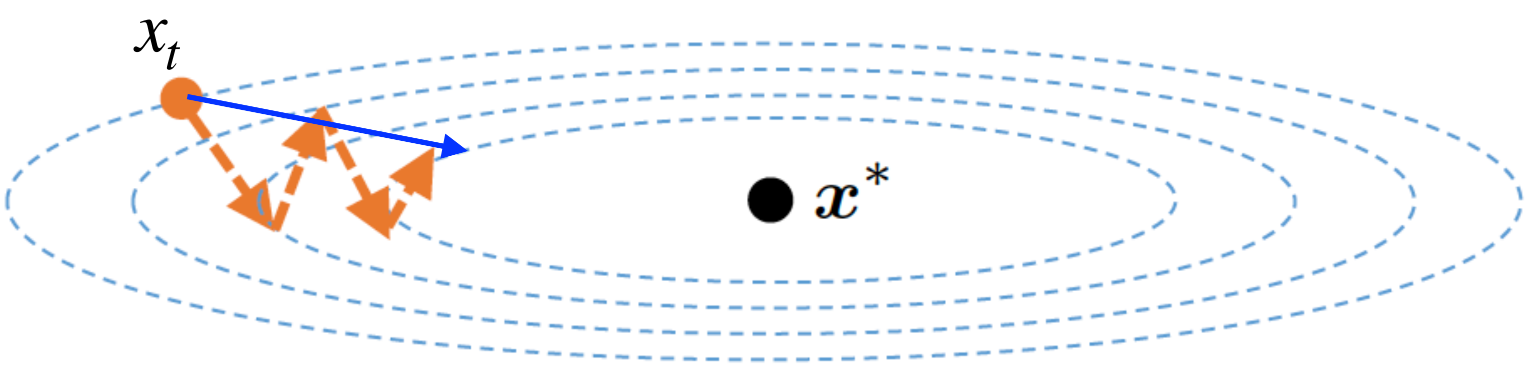
(HW2 problem)

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

A Primer for NPG: Scaled Gradient

$$\text{minimize}_{x \in \mathbb{R}^2} \quad f(x) := \frac{1}{2}(x - x^*)^\top Q(x - x^*)$$

Suppose $Q = [Q_{11}, 0; 0, Q_{22}]$ is a diagonal matrix with $0 < Q_{11} \ll Q_{22}$



Idea: Accelerate gradient descent by ***scaling the gradient***

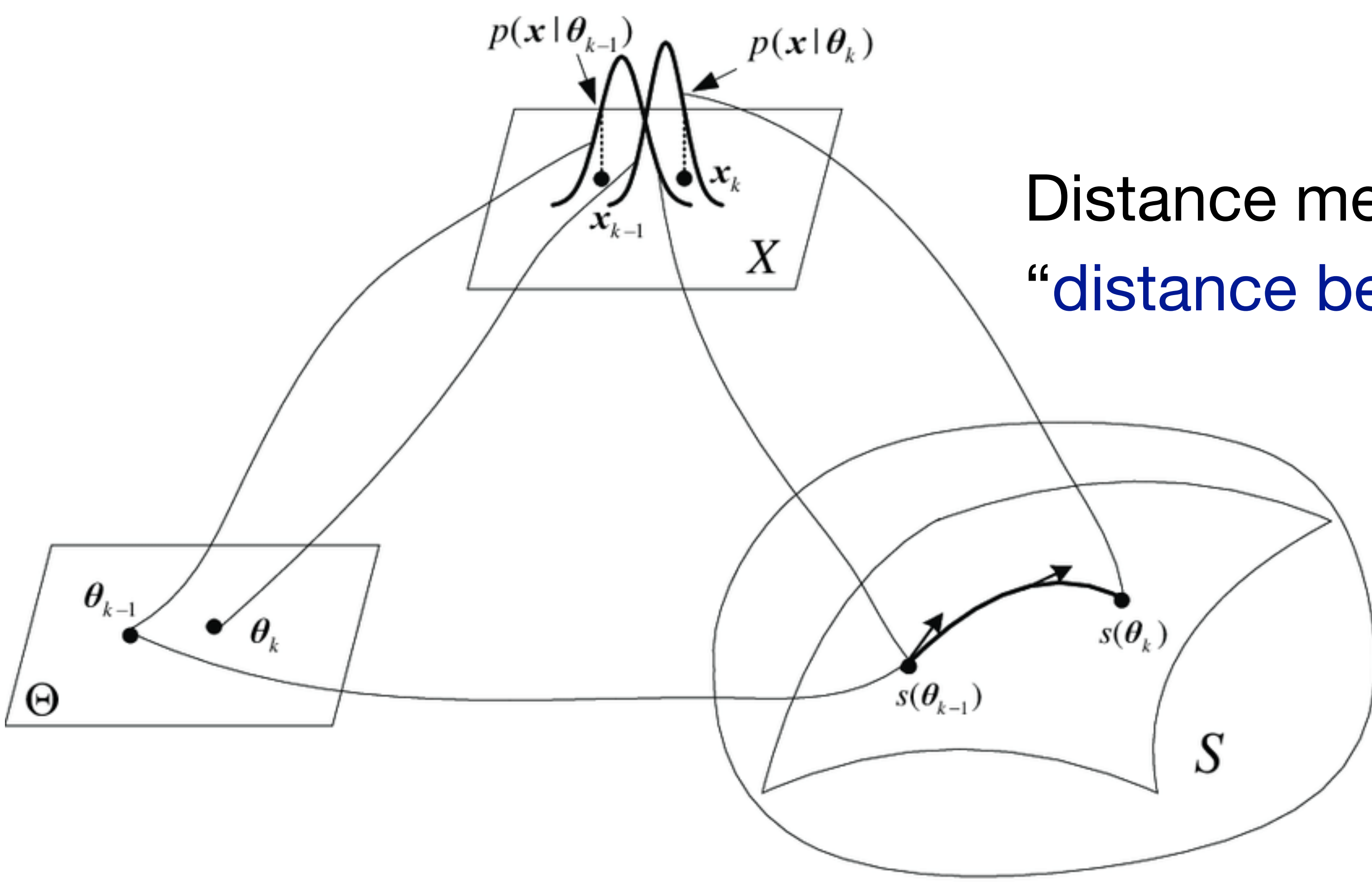
$$x_{t+1} = x_t - \eta_t Q^{-1} \nabla f(x_t) = x_t - \eta_t (x_t - x^*)$$

Natural Policy Gradient (NPG)

NPG: Use Fisher information matrix to scale the gradient

$$\theta_{k+1} \leftarrow \theta_k + \eta \cdot H_{\theta_k}^{-1} \nabla_\theta V^{\pi_\theta}(\mu) \Big|_{\theta=\theta_k}$$

$$H_\theta := \mathbb{E}_{s \sim d_\mu^{\pi_\theta}}\left[\left(\nabla_\theta \log \pi_\theta(a|s) \right) \left(\nabla_\theta \log \pi_\theta(a|s) \right)^\top \right]$$ **(Fisher information matrix)**



Distance metric of θ does not fully capture "distance between probability distributions"

(For ease of presentation, assume F_θ has full rank; otherwise use pseudo inverse)

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!