

# Trust Region Policy Optimization (TRPO)

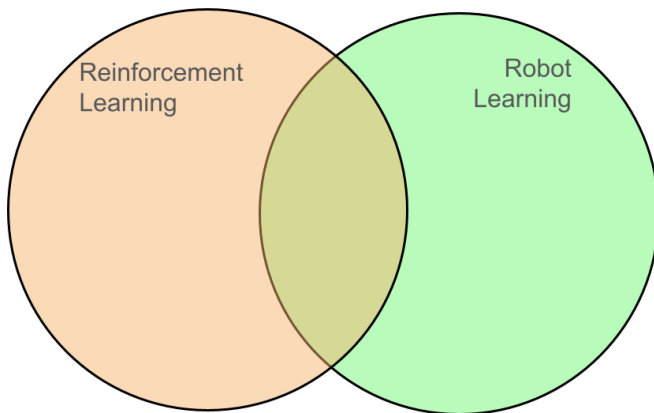
Original Paper by Schulman et al. [2017]

Matthew Vandergrift

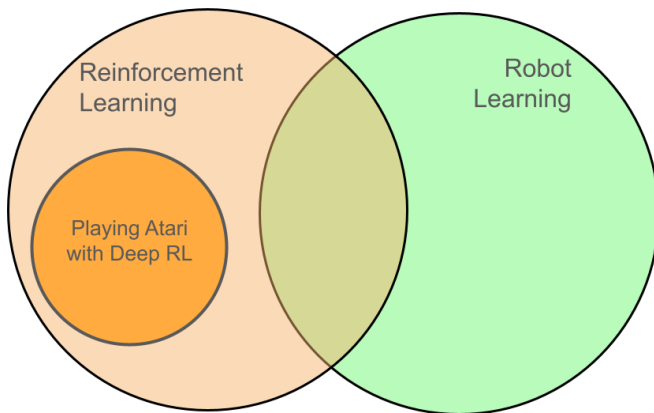
Robot Learning Seminar Presentation

March 2025

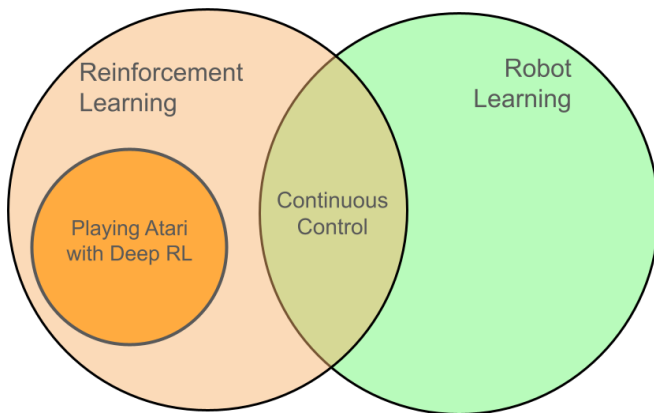
# Motivation



# Motivation



# Motivation



# Existing Solutions

- Reinforce
- Basic Actor-Critic Algorithms
- Natural Policy Gradients
- Derivative Free Methods: cross-entropy method, covariance matrix adaptation.

# Once again, ... RL

"RL is computational framework for learning from interaction"  
Sutton and Barto [2018]. The agent interacts with an environment with the goal of maximizing expected return. Let us denote expected return for a particular policy by  $\eta(\pi)$ .

# Advantage Function

Recall the advantage function  $A_\pi(s, a)$  defined as,

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s)$$

This function tells us how "good" taking action is compared to what we would have done otherwise.

# Policy Improvement via Advantage

Since policies are just collections of actions, we can use advantage function to evaluate them. Let  $\pi$  and  $\pi'$  be two different policies. Equation 1 gives a way to write the performance of  $\pi'$  using the performance of  $\pi$  and the advantage function.

$$\eta(\pi') = \eta(\pi) + \sum_s \mu_{\pi'}(s) \sum_a \pi'(a|s) A_{\pi}(s, a). \quad (1)$$

Proof in appendix



# RL is Solved!

At first glance we have a solution!

---

## Algorithm The Perfect RL Algorithm

---

**Require:**  $\pi$  and  $\eta(\pi)$

$$1: \max_{\pi'} (\eta(\pi) + \sum_s \mu_{\pi'}(s) \sum_a \pi'(a|s) A_{\pi}(s, a))$$

---

This doesn't work because we have a dependence on the policy distribution which is not something we have access to when considering  $\pi'$ .

# Dealing with $\mu_{\pi'}$

- Let's use  $\mu_{\pi}$  instead of  $\mu_{\pi'}$
- Define  $L_{\pi}(\pi') = \eta(\pi) + \sum_s \mu_{\pi}(s) \sum_a \pi'(a|s) A_{\pi}(s, a)$
- Assume  $\pi$  is parameterized by weights  $\theta$ .
- Gives us a local first order approximation,  
$$\nabla_{\theta} L_{\pi_{\theta_0}}(\theta_{\theta})|_{\theta=\theta_0} = \nabla_{\theta} \eta_{\pi_{\theta_0}}(\theta_{\theta})|_{\theta=\theta_0}$$
- If we take **small** steps in  $\theta$  then we can use our 'Perfect RL algorithm'.

# What is a small step?

A Major Contribution of the Paper is the following bound,

## Theorem

*Let  $D_{KL}^{max}(\pi, \pi') := \max_s D_{KL}(\pi(*|S) || \pi'(*|s))$ . We then have that,  $\eta(\pi') \geq L_{\pi}(\pi') - C D_{KL}^{max}(\pi, \pi')$  where  $C = \frac{4\epsilon\gamma}{(1-\gamma)^2}$*

This means we can bound the improvement between any-two policies based on their KL divergence. This means if we optimize within a certain KL distance we can *guarantee improvement*.

# A More Perfect RL Algorithm

---

**Algorithm 1** Policy iteration algorithm guaranteeing non-decreasing expected return  $\eta$

---

Initialize  $\pi_0$ .

**for**  $i = 0, 1, 2, \dots$  until convergence **do**

    Compute all advantage values  $A_{\pi_i}(s, a)$ .

    Solve the constrained optimization problem

$$\pi_{i+1} = \arg \max_{\pi} [L_{\pi_i}(\pi) - CD_{\text{KL}}^{\max}(\pi_i, \pi)]$$

$$\text{where } C = 4\epsilon\gamma/(1 - \gamma)^2$$

$$\text{and } L_{\pi_i}(\pi) = \eta(\pi_i) + \sum_s \rho_{\pi_i}(s) \sum_a \pi(a|s) A_{\pi_i}(s, a)$$

**end for**

---

The constraint is not computable due to  $\max_s f(s)$ .

# Make RL in Practice

- Computable constraint
- Cheap cost function
- Cheap constrained optimization solver

# Computable Constraint

We want to make our constrained optimization solvable.

- Get rid of max KL constraint over the whole state space.
- Define 'Average' KL,

$$\bar{D}_{KL} := \mathbb{E}_{s \sim \mu_{\pi_{\theta}}} [D_{KL}(\pi_{\theta}(*|S) || \pi_{\theta_{old}}(*|s))]$$

- Estimate this Expectation using roll-outs under the policy.

This gives us,

$$\begin{aligned} & \max_{\theta} L_{\theta_{old}}(\theta) \\ & \text{subject to } \bar{D}_{KL}(\theta_{old}, \theta) \leq \delta \end{aligned}$$

# Cheap Cost Function

We want to make  $L_{\theta_{old}}(\theta)$  fast to compute.

$$L_{\theta_{old}}(\theta) = \sum_s \mu_{\pi_{\theta_{old}}} \sum_a \pi_{\theta}(a|s) A_{\theta_{old}}(s, a) \quad (\text{Definition of } L)$$

We replace the sums by an expectation, and  $A$  with estimator  $\hat{A}$

$$L_{\theta_{old}}(\theta) = \mathbb{E}_{a \sim \pi_{\theta_{old}}, s \sim \pi_{\theta_{old}}} \left[ \frac{\pi_{\theta}(a|s)}{\pi_{\theta_{old}}(a|s)} \cdot \hat{A}_{\theta_{old}}(s, a) \right]$$

# Cheap constrained optimization solver

$$\max_{\theta} L_{\theta_{old}}(\theta) \text{ subject to } \bar{D}_{KL}(\theta_{old}, \theta) \leq \delta$$



# Cheap constrained optimization solver

$$\max_{\theta} L_{\theta_{old}}(\theta) \text{ subject to } \bar{D}_{KL}(\theta_{old}, \theta) \leq \delta$$

Actual  
Taylor expansion

$$f(x) = \boxed{f(0) + f'(0)x} + \frac{f''(0)}{2!}x^2 + \frac{f'''(0)}{3!}x^3 \dots$$

Lies invented by mathematicians  
to feel superior to physicists

# Cheap constrained optimization solver

$$\max_{\theta} L_{\theta_{old}}(\theta) \text{ subject to } \bar{D}_{KL}(\theta_{old}, \theta) \leq \delta$$

First Order of Taylor Expansion to  $L_{\theta_{old}}(\theta)$

# Cheap constrained optimization solver

$$\max_{\theta} L_{\theta_{old}}(\theta) \text{ subject to } \bar{D}_{KL}(\theta_{old}, \theta) \leq \delta$$

First Order of Taylor Expansion to  $L_{\theta_{old}}(\theta)$

$$L_{\theta_{old}}(\theta) \approx L_{\theta_{old}}(\theta_{old}) + g^T(\theta - \theta_{old})$$

$$L_{\theta_{old}}(\theta) \approx 0 + g^T(\theta - \theta_{old})$$

Let  $(\theta - \theta_{old}) = \Delta\theta$

$$L_{\theta_{old}}(\theta) \approx g^T \Delta\theta$$

# Cheap constrained optimization solvers

$$\max_{\theta} L_{\theta_{old}}(\theta) \text{ subject to } \bar{D}_{KL}(\theta_{old}, \theta) \leq \delta$$

Second Order of Taylor Expansion of Constraint

# Cheap constrained optimization solvers

$$\max_{\theta} L_{\theta_{old}}(\theta) \text{ subject to } \bar{D}_{KL}(\theta_{old}, \theta) \leq \delta$$

Second Order of Taylor Expansion of Constraint

$$\hat{D}_{KL}(\theta_{old}||\theta) \approx \hat{D}_{KL}(\theta_{old}||\theta_{old}) + \nabla_{\theta} \hat{D}_{KL}(\theta_{old}||\theta)|_{\theta_{old}} \cdot (\theta - \theta_{old}) + \frac{1}{2}(\theta - \theta_{old})^T H(\theta - \theta_{old})$$

$$\hat{D}_{KL}(\theta_{old}||\theta) \approx 0 + 0 + \frac{1}{2}\Delta\theta^T H\Delta\theta$$

$$\hat{D}_{KL}(\theta_{old}||\theta) \approx \frac{1}{2}\Delta\theta^T H\Delta\theta$$

# Cheap constrained optimization solvers

We now have a new constrained optimization problem,

$$\begin{aligned} & \operatorname{argmax}_{\theta} g^T \Delta \theta \\ & \text{subject to } \frac{1}{2} \Delta \theta^T H \Delta \theta \leq \delta \end{aligned}$$

Now since it's a nice quadratic/convex constraint we can solve,

$$\theta = \theta_{old} + \sqrt{\frac{2\delta}{g^t H^{-1} g}} H^{-1} g.$$

skip derivation

# Steps to Solve

Use Lagrangian to solve the constrained optimization

# Steps to Solve

Use Lagrangian to solve the constrained optimization

$$L = g^T \Delta\theta + \lambda \left( \frac{1}{2} (\theta - \theta_{old})^T H (\theta - \theta_{old}) - \delta \right)$$



# Steps to Solve

Use Lagrangian to solve the constrained optimization

$$L = g^T \Delta\theta + \lambda \left( \frac{1}{2} (\theta - \theta_{old})^T H (\theta - \theta_{old}) - \delta \right)$$

$$\nabla_{\theta} L = 0 \implies g^t + \frac{\lambda}{2} H \Delta\theta = 0$$

# Steps to Solve

Use Lagrangian to solve the constrained optimization

$$L = g^T \Delta\theta + \lambda \left( \frac{1}{2} (\theta - \theta_{old})^T H (\theta - \theta_{old}) - \delta \right)$$

$$\nabla_{\theta} L = 0 \implies g^t + \frac{\lambda}{2} H \Delta\theta = 0$$

$$\Delta\theta = -\frac{2}{\lambda} H^{-1} g$$

# Steps to Solve

Use Lagrangian to solve the constrained optimization

$$L = g^T \Delta\theta + \lambda \left( \frac{1}{2} (\theta - \theta_{old})^T H (\theta - \theta_{old}) - \delta \right)$$

$$\nabla_{\theta} L = 0 \implies g^t + \frac{\lambda}{2} H \Delta\theta = 0$$

$$\Delta\theta = -\frac{2}{\lambda} H^{-1} g$$

This looks like SGD, set  $\alpha = -\frac{2}{\lambda}$

# Steps to Solve

Use Lagrangian to solve the constrained optimization

$$L = g^T \Delta\theta + \lambda \left( \frac{1}{2} (\theta - \theta_{old})^T H (\theta - \theta_{old}) - \delta \right)$$

$$\nabla_{\theta} L = 0 \implies g^t + \frac{\lambda}{2} H \Delta\theta = 0$$

$$\Delta\theta = -\frac{2}{\lambda} H^{-1} g$$

This looks like SGD, set  $\alpha = -\frac{2}{\lambda}$

$$\Delta\theta = \alpha H^{-1} g$$

# Steps to Solve

We have  $\Delta\theta = \alpha H^{-1}g$ , and we solve for  $\alpha$  by setting constraint  $= \delta$

# Steps to Solve

We have  $\Delta\theta = \alpha H^{-1}g$ , and we solve for  $\alpha$  by setting constraint  $= \delta$

$$\frac{1}{2}\Delta\theta^T H \Delta\theta = \delta$$

# Steps to Solve

We have  $\Delta\theta = \alpha H^{-1}g$ , and we solve for  $\alpha$  by setting constraint  $= \delta$

$$\frac{1}{2}\Delta\theta^T H \Delta\theta = \delta$$

$$\frac{1}{2}\alpha H^{-1}g^T H \alpha H^{-1}g = \delta$$

# Steps to Solve

We have  $\Delta\theta = \alpha H^{-1}g$ , and we solve for  $\alpha$  by setting constraint  $= \delta$

$$\frac{1}{2}\Delta\theta^T H \Delta\theta = \delta$$

$$\frac{1}{2}\alpha H^{-1}g^T H \alpha H^{-1}g = \delta$$

Now we simply re-arrange



# Steps to Solve

We have  $\Delta\theta = \alpha H^{-1}g$ , and we solve for  $\alpha$  by setting constraint  $= \delta$

$$\frac{1}{2}\Delta\theta^T H \Delta\theta = \delta$$

$$\frac{1}{2}\alpha H^{-1}g^T H \alpha H^{-1}g = \delta$$

Now we simply re-arrange

$$\alpha^2 g^t H g = 2\delta$$

# Steps to Solve

We have  $\Delta\theta = \alpha H^{-1}g$ , and we solve for  $\alpha$  by setting constraint  $= \delta$

$$\frac{1}{2}\Delta\theta^T H \Delta\theta = \delta$$

$$\frac{1}{2}\alpha H^{-1}g^T H \alpha H^{-1}g = \delta$$

Now we simply re-arrange

$$\alpha^2 g^t H g = 2\delta$$

$$\alpha = \sqrt{\frac{2\delta}{g^t H g}}$$

# Steps to Solve

We have  $\Delta\theta = \alpha H^{-1}g$ , and we solve for  $\alpha$  by setting constraint  $= \delta$

$$\frac{1}{2}\Delta\theta^T H \Delta\theta = \delta$$

$$\frac{1}{2}\alpha H^{-1}g^T H \alpha H^{-1}g = \delta$$

Now we simply re-arrange

$$\alpha^2 g^t H g = 2\delta$$

$$\alpha = \sqrt{\frac{2\delta}{g^t H g}}$$

Subbing this into  $\Delta\theta = \theta - \theta_{old} = \alpha H^{-1}g$  we get

# Steps to Solve

We have  $\Delta\theta = \alpha H^{-1}g$ , and we solve for  $\alpha$  by setting constraint  $= \delta$

$$\frac{1}{2}\Delta\theta^T H \Delta\theta = \delta$$
$$\frac{1}{2}\alpha H^{-1}g^T H \alpha H^{-1}g = \delta$$

Now we simply re-arrange

$$\alpha^2 g^t H g = 2\delta$$
$$\alpha = \sqrt{\frac{2\delta}{g^t H g}}$$

Subbing this into  $\Delta\theta = \theta - \theta_{old} = \alpha H^{-1}g$  we get

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

$$H^{-1}g$$

# An Introduction to the Conjugate Gradient Method Without the Agonizing Pain

Edition 1  $\frac{1}{4}$

Jonathan Richard Shewchuk

August 4, 1994

# Line Search

Our approximations have gone back to bite us. We need to check that our updates don't exceed our *actual* constraint using a line search.

# Trust Region Policy Optimization

---

## Algorithm 3 Trust Region Policy Optimization

---

Input: initial policy parameters  $\theta_0$

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

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

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

Form sample estimates for

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

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

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

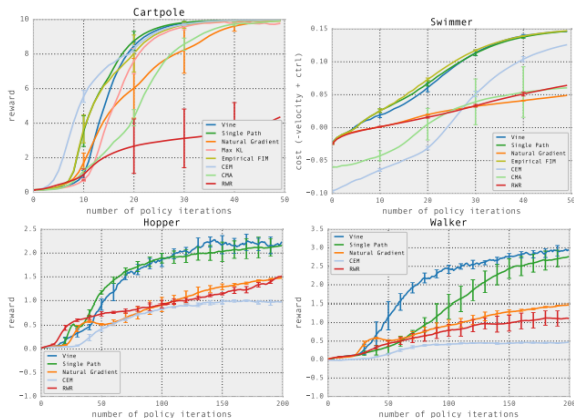
Perform backtracking line search with exponential decay to obtain final update

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

**end for**

---

# Experimental Results in TRPO Paper



*Figure 4. Learning curves for locomotion tasks, averaged across five runs of each algorithm with random initializations. Note that for the hopper and walker, a score of  $-1$  is achievable without any forward velocity, indicating a policy that simply learned balanced standing, but not walking.*



# External TRPO Robotics Applications

Mahmood et al. [2018] wrote a paper benchmarking policy gradient algorithms for robotics, including TRPO.

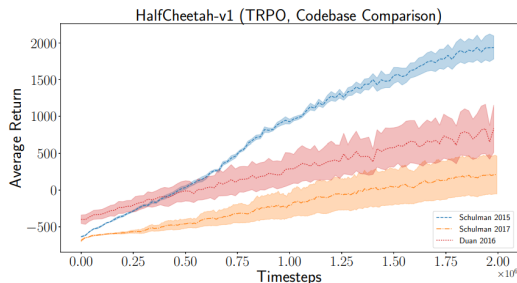
- "TRPO achieving near-best final learning performance in all tasks."
- "Among these algorithms, the final performance of TRPO was never substantially worse compared to the best in each task."
- "TRPO's performance was the least sensitive to hyper-parameter variations with the smallest interquartile range on both tasks."

# Next Steps

- The Chosen Path
  - Clipping
  - Hyperparameter tuning
- Other Approaches:
  - More precise approximations → no more line search
  - Assumptions about policy structure → tighter bound
  - Assumptions about function approximator → better notation of small step

# Critiques of the Paper

- They leave out critical implementation details [Henderson et al., 2019].
- They compare 'Vine' and 'Path', and vine is a different problem formulation
- TRPO-Blog for Details



# Thank you for listening

Robots following TRPO Policies

# References

- Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters, 2019. URL <https://arxiv.org/abs/1709.06560>.
- A. Rupam Mahmood, Dmytro Korenkevych, Gautham Vasan, William Ma, and James Bergstra. Benchmarking reinforcement learning algorithms on real-world robots, 2018. URL <https://arxiv.org/abs/1809.07731>.
- John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2017. URL <https://arxiv.org/abs/1502.05477>.
- Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249.