Trust Region Policy Optimization

Yixin Lin

Duke University

yixin.lin@duke.edu

March 28, 2017

Overview

Introduction

- Reinforcement learning is the problem of *sequential decision making in dynamic environment*
- Goal: capture the most important aspects of an agent making decisions
  - Input (sensing the state of the environment)
  - Action (choosing to affect on the environment)
  - Goal (prefers some states of the environment over others)
- This is *incredibly* general
- Examples
  - Robots (and their components)
  - Games
  - Better A/B testing

The Markov Decision Process (MDP)

- S: set of possible **states** of the environment
  - $p(s_{init}), s_{init} \in S$: a distribution over initial state
  - Markov property: we assume that the current state summarizes everything we need to remember
- A: set of possible **actions**
  - $P(s_{new}|s_{old}, a)$: state transitions, for each state s and action a
- $R : S \to \mathbb{R}$: **reward**
  - $\gamma \in [0, 1]$: discount factor

Policies and value functions

- π: a policy (what action to do, given a state)
- Return: (possibly discounted) sum of future rewards
  $r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} \dots$
- Performance of policy: $\eta(\pi) = \mathbb{E}[\text{return}]$
- $V^\pi(s) = E_\pi[\text{return} \mid s]$
  - How good is a state, given a policy?
- $Q^\pi(s, a) = E_\pi[\text{return} \mid s, a]$
  - How good is an action at a state, given a policy?

Policy iteration

- Assume perfect model of MDP
- Alternate between the following until convergence
  - Evaluating the policy (computing V^π)
    - For each state s, $V(s) = \mathbb{E}[\sum_{s',r} r + \gamma V(s')]$
    - Repeat until convergence (or just once for *value iteration*)
  - Setting policy to be greedy ($\pi(s) = \arg\max_a \mathbb{E}[r + \gamma V^\pi(s')]$)
- Guaranteed convergence (for both policy and value iteration)

Policy gradients

- Policy iteration scales very badly: have to repeatedly evaluate policy on all states
- Parameterize policy $a \sim \pi|\theta$
- We can sample instead

Policy gradients

- Sample a lot of *trajectories* (simulate your environment under the policy) under the current policy
- Make good actions more probable
  - Specifically, estimate gradient using *score function gradient estimator*
  - For each trajectory τ_i, $\hat{g}_i = R(\tau_i)\nabla_\theta \log p(\tau_i|\theta)$
  - Intuitively, take the gradient of log probability of the trajectory, then weight it by the final reward
  - Reduce variance by temporal structure and other tricks (e.g. baseline)
    - Replace reward by the advantage function $A_\pi = Q_\pi(s,a) - V_\pi(s)$
    - Intuitively, how much better is the action we picked over the average action?
- Repeat

Vanilla policy gradient algorithm/REINFORCE

Initialize policy $\pi|\theta$
while gradient estimate has not converged **do**
    Sample trajectories using π
    for each timestep **do**
        Compute return and advantage estimate
    end for
    Refit optimal baseline
    Update the policy using gradient estimate $\hat{g}$
end while

Connection to supervised learning

- Minimizing $L(\theta) = \sum_t \log \pi(a_t|s_t;\theta)\hat{A}_t$
  - In the paper, they use cost functions instead of reward functions
- Intuitively, we have some parameterized policy ("model") giving us a distribution over actions
- We don't have the correct action ("label"), so we just use the reward at the end as our label
- We can do better. How do we do credit assignment?
  - Baseline (roughly encourage half of the actions, not just all of them)
  - Discounted future reward (actions affect near-term future), etc.

Kakade & Langford: conservative policy iteration

- "A useful identity" for $\eta_{\tilde{\pi}}$, the expected discounted cost of a new policy $\tilde{\pi}$
- $\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}[\sum_{t=0} \gamma^t A_\pi(s_t, a_t)] =$
  $\eta(\pi) + \sum_s \rho_{\tilde{\pi}}(s) \sum_a \tilde{\pi}(a|s) A_\pi(s, a)$
- Intuitively: the expected return of a new policy is the expected return of the old policy, plus how much better the new policy is at each state
- Local approximation: switch out $\rho_{\tilde{\pi}}$ for ρ_π, since we only have the state visitation frequency for the old policy, not the new policy
  - $L_\pi(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_\pi(s) \sum_a \tilde{\pi}(a|s) A_\pi(s, a)$
- Kakade & Langford proved that optimizing this local approximation is good for small step sizes, but for mixture policies only

Natural policy gradients

- In this paper, they prove that $\eta(\tilde{\pi}) \leq L_{\pi}(\tilde{\pi}) + CD_{KL}^{\max}(\pi, \tilde{\pi})$, C is a constant dependent on γ
- Intuitively, we optimize the approximation, but regularize with the KL divergence between old and new policy
- Algorithm called the natural policy gradient
- Problem: choosing hyperparameter C is difficult

Overview of TRPO

- Instead of adding KL divergence as a cost, simply use it as an optimization constraint
- TRPO algorithm: minimize $L_\pi(\tilde{\pi})$, constraint that $D_{KL}^{\max} \leq \delta$ for some easily-picked hyperparameter δ

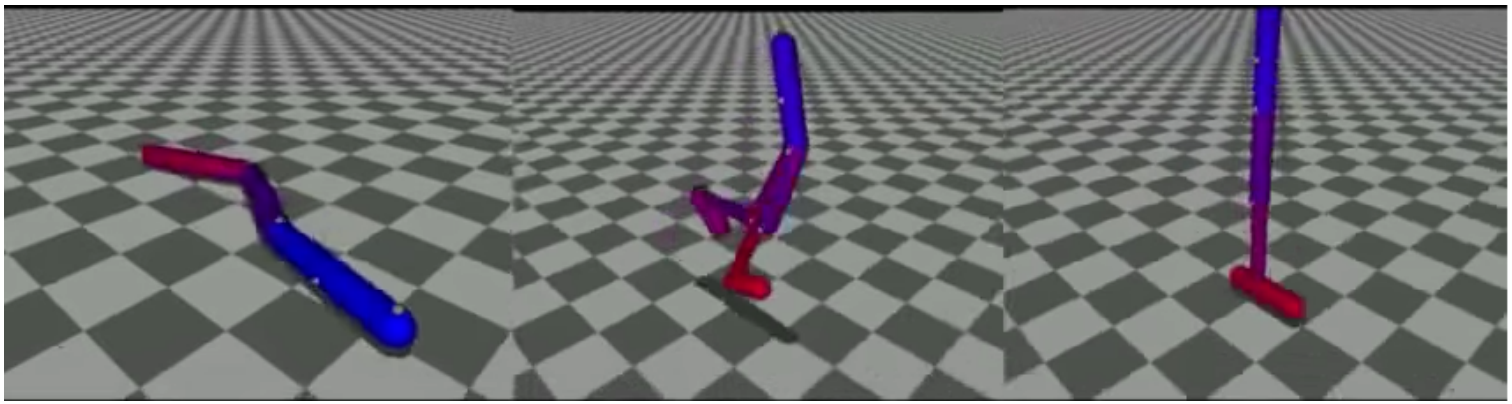
Practical considerations

- How do we sample trajectories?
  - Single-path: simply run each sample to completion
  - "Vine": for each sampled trajectory, pick random states along the trajectory and perform small rollout
- How do we compute gradient? Use conjugate gradient algorithm followed by line search

Algorithm

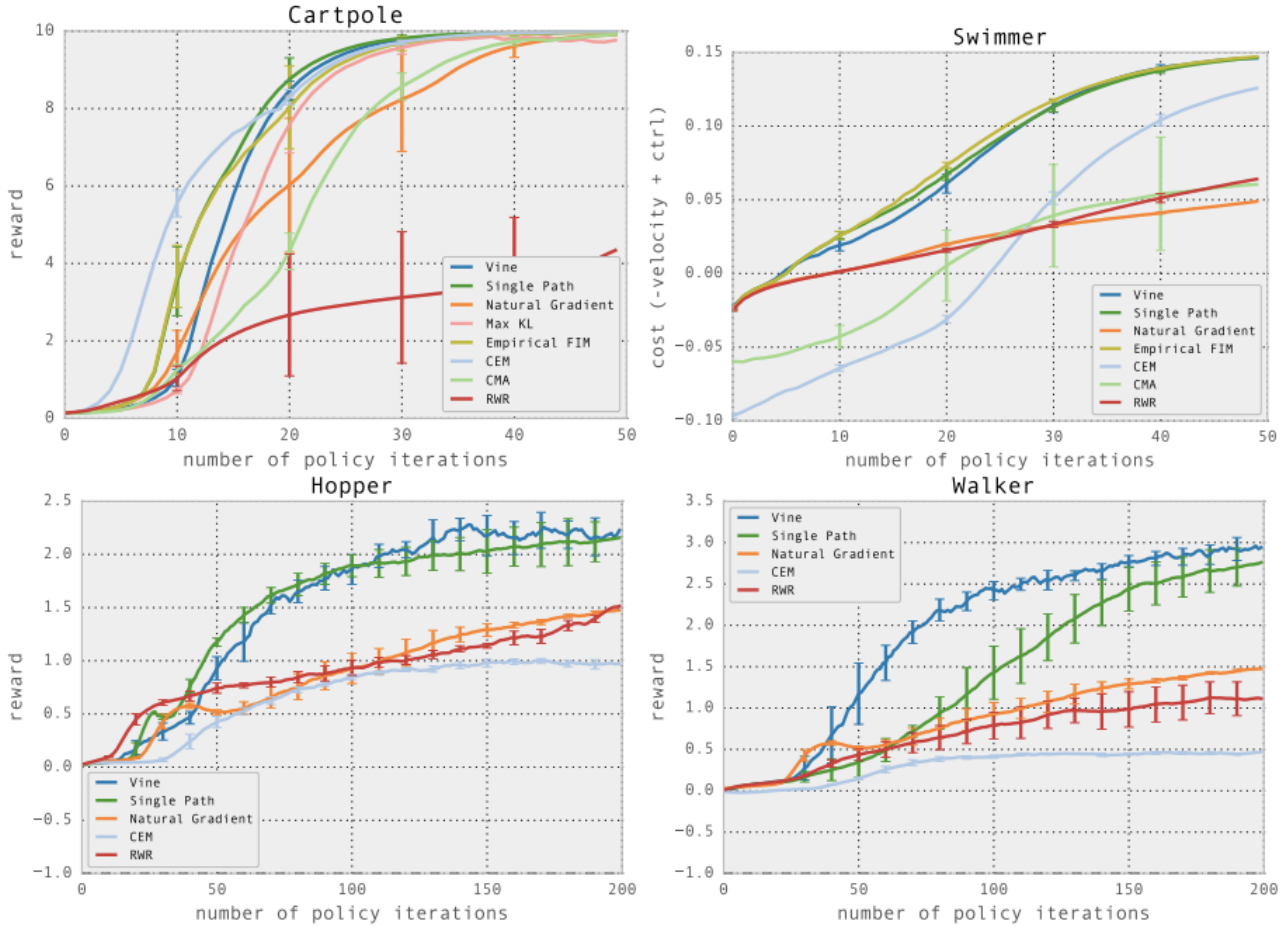
while gradient not converged **do**
    Collect trajectories (either single-path or vine)
    Estimate advantage function
    Compute policy gradient estimator
    Solve quadratic approximation to $L(\pi_\theta)$ using CG
    Rescale using line search
    Apply update
end while

Experiments - MuJoCo robotic locomotion



- Link to demonstration
- Same δ hyperparameter across experiments

Experiments - MuJoCo learning curves



- Link to demonstration
- Same δ hyperparameter across experiments

Experiments - Atari

Trust Region Policy Optimization

	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

- Not always better than previous techniques, but consistently decent
- Very little problem-specific engineering

Takeaway

- TRPO is a good default policy gradient technique which scales well and has minimal hyperparameter tuning
- Just use KL constraint on gradient approximator

References


RS Sutton. Introduction to reinforcement learning


Kakade and Langford. Approximately optimal approximate reinforcement learning


Schulman et al. Trust Region Policy Optimization


Schulman, Levine, Finn. Deep Reinforcement Learning course: link


Andrej Karpathy. Deep Reinforcement Learning: From Pong to Pixels: link


Trust Region Policy Optimization summary: link

Thanks!