Trust Region Policy Optimization

Yixin Lin
Duke University
yixin.lin@duke.edu

March 28, 2017
Overview

1 Preliminaries
   • Markov Decision Processes
   • Policy iteration
   • Policy gradients

2 TRPO
   • Kakade & Langford
   • Natural policy gradients
   • Overview of TRPO
   • Practice
   • Experiments

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

- $\pi$: a policy (what action to do, given a state)
- Return: (possibly discounted) sum of future rewards
  \[ r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} \ldots \]
- 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 $V^\pi$)
    - For each state $s$, $V(s) = \mathbb{E} \left[ \sum_{s', r} r + \gamma V(s') \right]$
    - 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 iteration scales very badly: have to repeatedly evaluate policy on all states
Parameterize policy $a \sim \pi|\theta$
We can sample instead
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 $\tau_i$, $\hat{g}_i = R(\tau_i) \nabla_\theta \log p(\tau_i|\theta)$
- 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 $\pi$
    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.
“A useful identity” for $\eta_{\tilde{\pi}}$, the expected discounted cost of a new policy $\tilde{\pi}$

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t)] = \eta(\pi) + \sum_s \rho_{\tilde{\pi}}(s) \sum_a \tilde{\pi}(a|s) A_\pi(s, a)$$

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 $\rho_\pi$, 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
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 $\gamma$

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 $\delta$. 
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
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 $\delta$ hyperparameter across experiments
Experiments - MuJoCo learning curves

- Link to demonstration
- Same $\delta$ hyperparameter across experiments
Experiments - Atari

Not always better than previous techniques, but consistently decent

Very little problem-specific engineering

<table>
<thead>
<tr>
<th></th>
<th>B. Rider</th>
<th>Breakout</th>
<th>Enduro</th>
<th>Pong</th>
<th>Q'bert</th>
<th>Seaquest</th>
<th>S. Invaders</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>354</td>
<td>1.2</td>
<td>0</td>
<td>-20.4</td>
<td>157</td>
<td>110</td>
<td>179</td>
</tr>
<tr>
<td>Human (Mnih et al., 2013)</td>
<td>7456</td>
<td>31.0</td>
<td>368</td>
<td>-3.0</td>
<td>18900</td>
<td>28100</td>
<td>3690</td>
</tr>
<tr>
<td>Deep Q Learning (Mnih et al., 2013)</td>
<td>4092</td>
<td>168.0</td>
<td>470</td>
<td>20.0</td>
<td>1952</td>
<td>1705</td>
<td>581</td>
</tr>
<tr>
<td>UCC-I (Guo et al., 2014)</td>
<td>5702</td>
<td>380</td>
<td>741</td>
<td>21</td>
<td>20025</td>
<td>2995</td>
<td>692</td>
</tr>
<tr>
<td>TRPO - single path</td>
<td>1425.2</td>
<td>10.8</td>
<td>534.6</td>
<td>20.9</td>
<td>1973.5</td>
<td>1908.6</td>
<td>568.4</td>
</tr>
<tr>
<td>TRPO - vine</td>
<td>859.5</td>
<td>34.2</td>
<td>430.8</td>
<td>20.9</td>
<td>7732.5</td>
<td>788.4</td>
<td>450.2</td>
</tr>
</tbody>
</table>
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!

Link to presentation: yixinlin.net/trpo