Trust Region Policy Optimization

June 11, 2016

TRPO is a gradient-based algorithm for policy optimization. It does not necessarily follow known patterns of reinforcement learning (i.e. policy iteration, policy gradient or actor-critic), but I hear it is wonderfully easy to use, and gives you great results.

1 Kakade & Langford's Policy Improvement

TRPO builds upon Kakade & Langford's result on policy improvement. This work starts on the following identity, which expresses the expected discount cost (or reward) η of a modified policy $\tilde{\pi}$ using expected costs of the original policy π.

$$\eta\left(\tilde{\pi}\right)=\eta\left(\pi\right)+\mathbb{E}_{s_0,a_0,s_1,a_1,\cdots}\left[\sum_{t=0}^{\infty}\gamma^t A_\pi\left(s_t,a_t\right)\right]$$

where $A_\pi\left(s_t,a_t\right)$ is the usual advantage function, defined as $Q_\pi\left(s_t,a_t\right)-V\left(s_t,a_t\right)$. It should be noted that $a_t\sim\tilde{\pi}\left(a_t|s_t\right)$, which means the trajectory follows $\tilde{\pi}$, not π. I recognize this identity from when we proved the correctness (or convergence?) of policy iteration: if we can keep all advantage functions nonpositive, we can guarantee improvement.

There's a problem: with function approximators, you cannot guarantee $\sum_a\tilde{\pi}\left(a|s\right)A_\pi\left(s,a\right)$ will be nonpositive for all states. We overcome this by forming a local approximation:

$$L_\pi\left(\tilde{\pi}\right)=\eta\left(\theta\right)+\sum_s\rho_\pi\left(s\right)\sum_a\tilde{\pi}\left(a|s\right)A_\pi\left(s,a\right)$$

The difference here is that the state distribution function ρ uses π - which means we follow π, and only use $\tilde{\pi}$ for weighting advantage function values. Now, suppose our policies are parameterized by a parameter vector θ. Then $L_{\pi_{\theta_0}}\left(\pi_\theta\right)$ is a good linear approximation of $\eta\left(\pi_\theta\right)$ when θ is close to θ_0 - the value matches (obviously $L_{\pi_{\theta_0}}\left(\pi_{\theta_0}\right)=\eta\left(\pi_{\theta_0}\right)$), and the gradients match as well.

Kakade & Langford now creates a new policy π_{new} that is close to π_{old} which minimizes $L_{\pi_{old}}\left(\pi_{new}\right)$ which will hopefully minimize $\eta\left(\pi_{new}\right)$. Unfortunately, they only had guarantees for π_{new} which is in the form of a mixed policy:

$$\pi_{new}=\left(1-\alpha\right)\pi_{old}\left(a|s\right)+\alpha\pi'\left(a|s\right)$$

This is of course not desirable as policies are bloated over time and the changes we can make are pretty small (?).

2 TRPO

TRPO realizes that K&L's guarantee works when we plug in $D_{TV}^{\max}(\pi_{old}, \pi_{new})$ as α. D_{TV} is a distance measure between probability distributions, which is upper bounded by KL divergence. The paper proves:

$$\eta(\tilde{\pi}) \leq L_{\pi}(\tilde{\pi}) + C \cdot D_{KL}^{\max}(\pi, \tilde{\pi})$$

for "some" C (they have the formula for that, I don't find it important..). $D_{KL}^{\max}$ denotes the maximum KL divergence of $\pi(\cdot|s)$ and $\tilde{\pi}(\cdot|s)$ among all states. Now, we can employ some majorization-minimization argument (reminds me of FISTA?) to prove that minimizing the RHS of above inequality improves η as well!

The actual TRPO algorithm is an approximation of minimizing the above quantity. The problems they were trying to solve was the step sizes were too small, and $D_{KL}^{\max}$ is not always well defined. There are two approximations they used to deal with these:

1. Simply minimize $L_{\pi}(\tilde{\pi})$ with a constraint that $D_{KL}^{\max}$ is lower than a certain threshold.
2. Use average KL divergence (weight each state by the distribution function ρ).

The second approximation allows us to estimate KL divergence from the samples we actually get! Tada!

Here is the outline of the TRPO algorithm.

- We sample (from a simulator or the real environment) one or more rollout sets using π. Now, pick some state-action pairs, and the rewards observed, and use constrained optimization to find the next generation policy that minimizes $L_{\pi}(\tilde{\pi})$ with the KL divergence constraint.
- When calculating $L_{\pi}(\tilde{\pi})$, use importance sampling (this comes up quite often!) to adjust differences between π and $\tilde{\pi}$. (Actually, we don't even have to rollout using π - we just need its advantage functions. We can take a different sampling distribution $q(a|s)$.. for discrete action spaces they say uniform q works better than taking π.)
- Replace advantage functions with action values (this only shifts the objective function by the value function, which ends up being a constant after expectations are taken of).

With these changes, at each iteration, we solve:

$$\begin{array}{ll} \text{minimize}_{\theta} & \mathbb{E}_{s\sim\rho_{\theta_{old}}, a\sim q}\left[\frac{\pi_{\theta}(a|s)}{q(a|s)} Q_{\theta_{old}}(s,a)\right] \\ \text{subject to} & \mathbb{E}_{s\sim\rho_{\theta_{old}}}\left[D_{KL}\left(\pi_{\theta_{old}}(\cdot|s), \pi_{\theta}(\cdot|s)\right)\right] \leq \delta \end{array}$$

Now we can estimate all of these from samples (Q, take s from samples gained using $\pi_{\theta_{old}}$, etc).

2.1 Sampling Strategies

The paper introduces two strategies: single-path (simulate a single trajectory), vine (simulate multiple trajectories, sample random states along those trajectories, sample a number of short rollouts from those to estimate rewards). Of course, vine works better usually.

One interesting tidbit mentioned: using a common random number source across different short rollouts in vine reduces variance. I am not entirely sure what that means...

2.2 Solving the Optimization Problem

They recommend using conjugate gradient algorithm followed by a line search. They talk a little about calculating Hessian analytically, which I guess makes sense, except that I don't know how to do that with general policies... I should look into actual implementations..

I looked at performance graphs. They are good, obviously, and holy crap, CEM is so good for a gradient free method.