Advanced Policy Gradient Methods: Natural Gradient, TRPO, and More

March 8, 2017

Defining a Loss Function for RL

- Let $\eta(\pi)$ denote the expected return of π

$$\eta(\pi) = \mathbb{E}_{s_0 \sim \rho_0, a_t \sim \pi(\cdot \mid s_t)} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]$$

- We collect data with π_{old}. Want to optimize some objective to get a new policy π
- A useful identity[1]:

$$\eta(\pi) = \eta(\pi_{\text{old}}) + \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t A^{\pi_{\text{old}}}(s_t, a_t) \right]$$

[1] S. Kakade and J. Langford. "Approximately optimal approximate reinforcement learning". *ICML*. 2002.

Proof of Useful Identity

First note that $A^{\pi_{\text{old}}}(s, a) = \mathbb{E}_{s' \sim P(s' \mid s, a)} \left[r(s) + \gamma V^{\pi_{\text{old}}}(s') - V^{\pi_{\text{old}}}(s) \right]$.

$$
\begin{aligned}
&\mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t A_{\pi_{\text{old}}}(s_t, a_t) \right] \\
&= \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t \left(r(s_t) + \gamma V^{\pi_{\text{old}}}(s_{t+1}) - V^{\pi_{\text{old}}}(s_t) \right) \right] \\
&= \mathbb{E}_{\tau \sim \pi} \left[-V^{\pi_{\text{old}}}(s_0) + \sum_{t=0}^{\infty} \gamma^t r(s_t) \right] \\
&= -\mathbb{E}_{s_0} \left[V^{\pi_{\text{old}}}(s_0) \right] + \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t) \right] \\
&= -\eta(\pi_{\text{old}}) + \eta(\pi)
\end{aligned}
$$

Surrogate Loss Function

- Want to manipulate $\eta(\pi)$ into an objective that we can estimate from sampled data

$$\begin{aligned}\eta(\pi) &= const + \mathbb{E}_{s\sim\pi, a\sim\pi}\left[A^{\pi_{\text{old}}}(s,a)\right] \\ &= const + \mathbb{E}_{s\sim\pi, a\sim\pi_{\text{old}}}\left[\frac{\pi(a\mid s)}{\pi_{\text{old}}(a\mid s)}A^{\pi_{\text{old}}}(s,a)\right]\end{aligned}$$

- Define $L_{\pi_{\text{old}}}(\pi)$ to be the "surrogate objective" that ignores change in state distrib:

$$\begin{aligned}L(\pi) &= \mathbb{E}_{s\sim\pi_{\text{old}}, a\sim\pi}\left[A^{\pi_{\text{old}}}(s_t,a_t)\right] \\ &= \mathbb{E}_{s\sim\pi_{\text{old}}, a\sim\pi_{\text{old}}}\left[\frac{\pi(a\mid s)}{\pi_{\text{old}}(a\mid s)}A^{\pi_{\text{old}}}(s,a)\right]\end{aligned}$$

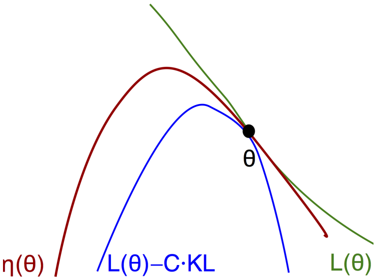
Matches to first order for parameterized policy

$$\begin{aligned}\nabla_\theta L(\pi_\theta)\big|_{\theta_{\text{old}}} &= \mathbb{E}_{s,a\sim\pi_{\text{old}}}\left[\frac{\nabla_\theta\pi_\theta(a\mid s)}{\pi_{\text{old}}(a\mid s)}A^{\pi_{\text{old}}}(s,a)\right]\Big|_{\theta_{\text{old}}} \\ &= \mathbb{E}_{s,a\sim\pi_{\text{old}}}\left[\nabla_\theta\log\pi_\theta(a\mid s)A^{\pi_{\text{old}}}(s,a)\right]\Big|_{\theta_{\text{old}}} = \nabla_\theta\eta(\pi_\theta)\big|_{\theta=\theta_{\text{old}}}\end{aligned}$$

- Local approximation to the performance of the policy

Improvement Theory

- Theory: bound the difference between $L_{\pi_{\mathrm{old}}}(\pi)$ and $\eta(\pi)$, the performance of the policy (error due because we're ignoring state distrib. change)
- Result: $\eta(\pi) \geq L_{\pi_{\mathrm{old}}}(\pi) - C \cdot \max_s \mathrm{KL}[\pi_{\mathrm{old}}(\cdot \mid s), \pi(\cdot \mid s)]$, where $c = 2\epsilon\gamma/(1-\gamma)^2$
- Monotonic improvement guaranteed (MM algorithm)



Practical Approximations

Theory: should maximize $L_{\pi_{\text{old}}}(\pi) - C \cdot \max_s \mathrm{KL}[\pi_{\text{old}}(\cdot \mid s), \pi(\cdot \mid s)]$.
Approximations:

- Estimate $L_{\pi_{\text{old}}}(\pi)$ using trajectories sampled from π_{old}
  - $\hat{L}_{\pi_{\text{old}}}(\pi) = \sum_n \frac{\pi(a_n \mid s_n)}{\pi_{\text{old}}(a_n \mid s_n)} \hat{A}_n$
  - If just gradient needed, can use $\hat{L}_{\pi_{\text{old}}}(\pi) = \sum_n \log \pi(s_n \mid s_n) \hat{A}_n$
- Use mean KL divergence $\mathbb{E}_{s \sim \pi_{\text{old}}}[\mathrm{KL}[\pi_{\text{old}}(\cdot \mid s), \pi(\cdot \mid s)]]$ instead of $\max_s \mathrm{KL}[\ldots]$
  - Define $\overline{\mathrm{KL}}_{\pi_{\text{old}}}(\pi) = \mathbb{E}_{s \sim \pi_{\text{old}}}[\mathrm{KL}[\pi_{\text{old}}(\cdot \mid s), \pi(\cdot \mid s)]]$
  - Use estimate from samples, $\sum_n \mathrm{KL}[\pi_{\text{old}}(\cdot \mid s_n), \pi(\cdot \mid s_n)]$
- C is too pessimistic and provides guarantee for discounted return
  - Natural policy gradient and PPO: use fixed or adaptive coefficient C
  - TRPO: use hard constraint with fixed KL penalty

Solving KL-Penalized Problem

- maximize$_\theta$ $L_{\pi_{\theta_{\text{old}}}}(\pi_\theta) - C \cdot \overline{\mathrm{KL}}_{\pi_{\theta_{\text{old}}}}(\pi_\theta)$
- Make linear approximation to $L_{\pi_{\theta_{\text{old}}}}$ and quadratic approximation to KL term:

$$\underset{\theta}{\text{maximize}} \quad g \cdot (\theta - \theta_{\text{old}}) - \tfrac{C}{2}(\theta - \theta_{\text{old}})^T F(\theta - \theta_{\text{old}})$$

$$\text{where} \quad g = \frac{\partial}{\partial \theta} L_{\pi_{\theta_{\text{old}}}}(\pi_\theta)\Big|_{\theta=\theta_{\text{old}}}, \quad F = \frac{\partial^2}{\partial^2 \theta} \overline{\mathrm{KL}}_{\pi_{\theta_{\text{old}}}}(\pi_\theta)\Big|_{\theta=\theta_{\text{old}}}$$

  - Quadratic part of L is negligible compared to KL term
  - F is positive semidefinite, but not if we include Hessian of L
- Solution: $\theta - \theta_{\text{old}} = \frac{1}{C} F^{-1} g$

Solving Linear Systems using Conjugate Gradient

- Previous slide: $\theta - \theta_{\text{old}} = \frac{1}{c} F^{-1} g$. Don't want to form full Hessian matrix $F = \frac{\partial^2}{\partial^2 \theta} \overline{\mathrm{KL}}_{\pi_{\theta_{\text{old}}}}(\pi_\theta)\big|_{\theta=\theta_{\text{old}}}$ (memory and computation)
- Can compute $F^{-1} g$ approximately using conjugate gradient algorithm without forming F explicitly

Truncated Newton Method

- Conjugate gradient algorithm approximately solves for $x = A^{-1}b$, without explicitly forming matrix A, just reads A through matrix-vector products $v \to Av$.
  - After k iterations, CG has minimized $\frac{1}{2}x^T Ax - bx$ in subspace spanned by $b, Ab, A^2b, \ldots, A^{k-1}b$
- Given vector v with same dimension as θ, want to compute $H^{-1}v$, where $H = \frac{\partial^2}{\partial^2\theta}f(\theta)$.
- To perform CG, Hessian-vector products $v \to Hv$. Can form this function using autodiff software like Tensorflow. Example:

```
theta = tf.placeholder(...) # parameter vector
f_of_theta = ... # scalar
vector = tf.placeholder([dim_theta])
gradient = tf.grad(f, theta)
gradient_vector_product = tf.sum( gradient * vector )
hessian_vector_product = tf.grad(gradient_vector_product, theta)
```

- Hessian vector product computation takes 1-2 times as long as gradient computation

S. J. Wright and J. Nocedal. *Numerical optimization*. Springer New York, 1999

Truncated Newton Method

- Hessian-vector can be formed as follows if KL Hessian (F) is computed using KL

```
theta = tf.placeholder(...) # parameter vector
actprob = ... # batchsize x n_actions.
              # E.g., outputs of neural network with softmax
oldactprob = tf.stop_gradient(actprob)
vector = tf.placeholder([dim_theta])
kl = tf.sum( oldactprob * tf.log(oldactprob / actprob), axis=1)
gradient = tf.grad(kl, theta)
gradient_vector_product = tf.sum( gradient * vector )
hessian_vector_product = tf.grad(gradient_vector_product, theta)
```

S. J. Wright and J. Nocedal. *Numerical optimization.* Springer New York, 1999

Solving KL-Penalized Problem: Summary

- maximize$_\theta$ $L_{\pi_{\theta_{\text{old}}}}(\pi_\theta) - C \cdot \overline{\text{KL}}_{\pi_{\theta_{\text{old}}}}(\pi_\theta)$
- Make linear approximation to $L_{\pi_{\theta_{\text{old}}}}$ and quadratic approximation to KL term:

$$\underset{\theta}{\text{maximize}} \quad g \cdot (\theta - \theta_{\text{old}}) - \tfrac{C}{2}(\theta - \theta_{\text{old}})^T F(\theta - \theta_{\text{old}})$$

$$\text{where} \quad g = \frac{\partial}{\partial \theta} L_{\pi_{\theta_{\text{old}}}}(\pi_\theta)\big|_{\theta=\theta_{\text{old}}}, \quad F = \frac{\partial^2}{\partial^2 \theta} \overline{\text{KL}}_{\pi_{\theta_{\text{old}}}}(\pi_\theta)\big|_{\theta=\theta_{\text{old}}}$$

- Solution: $\theta - \theta_{\text{old}} = \frac{1}{C} F^{-1} g$
- Solve for $F^{-1}g$ approximately using conjugate gradient algorithm by forming Hessian-vector product function

Truncated Natural Policy Gradient Algorithm

for iteration$=1, 2, \ldots$ **do**

Run policy for T timesteps or N trajectories

Estimate advantage function at all timesteps

Compute policy gradient g

Use CG (with Hessian-vector products) to compute $F^{-1}g$

Update policy parameter $\theta = \theta_{\text{old}} + \alpha F^{-1}g$

end for

TRPO: KL-Constrained Problem

- Unconstrained problem: maximize $L_{\pi_{\theta_{\text{old}}}}(\pi_\theta) - C \cdot \overline{\text{KL}}_{\pi_{\theta_{\text{old}}}}(\pi_\theta)$
- Constrained problem: maximize $L_{\pi_{\theta_{\text{old}}}}(\pi_\theta)$ subject to $\overline{\text{KL}}_{\pi_{\theta_{\text{old}}}}(\pi_\theta) \le \delta$
- Often easier to set hyperparameter δ rather than C, can remain fixed over whole learning process
- We'll solve constrained quadratic problem: compute $F^{-1}g$, and then rescale step to get correct KL
  - Take linear and quadratic constraint:
    maximize$_\theta$ $g \cdot (\theta - \theta_{\text{old}})$ subject to $\frac{1}{2}(\theta - \theta_{\text{old}})^T F(\theta - \theta_{\text{old}}) \le \delta$
  - Form Lagrangian $\mathcal{L}(\theta, \lambda) = g \cdot (\theta - \theta_{\text{old}}) - \frac{\lambda}{2}[(\theta - \theta_{\text{old}})^T F(\theta - \theta_{\text{old}}) - \delta]$
  - Differentiate wrt θ, get $\theta - \theta_{\text{old}} = \frac{1}{\lambda}F^{-1}g$
  - To satisfy constraint, want $\frac{1}{2}s^T F s = \delta$.
  - Given candidate step s_{unscaled}, rescale to $s = \sqrt{\frac{2\delta}{s_{\text{unscaled}} \cdot (F s_{\text{unscaled}})}} s_{\text{unscaled}}$

TRPO: KL-Constrained Problem

- Compute $s_{\text{unscaled}} = F^{-1} g$
- Rescale: $s = \sqrt{\frac{2\delta}{s_{\text{unscaled}} \cdot (H s_{\text{unscaled}})}} s_{\text{unscaled}}$
- Now do backtracking line search on original problem (before quadratic constraint) maximize $L_{\pi_{\theta_{\text{old}}}}(\pi_\theta) - \mathbf{1}[\overline{\mathrm{KL}}_{\pi_{\theta_{\text{old}}}}(\pi_\theta) \leq \delta]$
  - Use steps $s, s/2, s/4, \ldots$ until line search objective improves

TRPO Algorithm

for iteration$=1, 2, \ldots$ **do**

Run policy for T timesteps or N trajectories

Estimate advantage function at all timesteps

Compute policy gradient g

Use CG (with Hessian-vector products) to compute $F^{-1}g$

Compute rescaled step $s = \alpha F^{-1}g$ with rescaling and line search

Apply update: $\theta = \theta_{\text{old}} + \alpha F^{-1}g$

end for

Alternative Method for Calculating Natural Gradients

- Given parameterized probability density $p_\theta(x)$
- Fisher information matrix

$$\frac{\partial}{\partial^2\theta} \mathrm{KL}[p_{\theta_{\mathrm{old}}}, p_\theta] = \mathbb{E}_{x \sim p_{\theta_{\mathrm{old}}}} \left[\left(\frac{\partial}{\partial\theta} \log p_\theta(x) \right)^T \left(\frac{\partial}{\partial\theta} \log p_\theta(x) \right) \right] \Big|_{\theta=\theta_{\mathrm{old}}}$$

- FIM forms a "metric" on policy's parameter space, induced by KL divergence. Makes step invariant to reparameterization of coordinates ($\theta' = f(\theta)$), whereas gradient is not invariant.
- In policy optimization setting, instead of forming Fisher by differentiating KL, can explictly form $\sum_n \left(\frac{\partial}{\partial\theta} \log \pi_\theta(a_n \mid s_n)\right)^T \left(\frac{\partial}{\partial\theta} \log \pi_\theta(a_n \mid s_n)\right)$

"Proximal" Policy Optimization

- Back to penalty instead of constraint

$$\underset{\theta}{\text{maximize}} \sum_{n=1}^{N} \frac{\pi_\theta(a_n \mid s_n)}{\pi_{\theta_{\text{old}}}(a_n \mid s_n)} \hat{A}_n - C \cdot \overline{\mathrm{KL}}_{\pi_{\theta_{\text{old}}}}(\pi_\theta)$$

- Pseudocode:

  for iteration$=1, 2, \ldots$ **do**
      Run policy for T timesteps or N trajectories
      Estimate advantage function at all timesteps
      Do SGD on above objective for some number of epochs
      If KL too high, increase β. If KL too low, decrease β.
  end for

- $\approx$ same performance as TRPO, but only first-order optimization

Approximations in Supervised vs Reinforcement Learning

- Supervised learning
  - Linear approximation given by gradient $f(\theta) \approx f(\theta_0) + (\theta - \theta_0) \cdot g$
  - Training loss approximates test loss
- Reinforcement learning (policy gradients)
  - Linear approximation given by gradient of surrogate $f(\theta) \approx f(\theta_0) + (\theta - \theta_0) \cdot g$
  - Training surrogate approximates test surrogate (sampled data is representative of visitation distribution)
  - State distribution doesn't change much

Further Reading

- S. Kakade. "A Natural Policy Gradient." *NIPS*. 2001
- S. Kakade and J. Langford. "Approximately optimal approximate reinforcement learning". *ICML*. 2002
- J. Peters and S. Schaal. "Natural actor-critic". *Neurocomputing* (2008)
- J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. "Trust Region Policy Optimization". *ICML* (2015)
- Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. "Benchmarking Deep Reinforcement Learning for Continuous Control". *ICML* (2016)
- J. Martens and I. Sutskever. "Training deep and recurrent networks with Hessian-free optimization". *Neural Networks: Tricks of the Trade*. Springer, 2012

That's all. Questions?