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

John Schulman

OpenAI

August 26, 2017

Two Limitations of "Vanilla" Policy Gradient Methods

- Hard to choose stepsizes
  - Input data is nonstationary due to changing policy: observation and reward distributions change
  - Bad step is more damaging than in supervised learning, since it affects visitation distribution
    - Step too far $\rightarrow$ bad policy
    - Next batch: collected under bad policy
    - Can't recover—collapse in performance
- Sample efficiency
  - Only one gradient step per environment sample
  - Dependent on scaling of coordinates

Reducing reinforcement learning to optimization

- Much of modern ML: reduce learning to numerical optimization problem
  - Supervised learning: minimize training error
- RL: how to *use all data so far and compute the best policy*?
  - Q-learning: can (in principle) include all transitions seen so far, however, we're optimizing the wrong objective
  - Policy gradient methods: yes stochastic gradients, but no optimization problem*
  - This lecture: write down an optimization problem that allows you to do a small update to policy π based on data sampled from π (*on-policy* data)

What Loss to Optimize?

- Policy gradients

$$\hat{g} = \hat{\mathbb{E}}_t \left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \hat{A}_t \right]$$

- Can differentiate the following loss

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t \left[\log \pi_\theta(a_t \mid s_t) \hat{A}_t \right].$$

  but don't want to optimize it too far

- Equivalently differentiate

$$L^{IS}_{\theta_{\mathrm{old}}}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\mathrm{old}}}(a_t \mid s_t)} \hat{A}_t \right].$$

  at $\theta = \theta_{\mathrm{old}}$, state-actions are sampled using θ_{old}. (IS = importance sampling)

  Just the chain rule: $\nabla_\theta \log f(\theta)\big|_{\theta_{\mathrm{old}}} = \frac{\nabla_\theta f(\theta)\big|_{\theta_{\mathrm{old}}}}{f(\theta_{\mathrm{old}})} = \nabla_\theta \left(\frac{f(\theta)}{f(\theta_{\mathrm{old}})} \right)\big|_{\theta_{\mathrm{old}}}$

Surrogate Loss: Importance Sampling Interpretation

- Importance sampling interpretation

$$
\begin{aligned}
&\mathbb{E}_{s_t \sim \pi_{\theta_{\text{old}}}, a_t \sim \pi_\theta} \left[A^\pi(s_t, a_t) \right] \\
&= \mathbb{E}_{s_t \sim \pi_{\theta_{\text{old}}}, a_t \sim \pi_{\theta_{\text{old}}}} \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} A^{\pi_{\theta_{\text{old}}}}(s_t, a_t) \right] \quad \text{(importance sampling)} \\
&= \mathbb{E}_{s_t \sim \pi_{\theta_{\text{old}}}, a_t \sim \pi_{\theta_{\text{old}}}} \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t \right] \quad \text{(replace } A^\pi \text{ with estimator)} \\
&= L^{IS}_{\theta_{\text{old}}}(\theta)
\end{aligned}
$$

- Kakade et al.[1] and S. et al.[2] analyze how L^{IS} approximates the actual performance difference between θ and θ_{old}.
- In practice, L^{IS} is not much different than the logprob version $L^{PG}(\theta) = \hat{\mathbb{E}}_t \left[\log \pi_\theta(a_t \mid s_t) \hat{A}_t \right]$, for reasonably small policy changes.

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

[2] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. "Trust Region Policy Optimization". *ICML* (2015).

Simple Monte Carlo

- Statistical sampling can be applied to any expectation
- In general we can find the expectation of *f(x)* by sampling:

$$\int f(x)P(x)dx \approx \frac{1}{S}\sum_{s=1}^{S} f(x^{(s)}) \quad x^{(s)} \sim P(x)$$

- The function *f(x)* is arbitrary, so this is very general

- Example: making predictions

$$P(x|\mathcal{D}) = \int P(x|\theta,\mathcal{D})P(\theta|\mathcal{D})d\theta$$

$$\approx \frac{1}{S}\sum_{s=1}^{S} P(x|\theta^{(s)},\mathcal{D}) \quad \theta^{(s)} \sim P(\theta|\mathcal{D})$$

Properties of Monte Carlo

- Estimator:

$$\int f(x)P(x)dx \approx \frac{1}{S}\sum_{s=1}^{S} f(x^{(s)}) \quad x^{(s)} \sim P(x)$$

- Estimator is unbiased

$$\mathbb{E}_{P(x^{(s)})}[\hat{f}] = \frac{1}{S}\sum_{s=1}^{S} \mathbb{E}_{P(x)}[f(x)] = \mathbb{E}_{P(x)}[f(x)]$$

- Variance shrinks $\propto 1/S$:

$$\mathrm{var}_{P(x^{(s)})}[\hat{f}] = \frac{1}{S^2}\sum_{s=1}^{S} \mathrm{var}_{P(x)}[f(x)] = \mathrm{var}_{P(x)}[f(x)]/S$$

Importance sampling

Computing both $\tilde{P}(x), \tilde{Q}(x)$ then throwing *x* away seems wasteful

Instead rewrite the integral as an expectation under Q:

$$\int f(x)P(x)dx = \int f(x)\frac{P(x)}{Q(x)}Q(x)dx$$
$$\approx \frac{1}{S}\sum_{s=1}^{S} f(x^{(s)})\frac{P(x^{(s)})}{Q(x^{(s)})}$$

This is just simple Monte Carlo again, so it is unbiased

$r^{(s)} = \dfrac{P(x^{(s)})}{Q(x^{(s)})}$ is called the *importance weight*

Importance sampling also applies when integral is not an expectation

Divide and multiply any integrand by a convenient distribution

Trust Region Policy Optimization

- Define the following trust region update:

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right] \\ \text{subject to} \quad & \hat{\mathbb{E}}_t[\mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]] \leq \delta. \end{aligned}$$

- Also worth considering using a penalty instead of a constraint

$$\underset{\theta}{\text{maximize}} \qquad \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right] - \beta\hat{\mathbb{E}}_t[\mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]]$$

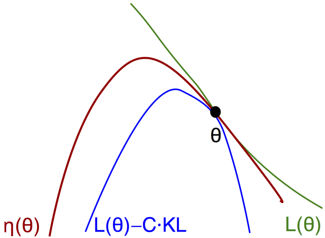
- Method of Lagrange multipliers: optimality point of δ-constrained problem is also an optimality point of β-penalized problem for some β.
- In practice, δ is easier to tune, and fixed δ is better than fixed β

Monotonic Improvement Result

- Consider KL penalized objective

$$\underset{\theta}{\text{maximize}} \quad \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right] - \beta\hat{\mathbb{E}}_t[\mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]]$$

- Theory result: if we use max KL instead of mean KL in penalty, then we get a lower (=pessimistic) bound on policy performance



Trust Region Policy Optimization: Pseudocode

- Pseudocode:

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

  Run policy for T timesteps or N trajectories

  Estimate advantage function at all timesteps

$$
\begin{aligned}
&\underset{\theta}{\text{maximize}} && \sum_{n=1}^{N} \frac{\pi_\theta(a_n \mid s_n)}{\pi_{\theta_{\text{old}}}(a_n \mid s_n)} \hat{A}_n \\
&\text{subject to} && \overline{\mathrm{KL}}_{\pi_{\theta_{\text{old}}}}(\pi_\theta) \leq \delta
\end{aligned}
$$

  end for

- Can solve constrained optimization problem efficiently by using conjugate gradient
- Closely related to natural policy gradients (Kakade, 2002), natural actor critic (Peters and Schaal, 2005), REPS (Peters et al., 2010)

Solving KL Penalized Problem

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

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

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

  - Quadratic part of L is negligible compared to KL term
  - F is positive semidefinite, but not if we include Hessian of L
- Solution: $\theta - \theta_{\text{old}} = \frac{1}{\beta} F^{-1} g$, where F is Fisher Information matrix, g is policy gradient. This is called the **natural policy gradient**[3].

[3] S. Kakade. "A Natural Policy Gradient." *NIPS*. 2001.

Solving KL Constrained Problem

- Method of Lagrange multipliers: solve penalized problem to get $\theta^*(\beta)$. Then substitute $\theta^*(\beta)$ into original problem and solve for β.
- β only affects scaling of solution, not direction. Compute scaling as follows:
  - Compute $s = F^{-1}g$ (soln with $\beta = 1$)
  - Rescale $\theta - \theta_{\text{old}} = \alpha s$ so that constraint is satisfied. Quadratic approx $\overline{\text{KL}}_{\pi_{\theta_{\text{old}}}}(\pi_\theta) \approx \frac{1}{2}(\theta - \theta_{\text{old}})^T F(\theta - \theta_{\text{old}})$, so we want

$$\frac{1}{2}(\alpha s)^T F(\alpha s) = \delta$$

$$\alpha = \sqrt{2\delta/(s^T F s)}$$

- Even better, we can do a line search to solve the original *nonlinear* problem.

$$\text{maximize}\ L_{\pi_{\theta_{\text{old}}}}(\pi_\theta) - \mathbf{1}[\overline{\text{KL}}_{\pi_{\theta_{\text{old}}}}(\pi_\theta) \leq \delta]$$

  Try $\alpha, \alpha/2, \alpha/4, \ldots$ until line search objective improves

"Proximal" Policy Optimization: KL Penalty Version

- Use 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

Review

- Suggested optimizing surrogate loss L^{PG} or L^{IS}
- Suggested using KL to constrain size of update
- Corresponds to natural gradient step $F^{-1}g$ under linear quadratic approximation
- Can solve for this step approximately using conjugate gradient method

Connection Between Trust Region Problem and Other Things

$$\begin{aligned} \underset{\theta}{\text{maximize}} & \sum_{n=1}^{N} \frac{\pi_\theta(a_n \mid s_n)}{\pi_{\theta_{\text{old}}}(a_n \mid s_n)} \hat{A}_n \\ \text{subject to} & \quad \overline{\mathrm{KL}}_{\pi_{\theta_{\text{old}}}}(\pi_\theta) \le \delta \end{aligned}$$

- Linear-quadratic approximation + penalty $\Rightarrow$ natural gradient
- No constraint $\Rightarrow$ policy iteration
- Euclidean penalty instead of KL $\Rightarrow$ vanilla policy gradient

Limitations of TRPO

- Hard to use with architectures with multiple outputs, e.g. policy and value function (need to weight different terms in distance metric)
- Empirically performs poorly on tasks requiring deep CNNs and RNNs, e.g. Atari benchmark
- CG makes implementation more complicated

Calculating Natural Gradient Step with KFAC

- Summary: do blockwise approximation to FIM, and approximate each block using a certain factorization
- Alternate expression for FIM as outer product (instead of second deriv. of KL):

$$\hat{\mathbb{E}}_t \left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)^T \nabla_\theta \log \pi_\theta(a_t \mid s_t) \right]$$

R. Grosse and J. Martens. "A Kronecker-factored approximate Fisher matrix for convolution layers". 2016
J. Martens and R. Grosse. "Optimizing neural networks with Kronecker-factored approximate curvature". 2015

Calculating Natural Gradient Step with KFAC

- Consider network with weight matrix W appearing once in network, with $y = Wx$. Then

$$\begin{aligned}\nabla_{W_{ij}} L = x_i \nabla_{y_j} L = x_i \overline{y}_j \\ F = & \hat{\mathbb{E}}_t \left[\nabla_{W_{ij}} \log \pi_\theta(a_t \mid s_t)^T \nabla_{W_{ij}} \log \pi_\theta(a_t \mid s_t) \right] \\ F_{ij,i'j'} = & \hat{\mathbb{E}}_t \left[x_i \overline{y}_j x_{i'} \overline{y}_{j'} \right]\end{aligned}$$

- KFAC approximation:

$$F_{ij,i'j'} = \hat{\mathbb{E}}_t \left[x_i \overline{y}_j x_{i'} \overline{y}_{j'} \right] \approx \hat{\mathbb{E}}_t [x_i x_{i'}] \hat{\mathbb{E}}_t \left[\overline{y}_j \overline{y}_{j'} \right]$$

- Approximating Fisher block as tensor product $A \otimes B$, where A is $nin \times nin$ and B is $nout \times nout$. $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$ and we can compute matrix-vector product without forming full matrix.
- Maintain running estimate of covariance matrices $\hat{\mathbb{E}}_t [x_i x_j], \hat{\mathbb{E}}_t \left[\overline{y}_{i'} \overline{y}_{j'} \right]$ and periodically compute matrix inverses (small overhead)

ACKTR: Combine A2C with KFAC Natural Gradient

- Combined with A2C, gives excellent on Atari benchmark and continuous control from images[4].
- Note: we're already computing $\nabla_\theta \log \pi_\theta(a_t \mid s_t)$ for policy gradient: no extra gradient computation needed
- Matrix inverses can be computed asynchronously
- Limitation: works straightforwardly for feedforward nets (including convolutions), less straightforward for RNNs or architectures with shared weights

[4] Y. Wu, E. Mansimov, S. Liao, R. Grosse, and J. Ba. "Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation". (2017).

Proximal Policy Optimization: KL Penalized Version

- 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

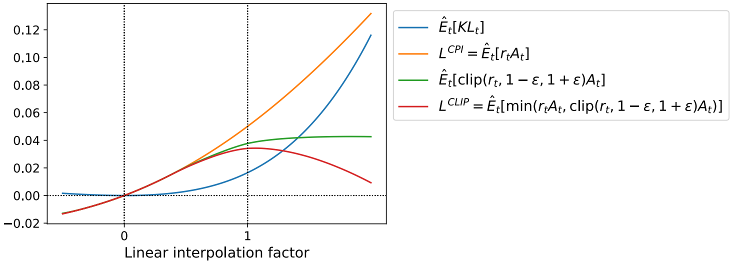
Proximal Policy Optimization: Clipping Objective

- Recall the surrogate objective

$$L^{IS}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right] = \hat{\mathbb{E}}_t\left[r_t(\theta)\hat{A}_t\right]. \quad (1)$$

- Form a lower bound via clipped importance ratios

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right] \quad (2)$$



- Forms pessimistic bound on objective, can be optimized using SGD

Proximal Policy Optimization

- Pseudocode:

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

  Run policy for T timesteps or N trajectories

  Estimate advantage function at all timesteps

  Do SGD on $L^{CLIP}(\theta)$ objective for some number of epochs

  end for

- A bit better than TRPO on continuous control, much better on Atari
- Compatible with multi-output networks and RNNs

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". Springer, 2012
- Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, et al. "Sample Efficient Actor-Critic with Experience Replay". (2016)
- Y. Wu, E. Mansimov, S. Liao, R. Grosse, and J. Ba. "Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation". (2017)
- J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. "Proximal Policy Optimization Algorithms". (2017)
- `blog.openai.com`: recent posts on baselines releases

That's all. Questions?