RL seminar #7: Advanced PG algorithms

Maksim Kretov

MIPT, Deep learning lab & iPavlov.ai

16 Dec 2017

Outline

Class information
- Results

Advanced Policy Gradient
- Drawbacks of PG
- Monotonic improvement theory
- Natural Policy Gradient
- Truncated Natural Policy Gradient
- Algorithms

Table of Contents

Class information
Results

Advanced Policy Gradient
Drawbacks of PG
Monotonic improvement theory
Natural Policy Gradient
Truncated Natural Policy Gradient
Algorithms

Results

Home assignments

Rating and course program: https://goo.gl/yq7fnf

- More than 30 points: 2 students
- [20-30) points: 0 students
- [10-20) points: 6 students
- [5-10) points: 5 students

Exam

Write at kretovmk@gmail.com, arrange time within 16-25 Dec 2017.

Feedback

is very welcome (format, tempo etc).
Please, send your thoughts at kretovmk@gmail.com.

Table of Contents

Class information

Results

Advanced Policy Gradient

Drawbacks of PG

Monotonic improvement theory

Natural Policy Gradient

Truncated Natural Policy Gradient

Algorithms

Drawbacks of PG

- Poor sample efficiency (on-policy)
- Updating procedure is brittle: small change in parameters of policy may cause significant policy change

Let's focus now on the second issue and try to ensure monotonic improvement of policy during training process.

Monotonic improvement theory

RL control task:

$$\pi^{opt} = \mathrm{argmax}_\pi J(\pi) \quad \pi = \pi(\theta) \quad \pi' = \pi(\theta')$$

Discounted future state distribution:

$$d^\pi(s) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t P(s_t = s|\pi)$$

Performance improvement:

$$J(\pi') - J(\pi) = \mathbb{E}_{\tau\sim\pi'}\Big[\sum_{t=0}^{\infty}\gamma^t A^\pi(s_t, a_t)\Big]$$

Problem: Need to sample whole trajectories from new policy π'.

Monotonic improvement theory

$$J(\pi') - J(\pi) = \mathbb{E}_{\tau \sim \pi'}\Big[\sum_{t=0}^{\infty} \gamma^t A^{\pi}(s_t, a_t)\Big] = \frac{1}{1-\gamma}\mathbb{E}_{s \sim d^{\pi'}, a \sim \pi'}[A^{\pi}(s, a)]$$

Exercise: prove last equality.

Re-write using importance sampling ratio:

$$J(\pi') - J(\pi) = \frac{1}{1-\gamma}\mathbb{E}_{s \sim d^{\pi'}, a \sim \pi}\Big[\frac{\pi'(a|s)}{\pi(a|s)} A^{\pi}(s, a)\Big]$$

Why this formula is better?

- We don't need sample from new policy π'

Still, there is distribution over states $s \sim d^{\pi'}$ that involves sampling from something that depends on new policy π'.

Monotonic improvement theory

$$J(\pi') - J(\pi) = \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^{\pi'}, a\sim\pi}\Big[\frac{\pi'(a|s)}{\pi(a|s)}A^{\pi}(s,a)\Big]$$

Lower bound (replaced π' with π in expectation) on policy performance improvement:

$$L_{\pi}(\pi') = \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^{\pi}, a\sim\pi}\Big[\frac{\pi'(a|s)}{\pi(a|s)}A^{\pi}(s,a)\Big]$$

$$J(\pi') - J(\pi) \geq L_{\pi}(\pi') - C\sqrt{\mathbb{E}_{s\sim d^{\pi}}[D_{KL}(\pi'(a|s)||\pi(a|s))]}$$

We eliminated any sampling from π'!

Monotonic improvement theory

So now we need to select π_{k+1}. At point $\pi_{k+1} = \pi_k$ lower bound on improvement is zero $\Rightarrow$ at each step k:

$$J(\pi_{k+1}) - J(\pi_k) \geq 0$$

Thus monotonic improvement is achieved.

But C is huge, because $C \sim (1-\gamma)^2 \Rightarrow$ subsequent policies are almost the same.

Monotonic improvement theory

Let's introduce approximation: trust regions. Replace original task:

$$\pi_{k+1} = \mathrm{argmax}_{\pi'}\left\{L_{\pi_k}(\pi') - C\sqrt{\mathbb{E}_{s\sim d^{\pi_k}}[D_{KL}(\pi'(a|s)||\pi_k(a|s))]}\right\}$$

With constrained optimization task:

$$\pi_{k+1} = \mathrm{argmax}_{\pi'} L_{\pi_k}(\pi')$$

$$\text{s.t. } \mathbb{E}_{s\sim d^{\pi_k}}[D_{KL}(\pi'(a|s)||\pi_k(a|s))] = D_{KL}^{av}(\pi'(a|s)||\pi_k(a|s)) \leq \delta$$

This is basic equations for the following algorithms.

Natural Policy Gradient

Idea: Let's derive update rule in closed form, using Lagrange multipliers technique.

In order to do that, make Taylor expansion of both target function and constraint ($\pi_k \to \theta_k,\ \pi' \to \theta$):

$$L_{\theta_k}(\theta) \approx L_{\theta_k}(\theta_k) + g^T(\theta_k - \theta)$$

$$D_{KL}^{av}(\theta||\theta_k) \approx \frac{1}{2}(\theta - \theta_k)^T H(\theta - \theta_k)$$

In the last formula there is no constant and linear terms because $D_{KL}^{av}(\theta||\theta_k)|_{\theta=\theta_k} = 0$ this KL-divergence is non-negative (so θ_k is minimum).

Natural Policy Gradient

Updated task:

$$\theta_{k+1} = \mathrm{argmax}_\theta \ g^T(\theta_k - \theta)$$
$$\text{s.t. } \frac{1}{2}(\theta - \theta_k)^T H(\theta - \theta_k) \leq \delta$$

Exercise: find solution using Lagrange multipliers.

Solution of approximate task:

$$\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g$$

This is Natural Policy Gradient algorithm.

Next: efficient calculation of Hessian ($\to$ TNPG).

Truncated Natural Policy Gradient

Dimension of H is NxN, and after that we need to invert it to use derived formula (complexity is $O(\log N^3)$).

Alternative

We need to find **search direction** $a = H^{-1}g$ and don't actually need whole Hessian matrix.

Approach

- Search direction a is a solution of linear system $Hx = g$.
- Use conjugate gradients algorithm. To use it, we only need to have access to function $f(v) = Hv$.

Truncated Natural Policy Gradient

How can we calculate $f(v) = Hv$ efficiently?[1]

$$(H\vec{v})_i = \sum_{j=1}^{N} \frac{\partial F(x)}{\partial x_i \partial x_j} v_j = \nabla \frac{\partial F(x)}{\partial x_i} \cdot \vec{v} = \nabla_v \frac{\partial F(x)}{\partial x_i}$$

And this is just directional derivative of function $\frac{\partial F(x)}{\partial x_i}$. We can calculate directional derivative with finite differences:

$$H\vec{v} \approx \frac{\nabla F(x + \epsilon \vec{v}) - \nabla F(x)}{\epsilon}$$

(in vectorized form)

So, we just need to calculate gradient two times in order to estimate $H\vec{v}$ for given $\vec{v} \Rightarrow$ We don't need to calculate H explicitly and invert it $\Rightarrow$ TNPG.

[1] andrew.gibiansky.com/blog/machine-learning/hessian-free-optimization

Algorithms

- Gradient descent: PG
- Natural gradients: PG $\rightarrow$ NPG
- Using CG to calculate $H^{-1}g$ efficiently: NPG $\rightarrow$ TNPG
- Line search to ensure improvement: TNPG $\rightarrow$ TRPO
- Simplify by appr. enforcing KL constraint: TRPO $\rightarrow$ PPO