Structured Priors for Policy Optimisation

Sihui Wang, Bill Byrne, Tom Gunter
June 19, 2017

Motivation

Why policy optimization?
- Greater efficiency and better convergence guarantees
- Able to learn stochastic policies
- Easy to use, value function might complicated

Why structured priors?
- We hope to achieve better sample efficiency by explicitly structuring policy search as a hierarchical problem with options.
- Hierarchical structured policies are suitable for many problems of interest, e.g. dialogue, robotics

Gradient-based Optimization

Policy function $\pi(a|s, \theta)$ is a mapping from $S \times A \rightarrow [0, 1]$.

Objective function, expected return of $\pi$:
$$\eta(\pi) = \eta(\pi_{old}) + \mathbb{E}_{s \sim \pi_{old}}[L_{\pi_{old}}(\pi, s, a)]$$
$$= \eta(\pi_{old}) + \mathbb{E}_{s \sim \pi_{old}}[\pi(a|s)]A_{\pi_{old}}(s, a)$$

However the state depends on the new policy parameter which makes it difficult to optimize.

Local approximation, $L(\pi)$:
$$L_{\pi_{old}}(\pi) = \eta(\pi_{old}) + \mathbb{E}_{s \sim \pi_{old}}[\pi(a|s)]A_{\pi_{old}}(s, a)$$

$\eta$ to first order
$$\nabla \eta L_{\pi_{old}}(\pi)|_{\theta = \theta_{old}} = \nabla \eta \pi(a|s)|_{\theta = \theta_{old}}$$

Monotonic improvement by updating surrogate function $M[1]$:
$$\eta(\pi) \geq L_{\pi_{old}}(\pi) - CD_{KL}(\pi_{old}, \pi) = M_{\pi_{old}}(\pi)$$

where $C = \frac{2}{\lambda_{max}} < 1$ is maximum $|\mathbb{E}_{s \sim \pi_{old}}[A_{\pi_{old}}(s, a)]|$.

Minorization-Maximization algorithm:
$$\eta(\pi) - \eta(\pi_{old}) \geq M_{\pi_{old}}(\pi) - M_{\pi_{old}}(\pi_{old})$$

Reinforcement learning problem to optimization problem:
$$\max_{\theta} L_{\pi_{old}}(\theta) - CD_{KL}(\theta_{old}, \theta) \leq \delta$$

Practical Approximation: TRPO

- Use a hard constrain on KL divergence to allow large update

$$\max_{\theta} \mathbb{E}_{s \sim \pi_{old}}[\pi(a|s)]A_{\pi_{old}}(s, a)$$

subject to $L_{\pi_{old}}(\theta_{old}, \theta) \leq \delta$

Policy improvement:
$$\pi(a|s) \sim \mathcal{N}(\text{mean} = \text{NeuralNet}(s; \{W, b\}), \text{std} = \exp(w_{\text{std}}))$$

Swimmer: swim forward as fast as possible. [State dim: 8, action dim: 2]

Structural TRPO

The idea of hierarchical policy is introduced in [2]. A hierarchical policy $\pi(a|s)$ consists of a set of several sub-policies and a gating network $\pi(o|s)$.
$$\pi(a|s) = \sum_{o} \pi(o|s)\pi(a|s, o)$$

Algorithm 1. Vanilla Hierarchical TRPO
1. Initialize both gating network and sub-policy parameters
2. for $i = 1 : N$
3. Reset the environment and sample $o \sim \pi(o|s)$
4. for $j = 1 : M$
5. Follow $a \sim \pi(a|s, o)$
6. If done: reset environment and start a new episode
7. end for
8. Using TRPO to update parameters of a particular sub-policy
9. end for
10. Using TRPO to update gating network parameters
11. Go back to step 2

Future Work
- Learn to use different option within a single episode.
- Reduce the variance in policy gradient method.
- Consider using boosting methods to include multiple sub-policies.

References


Results from OpenAI Gym

Continuous state space and discrete action: $\pi(\cdot|s) = \text{softmax}(a^T s + b)$
CartPole-v1: Balance a pole on a cart [State dim: 4, No. of actions: 2]

Acrobot-v1: Swing up a two link robot. [State dim: 6, No. of actions: 3]