Note: Proximal Policy Optimization

Yanqing Wu

Viwistar Robotics

Update: November 22, 2021

PPO: first-order method, on-policy learning, actor-critic structure

1 Background

In normal policy gradient, training can be unstable due to sparse reward and sampled target, causing dramatic changes on the policy; this eventually collapses performance. To enhance training stability, we want to limit the parameter updates that would change the policy too much at one step. With this goal, True Region Policy Optimization (TRPO) and Proximal Policy Optimization (PPO) was introduced. TRPO delimit the KL divergence with a constraint. More specifically, TRPO updates policies by taking the largest step possible to improve performance while satisfying the constraint on how close the new and old policies are allowed to be. However, TRPO is complicated to implement and is computational-heavy (from the computation of second-order derivative of KL-divergence). PPO simplifies it by using a **clipped surrogate objective** while retaining similar constraints and similar performance. Instead of enforcing a hard constraint as in TRPO, PPO formalized the constraint as a penalty in the objective function.

There are two primary variants of PPO: PPO-Penalty and PPO-Clip.

PPO-Penalty approximately solves a KL-constrained update like TRPO, but penalizes the KL-divergence in the objective function instead of making it a hard constraint, and automatically adjusts the penalty coefficient throughout training so that it is scaled appropriately.

PPO-Clip does not have a KL-divergence term in the objective and does not have a constraint at all. Instead relies on specialized clipping in the objective function to remove incentives for the new policy to drift from the old policy, PPO does hard clipping the policy ratio to be within a small range around 1.0, where 1.0 means the new policy is the same as old.

According to the PPO paper, PPO-Penalty performs worse than PPO-Clip.

2 Implementation

2.1 OpenAI PPO

Algorithm 1 PPO, Actor-Critic Style

for iteration=$1, 2, \ldots$ **do**
  for actor=$1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

Figure 1: Screenshot of OpenAI PPO algorithm

Objective loss is given by

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF} + c_2 S[\pi_\theta](s_t)]$$

where c_1, c_2 are coefficients, θ is a vector of policy parameters, S is entropy bonus, L^{CLIP} is given by

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

and squared-error loss L^{VF} is given by

$$(V_\theta(s_t) - V_t^{target})^2$$

Truncated version of advantage estimator $\hat{A}_t$ (for policy gradient implementation) is given by

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1}$$

where $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$. The surrogate loss L is optimized with minibatch Stochastic Gradient Descent (SGD) on NT timesteps of data (where N is the number of parallel actors and T is timesteps).

Another version of L^{CLIP} is as follows:

$$L(s, a, \theta_k, \theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a), \quad g(\epsilon, A^{\pi_{\theta_k}}(s, a))\right)$$

where $g(\epsilon, A) = (1+\epsilon)A$ when $A \geq 0$ and $g(\epsilon, A) = (1-\epsilon)A$ when $A < 0$. Through some simplifications, we see that probability ratio $\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}$ is clipped at $1+\epsilon$ $(A > 0)$ and $1-\epsilon$ $(A < 0)$. The hyperparameter ϵ specifies how much the new policy is allowed to change from the old policy (while still profiting the objective). Therefore, the clipping acts as a regularizer by discouraging the policy to change dramatically.

2.2 DeepMind PPO

DeepMind paper provides a more detailed pseudocode than OpenAI. According to DeepMind pseudo-code of PPO (2)

Algorithm 1 Proximal Policy Optimization (adapted from [8])

for $i \in \{1, \cdots, N\}$ **do**
  Run policy π_θ for T timesteps, collecting $\{s_t, a_t, r_t\}$
  Estimate advantages $\hat{A}_t = \sum_{t'>t} \gamma^{t'-t} r_{t'} - V_\phi(s_t)$
  $\pi_{\text{old}} \leftarrow \pi_\theta$
  for $j \in \{1, \cdots, M\}$ **do**
    $J_{PPO}(\theta) = \sum_{t=1}^{T} \frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)} \hat{A}_t - \lambda \text{KL}[\pi_{old}|\pi_\theta]$
    Update θ by a gradient method w.r.t. $J_{PPO}(\theta)$
  end for
  for $j \in \{1, \cdots, B\}$ **do**
    $L_{BL}(\phi) = -\sum_{t=1}^{T} (\sum_{t'>t} \gamma^{t'-t} r_{t'} - V_\phi(s_t))^2$
    Update ϕ by a gradient method w.r.t. $L_{BL}(\phi)$
  end for
  if $\text{KL}[\pi_{old}|\pi_\theta] > \beta_{high} \text{KL}_{target}$ **then**
    $\lambda \leftarrow \alpha\lambda$
  else if $\text{KL}[\pi_{old}|\pi_\theta] < \beta_{low} \text{KL}_{target}$ **then**
    $\lambda \leftarrow \lambda/\alpha$
  end if
end for

Figure 2: Screenshot of DeepMind PPO algorithm

In this algorithm, KL_{target} is the desired change in the policy per iteration. The Actor maximizes J_{PPO}, and Critic minimizes L_{BL}.

3 Reference

Will update later.