1 Proximal Policy Optimization

Policy Gradient methods, compute an estimate of the gradient of the objective function with respect to a parametrized policy (rather than action-value function in q-learning) to go stochastic gradient ascent directly on the policy. The most natural form of the gradient is

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

, where π_θ is the stochastic parameterized policy and$\bar{\mathrm{A}}_t$ is the emprirical advantage function at time 't', $\hat{\mathrm{E}}[\dots]$ is the empirical advantage, and $\hat{g}$ can be obtained by taking a gradient of $L^{PG}(\theta) = \hat{\mathrm{E}} \left[\log \pi_\theta(a_t|s_t) \, \bar{\mathrm{A}}_t \right]$

1.1 Trust Region Methods

In this cases we often try to define a surrogate objective function that is a lower bound of the true objective function, and maximize iteratively on this lower bound. They are based on a constrained optimization of the following objective

$$\underset{\theta}{\text{maximize}} \, \hat{\mathrm{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \, \bar{\mathrm{A}}_t \right]$$
$$\text{subject to } \hat{\mathrm{E}}_t \left[\mathrm{KL}[\pi_{\theta_{\text{old}}}(.|s_t), \pi_\theta(.|s_t)] \right] \leq \delta$$

, and the paper mentions that the objective could be solved by making a linear approximation of objective and quadratic approximation of the constrained using most probably a Taylor series expansion of the same. TRPO proposes to make it a penalty based optimization problem rather than a constrained based

$$\underset{\theta}{\text{maximize}} \, \hat{\mathrm{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \, \bar{\mathrm{A}}_t - \beta \mathrm{KL}[\pi_{\theta_{\text{old}}}(.|s_t), \pi_\theta(.|s_t)] \right] \tag{1}$$

1.2 Clipped Surrogate Objective

From here the contribution of the paper start where they introduce another surrogate objective which doesn't require calculating a second order derivative, $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$.

TRPO maximizes the following surrogate with a constraint

$$L^{CPI} = \hat{\mathrm{E}} \left[r_t(\theta) \, \bar{\mathrm{A}}_t \right], \quad r(\theta_{old}) = 1$$

, the propose a new objective which tries to restrict large changes in the policy i.e.

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

where ϵ is a hyper-parameter, this clips the change in the policy by moving out in the factor of $1+\epsilon$ in the case of improvement and $1-\epsilon$ in the case where we want to decrease the probability of the action.

1.3 Adaptive KL Penalty Coefficient

Knowing the problem of setting in β in Eq:1, we introduce an adaptive KL penalty, to achieve some target KL difference d_{targ} each update, i.e. using the following routine. Compute : $d = \hat{\mathrm{E}}_t \left[\mathrm{KL}[\pi_{\theta_{\text{old}}}(.|s_t), \pi_\theta(.|s_t)] \right]$, if $d < d_{targ}/1.5, \beta \leftarrow \beta/2$ if $d > d_{targ}/1.5, \beta \leftarrow \beta \times 2$.

1.4 Calculating Advantage $\bar{\mathrm{A}}_t$

Most techniques for computing variance reduced advantage function use a state value function $V(s)$. If our function approximation shares parameters for the policy and the value function, we must use a objective function with value function error as well. Entropy can also be added to encourage exploration.

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

where c_1, c_2 are coefficients, S denotes entropy bonus, and L_t^{VF} is a squared-error loss $(V_\theta(s_t) - V_t^{targ})^2$. The advantage function is calculated as

$$\begin{aligned} \bar{\mathrm{A}}_t &= -V(s_t) + r_t + \gamma r_{t+1} + \dots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V(s_T) \\ &= \delta - t + (\gamma\lambda)\delta_{t+1} + \dots + (\gamma\lambda)^{T-t+1} \delta_{T-1} \\ &\text{where } \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \end{aligned}$$

where $tin[0, T]$, where length T trajectory segment and for $\lambda = 1$ we get the same expression as above. The best performing parameters for PPO is $\epsilon = 0.2$

2 Deep Reinforcment Learning that Matters

This section discusses challenges posted by reproducibility, proper experimental techniques and reporting procedures. This will particularly focus on investigation of Policy Gradients methods in the domain of continuous control, using deep neural networks.

2.1 PG Objectives

This study will mainly pertain to 4 algorithms i.e. Trust Region Policy Optimization (TRPO), Deep Deterministic Policy Gradients (DDPG), Proximal Policy Optimization (PPO) and Actor Critic using Kronecker-Factored Trust Region (ACKTR). The general objective is often given by $p(\theta, s_0) = \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t r(s_t)|s_0]$, and using PG theorem we can get the following gradient update $\frac{\partial p(\theta, s_0)}{\partial \theta} = \sum_s \mu_{\pi_\theta}(s|s_0) \sum_a \frac{\partial \pi_\theta(a|s)}{\partial \theta} Q_{\pi_\theta}(s, a)$, here $\mu_{\pi_\theta}(s|s_0) = sum_{t=0}^{\infty} \gamma^t P(s_t = s|s_0)$where the state transition distribution and action value functions gradients are not required simplifying the problem to a large extent. The PPO and TRPO introduce surrogate objective functions that are explained in the previous section, which act as the lower bound for the objective functions.

2.2 Experimental Analysis

The use OpenAI baselines implementation on the some of the OpenAI Gym Environments and Mujoco envs. A 2 layer MLP is used with different activation functions, DDPG - (64,64, ReLU), TRPO and PPO - (64,64,tanh), ACKTR - (64,64,tanh) actor and (64,64, ELU) for the critic.

HyperParameters : We will study the effect of hyper parameters on the performance of RL algorithms, starting from the Network Architecture, Reward, Random Seeds, and Environment and Codebases. Using reward normalization is something which is highly dependent on the environment. Random Seed can not often perform same i.e. algorithm performance can actually depend on the the random seed. Environment, when proposing a new novel algorithm it is indispensable to show it across multiple domains. Codebases also have a great affect on the performance.