Relative Entropy Regularized Policy Iteration

Abbas Abdolmaleki, Jost Tobias Springenberg, Jonas Degrave, Steven Bohez, Yuval Tassa, Dan Belov, Nicolas Heess, Martin Riedmiller
DeepMind, London, UK
{aabdolmaleki,springenberg,grave,sbohez,danbelov,heess,riedmiller}@google.com

【强化学习 84】RERPI

张楚珩
清华大学 交叉信息院博士在读

RERPI 不是官方的简称，个人标记一下，算法全称叫做 Relative Entropy Regularized Policy Iteration。

原文传送门

Abdolmaleki, Abbas, et al. "Relative entropy regularized policy iteration." arXiv preprint arXiv:1812.02256 (2018).

特色

前一讲做 MPO 那一帮人做的后续工作，思路比较类似，大致有以下两点改进：1）在 policy evaluation 步骤中，不再使用比较复杂的 Retrace，而是使用更为简单的 TD(0)，事实证明，效果并不会更差；2）在 non-parametric 的 policy improvement 步骤中，使用一些演化算法的想法（CMA-ES），相比于 MPO 更 robust（比如能够 invariant to rewards scaling）。

过程

1. 策略循环框架

总体框架还是考虑一个 policy iteration，包含 policy improvement 和 policy evaluation：

Algorithm 1 Actor-Critic

Initialize $\pi^{(0)}, Q^{\pi^{(-1)}}, k \leftarrow 0$
repeat
  $Q^{\pi^{(k)}} \leftarrow \text{PolicyEvaluation}(\pi^{(k)}, Q^{\pi^{(k-1)}})$
  $\pi^{(k+1)} \leftarrow \text{PolicyImprovement}(\pi^{(k)}, Q^{\pi^{(k)}})$
  $k \leftarrow k+1$
until convergence

其中 policy evaluation 步骤就是用简单的 TD(0) 作为误差函数：

$$\min_\phi \left(r_t + \gamma Q_{\phi'}^{\pi^{(k-1)}}\left(s_{t+1}, a_{t+1} \sim \pi^{(k-1)}(a|s_{t+1})\right) - Q_\phi^{\pi^{(k)}}\left(s_t, a_t\right)\right)^2,$$

2. 策略改进步

和 MPO 类似，还是分为 E-step 和 M-step，先找到一个 non-parametric 的分布，然后把这个分布泛化到一个策略网络中。policy improvement step 仍然是优化近似的『一步』优化目标 $J(s,\pi)=\mathbb{E}_\pi[Q^{\pi^{(k)}}(s,a)]$ ，即 state 分布还是从原来的轨迹分布中得到，但是 action 的分布从新的策略中得到。然后找到一个 non-parametric 的 q 使得 $J(s,q)\geq J(s,\pi^{(k)})$ 。这里 $\pi^{(k)}$ 表示前一轮的旧策略。接下来，把得到的这个 non-parametric 的 q 泛化到策略网络中：

$$\pi^{(k+1)} = \operatorname{argmin}_{\pi_{\boldsymbol{\theta}}} \mathbb{E}_{\mu_\pi(s)}\left[\mathrm{KL}\left(q(a|s)\|\pi_{\boldsymbol{\theta}}(a|s)\right)\right] \tag{2}$$

先讲一下 E-step。

每一轮从 replay buffer 里面采样样本 $\{s_j\}_{j=1,\cdots,K}$ ，然后对于每个样本采样行动 $\{a_i\}_{i=1,\cdots,N}\sim\pi^{(k)}(a|s_j)$ ，并且把这些 state-action pair 送到 Q 函数中得到相应的估计值。接下来，我们需要决定这 NK 个 state-action pair 的 $q_{ij}=q(a_i|s_j),\forall s_j,a_i$ 。文章提出了使用三种不同的方法来设定 q_{ij} ：

使用 CMA-ES

根据每个状态下不同 action 的 Q 估计值的排序，来设定它们对应的采样概率： $q_{ij}\propto\ln(\frac{N+\eta}{rank(i)})$ ，其中 η 是一个温度参数，比如 CMA-ES 的算法里面它为 0.5， $rank(i)$ 表示第 i 个行动的 Q 估计值在该状态下的所有行动的 Q 估计值的排序，可以看出排序越靠前的行动，设定的下一步选中概率越高。

使用指数变换

解如下优化问题：

$$\begin{aligned} q_{ij} = &\underset{q(a_i|s_j)}{\operatorname{argmax}} \sum_j^K \sum_i^N q(a_i|s_j) Q^{\pi^{(k)}}(s_j, a_i) \\ &s.t. \frac{1}{K}\sum_j^K\sum_i^N q(a_i|s_j)\log\frac{q(a_i|s_j)}{\frac{1}{N}} < \epsilon, \qquad \forall_j \sum_i^N q(a_i|s_j) = 1 \end{aligned}$$

即，不仅要求分布 q （作为一个加权平均的权重）能够最大化在估计的 Q 函数上的加权平均值，还要求它具有较高的 entropy。如果没有后面的约束项，容易看到，分布 q 会把权重全部加到最大的那一部分 aciton 上。上述优化问题有闭式解：

$$q_{ij} = q(a_i, s_j) = \exp\left(Q^{\pi^{(k)}}(s_j, a_i)/\eta\right)/Z(j),$$

where $Z(j) = \sum_i \exp\left(Q^{\pi^{(k)}}(s_j, a_i)/\eta\right)$. The temperature η corresponding to the constraint ϵ can be found automatically by solving the following convex dual function alongside our policy optimization:

$$\eta = \operatorname{argmin}_\eta \eta\epsilon + \eta \sum_j^K \frac{1}{K} \log\left(\sum_i^N \frac{1}{N} \exp\left(\frac{Q(s_j, a_i)}{\eta}\right)\right)$$

这其实和前面的 MPO 类似。

使用恒等变化

即 $q_{ij} \propto Q^{\pi^{(k)}}(s_j, a_i)$ ，这和史前的 expected policy gradient algorithm 类似。

下面讲一下 M-step，这其实和 MPO 一样，只不过这篇 pape 里面在正文中讲的比较详细，主要的方法还是使用 Lagrangian Relaxation 解如下优化问题

$$\pi^{(k+1)} = \underset{\pi_\theta}{\operatorname{argmax}} \sum_j^K \sum_i^N q_{ij} \log \pi_{\boldsymbol{\theta}}(a_i|s_j), \quad \text{s.t.} \sum_j^K \frac{1}{K} \mathrm{KL}(\pi^{(k)}(a|s_j) \,\|\, \pi_{\boldsymbol{\theta}}(a|s_j)) < \epsilon_\pi, \quad (4)$$

即：

$$\max_{\boldsymbol{\theta}} \min_{\alpha > 0} L(\boldsymbol{\theta}, \eta) = \sum_j \sum_i q_{ij} \log \pi_{\boldsymbol{\theta}}(a_i|s_j) + \alpha\Big(\epsilon_\pi - \sum_j^K \frac{1}{K} \mathrm{KL}(\pi^{(k)}(a|s_j) \,\|\, \pi_{\boldsymbol{\theta}}(a|s_j))\Big).$$

由于两个高斯分布 KL 散度能够写成两部分，一部分主要限制均值的变化，一部分主要限制协方差矩阵的变化，因此文章发现，与其直接限制总体 KL 散度的变化 ϵ_π ，不如把它拆开为两部分分别限制这两部分的变化，这样可以分别控制探索和利用，即：

$$\begin{aligned}
\pi^{(k+1)} &= \underset{\mu_\theta, \Sigma_\theta}{\operatorname{argmax}} \sum_j^K \sum_i^N q_{ij} \log \pi_{\boldsymbol{\theta}}(a_i|s_j; \Sigma = \Sigma^k) + \sum_j^K \sum_i^N q_{ij} \log \pi_{\boldsymbol{\theta}}(a_i|s_j; \mu = \mu^k) \\
\text{s.t. } \epsilon_\mu &> \frac{1}{K} \sum_j^K \mathrm{KL}(\pi^{(k)}(a|s_j) \,\|\, \pi_{\boldsymbol{\theta}}(a|s_j; \Sigma = \Sigma^k)), \\
\epsilon_\Sigma &> \frac{1}{K} \sum_j^K \mathrm{KL}(\pi^{(k)}(a|s_j) \,\|\, \pi_{\boldsymbol{\theta}}(a|s_j; \mu = \mu^k)).
\end{aligned}$$

实验

这篇工作测试了很多连续控制的环境，主要包括三大类环境：Parkour suite、DeepMind control suite 和 OpenAI Gym。根据文章给出的结果，个人感觉该算法效果很好。

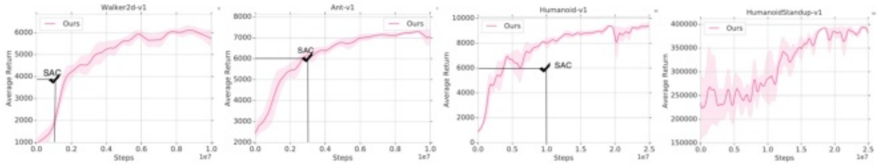

Figure 6: Comparison between our algorithm and SAC on walker, ant and humanoid from OpenAI gym. Check-mark shows the best reported performance of SAC [Haarnoja et al., 2018]. Results show that our method solve the tasks with same hyper parameters as before while achieving considerably better asymptotic performance than SAC with on-par sample efficiency. SAC does not report any result on humanoid-standup. Humanoid-standup is in particular interesting because of its very different reward scale with respect to other environments. Note that our method can also solve humanoid-stand with final return of 4000000 using the same hyper parameters.

发布于 2019-08-03