Data-Efficient Hierarchical Reinforcement Learning

Ofir Nachum
Google Brain
ofirnachum@google.com

Shixiang Gu*
Google Brain
shanegu@google.com

Honglak Lee
Google Brain
honglak@google.com

Sergey Levine†
Google Brain
slevine@google.com

【强化学习算法 19】HIRO

张楚珩
清华大学 交叉信息院博士在读

HIRO是Hlerarchical Reinforcement learning with Off-policy correction的缩写。

原文传送门：

Nachum, Ofir, et al. "Data-Efficient Hierarchical Reinforcement Learning." arXiv preprint arXiv:1805.08296 (2018).

特色：

提出了一种**general**并且**off-policy**的HRL算法。general是相比于当下有一些针对特定任务特殊设计的算法来说的。（可以参考本专栏前面讲到的NJUStarCraft和h-DQN）off-policy即呼应了标题里面提到的data-efficient。off-policy是出了名的不稳定，大家用来很多算法才让off-policy算法在一般RL问题上稳定；这里有两层策略，会带来新的不稳定，这里采用了一些修正让它稳定。

背景：

HRL一般的设定参考本专栏前面的文章【强化学习算法 18】FuN。

过程：

1. 算法大体框架

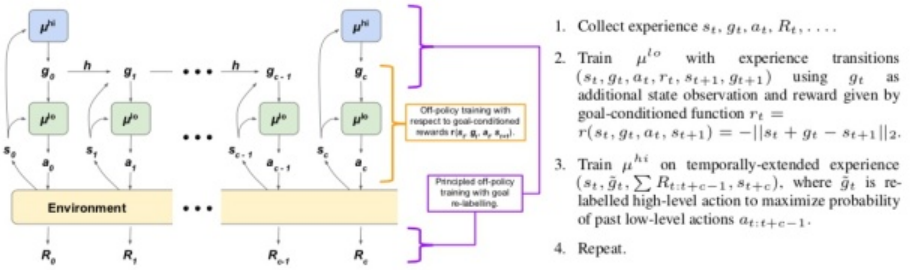

Figure 2: The design and basic training of HIRO. The lower-level policy interacts directly with the environment. The higher-level policy instructs the lower-level policy via high-level actions, or goals, $g_t \in \mathbb{R}^{d_s}$ which it samples anew every c steps. On intermediate steps, a fixed goal transition function h determines the next step's goal. The goal simply instructs the lower-level policy to reach specific states, which allows the lower-level policy to easily learn from prior off-policy experience.

分层为两层，分别是higher-level policy和lower-level policy。上层策略每隔c步调用一次，产生一个目标，下层策略在这c不里面尽量去完成这个目标，上层策略和下层策略分别用现有的off-policy RL算法去训练，这里使用的是TD3。

2. 定义上层策略产生的目标

上层策略产生的目标 g_t 定义为原状态空间里面的变化，即希望下层空间在c步里面由 s_t 变成 $s_t + g_t$ 。下层策略的状态除了原MDP的状态之外，还输入上层策略给定的目标 g_t 。由于下层策略的时间粒度更细，因此每一步都需要对上层空间给定的目标做一个转化，即 $g_{t+1} = h(s_t, g_t, s_{t+1}) = s_t + g_t - s_{t+1}$ 。

3. 定义下层策略的奖励

需要根据上层空间给定的这个含义明确的目标定义下层策略的奖励，定义方法很直观，就是状态空间里面的L2距离。

$$r(s_t, g_t, a_t, s_{t+1}) = -||s_t + g_t - s_{t+1}||_2. \tag{3}$$

4. off-policy修正

这里off-policy的实现方法是使用experience replay，通常而言，其存放的元素是 (s_t, a_t, r_t, s_{t+1}) 。标准的off-policy方法解决的问题是过去的策略在 s_t 时产生 a_t 的概率和当前策略不匹配的问题。但是在训练上层策略的时候，还会面临另外的不匹配，那就是过去采取行动 a_t （对于上层策略来讲就是 g_t ）之后获得的奖励 r_t （对于上次策略来讲就是 $\sum R_{t:t+c-1}$ ）和当前策略的不匹配。为什么会这样呢？因为下层的策略也在不断的学习之中，因此相同的 g_t 对应实际的动作序列也在发生变化。

如何对此进行修正呢？

文章里面使用了一个个人觉得比较傻的办法，即固定奖励 $\sum R_{t:t+c-1}$ 不变，但是去找**在当前条件下给一个怎样的 $\tilde{g}_t$ 可以最大可能产生以前下层策略动作序列和奖励**。即 $\tilde{g}_t = arg\max \mu^{lo}(a_{t:t+c-1} | s_{t:t+c-1}, \tilde{g}_{t:t+c-1})$ 。然后做近似

$$\log \mu^{lo}(a_{t:t+c-1}|s_{t:t+c-1}, \tilde{g}_{t:t+c-1}) \propto -\frac{1}{2}\sum_{i=t}^{t+c-1} ||a_i - \mu^{lo}(s_i, \tilde{g}_i)||_2^2 + \text{const.} \quad (5)$$

做完近似也求不出来这个 $argmax$ 呀。然后就随机采样（其中包含两个特殊样本原来的 g_t 和 $s_{t_c} - s_t$ ）找一个最大的。（看到这里感觉有点粗暴了啊，不过人家还对比了很多其他方法，这个实验效果最好）

结果：

最后结果没太大值得说的，就不贴图了。不过值得一说的是，文章一直跟FuN比较说自己做实验在原空间更好。但我就想说，本文的实验环境和FuN的都不一样呀。最大的区别是FuN用的是Video Input，这里用的是低维的输入（参见其提供的代码）。

发布于 2018-10-17

算法 机器学习 强化学习 (Reinforcement Learning)