FeUdal Networks for Hierarchical Reinforcement Learning

Alexander Sasha Vezhnevets VEZHNICK@GOOGLE.COM
Simon Osindero OSINDERO@GOOGLE.COM
Tom Schaul SCHAUL@GOOGLE.COM
Nicolas Heess HEESS@GOOGLE.COM
Max Jaderberg JADERBERG@GOOGLE.COM
David Silver DAVIDSILVER@GOOGLE.COM
Koray Kavukcuoglu KORAYK@GOOGLE.COM
DeepMind

【强化学习算法 18】FuN

张楚珩
清华大学 交叉信息院博士在读

很老的一篇文章提出了feudal RL，FuN指的是把它用于分层强化学习。

原文传送门：

Vezhnevets, Alexander Sasha, et al. "Feudal networks for hierarchical reinforcement learning." arXiv preprint arXiv:1703.01161 (2017).

背景：

分层强化学习被认为是解决复杂强化学习问题的重要方法，一般来讲大家就分为两层，使用这篇文章里面的表示方法就是Manager和Worker。如果不分层，有很多off-the-shelf的强化学习算法已经能够解这样一个 $MDP=\{S,A,R,T,\gamma\}$ 了，其中各个元素分别代表状态空间、动作空间、奖励、状态转移概率、discount rate。当分层之后，自然就变成了两个MDP了，一个Manager的、一个Worker的，那么这两个MDP的这个五个元素该怎么定义就需要一一考虑一下了。

- Manager的状态空间一般也就可以直接用原来MDP的状态空间。当然对于高维状态空间（比如图像），会用CNN转化到embedded space里面；对于不完全可观测的、非马可夫的，可能还可以用RNN来增强可观测性（这篇文章里面也用到了）。
- Manager的动作空间就是一个很大的研究热点了。一种方案就是其动作空间就用来选择不同的worker去做操作，比如Manager这会让第一个worker去执行，过会再让第二个worker去执行，这对应的大概就是option-critic framework（可以参考本专栏里面讲的NJUStarCraft那篇）；第二种方案就是Manager制定一个目标，然后让Worker去执行，那么目标怎么定义又是一个问题了，这也是本文试图解决的一个问题。
- Manager的奖励也可以直接用原来MDP的奖励。需要注意到HRL是用来解决困难强化学习问题的工具，其中一个很大的困难体现在sparse reward，换句话说就是long-term credit assignment问题。Manager的目标就是在时间尺度上看的更粗略一些，这样原本稀疏的奖励在Manager看来就不那么稀疏了。
- Manager对应的Transition也会相应变化，以前的Transition Model是 $p(s_{t+1}|s_t,a_t)$ ，现在的差不多变成了 $p(s_{t+c}|s_t,g_t)$ ，一般Manager的动作用 g_t 表示（goal）。这就产生了这篇文章里面讲的transition policy gradient（可以等会回过头来看）。
- Manager希望看的更远，更加注重长远的奖励，因此它相比于Worker来讲可以有更大的 γ 。
- Worker的状态空间呢？一般来讲，把原MDP的状态和Manager产生的goal输进来作为Worker的状态。
- Worker的动作空间一般就是原问题的动作空间，也就是原问题的实际操作（脏活累活）都交给Worker来实际解决。

- Worker的奖励也是一个比较棘手的问题。如果直接用原问题的奖励，原问题就是因为奖励稀疏所以难学，现在还用原来的奖励的话，等于是来了个领导不干活，所有的任务都靠Worker来做，这样显然不科学。因此，目前大家要么1）使用Manager产生的goal演化出来的比较密集的奖励（这篇文章用的方法），要么2）使用只依赖环境动力学模型的intrinsic reward（也是一个活跃的研究方向）。
- Worker的transition model和discount rate和原问题类似，可以使用off-the-shelf的各种RL算法来做。
- 控制权的转移是分层带来的问题中的一个。
  - 一般大家每c步让Manager做一下决策，然后接下来c步都让Worker行动，这样做的好处是在时间上显式地划分了两者的层级。但c怎么选择？是不是什么sub-policy都是固定c步比较好呢？
  - 另外有方法是让Worker行动结束了告诉Manager，但是很容易收敛到极差的worker（所有的worker都只做原来行动空间里面的一步，主要的任务靠Manager来完成）或者极好的worker（一个worker把所有的活都干了）。这都不是我们想要的。一个粗暴的解决方法就是加regularizer使得worker不要太强或者太弱。
  - 这篇文章里面就Manager每步都出信号，但是也做了限制避免产生一个“朝令夕改”的领导。

有了这些框架性概念我们就可以进入这篇文章给我们提供的解决方案了。

过程：

1. 总体流程

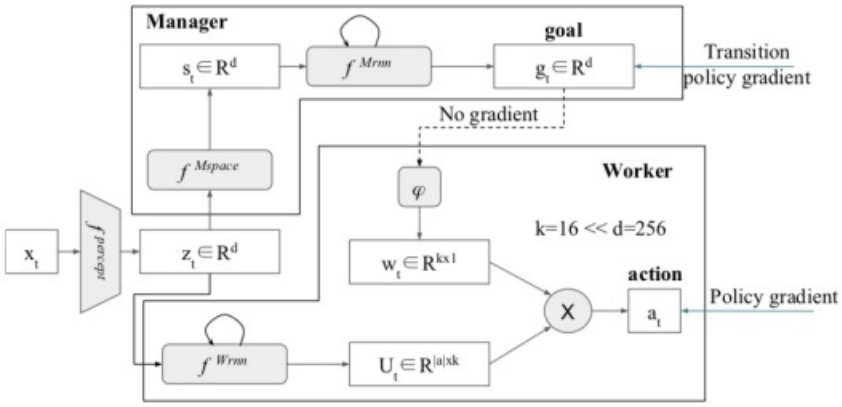


Figure 1. The schematic illustration of FuN (section 3)

由于高维输入，因此做了embedding（ $f^{percept}$ ）；Manager和Worker可能需要的对状态的embedding不一样，因此Manager又做了一个transform（ f^{Mspace} ）；任务可能non-Markovian，因此做了RNN的embedding（ f^{Mrnn} 和 f^{Wrnn} ）；Manager产生的goal扔给Worker，为了防止这个每步生

产的goal“朝令夕改”，因此对近c步的goal做pooling和linear projection之后再扔给Worker（ φ ）；两者拼起来通过神经网络（ x ）形成最后的动作。

这整个流程都是从头到尾都可求导的，直接用现有的RL算法就可以end-to-end训练了。那我们分层还有什么意义？？（惊！）

原来如果这样求解，Manager产生的goal就等于是一个hidden variable，没有对应的可以说的明白的含义了。因此，训练必须把两层分来来做训练。

2. 如何定义Manager的action（goal）？

要想把Manager和Worker分开训练，首先要做的就是给Manager输出的action（goal）一个确定的含义，而不是让它仅仅只是一个隐变量。这里规定goal是**学习到的低维状态表示空间中的方向**。学习到的低维状态表示即图上的 $z_t \in \mathbb{R}^d$ ，而输出的 $g_t \in \mathbb{R}^d$ 表示希望Worker能通过采取各种行动使得状态的表示往 $z_t + kg_t$ 方向变化。

3. 如何定义Worker的reward？

Manager制定的目标就需要Worker来完成，那么Worker就需要不仅受到外部奖励的激励，还需要受到内部奖励的激励，即 $R_t + \alpha R_t^I$ 。内部奖励的定义需要和Manager的目标关联，这里的定义是

$$r_t^I = 1/c \sum_{i=1}^{c} d_{\cos}(s_t - s_{t-i}, g_{t-i}) \qquad (8)$$

可以理解为当前状态相比于前 c 步中每一步目标完成情况的平均值。被称作**directional cosine similarity reward**。

4. 如何训练Manager？

刚刚提到要分成两部分来训练。对应训练Manager的方法就是这里所谓的**transition policy gradient**，它和普通Policy Gradient的区别就是普通的PG的transition model是 $\pi(s_{t+1}|s_t, a_t)$ ，而这里把每个连续的 c 步看做Manager的一步，把Worker连续 c 步的产生的状态变化当做一步transition，即 $\pi^{TP}(s_{t+c}|s_t) = p(s_{t+c}|s_t, g_t) = p(s_{t+c}|s_t, \mu(s_t, \theta))$ 。

对应产生的策略梯度公式就是

$$\nabla_\theta \pi_t^{TP} = \mathbb{E}\left[(R_t - V(s_t)) \nabla_\theta \log p(s_{t+c}|s_t, \mu(s_t, \theta))\right] \qquad (10)$$

由于内部奖励的形式是上面给出的形式，这就激励了Manager的transition分布更接近von Mises-Fisher分布，即 $p(s_{t+c}|s_t, g_t) \propto \exp d_{\cos}(s_{t+c} - s_t, g_t)$ ，因此可以进一步把上式化为

$$\nabla g_t = A_t^M \nabla_\theta d_{\cos}(s_{t+c} - s_t, g_t(\theta)), \qquad (7)$$

接下来就可以用策略梯度类方法来训练Manager了，文章中使用A3C方法。

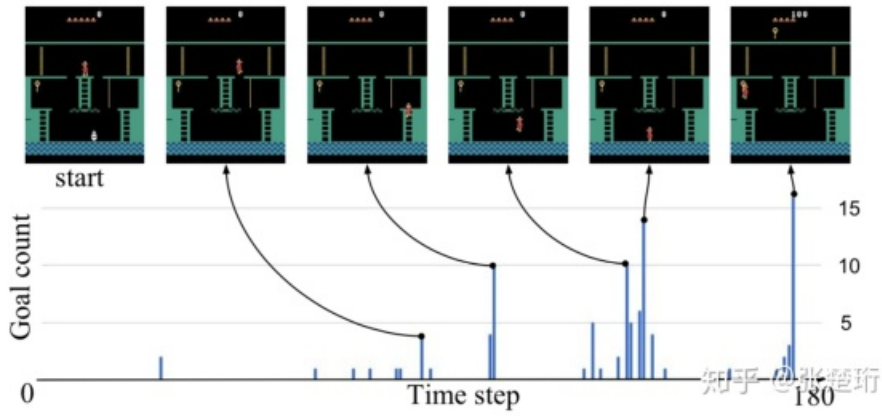
5. 如何训练Worker？

定义好了Worker的奖励之后，Worker的训练就变成了普通的RL问题了，使用A3C解决。

结果：

论文的positive result肯定都包含学习曲线，说明该算法的渐进性能和收敛速度更好，这里不贴出来了。

比较有意思的是一些说明Manager确实能学习到有意义的goal的实验。



这是在Montezuma's Revenge这个游戏上的实验，底下的count表明这一帧图像在前面的多少个时刻被当做goal，即可以把这样的bar比较高的位置对应的游戏局面理解为是Manager预想希望达到的局面。不难看出，这些帧基本上都是拿到钥匙的关键路径，比如先从楼梯边上跳下来，然后爬梯子、过河。

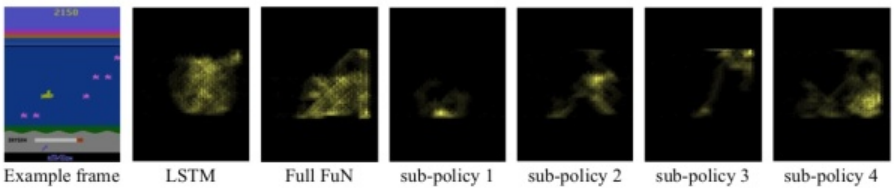


Figure 3. Visualisation of sub-policies learnt on sea quest game. We sample a random goal and feed it as a constant conditioning for the Worker and record its behaviour. We filter out only the image of the ship and average the frames, acquiring the heat-map of agents spatial location. From left to right: i) an example frame of the game ii) policy learnt by LSTM baseline iii) full policy learnt by FuN followed by set of different sub-policies. Notice how sub-policies are concentrated around different areas of the playable space. Sub-policy 3 is used to swim up for oxygen.

这个是在SeaQuest这个游戏上面的实验。后面的热力图表示不同策略下小船出现在不同地方的概率，可以看到选择不同的goal产生的不同sub-policy确实让小船局限在不同的区域移动，甚至比如sub-policy3的动作就是浮到水面上换气。

Manger训练的特殊技巧

回看算法的整体框图，Manager和Worker的策略产生都是使用一个RNN网络来做的，文章里面使用的LSTM。但是Manager希望在更粗粒度的时间尺度上观察问题，具体说来就是相对于Worker看问题的尺度有一个 c 倍的scale。为了让Manager确实能比Worker记忆的时间更长，自然想法就是每 c 步才喂一个数据到Manager的LSTM里面。但是这样可能漏掉中间的一些信息。然后自然会想到，弄 c 个这样的LSTM，每步数据进来的时候只训练其中的第 $t\%c$ 个模型（%表示取余）。这就是文章里面提到的**dilated LSTM**。

发布于 2018-10-17

算法 机器学习 强化学习 (Reinforcement Learning)

文章被以下专栏收录

强化学习前沿
读呀读paper

进入专栏