关于“分层强化学习”的前沿工作调研报告

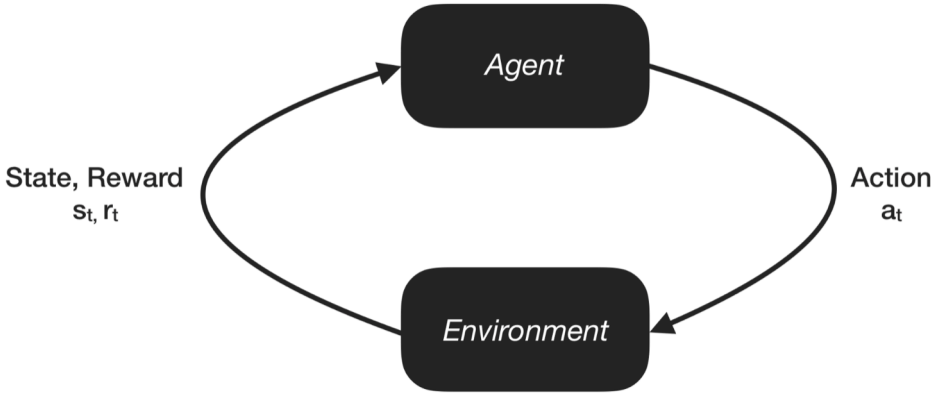
18340146 计算机科学与技术 宋渝杰

摘要

分层强化学习 (HRL) 是一种很有前途的方法，可以扩展传统的强化学习 (RL) 方法来解决更复杂的任务。而对于分层强化学习的研究，在每年的顶级会议都会收录不少相关的论文。在这片报告中，我将对其中近几年的几篇顶级会议论文进行调研，了解他们所做的前沿工作，并以我自己的理解和复述为主，引用论文原话为辅的方式，对这些工作进行调研汇报。

1 背景

近年来，强化学习 (RL) 方法在许多领域都取得了较好的研究结果，比如说从像素中学习玩雅达利游戏。最基础的强化学习模型如下：

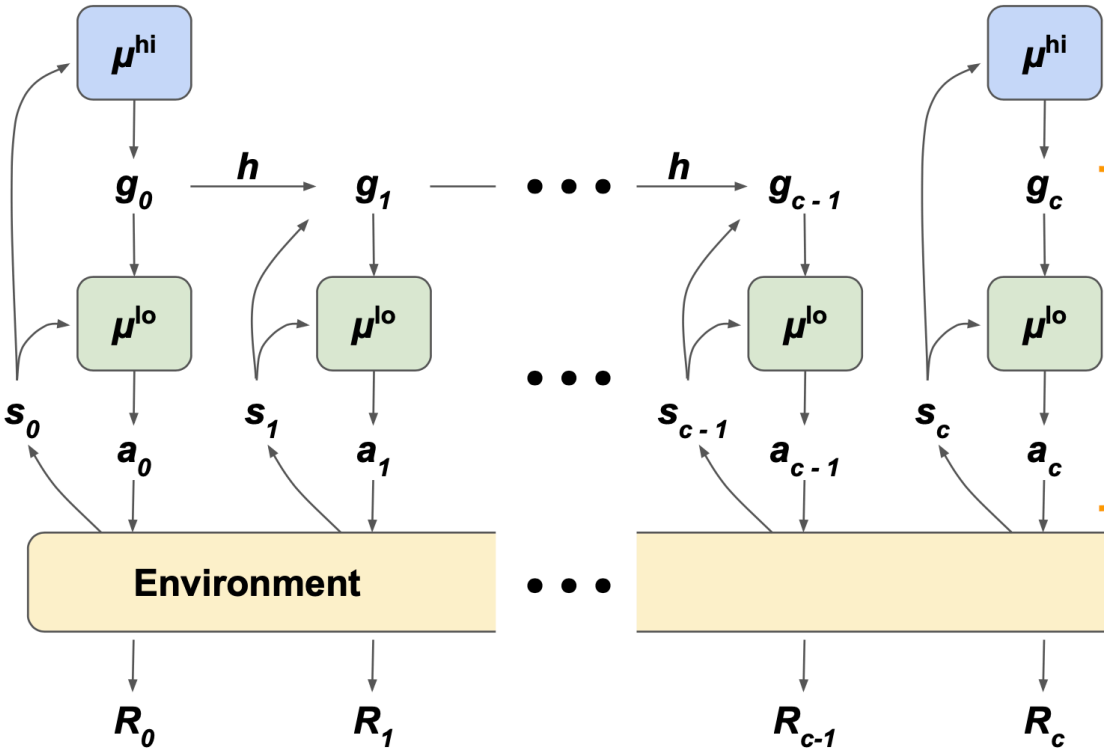


有一个智能体 Agent 和一个环境，智能体通过当前环境的状态 s_t 做出一个行动 a_t，环境给出下一个状态 s_{t+1} 和这一个行动得到的收益 r_t，重复上述过程，智能体通过一些算法（比如说 Q-learning）对这些样本元组 (s_t, a_t, r_t, s_{t+1}) 进行学习。

然而，基础的 RL 方法并不能完美地应用于所有问题。首先是具有大的动作或状态空间的问题，基础的方法会产生维数灾难；然后是多时间段问题：游戏的一个时间段有一个时间段对应的目标，而基础的方法也很难适应这种情况。

分层强化学习 (HRL) 是一种训练多层策略在连续更高的时间和行为抽象层次上执行决策和控制的方法，长期以来一直被认为能够学习这些困难的任务。分层主要解决的是稀疏奖励的问题，实际的强化问题往往环境奖励很稀疏，再加上庞大的状态空间和动作空间组合，导致直接训练往往训练不出来。而我们人类在解决一个复杂问题时，往往会将其分解为若干个容易解决的子问题，分而治之，分层的思想正是来源于此。

分层强化学习的基本框架如下：



每个时间点 t_0 顶层策略 μ^{hi} 会根据当前状态 s_0 提出一个目标 g_0，底层策略 μ^{lo} 根据当前的状态 s_0 和目标 g_0 来输出动作 a_0，并且会获得一个内部激励 R_0 和下一个状态 s_1，其中奖励 R_0 和与目标状态的接近程度成正比，越接近目标，奖励 R_0 越大。下一个时间点 t_1，目标转移函数 h 会输出下一个时间点的目标 g_1，底层策略 μ^{lo} 再根据状态 s_1 和目标 g_1 来输出动作 a_1...顶层策略每隔 c 个时间步重新提出一个目标。

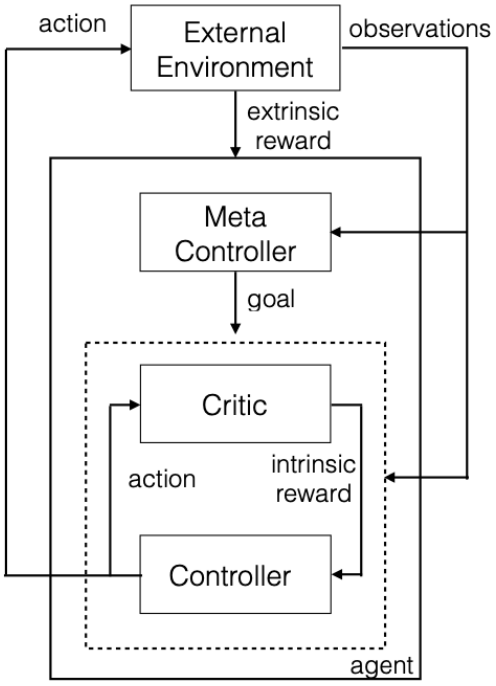
2 h-DQN

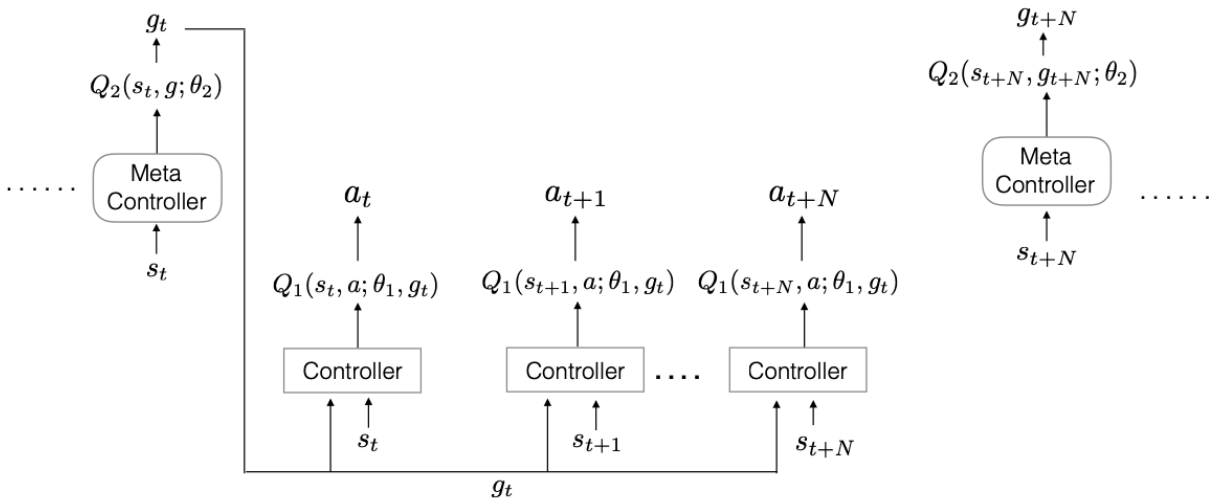
我调研的第一篇前沿论文，是 2016 年发表的 [Hierarchical Deep Reinforcement Learning : Integrating Temporal Abstraction andIntrinsic Motivation]

2.1 h-DQN 的模型

在著名的雅达利游戏“蒙特祖玛的复仇”中，玩家需要操控小人先去获得钥匙，然后才能打开通关的门。因此在这种游戏情况下，就有明显的多目标情况：想要通关就需要来到大门处，但想要打开大门就需要先去拿到钥匙。作者在论文中主要拿 DQN 来和自己的方法对比，并指出了 DQN 仍然存在的不足之处："Deep Q-Networks and its variants have been successfully applied to various domains including Atari games and Go, but still perform poorly on environments with sparse, delayed reward signals."

因此，作者 propose a framework with hierarchically organized deep reinforcement learning modules working at different time-scales. 模型分为两个层次结构：





(a) 顶层模块 (Meta-controller) 接收状态并选择一个新目标；它负责获取当前状态 s_t，然后从可能的子任务里面选取一个子任务 g_t 交代给下一个层级的 Controller 去完成。

(b) 低级模块 (Controller) 使用状态和选择的目标来选择操作，直到达到目标或事件结束；它负责接收上一个层级的子任务 g_t 以及当前的状态 s_t，然后选择一个或一系列可能的行动 a_t~a_{t+N} 去执行，直到达到目标或末端状态。当且仅当目标达到时，内部批评者 (Critic) 才会向 Controller 提供积极的奖励。

然后 Meta-controller 选择另一个目标，并重复步骤 (a-b)。

// 因此，h-DQN 的本质可以理解为双层 DQN。

2.2 h-DQN 的训练

论文使用 DQN 的框架来学习 Meta-controller 和 Controller 的策略。具体来说，Controller 的 Q 价值函数如下：

$$
\begin{aligned}
Q_1^*(s,a;g) &= \max_{\pi_{ag}} \mathrm{E}[\sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'} \mid s_t = s, a_t = a, g_t = g, \pi_{ag}] \\
&= \max_{\pi_{ag}} \mathrm{E}[r_t + \gamma \max_{a_{t+1}} Q_1^*(s_{t+1}, a_{t+1}; g) \mid s_t = s, a_t = a, g_t = g, \pi_{ag}]
\end{aligned} \tag{1}
$$

where g is the agent's goal in state s and $\pi_{ag} = P(a|s,g)$ is the action policy.

Similarly, for the Meta-controller, it has:

$$
Q_2^*(s,g) = \max_{\pi_g} \mathrm{E}[\sum_{t'=t}^{t+N} f_{t'} + \gamma \max_{g'} Q_2^*(s_{t+N}, g') \mid s_t = s, g_t = g, \pi_g] \tag{2}
$$

where N denotes the number of time steps until the controller halts given the current goal, g' is the agent's goal in state s_{t+N} , and $\pi_g = P(g|s)$ is the policy over goals.

之后是定义 Q_1, Q_2 的损失函数：论文首先将 Q_1 的经验样本 $(s_t, a_t, g_t, r_t, s_{t+1})$ 和 Q_2 的经验样本 (s_t, g_t, f_t, s_{t+N}) 分别保存在两个经验池 D_1, D_2 中，然后定义损失函数：

$$
L_1(\theta_{1,i}) = E_{(s,a,g,r,s') \sim D_1}[(y_{1,i} - Q_1(s,a,\theta_{1,i},g))^2] \tag{3}
$$

$$
L_2(\theta_{2,i}) = E_{(s,g,f,s') \sim D_2}[(y_{2,i} - Q_2(s,\theta_{2,i},g))^2] \tag{4}
$$

where i denotes the training iteration number and $y_{1,i} = r + \gamma \max_{a'} Q_1(s', a'; \theta_{1,i-1}, g)$, $y_{2,i} = r + \gamma \max_{a'} Q_2(s'; \theta_{2,i-1}, g)$

需要特别提出的是，论文并没有直接给出 $L_2(\theta_{2,i})$ 的计算公式，我凭借自己的理解写下了其公式，可能会有谬误。

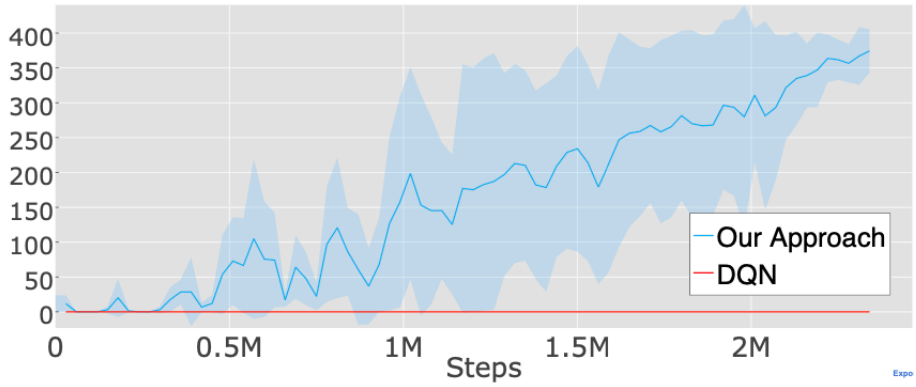
之后便使用神经网络常见的随机梯度下降方式对 DQN 进行训练和参数优化。

Algorithm 1 Learning algorithm for h-DQN

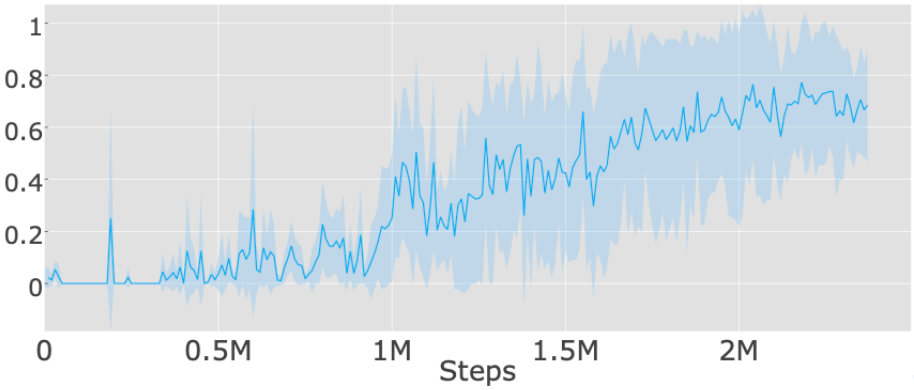
Initialize experience replay memories $\{\mathcal{D}_1, \mathcal{D}_2\}$ and parameters $\{\theta_1, \theta_2\}$ for the controller and meta-controller respectively.
Initialize exploration probability $\epsilon_{1,g} = 1$ for the controller for all goals g and $\epsilon_2 = 1$ for the meta-controller.
for $i = 1, num\_episodes$ **do**
    Initialize game and get start state description s
    $g \leftarrow \text{EPSGREEDY}(s, \mathcal{G}, \epsilon_2, Q_2)$
    while s is **not** terminal **do**
        $F \leftarrow 0$
        $s_0 \leftarrow s$
        while not (s is terminal **or** goal g reached) **do**
            $a \leftarrow \text{EPSGREEDY}(\{s, g\}, \mathcal{A}, \epsilon_{1,g}, Q_1)$
            Execute a and obtain next state s' and extrinsic reward f from environment
            Obtain intrinsic reward $r(s, a, s')$ from internal critic
            Store transition $(\{s, g\}, a, r, \{s', g\})$ in $\mathcal{D}_1$
            $\text{UPDATEPARAMS}(\mathcal{L}_1(\theta_{1,i}), \mathcal{D}_1)$
            $\text{UPDATEPARAMS}(\mathcal{L}_2(\theta_{2,i}), \mathcal{D}_2)$
            $F \leftarrow F + f$
            $s \leftarrow s'$
        end while
        Store transition (s_0, g, F, s') in $\mathcal{D}_2$
        if s is **not** terminal **then**
            $g \leftarrow \text{EPSGREEDY}(s, \mathcal{G}, \epsilon_2, Q_2)$
        end if
    end while
    Anneal ϵ_2 and adaptively anneal $\epsilon_{1,g}$ using average success rate of reaching goal g.
end for

2.3 h-DQN 的效果和不足之处

效果：论文在雅达利游戏“蒙特祖玛的复仇”中进行测试，由下图可以看出，h-DQN 能在不断的迭代训练中获得更高的分数（而普通的 DQN 基本训练不出任何效果），同时 reaching the goal 'key' 的成功率也在逐步上升。



(a) Total extrinsic reward



(b) Success ratio for reaching the goal 'key'

不足：论文对特定的情境进行了人为的设置，比如说论文在雅达利游戏"蒙特祖玛的复仇"中，人为设置了 Critic，判断规定为"小人是否到达某个位置"这样的 yes-or-no 的判断条件。这会使得模型依赖人为知识设定，对于其他任务并不是普遍适用。

同时，由于这是 2016 年的论文，在当时的前沿研究基本就只有 DQN，因此论文只能与其进行对比。而当后几年更多的专家研究出更好的分层强化学习模型之后，h-DQN 也只能沦为背景板。

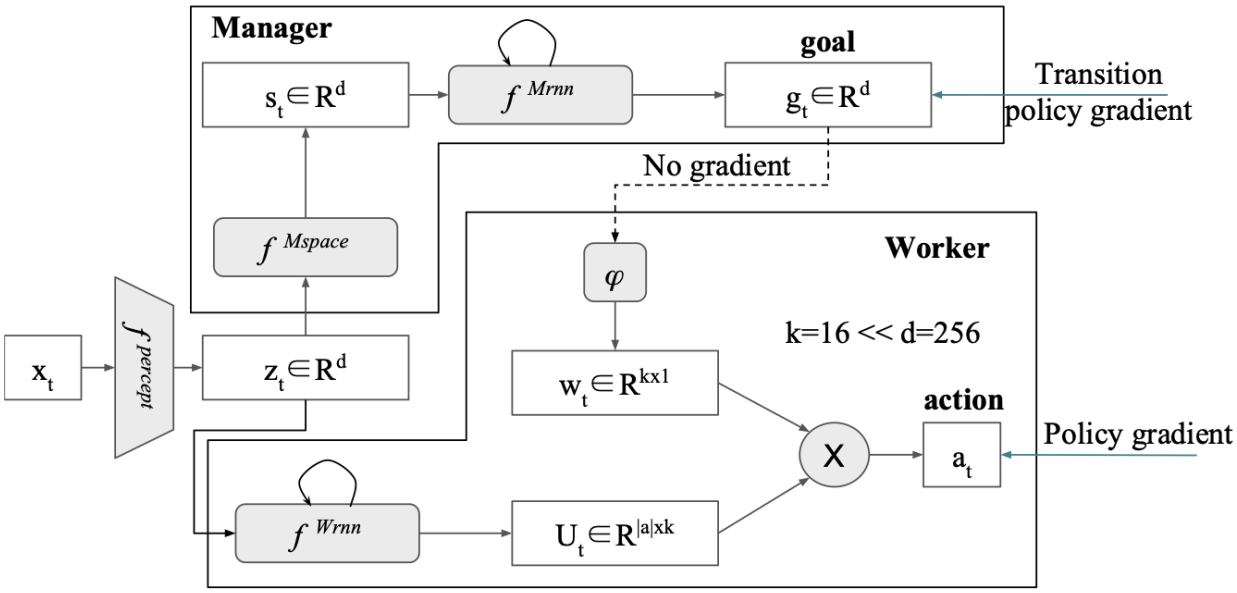
3 FuN

我调研的第二篇前沿论文，是 2017 年发表的 FeUdal Networks for Hierarchical Reinforcement Learning

3.1 FuN 的模型

FuN 的模型是基于 1993 年的论文 Feudal Reinforcement Learning，该论文的思想来源于封建等级制度，将控制层次分为三个等级，当前层为 manager，当前层的上一层为 super-manager，当前层的下一层为 sub-manager，属于典型的 multi-level control。论文还提出两个原则：(1) reward hiding，即只要满足 manager 即奖励，不论是否满足 super-manager；(2) information hiding，即 sub-manager 无需知道 super-manager 给 manager 定的目标，以及 super-manager 无需知道 manager 怎么做的。

上面这篇论文的意义在于提出了多层次任务划分的思想，而 FuN 的主要工作就在于将其用于分层强化学习：FuN 也由两个模块组成 - the Worker and the Manager。Manager 在内部计算一个潜在状态表示 s_t 并输出一个目标向量 g_t。Worker 根据外界观察、自己的状态和 Manager 的目标产生行动。具体的模型结构如下：



模型首先将输入 x_t 通过 CNN 卷积神经网络 ($f^{percept}$) 从高维图片转换为低维向量 z_t。z_t 传入 Manager 模块，经过一个全连接层 (f^{Mspace}) 成为 Manager 的状态空间（state space） s_t，之后经过作者精心设计的循环神经网络 Dilated LSTM 得到 Manager 层为 Worker 层设立的目标 g_t，再经过对近 c 步的 goal 进行线性变换 ϕ 得到传给 Worker 层的目标 w_t；同时 z_t 传入 Worker 模块，经过一个常见的循环神经网络 LSTM 得到 Worker 层的变换矩阵 U_t（即最后的神经网络 X 的系数矩阵），最后将其与目标 w_t 结合，经过最后的神经网络 X 得到最后的动作 a_t。

$$z_t = f^{percept}(x_t) \tag{5}$$
$$s_t = f^{Mspace}(z_t) \tag{6}$$
$$h_t^M, \hat{g}_t = f^{Mrnn}(s_t, h_{t-1}^M);\ g_t = \hat{g}_t / ||\hat{g}_t||; \tag{7}$$
$$w_t = \phi(\sum_{i=t-c}^{t} g_i) \tag{8}$$
$$h_W, U_t = f^{Wrnn}(z_t, h_{t-1}^W) \tag{9}$$
$$a_t = SoftMax(U_t w_t) \tag{10}$$

3.2 FuN 的训练

对于分层强化学习，论文也采取了对 Manager 层和 Worker 层进行分开训练的方式。对于 Manager 层，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。论文规定 goal 是学习到的低维状态表示空间中的方向："We propose instead to independently train Manager to predict advantageous directions (transitions) in state space and to intrinsically reward the Worker to follow these directions."

而训练 Manager 层的方法就是论文设计的 transition policy gradient，这种方法把每个连续的 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

$$\pi^{TP}(s_{t+c}|s_t) = p(s_{t+c}|s_t, \mu(s_t, \theta)) \tag{11}$$

对应产生的策略梯度公式就是

$$\nabla_\theta \pi_t^{TP} = E[(R_t - V(s_t))\nabla_\theta \log p(s_{t+c}|s_t, \mu(s_t,\theta))] \tag{12}$$

又存在 $p(s_{t+c}|s_t, g_t) \propto \exp d_{\cos}(s_{t+c} - s_t, g_t)$，因此上式可以转化为

$$\nabla g_t = A_t^M \nabla_\theta d_{\cos}(s_{t+c} - s_t, g_t(\theta)) \tag{13}$$

其中 $A_t^M = R_t - V_t^M(x_t, \theta)$ 是 Manager 的优势函数，$d_{\cos}$ 是余弦相似度。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。

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors θ and θ_v and global shared counter $T = 0$
// Assume thread-specific parameter vectors θ' and θ'_v
Initialize thread step counter $t \leftarrow 1$
repeat
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
  $t_{start} = t$
  Get state s_t
  repeat
    Perform a_t according to policy $\pi(a_t|s_t;\theta')$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t+1$
    $T \leftarrow T+1$
  until terminal s_t **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
  for $i \in \{t-1, \ldots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
    Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial\,(R - V(s_i;\theta'_v))^2/\partial\theta'_v$
  end for
  Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

而对于 Worker 层，首先也是定义其 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，论文的定义是

$$r_t^I = 1/c \sum_{i=1}^{c} d_{\cos}(s_t - s_{t-i}, g_{t-i}) \tag{14}$$

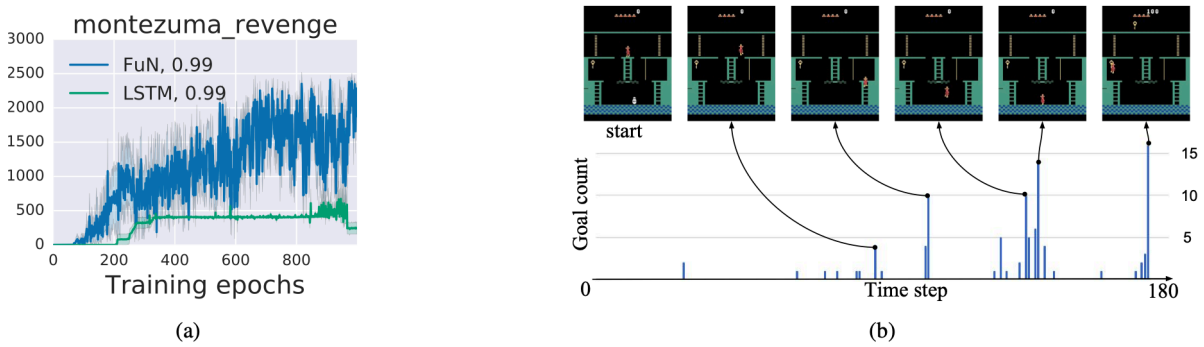
对应产生的策略梯度公式就是

$$\nabla_{\pi_t} = A_t^D \nabla_\theta \log \pi(a_t|x_t;\theta) \tag{15}$$

其中 $A_t^D = (R_t + \alpha R_t^I - V_t^D(x_t;\theta))$ 是 Worker 的优势函数。接下来就可以用策略梯度类方法来训练 Worker 了，文章中也是使用 A3C 方法。

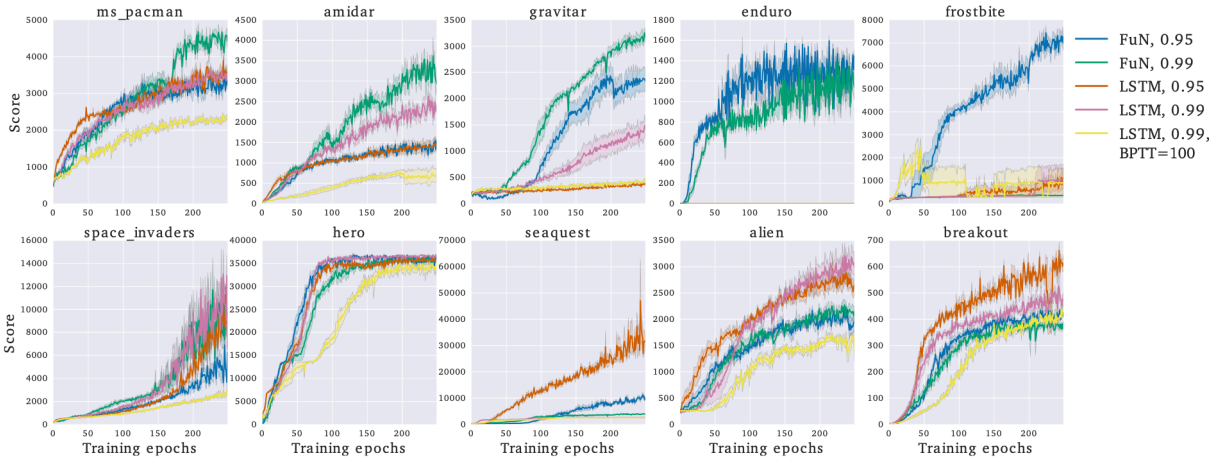
3.3 FuN 的效果和不足之处

效果：论文中所做的实验主要的目的是证明 FuN 模型能够学习到重要的、有帮助的、可解释的子策略和子目标："The goal of our experiments is to demonstrate that FuN learns non-trivial, helpful, and interpretable sub-policies and sub-goals"，同时也进行和基线“LSTM 模型”的实验对比，结果如下：



上图为在 Montezuma's Revenge 这个游戏上的实验：左图为 FuN 和基准 LSTM 的表现；右图的 Goal 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 count 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。可以看出，这些帧基本上都是拿到钥匙的关键路径。

不足：对于某些特定的游戏（如 space_invaders, hero, seaquest），FuN 表现和基准 LSTM 差别不大，甚至还劣于基准 LSTM。这说明 in these games long-term credit assignment is not important and the agent is better off optimising more immediate rewards in a greedy fashion.



4 HIRO

我调研的第三篇前沿论文，是 2018 年发表的 [Data-Efficient Hierarchical Reinforcement Learning]

4.1 HIRO 的模型

分层强化学习（Hierarchical Reinforcement Learning，HRL）方法，是传统强化学习（RL）方法的一种扩展，然而自从提出之后就存在着 3 个重要的难点：怎么训练低层策略来感应语义存在不同的行为；怎么定义高层策略的动作；怎么训练多个策略，在不过度收集数据的情况下。

因此目前大多数 HRL 方法都需要特定于任务的设计和**策略** on-policy 训练，这使得它们难以应用到现实场景中。而这篇论文提出了一种新的模式：HIRO，一种将 off-policy 算法应用在 HRL 框架上的方法，使得 HRL 能变得通用、无需人为设置繁杂的假设情况、数据得以重复和高效利用。因此论文的重点部分在于：**在顶层策略训练过程中使用 off-policy correction**

在之前 HRL 的框架中，顶层策略只能用 on-policy，若使用 off-policy 则会引起该策略的不稳定，这个不稳定问题是由 off-policy 对历史经验数据的重复使用造成的：站在 μ^{hi} 的角度，在状态 s_t 下，提出目标 g_t，得到的奖励是 $\sum R_{t:t+c-1}$，下一个状态是 s_{t+c}。这里的下一个状态 s_{t+c} 是经过 μ^{lo} 与环境互动 c 步后得到的，所以它与 μ^{lo} 输出的动作 a 有关。所以，对于 μ^{hi} 来说，μ^{lo} 也算是环境的一部分，因为

它会影响 μ^{hi} 的下一个状态 s_{t+c}。

然而，μ^{lo} 是不断在更新的，所以，在同样的状态 s_t 下，μ^{hi} 提出同样的目标 g_t，得到的奖励 $\sum R_{t:t+c-1}$ 以及下一个状态 s_{t+c} 是不断在变化的，而且是没有规律的，完全是随机的。因此如果只通过普通的 off-policy 可能完全学不到任何经验。

因此，这篇论文提出了 off-policy correction：既然 μ^{hi} 在 s_t 下提出目标 g_t，由于 μ^{lo} 的不断更新，下一个状态不再转移为 s_{t+c}，变得无规律性。那么如果这个时候 μ^{hi} 提出的是另一个目标 $\tilde{g}_t$，这个 $\tilde{g}_t$ 使得当前最新的 μ^{lo} 策略采取和之前未更新时相同的动作 a，那么在 c 步中，low-level policy μ^{lo} 与环境的互动关系保持不变，如此一来，s_{t+c} 也就不会改变了。

那么，下一步就是去寻找这个 $\tilde{g}_t$。论文的方法是：以 $s_{t+c} - s_t$ 为高斯采样中心，采取 8 个样本，然后加上这个采样中心 $s_{t+c} - s_t$ 以及原来 μ^{lo} 未更新时的 g_t，一共 10 个参考值。

论文选择的 $\tilde{g}_t$ 要使得 $\mu^{lo}_{new}(a_{t:t+c-1}|s_{t:t+c-1}, \tilde{g}_{t:t+c-1})$ 发生的概率最大，所以，将这 10 个参考值代入下式进行计算：

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

取获得上式值最大的那个参考值作为 $\tilde{g}_t$。

4.2 HIRO 的训练

由于论文中采取了 off-policy correction，因此采取了对应的 TD3 (Twin Delayed Deep Deterministic Policy Gradient) 方法，而论文并没有对此进行介绍，因此我参考了网上的各种博客对其的介绍，并在此讲述一下我的理解。

首先 TD3 方法基于 DDPG (Deep Deterministic Policy Gradient)，首先 DQN 用神经网络代替 Q 表格，loss 函数就是神经网络当前的输出与 target 之间的差距，然后对损失函数求导更新网络参数。而 target 的计算方式为 $r + \gamma \max_{a'} \hat{q}(s', a', w)$，即从下一个状态使用 max 函数选一个最大的动作 Q，显然 max 的操作不能处理连续动作的情况，因此 DDPG 采取了另一个神经网络来代替 $\max_{a'} \hat{q}(s', a', w)$。

DDPG 采取 Actor-Critic 方法对这个神经网络进行参数更新，同时再加上 DQN 自带的经验回放与双网络结构。而由于 DDPG 加入了噪声 ϵ，所以真实情况下，有误差的动作价值估计的最大值通常会比真实值更大。而在 TD3 算法中，它使用两个 Critic 网络来评估 Q 值，然后选取较小的那个网络的 Q 值来更新，这样就可以缓解 Q 值高估现象。同时也采用了延迟 Actor 网络更新和目标策略的平滑正则化 Target Policy Smoothing Regularization 方案，对 TD3 的结果进行优化。

Algorithm 1 TD3

Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta_1' \leftarrow \theta_1$, $\theta_2' \leftarrow \theta_2$, $\phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ **to** T **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
  Store transition tuple (s, a, r, s') in $\mathcal{B}$

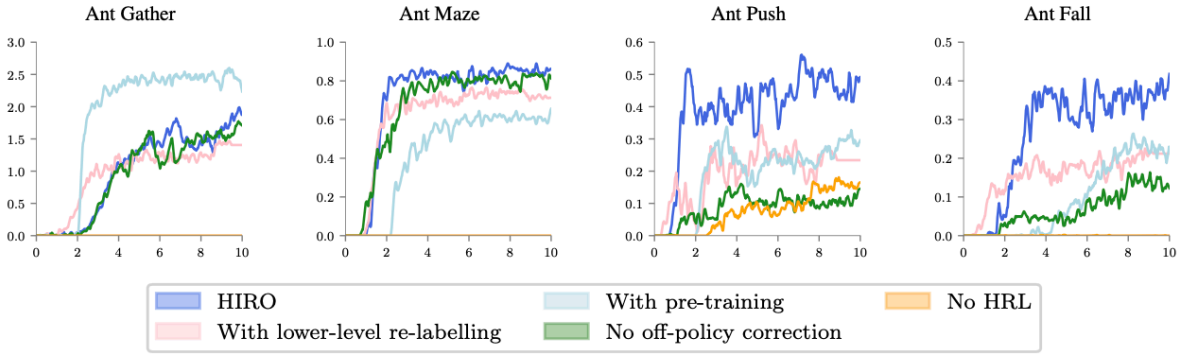
  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta_i'}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
  if t mod d **then**
    Update ϕ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta_i' \leftarrow \tau\theta_i + (1 - \tau)\theta_i'$
    $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
  end if
end for

4.3 HIRO 的效果和不足之处

效果：论文在 Ant Gather、Ant Maze、Ant Push、Ant Fall 几个环境中验证了 HIRO 与其他层次强化学习方法如 Feudal Network、VIME 等，最后得到了该算法的效果远好于其他几个算法。

	Ant Gather	**Ant Maze**	**Ant Push**	**Ant Fall**
HIRO	**3.02±1.49**	**0.99±0.01**	**0.92±0.04**	**0.66±0.07**
FuN representation	0.03 ± 0.01	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
FuN transition PG	0.41 ± 0.06	0.0 ± 0.0	0.56 ± 0.39	0.01 ± 0.02
FuN cos similarity	0.85 ± 1.17	0.16 ± 0.33	0.06 ± 0.17	0.07 ± 0.22
FuN	0.01 ± 0.01	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
SNN4HRL	1.92 ± 0.52	0.0 ± 0.0	0.02 ± 0.01	0.0 ± 0.0
VIME	1.42 ± 0.90	0.0 ± 0.0	0.02 ± 0.02	0.0 ± 0.0

其次论文也在几个环境中做了基础 HIRO 及其各种变体（Ablative Analysis）的测试，以理解其 HIRO 模型各种设计的重要性：



不足：该模型对 lower-level re-labelling 和 pre-training 支持不足：在其他论文中，提到 lower-level re-labelling 这种技术允许底层策略使用关于特定目标 g 收集的经验，用于学习关于任何备选目标 $\tilde{g}$ 的行为，而在本篇论文的实验中加入 With lower-level re-labelling 后，虽然训练前期学习速度显著，但是它的表现很快就趋于稳定，最终的结果也没有基础 HIRO 好。论文并没有对该问题进行深究，而只是简单地提出 lower-level re-labelling 的好处需要更多的研究，并鼓励未来的工作来研究更好的方法来利用它的好处。

而对于 pre-training，这是一种避免顶层策略训练过程中使用 off-policy 产生非平稳问题的方法，加入这种方法后，在简单的场景如 Ant Gather 中表现更好，而在困难的场景中则产生负收益。论文对其的解释为："这表明我们的 off-policy correction 仍然不完美， 通过改进它可以获得潜在的重大好处。"

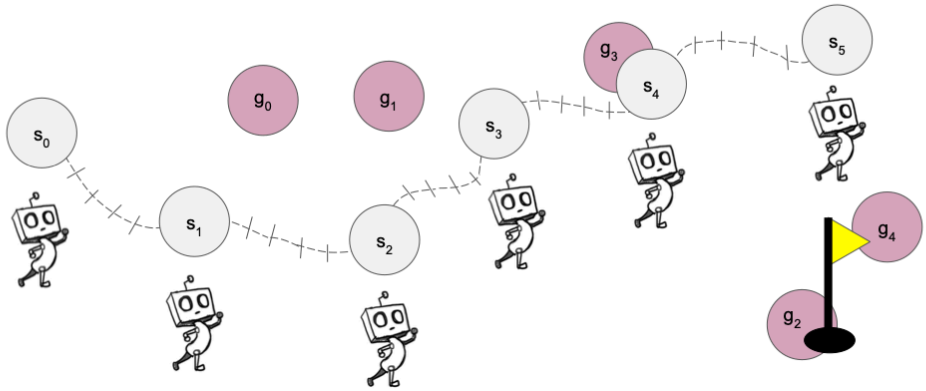
5 HAC

我调研的最后一篇前沿论文，是 2019 年发表的 Learning Multi-Level Hierarchies with Hindsight

5.1 HAC 的原理

分层强化学习在学习的过程中，存在着这样的一个问题：在层次结构中如果某一层次的策略发生了变化，可能会导致更高层次策略的过渡和奖励部分的变化，这称为 non-stationary 现象。分层强化学习出现 non-stationary 原因主要有两个方面：一、智能体在 H 个时间步到达的状态取决于下一层策略，上层在不同阶段提出相同的子目标，最终可能到达不同的状态，导致只有最底层的策略是稳定的。二、每当底层策略使用探索型的策略去完成子目标的时候，顶层的 transition and reward functions 可能会发生变化。这使得共同学习多个层次的策略变得困难。

对于这一问题，该论文提出了一个新的模型 Hierarchical Actor-Critic (HAC)，HAC 的基本结构为三层，高层读取环境状态 s_t，每隔 H 个时间步提出子目标 g_t，低层接收目标然后进行 H 步操作，每一步输出一个行动 a_{t+i}，$i \in [0, H)$，环境也将变为 s_{t+i+1}。当达到子目标或者执行了 H 步之后，高层继续提出新的子目标...而 HAC 的特别之处，就在于三个 transitions：Hindsight action transitions，Hindsight goal transitions，Subgoal testing transitions。



Hindsight action transitions：模拟底层使用最优策略时的顶层策略所用的 transitions，有两大关键部分组成：第一部分是将事后实现的子目标状态作为 transition 中的 action，而不是最初提出的子目标状态。第二个关键部分是在所有子目标层面使用的奖励函数（reward function）。这种奖励函数的第一个要求是，它应该激励最短路径实现目标，因为较短的路径可以更快地学习。第二个要求是，它应该独立于较低层次的路径。论文中举了一个例子：智能体初始状态为 s_0，顶层策略提出子目标 g_0，执行 $H = 5$ 步之后到达状态 s_1。则此时顶层策略的 Highsight action transitions 为 [initial state=s0, action=s1, reward=-1, next state=s1, goal=yellow flag, discount rate=gamma]。同理，顶层策略第二个时间步的 transitions 为 [initial state=s1, action=s2, reward=-1, next state=s2, goal=yellow flag, discount rate=gamma]

Hindsight goal transitions：为了补充除顶层之外其它层的 transitions。让每层都能更有效地学习稀疏奖励（the sparse reward）任务，方法是利用 Hindsight Experience Replay 的思想。考虑论文中的例子，假如底层在执行 $H = 5$ 步产生 5 个 transition 为 [initial state = s0, action = joint torques, reward = -1, next state = first tick mark, goal = g0, discount rate = γ], … , [initial state=4th tick mark, action=joint torques, reward=-1, next state=s1, goal=g0, discount rate=γ]，之后利用 HER，将 goal 修改为 5 个时间段中的某个 next state（例如 s1），即 [initial state = s0, action = joint torques, reward = -1, next state = first tick mark, goal = s1, discount rate = γ], … , [initial state = 4th tick mark, action = joint torques, reward = 0, next state = s1, goal = s1, discount rate = 0]（注意这里也修改了 reward，当且仅当 next state = goal 时修改为 0）

Subgoal testing transitions：在 Hindsight action transitions 中假设low-level已经达到最优，那么仅靠 action transition 则 high level 只会生成靠近 target 的子目标，但这些子目标离初始位置很远，我们假设 low-level 能达到，实际上到不了。因此为了让 high level 生成切合实际的子目标，我们要时不时地检测这个 subgoal 是否可到达，如果到达不了，就修改这个地方的 Q 值。例如：[initial state = s0, action = s1, reward = -H, next state = s1, goal = g0, discount rate = γ]

因此，HAC 的本质也许是一种数据增强。

Algorithm 1 Hierarchical Actor-Critic (HAC)

Input:

- Key agent parameters: number of levels in hierarchy k, maximum subgoal horizon H, and subgoal testing frequency λ.

Output:

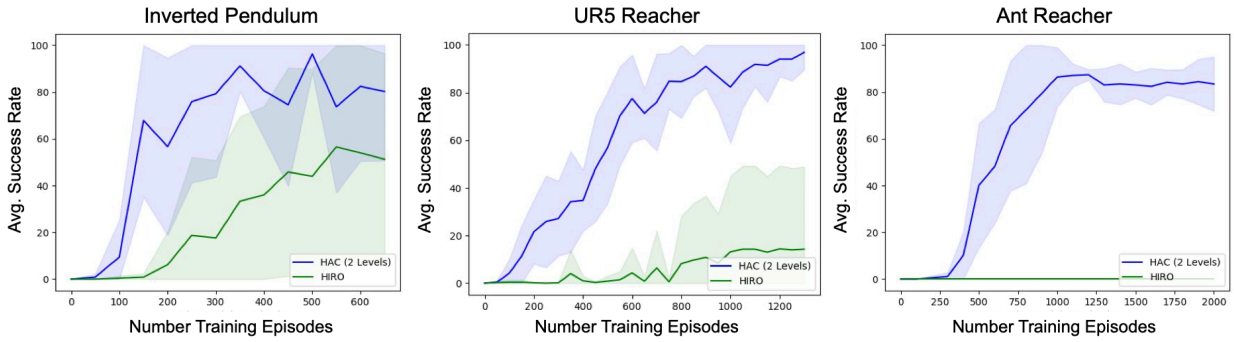
- k trained actor and critic functions $\pi_0, ..., \pi_{k-1}, Q_0, ..., Q_{k-1}$

for M episodes **do** ▷ Train for M episodes
  $s \leftarrow S_{init}$, $g \leftarrow G_{k-1}$ ▷ Sample initial state and task goal
  $train - level(k-1, s, g)$ ▷ Begin training
  Update all actor and critic networks
end for

function TRAIN-LEVEL($i :: level, s :: state, g :: goal$)
  $s_i \leftarrow s$, $g_i \leftarrow g$ ▷ Set current state and goal for level i
  for H attempts or until g_n, $i \leq n < k$ achieved **do**
    $a_i \leftarrow \pi_i(s_i, g_i)$ + $noise$ (if not subgoal testing) ▷ Sample (noisy) action from policy
    if $i > 0$ **then**
      Determine whether to test subgoal a_i
      $s_i^{'} \leftarrow train - level(i-1, s_i, a_i)$ ▷ Train level $i-1$ using subgoal a_i
    else
      Execute primitive action a_0 and observe next state $s_0^{'}$
    end if
    ▷ Create replay transitions
    if $i > 0$ and a_i missed **then**
      if a_i was tested **then** ▷ Penalize subgoal a_i
        $Replay\_Buffer_i \leftarrow [s = s_i, a = a_i, r = Penalty, s^{'} = s_i^{'}, g = g_i, \gamma = 0]$
      end if
      $a_i \leftarrow s_i^{'}$ ▷ Replace original action with action executed in hindsight
    end if
    ▷ Evaluate executed action on current goal and hindsight goals
    $Replay\_Buffer_i \leftarrow [s = s_i, a = a_i, r \in \{-1, 0\}, s^{'} = s_i^{'}, g = g_i, \gamma \in \{\gamma, 0\}]$
    $HER\_Storage_i \leftarrow [s = s_i, a = a_i, r = TBD, s^{'} = s_i^{'}, g = TBD, \gamma = TBD]$
    $s_i \leftarrow s_i^{'}$
  end for
  $Replay\_Buffer_i \leftarrow$ Perform HER using $HER\_Storage_i$ transitions
  return $s_i^{'}$ ▷ Output current state
end function

5.2 HAC 的效果

在论文中，作者将 HAC 模型和 HIRO 模型在三个游戏中进行测试，结果说明 HAC 不管是在训练速度和训练效果上都大大优于 HIRO。在论文中，作者也提及了 HIRO 的不足：(i) HIRO does not use Hindsight Experience Replay at either of the 2 levels and (ii) HIRO uses a different approach for handling the non-stationary transition functions. In other words, HIRO values subgoal actions with respect to a transition function that essentially uses the current lower level policy hierarchy, not the optimal lower level policy hierarchy as in our approach. Consequently, HIRO may need to wait until the lower level policy converges before the higher level can learn a meaningful policy.



6 总结

在这次的前沿工作调研报告中，我选择了“分层强化学习”这一主题，调研并阅读了近期的 4 篇论文。在课程中已经了解到 DQN 模型的基础之上，又学习到 h-DQN、FuN、HIRO、HAC 四种分层强化学习的模型，同时也大概了解了 AC、DDPG、TD3、A3C 这几种模型训练方法。当然，由于时间和自身能力的限制，我并没有实现对选定论文的复现，而调研的论文也不能说我实现了完全透彻的理解，这也是本次期末大作业我一个不足的地方。