【干货总结】分层强化学习(HRL)全面总结

原创 脆皮咕 深度强化学习实验室 昨天

深度强化学习实验室

来源：https://zhuanlan.zhihu.com/p/267524544
作者：脆皮咕(S.Q.Yang)
编辑：DeepRL

最近做分层强化学习的survey，系统地看了相关的经典论文，有老有新，将所看文章做个总结，由于做的项目涉及到很多分层相关，可能会长期研究这方面的内容，后续会不定期更新相关论文。其实有些论文读完也是云里雾里，包含了很多我的个人理解，会有一些错误和纰漏，希望各位不吝赐教，共同交流学习。

分层强化算是强化学习领域比较流行的研究方向，每年顶会论文中都有一定比例的分层论文。分层主要解决的是稀疏reward的问题，实际的强化问题往往reward很稀疏，再加上庞大的状态空间和动作空间组合，导致直接硬训往往训不出来，遇到头铁的agent更是如此。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。个人理解目前分层的解决手段大体分两种，一种是**基于目标**的(goal-reach)，主要做法是选取一定的goal，使agent向着这些goal训练，可以预见这种方法的难点就是如何选取合适的goal；另一种方式是**多级控制**(multi-level control)，做法是抽象出不同级别的控制层，上层控制下层，这些抽象层在不同的文章中可能叫法不同，如常见的option、skill、macro action等，这种方式换一种说法也可以叫做**时序抽象**(temporal abstraction)

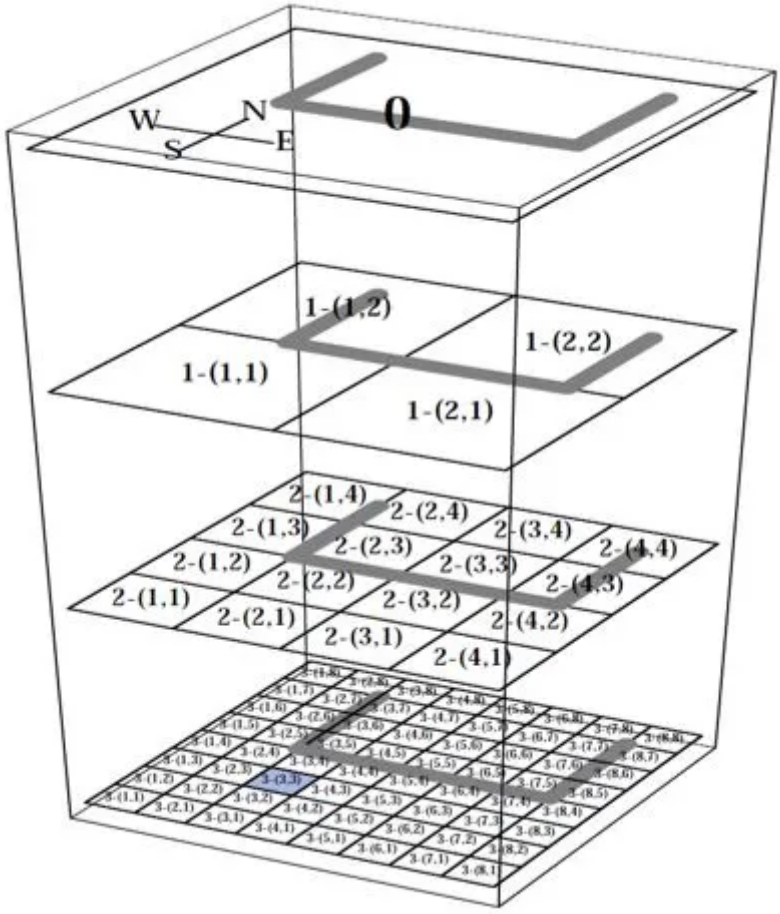
包含：**Feudal、HAM、MAXQ、Options、Option-Critic、A2OC、H-DRL、h-DQN、FuN、UVFA、HER、HAC、HIRO、Skill Chaining、Information-Constrained Primitives、DIAYN、DADS**

1、Feudal

Feudal Reinforcement Learning

比较古老的一篇论文，正如其名字所述，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。不同层次的控制遵循两个原则：(1)reward hiding，即只要满足manager即奖励，不论是否满足super-manager，通俗地说就是你只需要满足你的直属上级；(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，通俗地说就是底下干活的不用知道大领导和小领导之间的那些事，以及领导不管你咋干的只要干好就行，嗯，这可太封建官僚了，正如其名。

论文中给出的是迷宫导航问题，由于文章是92年发的，DRL还远远不成熟，因此都是使用Q-learning表格型解法来做，不同的是算Q的时候要以上级控制器的指令和下级控制器所处的位置为输入，体现层次关系。这篇文章的意义在于提出了多层次任务划分的思想，为以后的分层研究奠定了基础。



通过Feudal分层解决导航问题

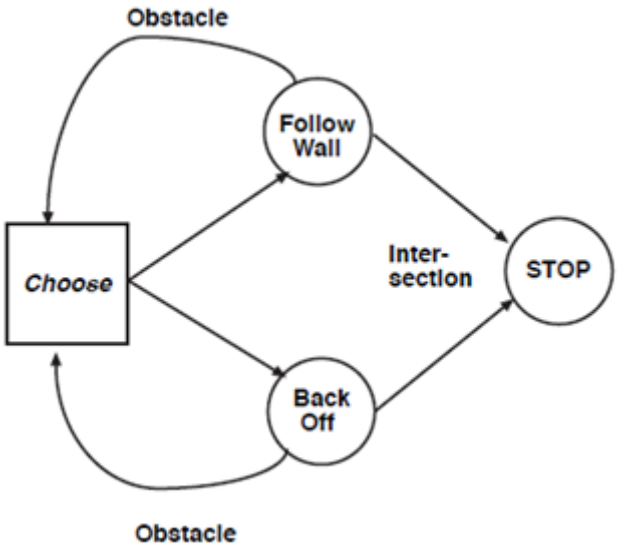
2、HAM

Reinforcement Learning with Hierarchies of Machines

HAM的意思是分层抽象机，核心思想是通过先验知识设计状态机，降低MDP复杂度，再求解简化MDP的最优策略，也是典型的multi-level control。这里引入了状态机的几种状态：(1)Action state: 与环境交互；(2)Call state: 调用其他状态机(与option类似)；(3)Choice state: 非确定地选择当前状态机的下一状态(需要学习的部分)；(4)Stop state: 终止当前状态机，返回到调用的地方。具体的学习算法也是Q-learning，不同的是将环境状态和状态机状态增广为新的状态空间，来学习使用当前状态机的哪种做法可以获得更好的reward。

本文的局限就是需要人工设计状态机，而这需要大量的先验领域知识，导致状态机设计十分复杂困难，在面对复杂问题时更是如此。其次本文的算法具有自上而下的call-and-return特点，即调用完一个状态机后就返回调用点，再继续后续的工作。

$$Q([s_c, m_c], a) \leftarrow Q([s_c, m_c], a) + \alpha[r_c + \beta_c V([t, n]) - Q([s_c, m_c], a)]$$



一个简单的规避障碍的状态机

3、MAXQ

Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition

本文的思想是对任务进行值函数分解，分解为多个子任务，每个子任务对应一个MDP。QMIX是对不同agent的Q进行分解，本文是对不同子任务进行Q分解，属于multi-level control。算法公式就是值函数的贝尔曼方程，一层一层向下分解就能写成所有子任务值的迭代和形式，可以把选取某个子任务看成传统的选择与环境互动的动作。优化方法和一般的Q-learning是一样的。

本文的局限是如何分解任务也是需要下一番功夫的，问题越难越不好分解，同时，本文的方法也具有call-and-return特点。

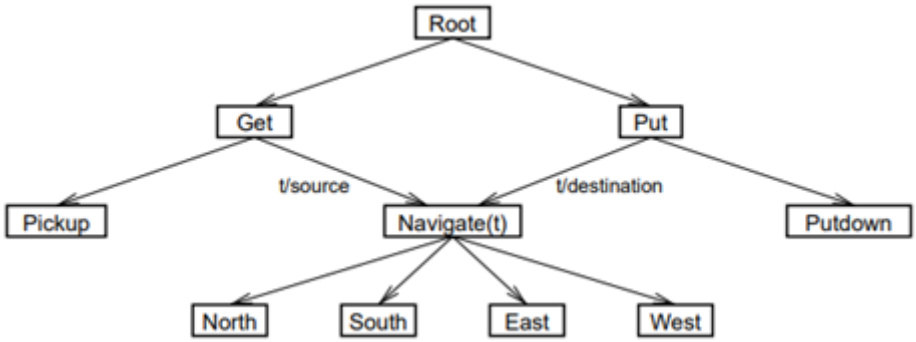
projected value function

$$V_i^{\pi}(s) = V_a^{\pi}(s) + \sum_{s'} P_i(s'|s,a) V_i^{\pi}(s')$$

completion function

$$C_i^{\pi}(s,a) = \sum_{s'} P(s'|s,a) V_i^{\pi}(s')$$

$$V_0^{\pi}(s) = V_{a_n}^{\pi}(s) + C_{a_{n-1}}^{\pi}(s, a_n) + \ldots + C_{a_1}^{\pi}(s, a_2) + C_0^{\pi}(s, a_1)$$



使用MAXQ的出租车接客任务分解

4、Options

Between MDPs and semi-MDPs：A framework for temporal abstraction in reinforcement learning

本文是Sutton在1999年发表的，本文的最大贡献就是提出了option的概念，并将temporal abstraction的思想和MDP与SMDP相结合，目前option是分层领域最主流的方法之一，十分值得阅读原文。这里不做具体的分析了，可以结合后面option-critic那篇文章的阅读笔记看。option相关的方法也属于multi-level control。

5、Option-Critic

The Option-Critic Architecture

本文将option的概念和AC框架相结合，使option方法应用于HRL成为可能，文中严格推导了相关的策略梯度定理，十分推荐仔细阅读并跟着原文推一遍，对理解MDP有很大帮助。下面是之前发的本文的读后笔记。https://zhuanlan.zhihu.com/p/217811126

6、A2OC

When Waiting is not an Option : Learning Options with a Deliberation Cost

这是option-critic的作者在之后又发的一篇option相关文章，主要思想是认为option的频繁切换会影响运行效果和效率，于是在option-critic的中断函数的策略更新中增加一个惩罚值，适当地降低option的切换频率。这个结论其实在option-critic的实验部分就已经提到了，理解上也很直观，很显然，然而经过一番包装也成了一篇改进文章。文中吃豆人实验的效果非常好，可以看出加了惩罚后，agent只在适当的拐弯处才切换option。

(a) Without a deliberation cost, options terminate instantly and are used in any scenario without specialization.

(b) Options are used for extended periods and in specific scenarios through a trajectory, when using a deliberation cost.

(c) Termination is sparse when using the deliberation cost. The agent terminates options at intersections requiring high level decisions.

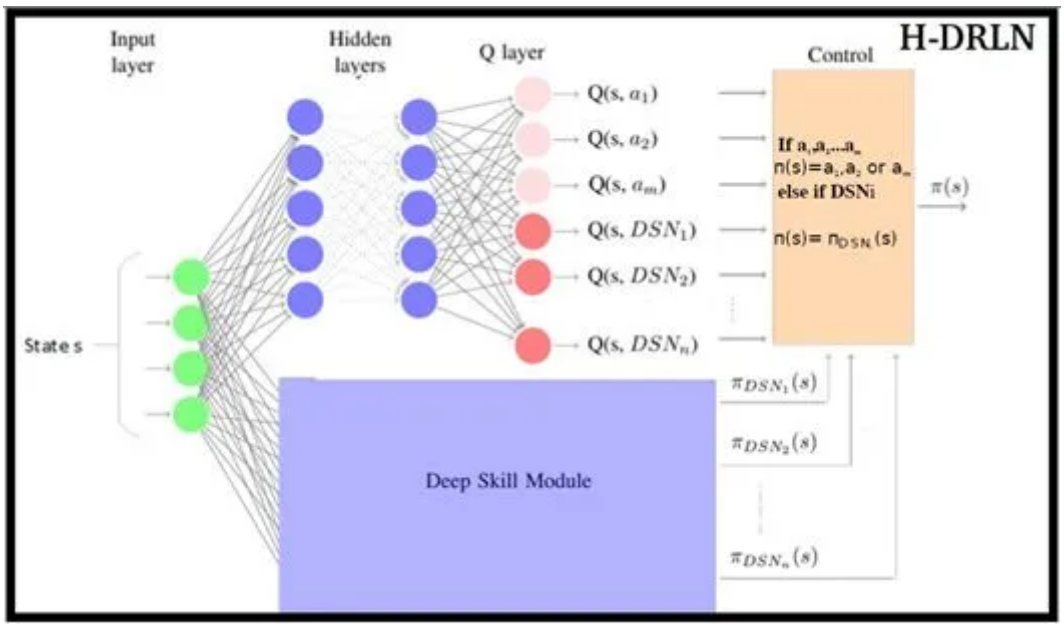
吃豆人实验

7、H-DRL

A Deep Hierarchical Approach to Lifelong Learning in Minecraft

本文以Minecraft作为实验环境，主打终身学习和策略蒸馏，因为Minecraft里有很多开放式的任务，十分适合skill相关的学习任务。文中提出了一种Deep skill network，就是一个能应对不同skill或task的综合网络，给出了两种实现形式，一种是针对每种skill或task，单独训练各自的DQN网络，用哪个就调哪个；另一种是几种skill或task共用一部分神经网络的隐层，再各自引出不同的输出头，通过策略蒸馏的方式使这个综合的大网络能更有普适性且易于部署使用。当然本算法不只支持skill的调用，也同时支持直接使用原始的action，不同的是调原始action的时候，只和环境互动一个step，调skill的时候会根据相应skill的需要和环境连续互动n个step。

本文的分层结构属于multi-level control，创新在将分层和策略蒸馏、终身学习等概念相结合，局限是其中某些skill可能需要预训练。



H-DRL的分层结构

8、h-DQN

Hierarchical Deep Reinforcement Learning : Integrating Temporal Abstraction andIntrinsic Motivation

Montezuma's Revenge应该是atari小游戏中最臭名昭著的了，在原始的DQNpaper中这个游戏在DQN下完全训不出来，得分为0，可见目前的强化学习在面对这种复杂的非反应式的闯关游戏时还是捉襟见肘，本文主要针对这个游戏环境做了相关的工作。本文将分层的层次划分为meta controller和controller，meta controller负责给出一个要达到的目标，接受的是环境反馈的稀疏的extrinsic reward，controller负责目标的具体实现，接受自定义的稠密的intrinsic reward。既然有目标，那么首要的问题就是如何选取目标，本文的做法是选取游戏画面中某些重要物体的图像作为目标，随state一起传入神经网络，当然这些目标图像都是人工选择好的。这也是本文的局限，就是针对Montezuma's Revenge这个环境做了不少工程化工作，很难有推广意义。

本文以某些goal为训练目标，因此属于goal-reach的范畴。

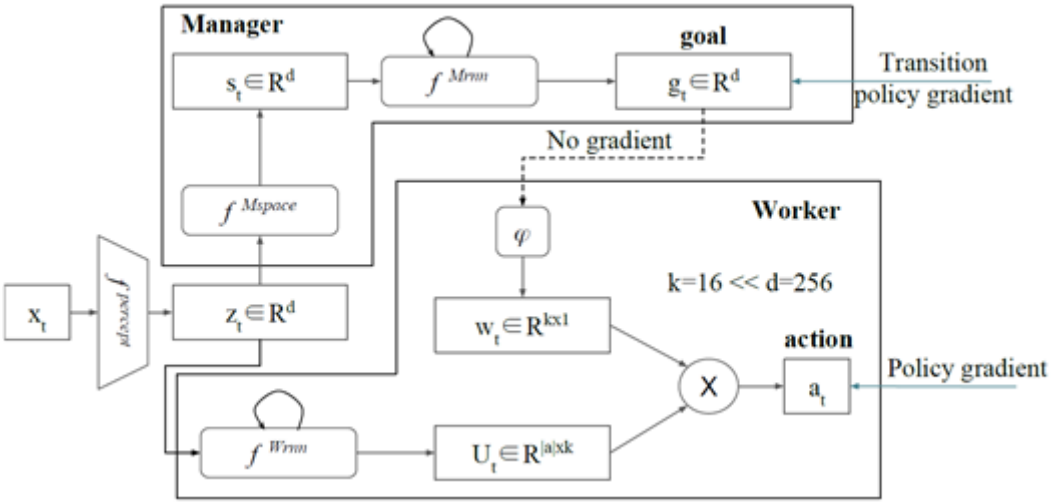
Montezumas Revenge

9、FuN

FeUdal Networks for Hierarchical Reinforcement Learning

之前介绍过Feudal，本文就是将Feudal的思想DRL化。将层次结构分为manager和worker，通过前面这些文章的总结，可以知道上层的控制器有两种控制选择：一种是选择不同的下层控制器来执行，比如option就是这种做法；另一种就是选取合适的goal来让下层控制器实现，上面的h-DQN就是这种。本文的做法是第二种，但创新在这个goal是自动生成的。文中将神经网络的隐层state作为goal，赋予这个隐变量的语义就是当前state在低维空间中所要改变的方向。worker的reward就顺理成章地变为前后两个step下state向量的变化方向与这个象征正确变化方向的goal的余弦相似度。

本文创新在goal实现了自动选取并赋予了很合理的语义解释。这里有一篇很好的分析可以一看。
https://zhuanlan.zhihu.com/p/46928498



FuN网络结构

10、UVFA

Universal Value Function Approximators

本文提出了一种统一的值函数，在原始的值函数 $V(s)$ 、 $Q(s,a)$ 基础上增加了goal作为输入变成 $V(s,g)$、$Q(s,a,g)$ ，这样值函数就变成在某一状态（或状态动作）某一目标下的价值。将给定状态和目标映射成对应的状态目标值的方法是，分别将状态和目标过自己的神经网络，拟合出对应的隐变量，然后将两个隐变量通过某种方式结合成一个标量就产生了对应的值，文中采用的方法是向量内积。在拟合指定状态和目标的隐变量时需要使用监督学习的方法，因此必然需要label，文中使用一种Hord of demons的方式可以产生不同目标对应的状态目标值，然后通过低秩分解的方式将其分解为状态对应的隐变量和目标对应的隐变量，于是label就有了。因此，整个UVFA的算法流程就是，通过某种方法获取各个状态目标对应的价值，然后通过低秩分解将这个值拆解为状态对应的部分和目标对应的部分作为label，再利用监督学习学习状态和目标对应隐变量的函数拟合，这样以后就可以通过任意状态和目标直接拟合出对应的隐变量，最后通过向量内积整合成一个状态目标值，起到泛化作用。
本文提出的统一值函数概念具有迁移意义，同时属于goal-reach范畴，但局限是文中提到比较困难的仍是goal如何选取，但本文并不打算讨论这个问题，因为本文的重点是提供这种考虑目标在内的广义值函数的概念，于是只是简单地选择某些state作为goal，可见goal如何合理选取将是这类分层问题的最大困难。

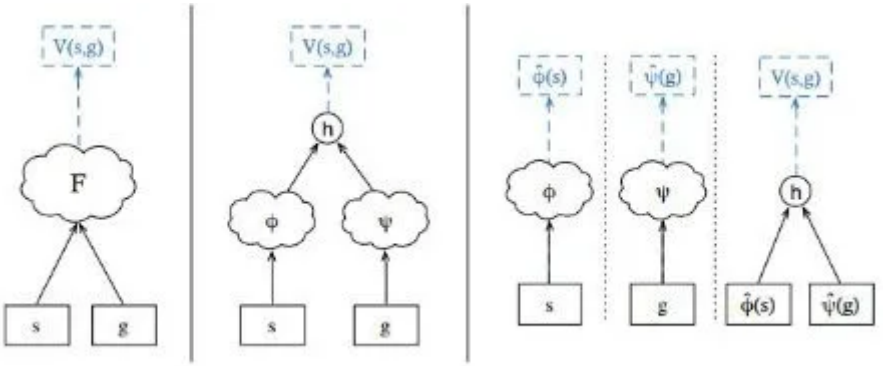


Figure 1. Diagram of the presented function approximation architectures and training setups. In blue dashed lines, we show the learning targets for the output of each network (cloud). **Left**: concatenated architecture. **Center**: two-stream architecture with two separate sub-networks ϕ and ψ combined at h. **Right**: Decomposed view of two-stream architecture when trained in two stages, where target embedding vectors are formed by matrix factorization (right sub-diagram) and two embedding networks are trained with those as multi-variate regression targets (left and center sub-diagrams).

统一值函数拟合架构

Algorithm 1 UVFA learning from Horde targets

Input: rank n, training goals $\mathcal{G}_T$, budgets b_1, b_2, b_3
Initialise transition history $\mathcal{H}$
for $t = 1$ **to** b_1 **do**
  $\mathcal{H} \leftarrow \mathcal{H} \cup (s_t, a_t, \gamma_{ext}, s_{t+1})$
end for
for $i = 1$ **to** b_2 **do**
  Pick a random transition t from $\mathcal{H}$
  Pick a random goal g from $\mathcal{G}_T$
  Update Q_g given a transition t
end for
Initialise data matrix $\mathbf{M}$
for $(s_t, a_t, \gamma_{ext}, s_{t+1})$ **in** $\mathcal{H}$ **do**
  for g **in** $\mathcal{G}_T$ **do**
    $\mathbf{M}_{t,g} \leftarrow Q_g(s_t, a_t)$
  end for
end for
Compute rank-n factorisation $\mathbf{M} \approx \hat{\boldsymbol{\phi}}^\intercal \hat{\boldsymbol{\psi}}$
Initialise embedding networks ϕ and ψ
for $i = 1$ **to** b_3 **do**
  Pick a random transition t from $\mathcal{H}$
  Do regression update of $\phi(s_t, a_t)$ toward $\hat{\boldsymbol{\phi}}_t$
  Pick a random goal g from $\mathcal{G}_T$
  Do regression update of $\psi(g)$ toward $\hat{\boldsymbol{\psi}}_g$
end for
return $Q(s, a, g) := h(\phi(s, a), \psi(g))$

收集transition

Horde of demons
获取不同goal的值函数

构造M矩阵

分解M得到监督信号

two-stream监督学习

重构值函数

11、HER

Hindsight Experience Replay

这一篇是建立在UVFA基础上的，hindsight是事后诸葛亮的意思，点明了本文算法的核心思想。本文解决的是reward稀疏和goal选取困难的问题，方法另辟蹊径，读完让人眼前一亮。由于reward稀疏，必然会导致很多条完整的transition都无法达到预设的goal，于是产生了很多失败的transition，虽然没法用于预设goal的训练，但我们可以对于每一条transition都去计算一下假如换做其他目标的话，该transition会变成什么样。

举例来说，假如有一条transition，其最后一个step并没有达到预设的goal，那么这一步将不会获得正的reward或者获得负的reward，但如果把goal改为已经达到的这个最后一步的state，那么这一步将获得正的reward或0reward，不止这一步，整条transition的goal都应修改为这个state，这就是hindsight的含义，通俗地说就是**“既然无法达到那个目标，那就把已经达到的当作目标吧”**。这么做合理的原因就在于，不管goal设置的是什么，都不会影响系统动力学，在相同的state下采取相同的action会达到相同的next state，因此你可以将这条已经产生的轨迹的goal重设为任何值。通过这种方式，在保留失败transition的基础上，样本池里的样本数量相对于以往会得到相应的扩充。

文中共提出了4种goal的重新选取方法：(1)每次只选择这一条轨迹的最后一个状态对应的目标作为新目标；(2)每次随机随机选择k个在这个轨迹上并且在这个transition之后的状态作为新目标；(3)每次选择k个在这个轨迹上的状态作为新目标；(4)每次在所有出现过的状态里面选择k个状态作为新目标(不同episode)。每种选择方法造成的样本扩展倍数不同，文中表示(2)的效果最好。

由于继承自UVFA，因此本文的方法属于goal-reach范畴，可以有效解决reward稀疏的问题，但是goal仍为一般的state，存在一定的局限性，不过这篇文章的点子还是让我感到很新奇。

Algorithm 1 Hindsight Experience Replay (HER)

Given:

- an off-policy RL algorithm $\mathbb{A}$, ▷ e.g. DQN, DDPG, NAF, SDQN
- a strategy $\mathbb{S}$ for sampling goals for replay, ▷ e.g. $\mathbb{S}(s_0, \ldots, s_T) = m(s_T)$
- a reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \to \mathbb{R}$. ▷ e.g. $r(s, a, g) = -[f_g(s) = 0]$

Initialize $\mathbb{A}$ ▷ e.g. initialize neural networks

Initialize replay buffer R

for episode = $1, M$ **do**

  Sample a goal g and an initial state s_0.

  for $t = 0, T-1$ **do**

    Sample an action a_t using the behavioral policy from $\mathbb{A}$:

    $a_t \leftarrow \pi_b(s_t||g)$ ▷ $||$ denotes concatenation

    Execute the action a_t and observe a new state s_{t+1}

  end for

  for $t = 0, T-1$ **do**

    $r_t := r(s_t, a_t, g)$

    Store the transition $(s_t||g, a_t, r_t, s_{t+1}||g)$ in R ▷ standard experience replay

    Sample a set of additional goals for replay $G := \mathbb{S}$(**current episode**)

    for $g' \in G$ **do**

      $r' := r(s_t, a_t, g')$

      Store the transition $(s_t||g', a_t, r', s_{t+1}||g')$ in R ▷ HER

    end for

  end for

  for $t = 1, N$ **do**

    Sample a minibatch B from the replay buffer R

    Perform one step of optimization using $\mathbb{A}$ and minibatch B

  end for

end for

12、HAC

Learning Multi-Level Hierachies with Hindsight

这一篇继承自HER，HRL中一般分多个层次进行控制，上下层同时训练的时候会出现non-stationary的问题，也就是下层的policy改变时会影响上层的transition，因此可能原来能达到某个goal，但现在达不到了，对学习造成很大影响。本文提出的做法是，假设下层的policy已经是最优，就不会因为下层policy的变化而影响上层，那么如何使还没有达到最优的下层policy看上去最优呢？通过HER来实现。

本文提出三种transition：(1)hindsight action transition，对于非最低层来说action就是选goal，或者说是选state，因此action space就是state space，如果上层设定的goal下层没达到，那就将已达到的state作为goal，相当于对上层的action做了hindsight，这样基于下层已经最优的假象进行上层的训练，从而暂时忽略掉下层policy的变化，这部分是对action的调整；(2)hindsight goal transition，这个就是原始的HER所干的事情，是对goal的调整，解决reward稀疏的问题；(3)subgoal testing transition，在上述两种调整下，agent用于更新的transition都是可到达的一些state，对于一些较远的目标，可能无法实现，因此以一定概率进行测试下层policy能否实现当前goal，当goal无法被下层完成时，即给出不切实际的遥远goal时，给以惩罚。

本文与UVFA和HER一脉相承，都属于goal-reach。

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** subgoal testing transition ▷ 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
    hindsight action transition ▷ 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 $s_i^{'}$ hindsight goal transition ▷ Output current state
end function

13、HIRO

Data-Efficient Hierarchical Reinforcement Learning

HIRO的全称是HIerarchical Reinforcement learning with Off-policy correction，本文关注off-policy下high/low level non-stationary的问题，off-policy本来就十分不稳定，很多算法采用很多技术才减弱了不稳定性，除此之外在HRL下也有它特有的不稳定性，就是上下层策略的不稳定性，HAC中是通过HER来解决这个问题的。

这篇文章的分层方式和FuN十分类似，都是上层产生goal，这个goal的语义是期望达到的状态与当前状态的残差，下层接受goal进行下层的控制。上层每隔c步输出一次，下层每步都输出，产生的样本放入样本池，这就导致在训练上层策略时，由于下层策略可能已经改变，不再是当初存这条样本时的样子，相同的

goal下对应的c步action序列和相应的的累积reward和当初存下来的不匹配，于是产生了non-stationary的问题。为了解决这个问题，文中提出一个做法，即固定已经产生的样本的奖励不变，但是去寻找在当前条件下给一个怎样的goal可以最大可能产生以前下层策略的c步动作序列和累积奖励，这看起来有一点最大似然的意思。文中给出一个近似公式，但也无法求解，于是通过采样从10个备选的goal中选一个能让下式最大的，其中两个特殊的是原始的goal g_t 和 $s_{t+c} - s_t$ ，另外8个通过以 $s_{t+c} - s_t$ 为中心的高斯分布采出来。

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

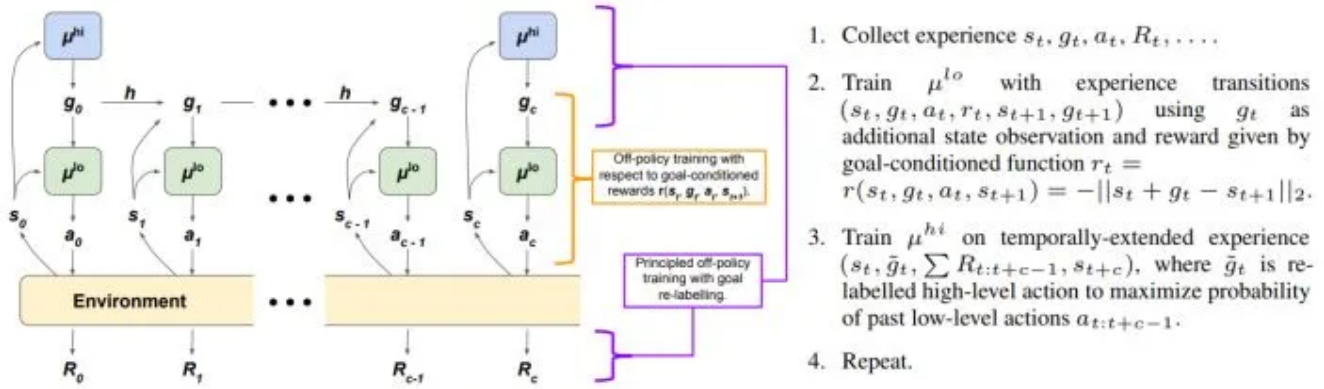


Figure 2: The design and basic training of HIRO. The lower-level policy interacts directly with the environment. The higher-level policy instructs the lower-level policy via high-level actions, or goals, $g_t \in \mathbb{R}^{d_g}$ which it samples anew every c steps. On intermediate steps, a fixed goal 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.

HIRO算法结构

14、Skill Chaining

Skill Discovery in Continuous Reinforcement Learning Domains using Skill Chaining

Option Discovery Using Deep Skill Chaining

这两篇属于skill discovery范畴，但本质还是分层，第二篇在第一篇基础上使用DRL来解决。这篇提出的创新点就是以往的option需要指定其数量，option的数量是一个超参，而这里无需指定数量，agent会自动学到需要多少option。自动学习option数量通过skill chaining来实现，skill chaining的思想就是，先确定在goal附近能通过某些途径达到goal的option，将其加入一个集合，再将这个option的initial set作为下一个需要学习的option的terminate state，这样不断从后往前推直到遇到全局的初始state，这样所有option连起来就形成了一条skill chain，可以解决goal的问题。虽然这些option是首尾相接的，但实际agent还是通过学习来决定选取哪一个option，并不是确定的选相连的下一个option。

本文有一个局限就是最开始需要找到一个或几个能达到goal的成功轨迹，然后才能有后续的工作，在openreview上我看有一个盲评评审也提到了这个问题。不过基于skill chaining的分层研究目前还不是太

多，看来还是有很多工作可以在这上面做。

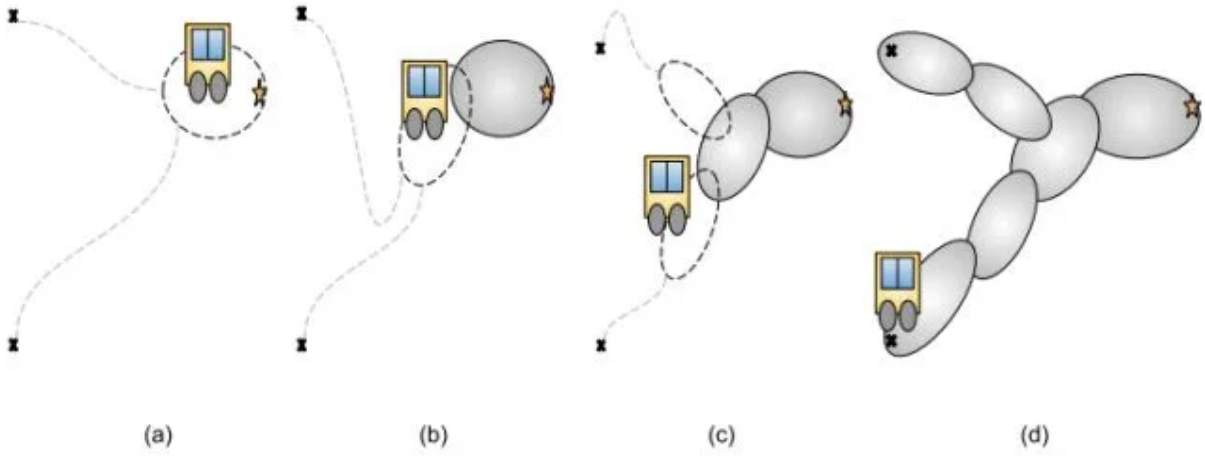


Figure 4: An illustration of the deep skill chaining algorithm. ⋆ represents the goal state, × represents the two start states. (a) Before the agent has discovered its first skill/option, it acts according to its global DDPG policy. Having encountered the goal state N times, the agent creates an option to trigger the goal from its local neighborhood. (b) Now, when the agent enters the initiation set of the first option, it begins to learn another option to trigger the first option. (c) Because the agent has two different start states, it learns two qualitatively different options to trigger the option learned in (b). (d) Finally, the agent has learned a skill tree which it can follow to consistently reach the goal.

15、Information-Constrained Primitives

Reinforcement Learning with Competitive Ensembles of Information-Constrained Primitives

一般的HRL都会分为两层，上层控制器通观全局决定下层控制器如何行动，但因为得啥状态都看，因此很不好学，于是这篇文章干脆将上层的控制器取消，只学习各个子策略，子策略自己决定从状态中获取多少信息来进行自己的行动，这样就去中心化了。这里的思想基于信息论中的信息瓶颈概念，子策略能从更关注的状态中获取更多的信息，因此能从环境状态中获取更多信息的策略更应该被选择。同时由于信息有限，需要对策略进行regularization以使用尽可能少的信息。这样就导致策略之间会相互竞争，并由于信息量的限制，使自己只关注自己专注的领域，实现不用上层控制器也能分工完成任务。

下面式子是需要最大化的目标函数，第一项代表策略获得的reward，与需要从当前state中获取信息的量成正比，信息量通过策略生成的隐变量与标准高斯分布的KL散度来衡量，这一项代表策略间的竞争；第二项是正则项，就是前面的KL散度，使当前策略不要无限制地从state中攫取信息；第三项是entropy的正则项，使各策略不要坍塌到一个策略上，保证多样性。

$$J_k(\theta) \equiv \mathbb{E}_{\pi_\theta}[r_k] - \beta_{\text{ind}}\mathcal{L}_k - \beta_{\text{reg}}\mathcal{L}_{\text{reg}}$$

最近的HRL尤其是skill discovery方面流行使用信息论的内容做创新，例如信息瓶颈、互信息等，以前没怎么接触过信息论，这篇论文看了好几遍才理解，还是需要恶补一下相关理论知识。

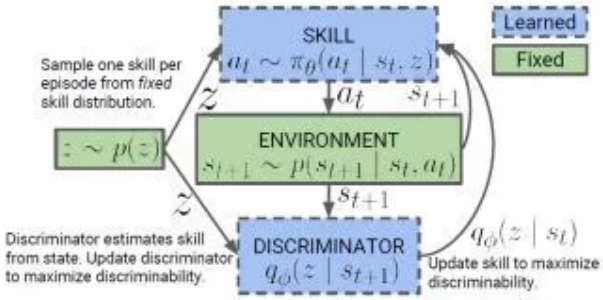
16、DIAYN

Diversity Is All You Need: Learning Skills Without A Reward Function

DIAYN的全称是Diversity is All You Need，这篇属于基于互信息的skill discovery，是目前比较火的研究方向，基于互信息的skill discovery的思路一般是基于某些互信息的目标函数学到skill的dynamic，或者说是skill与state的分布，这是无需环境reward支持的，然后将这些学好的skill用到其他应用中，比如作为hierarchical RL的下层policy，这篇文章也是这个思路。

本文主要提出了三个观点：（1）有用的skill可以用来控制agent访问到某些state，并且不同的skill导向的state不同，使得skill具有可分辨性；（2）通过state，而不是action，来分辨区分skill；（3）鼓励探索以保证skill具有足够的多样性，使得不同的skill之间具有可分辨性。因此定义的目标函数也对应了上述三个观点。将目标函数变形为变分下界，再将优化原目标变为优化下界，同时引入伪reward用于优化policy，最终学到skill和state的推断关系，这是基于互信息的skill discovery的常用做法。具体目标函数的组成可以看这篇分析。https://zhuanlan.zhihu.com/p/270017839

训练好的skill可以直接作为hierarchical RL的底层策略，在上层再学习一个meta controller用于skill的选择，由此可见，DIAYN通过互信息在无需reward的情况下学习到skill，再通过一个上层策略使用这些skill，这与以往的上下层策略一起通过环境reward训练是不同的思路。与之类似的skill discovery方法都是类似的做法。



Algorithm 1: DIAYN

while *not converged* **do**
  Sample skill $z \sim p(z)$ and initial state $s_0 \sim p_0(s)$
  for $t \leftarrow 1$ **to** $steps\_per\_episode$ **do**
    Sample action $a_t \sim \pi_\theta(a_t \mid s_t, z)$ from skill.
    Step environment: $s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)$.
    Compute $q_\phi(z \mid s_{t+1})$ with discriminator.
    Set skill reward $r_t = \log q_\phi(z \mid s_{t+1}) - \log p(z)$
    Update policy (θ) to maximize r_t with SAC.
    Update discriminator (ϕ) with SGD.

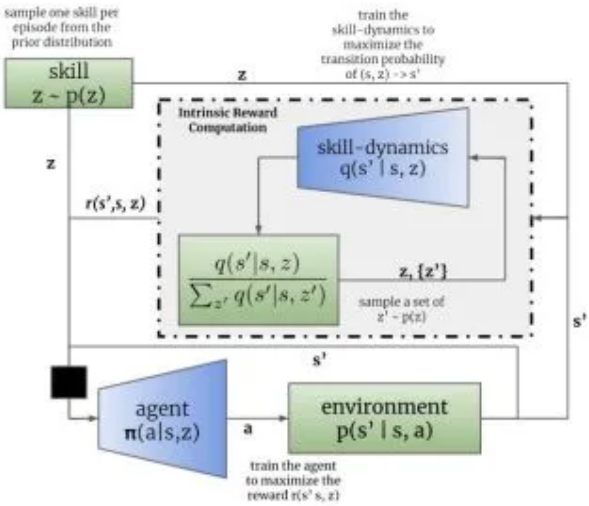
Figure 1: **DIAYN Algorithm**: We update the discriminator to better predict the skill, and update the skill to visit diverse states that make it more discriminable.

17、DADS

Dynamics-Aware Unsupervised Discovery of Skills

这是一篇ICLR2020的基于互信息的skill discovery论文，文章通过在没有外界reward支持的无监督学习条件下，通过发现具有可预测性的skill并学习skill的dynamic，获取多个专注不同领域的skill，然后通过模型预测控制(MPC)，直接基于已学好的模型做planning。本文整体的思路和DIAYN一致，都是通过最大化state和skill的互信息来学习skill dynamic，所不同的是本文在使用学到的skill时，采用了planning

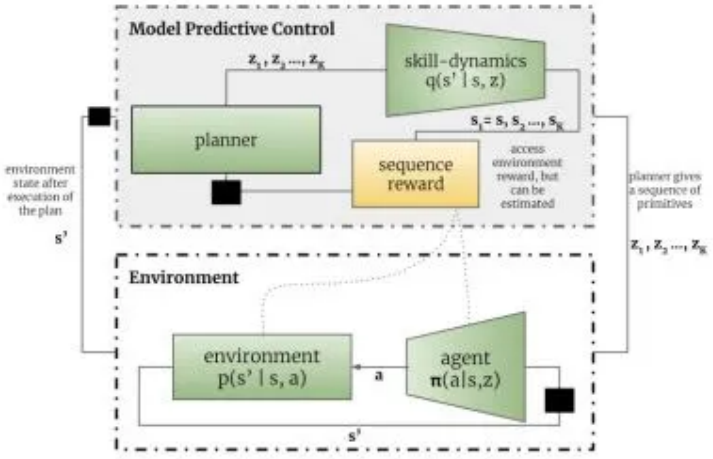
的做法，而没有像DIAYN一样又学习了一个上层的meta controller来选择调用不同的技能。具体的分析可以看下面链接。https://zhuanlan.zhihu.com/p/270147463



Algorithm 1: Dynamics-Aware Discovery of Skills (DADS)

Initialize π, q_ϕ;
while *not converged* **do**
  Sample a skill $z \sim p(z)$ every episode;
  Collect new M on-policy samples;
  Update q_ϕ using K_1 steps of gradient descent on M transitions;
  Compute $r_z(s, a, s')$ for M transitions;
  Update π using any RL algorithm;
end

Figure 2: The agent π interacts with the environment to produce a transition $s \rightarrow s'$. Intrinsic reward is computed by computing the transition probability under q for the current skill z, compared to random samples from the prior $p(z)$. The agent maximizes the intrinsic reward computed for a batch of episodes, while q maximizes the log-probability of the actual transitions of $(s, z) \rightarrow s'$.



Algorithm 2: Latent Space Planner

$s \leftarrow s_0$;
Initialize parameters $\mu_1, \ldots \mu_{H_P}$;
for $i \leftarrow 1$ **to** H_E/H_Z **do**
  for $j \leftarrow 1$ **to** R **do**
    $\{z_i, \ldots z_{i+H_P-1}\}_{k=1}^{K} \sim \mathcal{N}_i, \ldots \mathcal{N}_{i+H_P-1}$;
    Compute r_{env} for $\{z_i, \ldots z_{i+H_P-1}\}_{k=1}^{K}$;
    Update $\mu_i, \ldots, \mu_{i+H_P-1}$;
  end
  Sample z_i from $\mathcal{N}_i$;
  Execute $\pi(a|s, z_i)$ for H_Z steps;
  Initialize μ_{i+H_P};
end

Figure 3: At test time, the planner executes simulates the transitions in environment using skill-dynamics q, and updates the distribution of plans according to the computed reward on the simulated trajectories. After a few updates to the plan, the first primitive is executed in the environment using the learned agent π.

论文原文链接：https://github.com/YangShengqi/paper

完

总结1：周志华 || AI领域如何做研究-写高水平论文

总结2：全网首发最全深度强化学习资料(永更)

总结3：《强化学习导论》代码/习题答案大全

总结4：30+个必知的《人工智能》会议清单

总结5：2019年-57篇深度强化学习文章汇总

总结6：万字总结 || 强化学习之路

总结7：万字总结 || 多智能体强化学习(MARL)大总结

总结8：深度强化学习理论、模型及编码调参技巧

完

第87篇：165篇CoRL2020 accept论文汇总

第86篇：287篇ICLR2021深度强化学习论文汇总

第85篇：279页总结"基于模型的强化学习方法"

第84篇：阿里强化学习领域研究助理/实习生招聘

第83篇：180篇NIPS2020顶会强化学习论文

第82篇：强化学习需要批归一化(Batch Norm)吗？

第81篇：《综述》多智能体强化学习算法理论研究

第80篇：强化学习《奖励函数设计》详细解读

第79篇: 诺亚方舟开源高性能强化学习库“刑天”

第78篇：强化学习如何tradeoff"探索"和"利用"？

第77篇：深度强化学习工程师/研究员面试指南

第76篇：DAI2020 自动驾驶挑战赛(强化学习)

第75篇：Distributional Soft Actor-Critic算法

第74篇：【中文公益公开课】RLChina2020

第73篇：Tensorflow2.0实现29种深度强化学习算法

第72篇：【万字长文】解决强化学习"稀疏奖励"

第71篇：【公开课】高级强化学习专题

第70篇：DeepMind发布"离线强化学习基准“

第69篇：深度强化学习【Seaborn】绘图方法

第68篇：【DeepMind】多智能体学习231页PPT

第67篇：126篇ICML2020会议"强化学习"论文汇总

第66篇：分布式强化学习框架Acme，并行性加强

第65篇：DQN系列(3): 优先级经验回放(PER)

第64篇：UC Berkeley开源RAD来改进强化学习算法

第63篇：华为诺亚方舟招聘 || 强化学习研究实习生

第62篇：ICLR2020- 106篇深度强化学习顶会论文

第61篇：David Sliver 亲自讲解AlphaGo、Zero

第60篇：滴滴主办强化学习挑战赛:KDD Cup-2020

第59篇：Agent57在所有经典Atari 游戏中吊打人类

第58篇：清华开源「天授」强化学习平台

第57篇：Google发布"强化学习"框架"SEED RL"

第56篇：RL教父Sutton实现强人工智能算法的难易

第55篇：内推 || 阿里2020年强化学习实习生招聘

第13篇：OpenSpiel(28种DRL环境+24种DRL算法)

第12篇：模块化和快速原型设计Huskarl DRL框架

第11篇：DRL在Unity自行车环境中配置与实践

第10篇：解读72篇DeepMind深度强化学习论文

第9篇：《AutoML》：一份自动化调参的指导

第8篇：ReinforceJS库（动态展示DP、TD、DQN）

第7篇：10年NIPS顶会DRL论文(100多篇)汇总

第6篇：ICML2019-深度强化学习文章汇总

第5篇：深度强化学习在阿里巴巴的技术演进

第4篇：深度强化学习十大原则

第3篇："超参数"自动化设置方法---DeepHyper

第2篇：深度强化学习的加速方法

第1篇：深入浅出解读"多巴胺（Dopamine）论文"、环境配置和实例分析

第14期论文：2020-02-10(8篇)

第13期论文：2020-1-21(共7篇)

第12期论文：2020-1-10(Pieter Abbeel一篇,共6篇)

第11期论文：2019-12-19(3篇，一篇OpennAI)

第10期论文：2019-12-13(8篇)

第9期论文：2019-12-3(3篇)

第8期论文：2019-11-18(5篇)

第7期论文：2019-11-15(6篇)

第6期论文：2019-11-08(2篇)

第5期论文：2019-11-07(5篇，一篇DeepMind发表)

第4期论文：2019-11-05(4篇)

第3期论文：2019-11-04(6篇)

第2期论文：2019-11-03(3篇)

第1期论文：2019-11-02(5篇)

阅读原文

喜欢此内容的人还喜欢

图灵奖得主的大神之家：一家三口都是MIT博士，还联合发了一篇AI论文

新智元

NeurIPS 2020中国入选论文：新一代算法“鉴黄师”诞生，中科院计算所研究生一作

量子位

2020年“微软学者”奖学金获奖者公布！

微软学术合作