

Never Give Up (NGU)

LEARNING DIRECTED EXPLORATION STRATEGIES

组员：颜铭 杨宪 毛荣贞 姚文君



目录

CONTENTS

- 一. 研究背景
- 二. 研究内容
- 三. 论文实现细节分析
- 四. 思考与展望



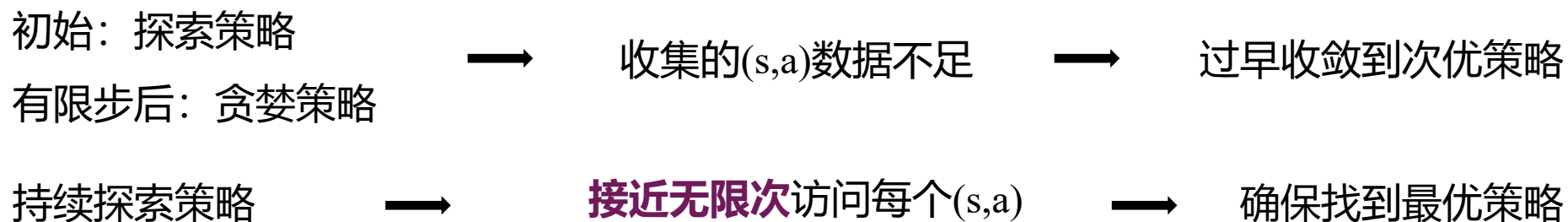
南开大学
Nankai University

01

研究背景

问题引入

探索问题仍然是深度强化学习领域的主要挑战之一



最简单的持续探索方法：以非零概率选择每个状态中的所有动作的随机策略

- 优点：对表格问题能找到最优策略
- 局限：效率低；所需的步骤会随着状态空间的增大呈指数增长
- 适用场景：密集奖励场景

稀疏奖励场景下
可能完全无法学习

相关研究

推动探索的方法：内在奖励(intrinsic reward)

- 主要实验环境：非表格问题

第一类：内在奖励与当前状态和已经访问过的状态之间的差异成比例

当智能体熟悉环境时，内在探索奖励消失，只由外在奖励来驱动学习

(Bellemare 等人, 2016; Ecoffet 等人, 2019; Stanton & Clune, 2018)

- 根本性的限制：

某状态的新颖性消失 \longrightarrow 不再访问该状态 \longrightarrow 丢失潜在的学习机会 \longrightarrow 限制智能体未来进一步探索和学习的能力

第二类：估计预测前向模型，并将预测误差作为驱动探索的内在动机

(Haber 等人, 2018; Houthoofd 等人, 2016; Oh 等人, 2015; Pathak 等人, 2017; Stadie 等人, 2015)

- 局限：成本高；容易出错；难以推广到任意环境（缺乏新颖信号时，算法降级为无向探索方案）
- 解决方案：调节学习算法的速度和奖励信号消失的速度之间的平衡

文章主要贡献

贡献一： 定义一个结合了life-long novelty(终生新颖性)和episodic novelty(分幕式新颖性)的探索奖励，用于学习探索策略

- 优点：使智能体可以在整个训练过程中保持探索——论文题目Never Give Up的来源

贡献二： 共同学习源自同一网络的单独的探索 and 开发策略

- 参数学习：使用UVFA框架，将内在奖励的权重 β 作为参数
- 特点：得到一族具有不同程度的探索行为的策略
- 优点：开发策略——专注于最大化外部奖励

$$r_t = r_t^e + \beta r_t^i$$

探究策略——保持探索，不会最终沦为无方向的策略

探索性策略的学习可以帮助构建一个即使在~~没有外部奖励~~的情况下也能继续发展的共享架构。

使用强化学习来近似与几个不同的内在奖励权重对应的最优价值函数。



文章主要贡献

贡献三：实验证据表明所提出的方法是可扩展的，并且在困难的探索任务中与SOTA方法的表现相同或更好。

- 不局限于导航任务，包含长期内在奖励，能够分离探索和开发政策

——与Savinov et al. (2018)相比

- 不依赖特权信息，结合分幕式和非分幕式的新颖性，获得优越的结果

(使用特权信息可能会导致过拟合或过度依赖于这些信息，使得智能体在面对新的情况时表现不佳)

——与Stanton & Clune (2018)相比

- 通过共享权重（而不是使用一个公共重放缓冲区）来学习多个策略，不需要精确的计数，可以扩展到Atari等更现实的领域

——与Beyer et al. (2019)相比



南開大學
Nankai University

02

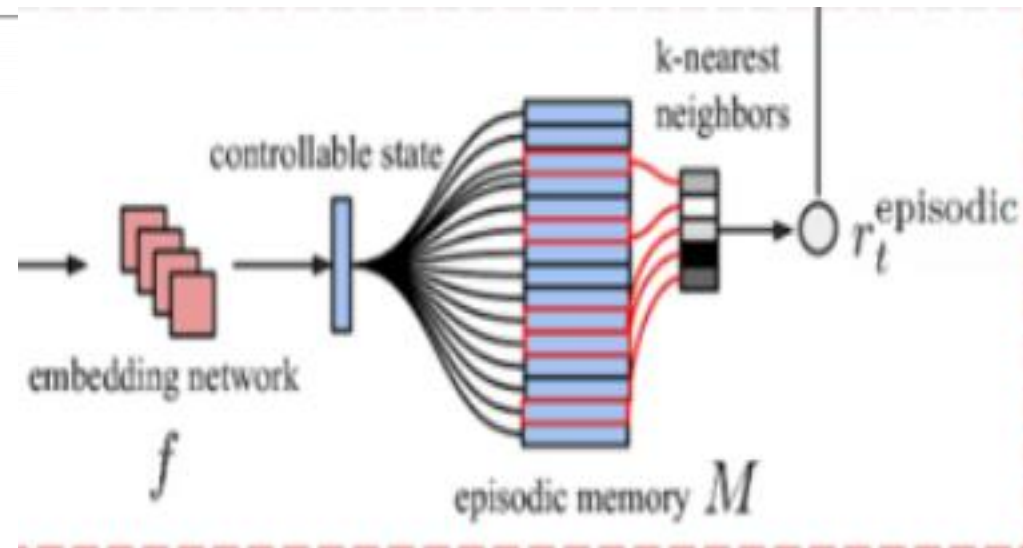
研究内容

研究内容

本文定义一种探索奖励，结合life-long novelty(终生新颖性)和episodic novelty(分幕式新颖性)来学习探索策略，可以在智能体的整个训练过程中保持探索，分幕式新颖性鼓励智能体人在几个情节中（而不是在同一情节中）定期重新访问熟悉（但可能未充分探索）的状态。终期的新颖性逐渐降低了状态，这些状态在许多情节中逐渐变得越来越熟悉。

具体地，分幕式新颖性使用的分幕式记忆充满了所有先前访问过的状态。采用自监督目标编码，避免状态空间的不可控。

对于终生式的新颖性，通过可乘性调制分幕相似信号，结合随机提纯网络误差(Random Network Distillation error)驱动梯度损失的学习。与分幕新颖性相反，终生(life-long)新颖性变化缓慢，这取决于由学习率决定的梯度下降优化的程度。



这种结合的新颖性概念能够在具有大型高维状态空间的复杂任务中进行泛化，在该状态空间中，从未两次观察到给定状态，并且在分幕内和分幕间都保持一致的探索。

内在奖励的具体含义

内在奖励，直接体现将分幕式和终生新颖性相结合的思想，明确鼓励智能体在一个episode中反复访问环境中的所有可控状态。

论文的内在奖励满足三个属性，记号为 r_t^i ：

- (i) 迅速阻止在同一分幕中再次访问同一状态；
- (ii) 缓慢阻止在分幕中多次访问过的状态访问；
- (iii) 状态的概念忽略了环境的各个方面不受智能体行为的影响。

由好奇心机制，外在奖励通过内在奖励（或探索奖励）得到增强

奖励机制的工作框架

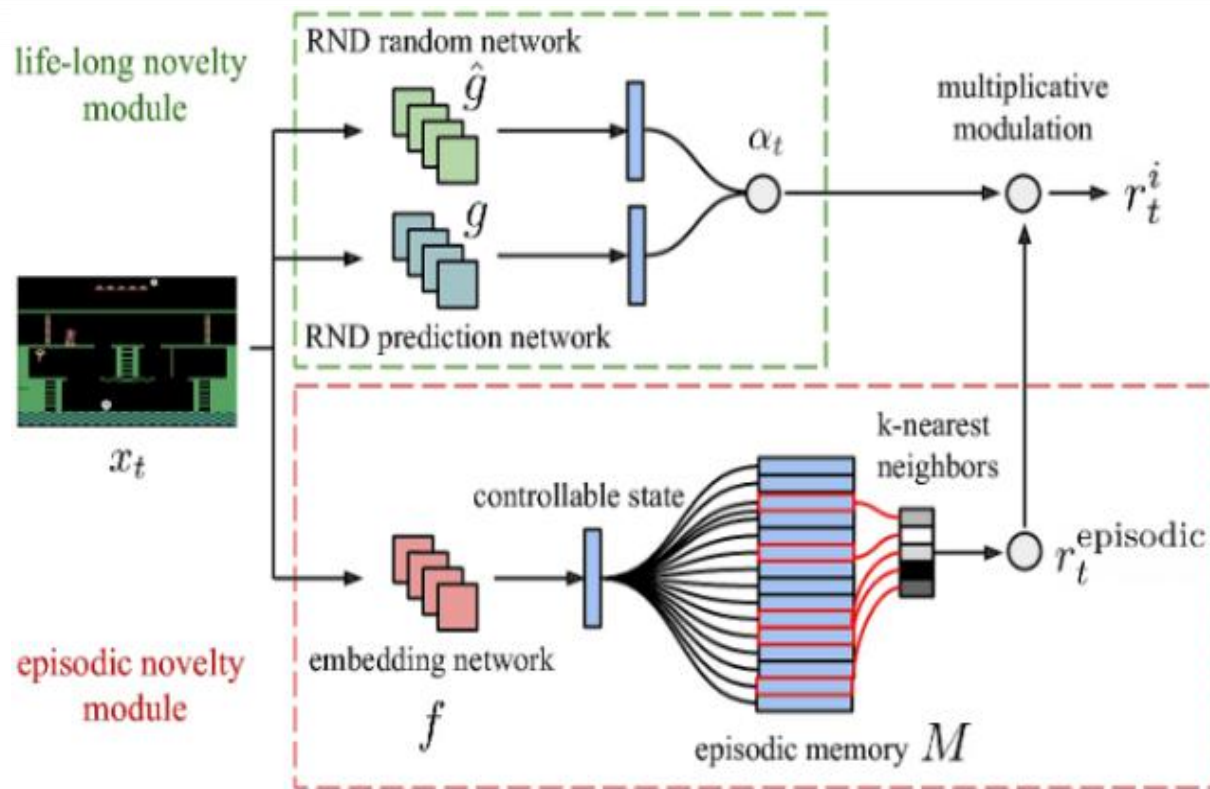
$$r_t = r_t^e + \beta r_t^i$$

上式为增强奖励的表达式定义，其中 r_t^e 为外部奖励， r_t^i 为内部奖励， β 为权重正相关性的正标量。

奖励由两个模块组成：分幕式新颖性模块和（可选）终身新奇模块，分别在右图中用红色和绿色表示。

每个步骤中，智能体都会计算分幕性内在奖励 $r_t^{episodic}$ 并将与当前观察值相对应的可控制状态附加到记忆存储M中。

当前观察结果与分幕记忆的内容进行比较。较大的差异会产生较大的分幕内在奖励。分幕性内在奖励约束力促进了特工在单个分幕中访问尽可能多的不同状态



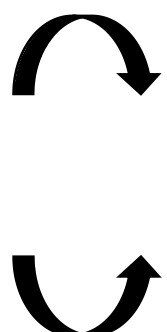
新颖性的概念体现在采样间隔相互作用是一致更新的：一个已经访问了数千次的状态，只要在当前分幕的历史上同样具有新颖性，它就会获得与一个全新状态相同的内在奖励



奖励机制的几点说明

- 1、终生(Long-term)的内在奖励指的是整个训练过程中对状态的陌生程度的评价。Long-term内在奖励用已有的好奇心驱动方法都是可以的 (ICM, RND) , 论文中是用RND实现的, 通过控制学习率, Long-term内在奖励可以缓慢的阻止agent访问熟悉的状态;
- 2、分幕式(Episodic)内在奖励指的是在一个episode中状态的陌生程度。它的好处是每个episode都重新计算, 不会随着训练而衰减的状态。episodic内在奖励通过episodic memory来实现, K-近邻距离越大内在奖励越大, 能促进agent在一个episode中访问不同, memory的方式能够使得episodic内在奖励能够快速变化, 促进agent在同一个episode内访问不同的状态。

由上述分析, 长期新颖性信号通过可控状态地控制各个分幕的探索量, 论文定义了非调制的奖励方法以使用好奇心的时序逻辑


$$r_t^i = r_t^{episodic} \cdot \min\{\max\{\alpha_t, 1\}, L\}$$

内在奖励由long-term和episodic
内在奖励的乘积计算, 这里引入了折扣因子 $L=5$

嵌入网络的设计原理

将当前观测值映射到与其可控状态相对应的 p 维向量。引入一个环境，使其具有与智能体程序的动作无关的多变性。使得智能体无需采取任何行动即可访问大量不同的状态（收集大量的累积内在奖励）。

为了避免无意义的探索，在连续两次观察的情况下，论文训练了一个孪生神经网络，以预测智能体采取的行动是从一个观察转到另一个观察。这是由先验知识启发的：**不受智能体采取的行动影响的环境中的所有可变性都将无助于做出此预测**。这也是部分马尔科夫性作用的模型迁移，论文使用条件概率似然化为：

$$p(a|x_t, x_{t+1}) = h(f(x_t), f(x_{t+1}))$$

终生新颖度的借鉴性工作

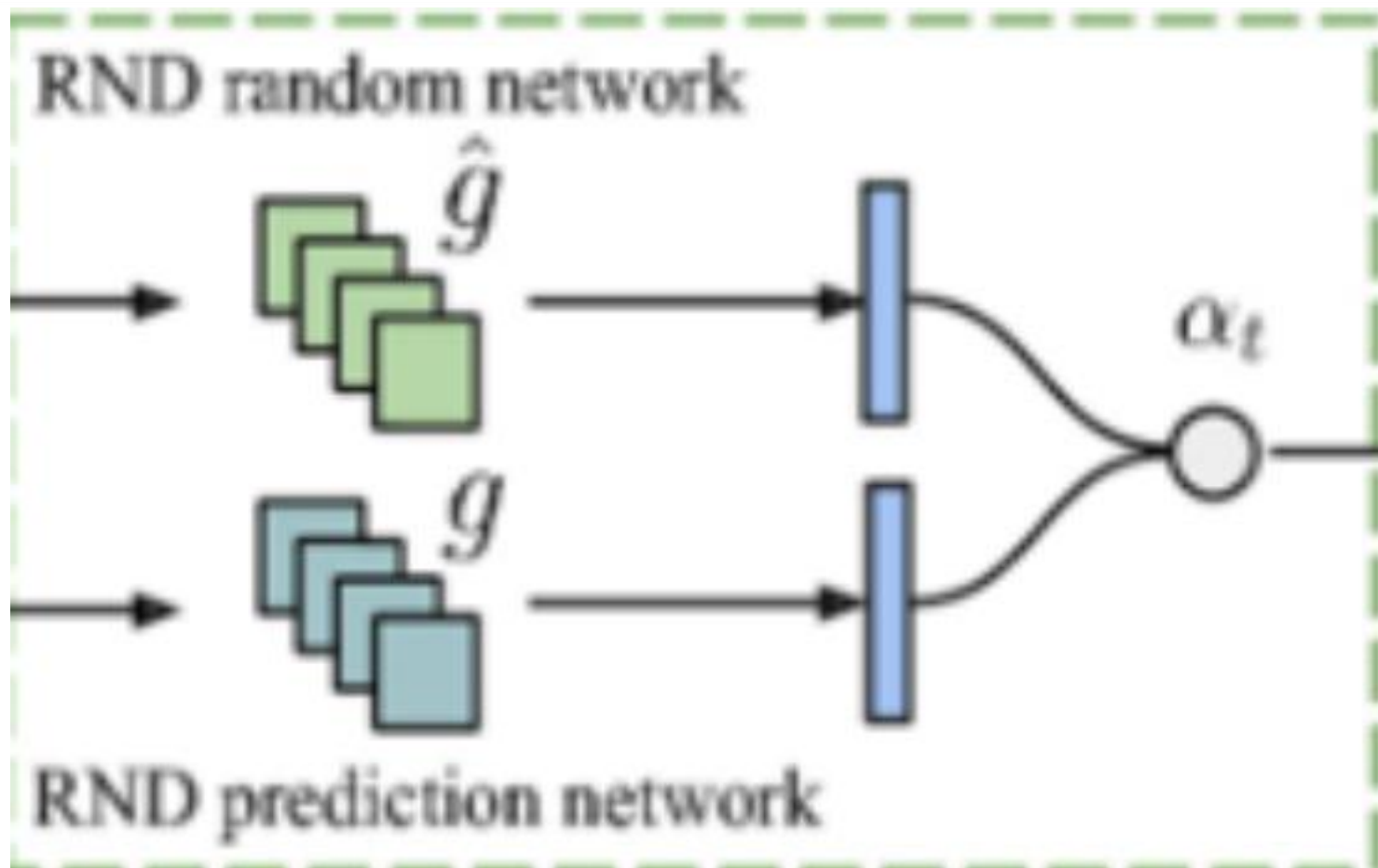
life-long novelty:

RND的主要思路就是随机初始化一个网络作为target network, 然后训练一个predictor network来预测状态转移, 使得predictor network的预测接近target network:

minimizing $\text{err}(x_t) = \|\hat{g}(x_t; \theta) - g(x_t)\|^2$

其中 α_t 是long-term内在奖励, 由RND中predict net与random net的预测误差计算:

$$\alpha_t = 1 + \frac{\text{err}(x_t) - \mu_e}{\sigma_e}$$



分幕好奇心的创新性工作

Episodic novelty:

Episodic novelty计算论文的情节内在奖励，它由分幕式记忆存储 M 和嵌入函数 F 组成，将当前观察结果映射到论文称为可控状态的学习表示。在每一幕开始时episodic memory开始完全为空。在每个步骤中，智能体都会计算每一幕内在奖励，在时间 t ，内存包含每个 $f(x)$ 中访问的所有观测值的可控制状态，论文定义了固有的奖励如下式

$$r_t^{\text{episodic}} = \frac{1}{\sqrt{n(f(x_t))}} \approx \frac{1}{\sqrt{\sum_{f_i \in N_k} K(f(x_t), f_i) + c}}$$

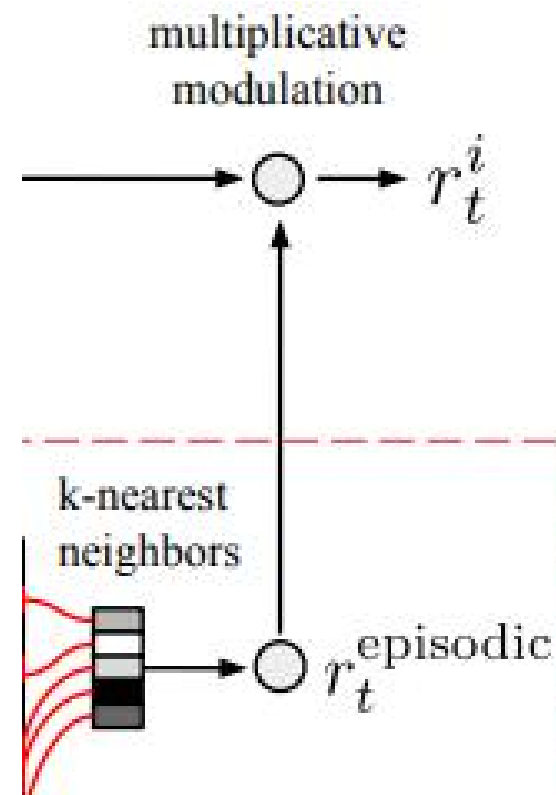
Algorithm 1: 情景内在奖励的计算时间: r_t^{episodic} 。

Input : $M; k; f(x_t); c; \epsilon; \xi; s_m; d_m^2$

Output : r_t^{episodic}

Compute the k -nearest neighbours of $f(x_t)$ in M and store them in a list N_k

Create a list of floats d_k of size k





“永不放弃”的保持探索智能 NGU Agent

体

(1) 体系架构

基于价值函数逼近器 (UVFA) 设计思想, 取同步估计优化值函数在一系列形式为增强的强化奖励公式的超参数。论文中使用混合演员数 N 的值作为模型的度量, 并在可控状态中传递参数 β 。在定性解构到模拟中可以简单地通过贪心策略实现探索行为的限制到积极开发(exploitation)的转化, 甚至不用观察到外部奖励就可以学习到性能强大的参数模型。

这里学习大量策略的优势来自于开发和探索策略从行为角度来看可能完全不同的事实, 引入大量可以平滑更新的策略可以进行更有效的深度强化学习网络的参数进化

(2) 强化学习损失函数

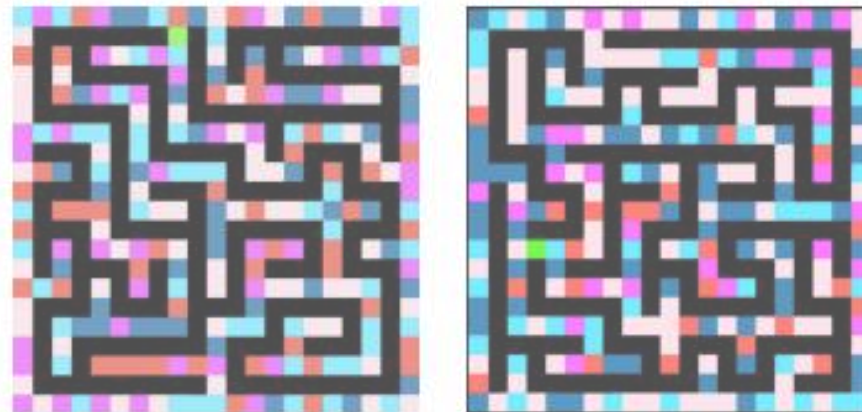
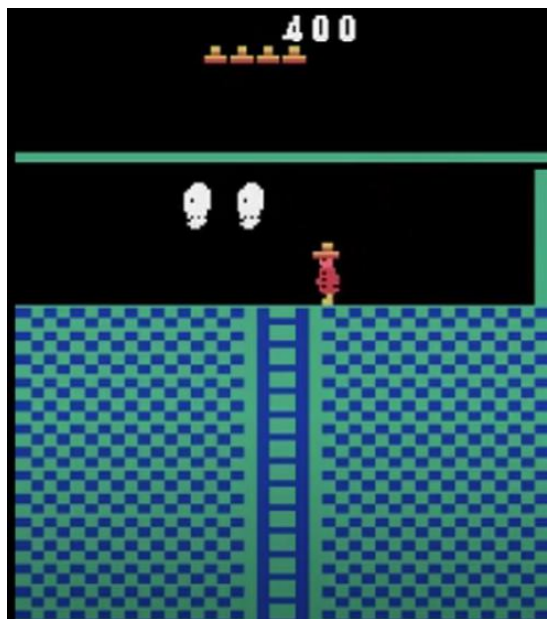
同经典的DQN网络一致使用均方误差函数作为训练损失, 论文在此基础上提出借鉴ettrace双Q学习损失优化。然后再结合分布式训练框架, 并做了相关结构上的改进, 就得到了 NGU 智能体, 另外值得注意的是, 为了促进探索, 在强化学习训练过程中使用内在奖励, 其实改变了任务对应的马尔可夫决策过程 (MDP)。

(3) 分布式开发

深度强化学习中的最新工作通过在分布式培训体系结构上运行获得了显著改善的性能, 该体系结构从在不同环境实例上并行运行的许多参与者收集了大量经验。文章中的智能体以Kapturowski等人的工作为基础。将表演与表演脱钩, actor将经验馈入分布式回放缓冲区, Learner以优先级排序方式从中随机采样批次进行训练。

NGU智能体在游戏中的实验成果

Hyperparameter	Value
Max episode length	30 min
Num. action repeats	4
Num. stacked frames	1
Zero discount on life loss	false
Random noops range	30
Sticky actions	false
Frames max pooled	3 and 4
Grayscaled/RGB	Grayscaled
Action set	Full



Hyperparameter	Value
Episodic memory capacity	5000
Learning rate (R2D2 and Action prediction)	0.001
Replay capacity	1e6
Intrinsic reward scale β	0.5
Trace length	50
Replay period	50
Retrace λ	0.97
Retrace loss transformation	identity
Num. action repeats	1
Target Q-network update period	100
Q-network filter sizes	(3, 3)
Q-network filter strides	(1, 1)
Q-network num. filters	(16, 32)
Action prediction network filter sizes	(3, 3)
Action prediction network filter strides	(1, 1)
Action prediction network num. filters	(16, 32)
Kernel ϵ	0.01
Evaluation ϵ	0

为了更好地理解完整NGU智能体的几种主要设计选择，我们进行了进一步的消解和融合：

消解融合指的是使用混合演员数量N设计出的不同超参数，在稀疏到密集的奖励的游戏上，部署到艰难探索的定向初始化决策的业务场景中，分别评估其对非策略数据的影响。

对于艰难探索决策有结论：

开发策略如何能够利用所有内在条件混合的权重来探索难以做到的游戏，但仍会朝着最终分幕得分最大化的方向进行优化

对于密集奖励游戏有结论：

由于内在奖励信号可能与游戏目标完全失调，因此可以预期到单从内在奖励导向的好奇心机制学习效果不佳

NGU智能体在游戏中的实验成果

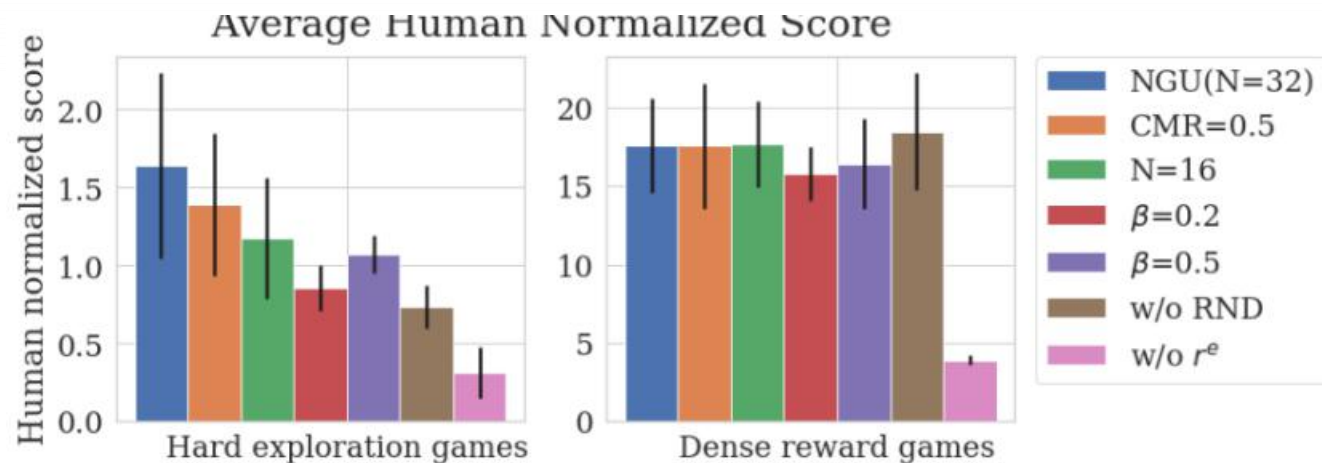


Figure 3: Human Normalized Scores on dense reward and hard exploration games.

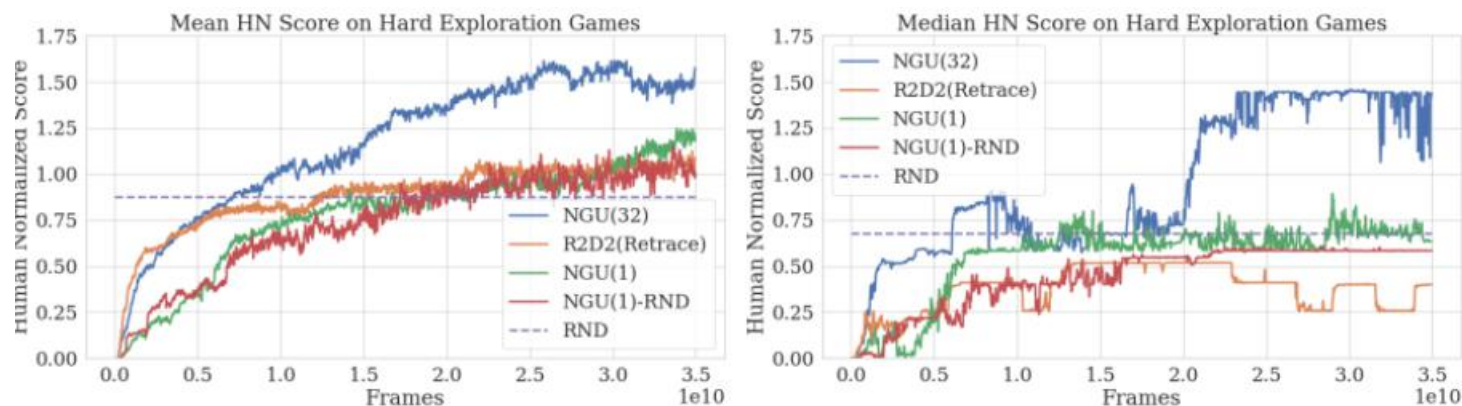


Figure 4: Human Normalized Scores on the 6 hard exploration games.

可视化结果分析中可以得到如下结论:

1. 首先, 分享所有演员的经验会轻微损害艰难探索游戏的整体平均表现。这表明, 通过不同的模型权重而不是通过共享数据, 可以获得针对不同条件混合物的不同行为能力
2. 在观察到的期望意义下, 对于艰苦的探索进行决策的游戏中, 增加混合演员数目N的数量会带来改善
3. β 值由密集奖励参数迁移到艰难探索上可控参数邻域的值范围缩小, 表明NGU要么没有足够高的探索性变体 (β 太低), 要么策略变得过于偏向探索性行为 (β 太高)



南开大学
Nankai University

03

论文实现细节分析



如何给出评估模型性能的算法?

类似于R2D2网络评估工作，结合并行的评估算法运行Q网络模型并分析环境交互中的learner和actor部分的智能体信息

设计算法
细节分析

模型实现中限定模型性能状态的术语相关运行方式:

(1) 学习者:Learner, 限定学习行为的步进算法

- 从经验回放的缓冲区中采样一系列增强的奖励 r_t ,内在奖励 r_t^i , 观察 x , 动作 a 和折扣率 γ_i
- 对于Q网络模型, 使用R2D2中依靠回溯(Retrace)学习过程,对网络模型输入的三元组 (x_t, x, a) 进行采样。
- 采样序列最后5帧动作预测网络进行动作优化, 而所有的step步长时间都用于训练Q网络损失

(2) 评估器和演员:Evaluator and actor, 限定策略函数的行为和评估模式

- 获得参数 x_t, r_t^e, r_{t-1}^i 和折扣因子
- 根据这些输入, 匹配向前传播的R2D2算法得到参数 α_t
- 根据 x_t 匹配参数为 r_t^i 的嵌入式网络(embedding network) $x_t, \alpha_t \cdot r_t = r_t^e + \beta_i r_t^i, \gamma_i$
- 对演员, 将 α_t 插入到经验回放数组中
- 对于环境交互的智能体实例根据参数 α_t 步进



为加速收敛到最优解，对于网络加强小批量更新。每个actor每秒在Atari环境中执行大约260步的探索。

关于经验回放公式中内奖励参数的选取,为每个actor赋值 $\beta_i\}_{i=0}^{N-1}$ 集合中的定值,对于第j个actor选取 $j \bmod (N-1)$ 下标的数值,集合定义如下:

$$\beta_i = \begin{cases} 0 & \text{if } i=0 \\ \beta & \text{if } i=N-1 \\ \beta \cdot \sigma(10^{\frac{2i-(N-2)}{N-2}}) & \text{otherwise} \end{cases}$$

这里 σ 指的是S型激活函数，如sigmoid和tanh函数等，这都被广泛运用在长短期记忆网络的门控神经元中，这也是基于马尔科夫性获取模型规律中结合循环神经的预处理手段

好奇心机制可以这样解释：探索性策略使用较小的折现因子，因为内在报酬密集且价值范围较小，而我们对开发性策略使用尽可能高的折现因子，以便尽可能接近优化未折现收益

超参数中最重要的 β 和 γ 中导出的循环状态会被用作初始化神经网络，再由采样回到时序组织的经验回放池中

论文使用的是基于优先权的经验回放池,与奖励策略参数 β 相关联的是折扣因子 γ : 其具体定义如下:

$$\gamma = 1 - \exp\left(\frac{(N-1-i) \log(1-\gamma_{max}) + i \log(1-\gamma_{min})}{N-1}\right)$$

这种形式允许存在折扣因子使折扣系数在最小折扣因子和最大折扣因子的对数空间中均匀分布。此外，通过同步的折扣因子调整,我们进一步得到智能体的好奇机制的定量分析

代表开发策略的参数 β_0 关联到最高折扣因子 $\gamma_0 = \gamma_{max}$

代表探索性策略的参数 β_{N-1} 关联到最低折扣因子

$$\gamma_0 = \gamma_{min}$$

“永不放弃的内部奖励算法”

我们有以下记号和分析

1. M 是由一系列部分记忆包含在时间 t 和之前的嵌入序列中 $\{f(x_0), f(x_1), \dots, f(x_{t-1})\}$

2. k 是最近邻的数目

3. $N_k = \{f_i\}_{i=1}^k$ 是在经验池 N 中的 k 近邻集合

4. K 定义了核 $K(x, y) = \frac{\epsilon}{\frac{d^2(x, y)}{d_m^2} + \epsilon}$

5. c 是伪计数常量

6. ξ 是聚类的距离

7. s_m 是最大相似度



开源代码的实现

```
def compute_intrinsic_reward(
    episodic_memory: List,
    current_c_state: Tensor,
    k=10,
    kernel_cluster_distance=0.008,
    kernel_epsilon=0.0001,
    c=0.001,
    sm=8,
) -> float:
    state_dist = [(c_state, torch.dist(c_state, current_c_state)) for c_state in episodic_memory]
    state_dist.sort(key=lambda x: x[1])
    state_dist = state_dist[:k]
    dist = [d[1].item() for d in state_dist]
    dist = np.array(dist)

    # TODO: moving average
    dist = dist / np.mean(dist)

    dist = np.max(dist - kernel_cluster_distance, 0)
    kernel = kernel_epsilon / (dist + kernel_epsilon)
    s = np.sqrt(np.sum(kernel)) + c

    if np.isnan(s) or s > sm:
        return 0
    return 1 / s
```

分幕式内在奖励匹配算法伪代码

Algorithm 1: Computation of the episodic intrinsic reward at time t : r_t^{episodic} .

Input : $M; k; f(x_t); c; \epsilon; \xi; s_m; d_m^2$

Output : r_t^{episodic}

- 1 Compute the k -nearest neighbours of $f(x_t)$ in M and store them in a list N_k
- 2 Create a list of floats d_k of size k
/* The list d_k will contain the distances between the embedding $f(x_t)$ and its neighbours N_k . */
- 3 **for** $i \in \{1, \dots, k\}$ **do**
- 4 $d_k[i] \leftarrow d^2(f(x_t), N_k[i])$
- 5 **end**
- 6 Update the moving average d_m^2 with the list of distances d_k
/* Normalize the distances d_k with the updated moving average d_m^2 . */
- 7 $d_n \leftarrow \frac{d_k}{d_m^2}$
/* Cluster the normalized distances d_n i.e. they become 0 if too small and 0_k is a list of k zeros. */
- 8 $d_n \leftarrow \max(d_n - \xi, 0_k)$
/* Compute the Kernel values between the embedding $f(x_t)$ and its neighbours N_k . */
- 9 $K_v \leftarrow \frac{\epsilon}{d_n + \epsilon}$
/* Compute the similarity between the embedding $f(x_t)$ and its neighbours N_k . */
- 10 $s \leftarrow \sqrt{\sum_{i=1}^k K_v[i] + c}$
/* Compute the episodic intrinsic reward at time t : r_t^i . */
- 11 **if** $s > s_m$ **then**
- 12 $r_t^{\text{episodic}} \leftarrow 0$
- 13 **else**
- 14 $r_t^{\text{episodic}} \leftarrow \frac{1}{s}$

算法效果的分析

Pitfall!

由前文的定性分析，在缺少纯粹的开发策略的情形下，模型消融能力越来越关键，对于演员数目为 $N=1$ 的融合模型在密集奖励的游戏中表现不佳。

论文中由算法实现和可视化结果分析由侧面说明的，对于K近邻数目变动在稀疏奖励游戏中表现的影响浮动较小。且可控制的平稳状态(性能直观度量)也基本没有变动

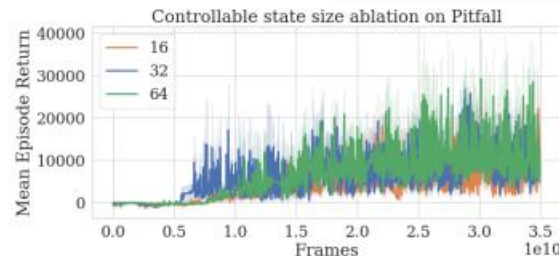


Figure 8: Mean episodic return for agents trained *Pitfall!*.

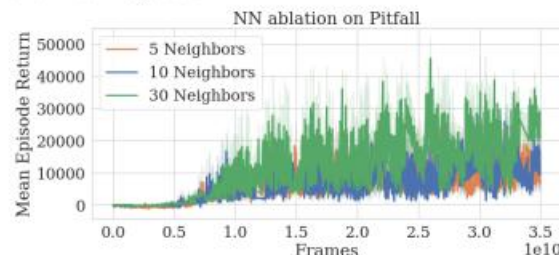


Figure 10: Mean episodic return for agents trained *Pitfall!*.

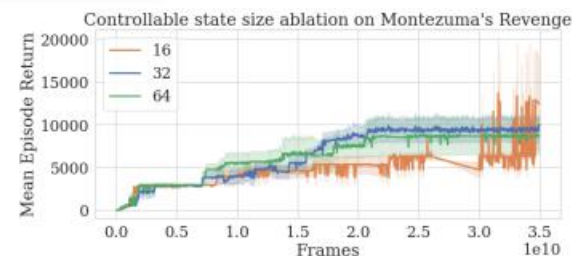


Figure 9: Mean episodic return for agents trained *Montezuma's Revenge*.

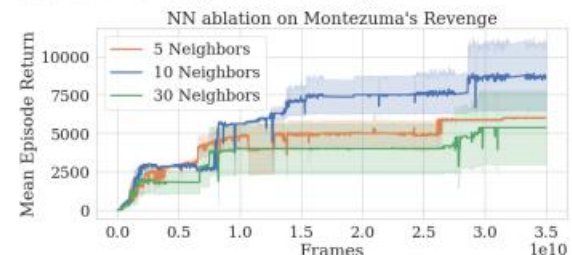


Figure 11: Mean episodic return for agents trained *Montezuma's Revenge*.

Montezuma's Revenge

算法报告需要回归到准备的目标输出的公式中:

$$r_t^i = r_t^{episodic} \cdot \min(\max(\alpha_t, 1), L)$$

作为端到端的训练我们必须要对输出预设条件的裁剪因子 L 进行解释:
为什么备择中要固定非退化的因子 L 呢?($L=5$)

实验证明在以上两个实验中参数的值在定量强调时多次表现鲁棒性，且通过由上到下裁剪因子 L 模拟算法性能评估，在平稳状态受限的NGU($N=1$)上有不利影响。其本质上是轨迹变动的意义下无需外部奖励即可获得高分。这是**因为探索性策略学会了生存，最终导致了高分。**

此外对于稀疏奖励对应的艰苦探索的扩展，学习策略更表现出侧重开发的特性。此时外部的参数表现出较少的变化特性

算法优化: Retrace回溯算法信息

Retrace是一种轨策略的强化学习算法, 可用于评估或控制。在评估设置中, 目标主要是根据从行为策略 μ 绘制的轨迹来估计目标策略 π 的作用值

Q^π

设置目标策略, 或者使用更加精确的目标策略序列, 依赖于Q函数的序列如何生成估计值, 论文指出可以认为轨迹 τ 可以通过估计 Q^* 过程生成。

损失函数仍采用均方误差形式

$$L(x_t, a_t, \theta) = (Q(x_t, a_t; \theta) - \hat{y}_t)^2$$

更一般的有, 以及实值函数 h 的定义

$$\delta_t^h = r_t \gamma \sum_{a \in A} \pi(a|x_{t+1}) h^{-1}(Q(x_{t+1}, a)) - h^{-1}(Q(x_t, a_t))$$

$$\forall z \in \mathbb{R}, h(z) = \text{sign}(z)(\sqrt{|z| + 1} - 1) + \epsilon z,$$

$$\forall z \in \mathbb{R}, h^{-1}(z) = \text{sign}(z)((\frac{\sqrt{1 + 4\epsilon(|z| + 1 + \epsilon)} - 1}{2\epsilon}) - 1)$$

为达到这一步, 考虑将轨迹 τ 从开始的状态-价值对 (x, a) 中生成, 有公式:

$$\tau = (x_t, a_t, r_t, x_{t+1})_{t \in N}$$

进一步回溯算子根据 μ 和 π 有如下定义:

$$\mathcal{T}Q(x, a) = Q(x, a) + \mathbb{E}_\mu[\sum_{t \geq 0} \gamma^t (\prod_{s=1}^t) \delta_t]$$

相应差异 δ_t 的定义如下:

$$\delta_t = r_t + \gamma \sum_{a \in A} \pi(a|x_{t+1}) Q(x_{t+1}|a) - Q(x_t, a_t)$$

在实现中仿照DDQN采取两个Q网络:

一个目标网络为 $Q(x, a, \theta^-)$, 一个在线网络为 $Q(x, a, \theta)$

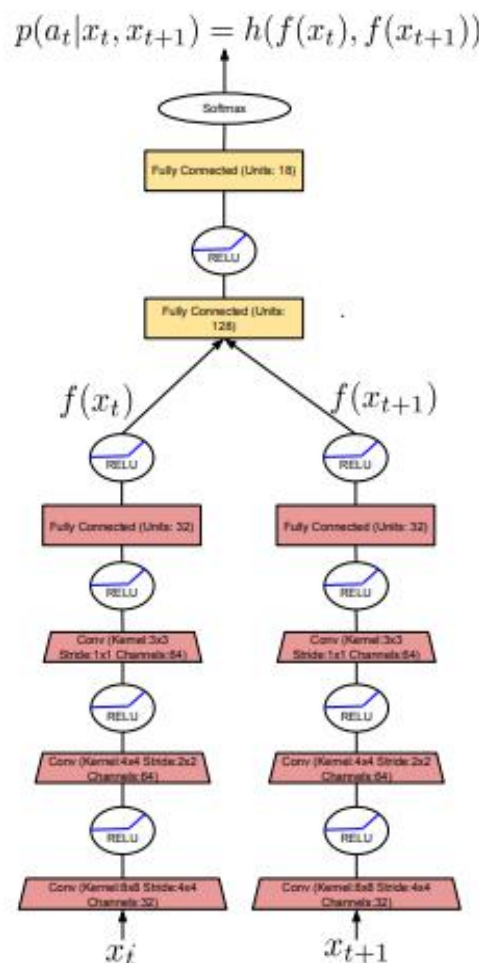
目标网络是在更新权重参数之外, 对预测值进行拟合:

$$\hat{y}_t = \hat{T}Q(x_t, a_t; \theta^-)$$

$$= Q(x_t, a_t; \theta^-) \sum_{s=t}^{t+k-1} \gamma^{s-t} (\prod_{i=t+1}^s c_i) (r_s + \gamma \sum_{a \in A} \pi(a|x_{s+1}) Q(x_{s+1}, a; \theta^-) - Q(x_s, a_s; \theta^-))$$

集成网络架构分析

图.具有逆动态预测的嵌入网络体系结构



```
class EmbeddingModel(nn.Module):
    def __init__(self, obs_size, num_outputs):
        super(EmbeddingModel, self).__init__()
        self.obs_size = obs_size
        self.num_outputs = num_outputs

        self.fc1 = nn.Linear(obs_size, 32)
        self.fc2 = nn.Linear(32, 32)
        self.last = nn.Linear(32 * 2, num_outputs)

        self.optimizer = optim.Adam(self.parameters(), lr=1e-5)

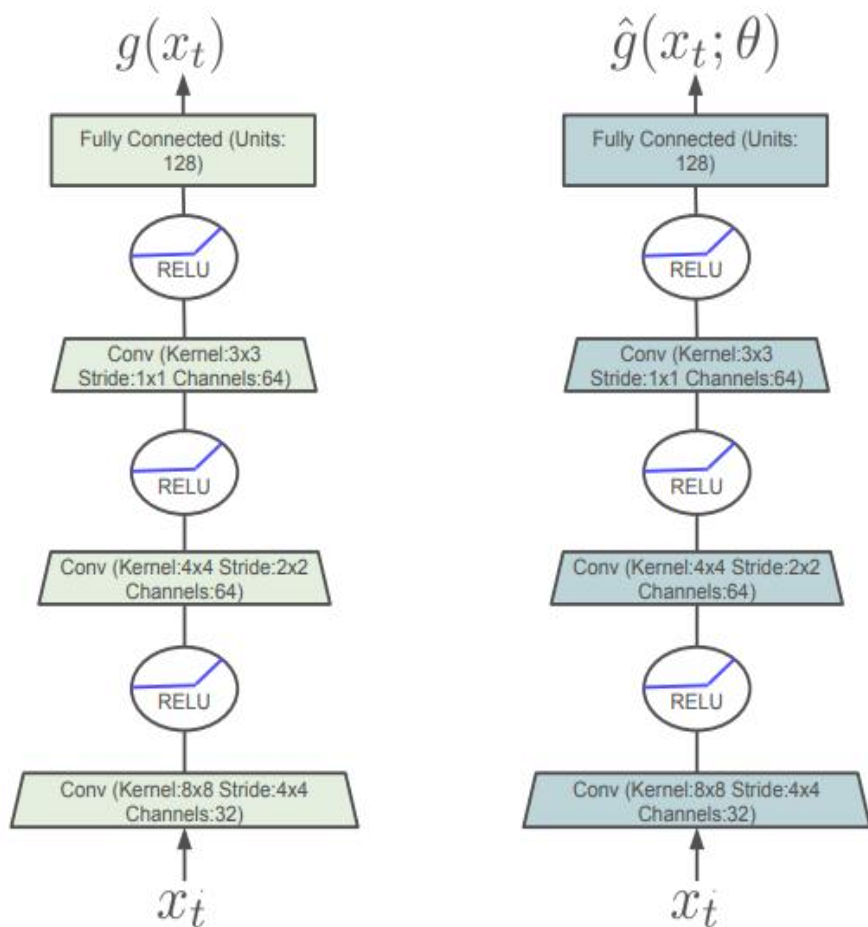
    def forward(self, x1, x2):
        x1 = self.embedding(x1)
        x2 = self.embedding(x2)
        x = torch.cat([x1, x2], dim=2)
        x = self.last(x)
        return nn.Softmax(dim=2)(x)

    def embedding(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        return x

    def train_model(self, batch):
        batch_size = torch.stack(batch.state).size()[0]
        # last 5 in sequence
        states = torch.stack(batch.state).view(batch_size, config.sequence_length, self.obs_size)[: , -5: , :]
        next_states = torch.stack(batch.next_state).view(batch_size, config.sequence_length, self.obs_size)[: , -5: , :]
```

强化学习中，通常需要将状态和动作表示为向量形式，以便能够输入到神经网络中进行处理。而 Embedding 网络是一种将离散的输入映射到连续的向量空间中的技术，可以将离散的状态和动作转换为连续的向量表示。这样做的好处是可以使神经网络更好地处理离散的输入，同时也可以将相似的输入映射到相近的向量空间中，从而更好地学习到它们之间的关系。

图.RND(随机提纯网络)网络架构

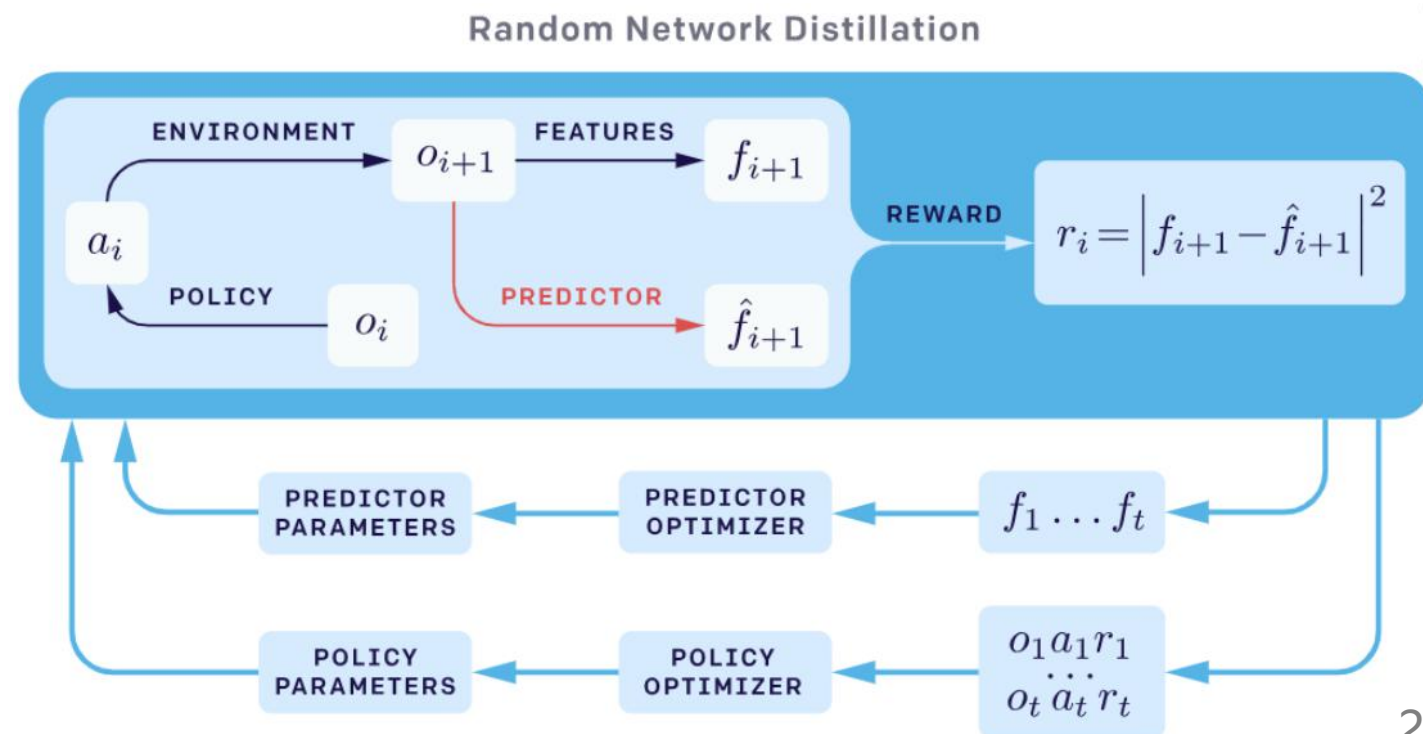


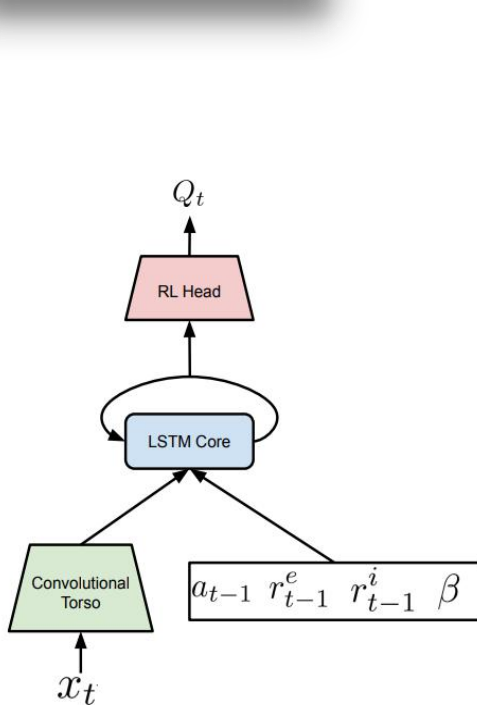
对于预测模型，一般来说，预测误差和四个因素有关：

- 预测器无法泛化，因为训练集不够
- 预测目标是随机的
- 缺少必要的输入信息
- 模型能力不足以适应目标函数

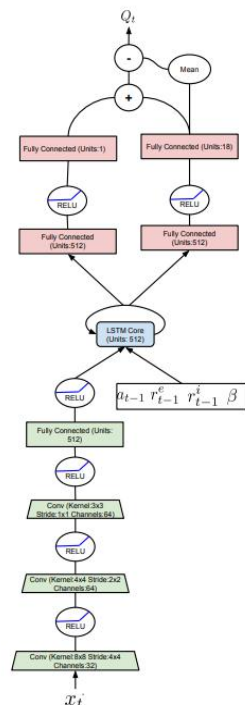
第一点是必要的，泛化能力差就代表了预测误差高也就是好奇心高。对于剩下三点要想办法消除。

RND算法，引入的内在 reward 是基于预测下一状态下 固定且随机初始化的神经网络 的输出。这也是NGU算法模型的雏形





(a) Sketch of the R2D2 Agent



(b) Detailed R2D2 Agent

```
class R2D2(nn.Module):
    def __init__(self, num_inputs, num_outputs):
        super(R2D2, self).__init__()
        self.num_inputs = num_inputs
        self.num_outputs = num_outputs

        self.lstm = nn.LSTM(input_size=num_inputs, hidden_size=config.hidden_size, batch_first=True)
        self.fc = nn.Linear(config.hidden_size, 128)
        self.fc_adv = nn.Linear(128, num_outputs)
        self.fc_val = nn.Linear(128, 1)

    def forward(self, x, hidden=None):
        # x [batch_size, sequence_length, num_inputs]

        batch_size = x.size()[0]
        sequence_length = x.size()[1]
        out, hidden = self.lstm(x, hidden)

        out = F.relu(self.fc(out))
        adv = self.fc_adv(out)
        adv = adv.view(batch_size, sequence_length, self.num_outputs)
        val = self.fc_val(out)
        val = val.view(batch_size, sequence_length, 1)

        qvalue = val + (adv - adv.mean(dim=2, keepdim=True))

        pred = pred.gather(2, actions)

        _, next_pred_online_action = next_pred_online.max(2)

        target = rewards + masks * pow(config.gamma, steps) * next_pred.gather(2, next_pred_online_action.unsqueeze(2))

        td_error = pred - target.detach()

        for idx, length in enumerate(lengths):
            td_error[idx][length - config.burn_in_length_ww:][:] = 0

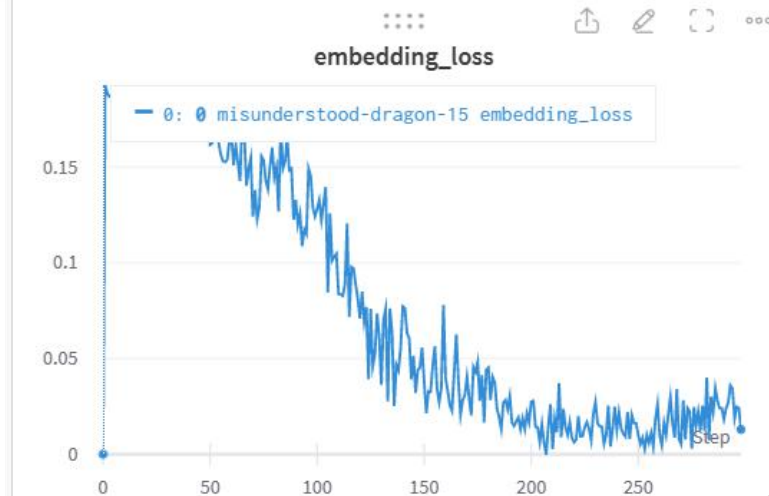
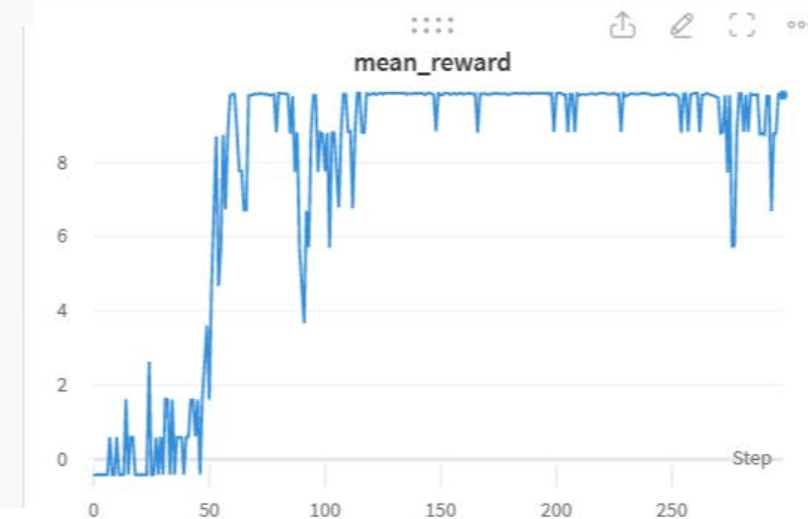
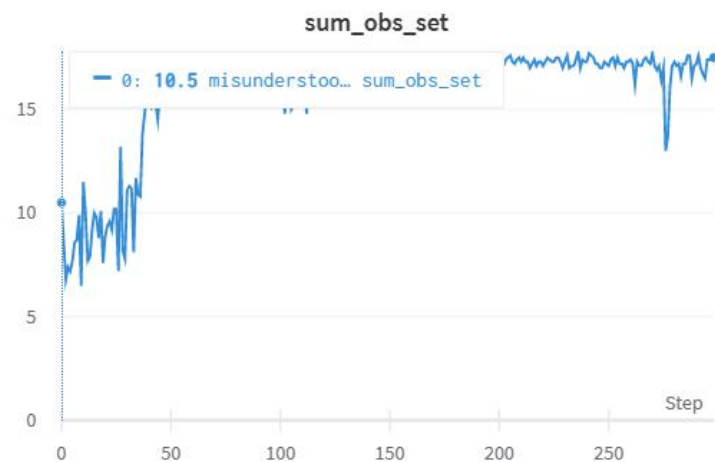
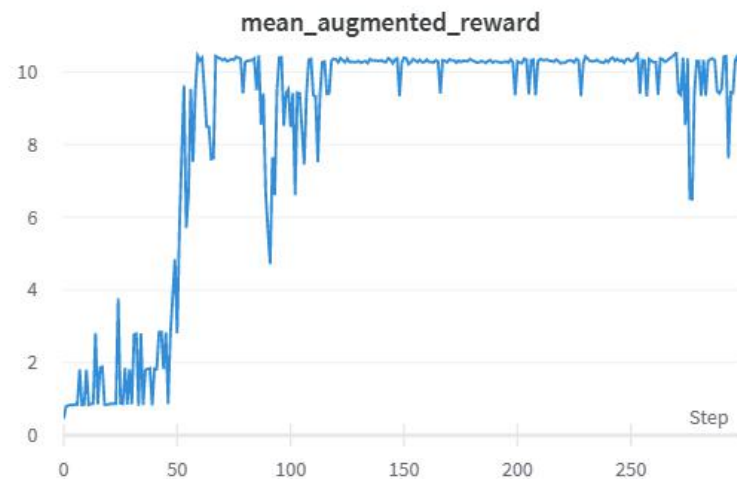
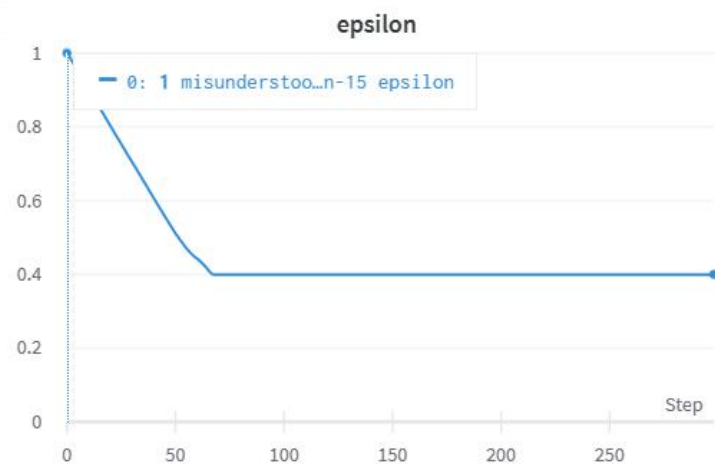
        return td_error
```

```
@classmethod
def get_td_error(cls, online_net, target_net, batch, lengths):
    def slice_burn_in(item):
        return item[:, config.burn_in_length_ww, :]
```

R2D2主体为lstm网络的循环特征消息处理机制，创新使用分布式训练和优先经验回放
NGU模型侧重改良好奇心机制的超参数效应，是对R2D2的直接继承和发展



3000幕---训练效果可视化图表





南开大学
Nankai University

04

思考与展望



思考

经典探索策略

ϵ -贪婪：智能体以较小的概率 ϵ 进行随机探索，在大多数情况以概率 $1 - \epsilon$ 选择当前的最优动作。

玻尔兹曼探索策略：智能体根据学习到的由温度参数 τ 调节的Q值，从玻尔兹曼分布（softmax函数）中选择动作。

汤普森采样：智能体将追踪记录的最优动作概率作为先验分布，然后从这些分布中采样。

硬探索问题

“硬探索”问题是指在奖励非常稀少甚至具有欺骗性的环境中进行探索。在这种情况下进行随机探索基本无法找到成功的状态或获得有意义的反馈。



思考

内在奖励作为额外的探索奖励

根据当前任务定义，策略的训练由两项组成，包括来自环境的外部奖励和当前时刻 t 的内在探索奖励， β 是调整探索与利用之间平衡的超参数：

$$r_t = r_t^e + \beta r_t^i$$

原始外在奖励和内在奖励之间的相对权重：

稀疏奖励环境：由于只有智能体达到目标位置时，智能体才获得一个正的0到1之间的奖励，其他时刻奖励都为零。在这种环境上，累计折扣内在奖励的幅度会远大于原始的 $[0, 1]$ 之间的数，造成智能体学习的目标偏差太大。

密集奖励环境：每一步都会有原始外在奖励，且幅度远大于1。在这种环境上，累计外在奖励的幅度会远大于内在奖励，内在奖励对智能体的驱动不足。



展望

1. 学习超越简单逆动力学模型的有效可控状态。
2. β 的动态适应。

如何自动平衡内在奖励和环境本身的外在奖励？

不同类型的环境，探索策略也应该不同，是否有一种通用方案可以解决一大批环境上的探索问题？

谢谢大家

组员分工:

- **颜铭**-论文翻译和算法公式的阅读整理，研究开源代码并分析结果，组织分工。负责第三部分PPT制作以及第二部分PPT修改，负责该部分的PPT发言。
- **杨宪**-负责整理博客资料，分享论文学习心得，负责PPT第二部分制作。
- **毛荣贞**-阅读论文并负责第四部分PPT制作，分享心得体会并负责部分PPT发言。
- **姚文君**-阅读论文并负责第一部分PPT制作与PPT学术化修改工作。