

# PPO × Family 第四讲文字稿

PPO × Family 系列课程的第一讲（概述课）系统性地讲解了决策智能的核心技术——深度强化学习，并深入浅出地介绍了最强大通用的算法之一——PPO。第二讲（解构复杂动作空间）和第三讲（表征多模态观察空间）分别从决策输出动作设计和决策输入表征建模问题展开，进一步深入拓展 PPO × Family 的各种知识与技巧。而第四讲，将会进入到 MDP 的第三大核心元素——奖励函数，从指导智能体探索和利用的角度，介绍奖励空间上的“两朵乌云”及衍生的“算法-代码-实践”知识。



(图 1：奖励空间上的两朵乌云——稀疏性和多尺度变化。)

## 4.1 奖励空间概述

如果想高效稳定地指导决策智能体的探索和利用，关键在于如何将自然决策问题的优化目标转化为适合于强化学习优化的奖励函数。换言之，奖励空间设计的终极目的，就是得到对于智能体决策行为轨迹客观的、合理的评价指标。而这一研究方向主要面临着两大难题，即所谓的“两朵乌云”：

- 奖励的稀疏性
- 奖励的多尺度变化

### 4.1.1 奖励的稀疏性

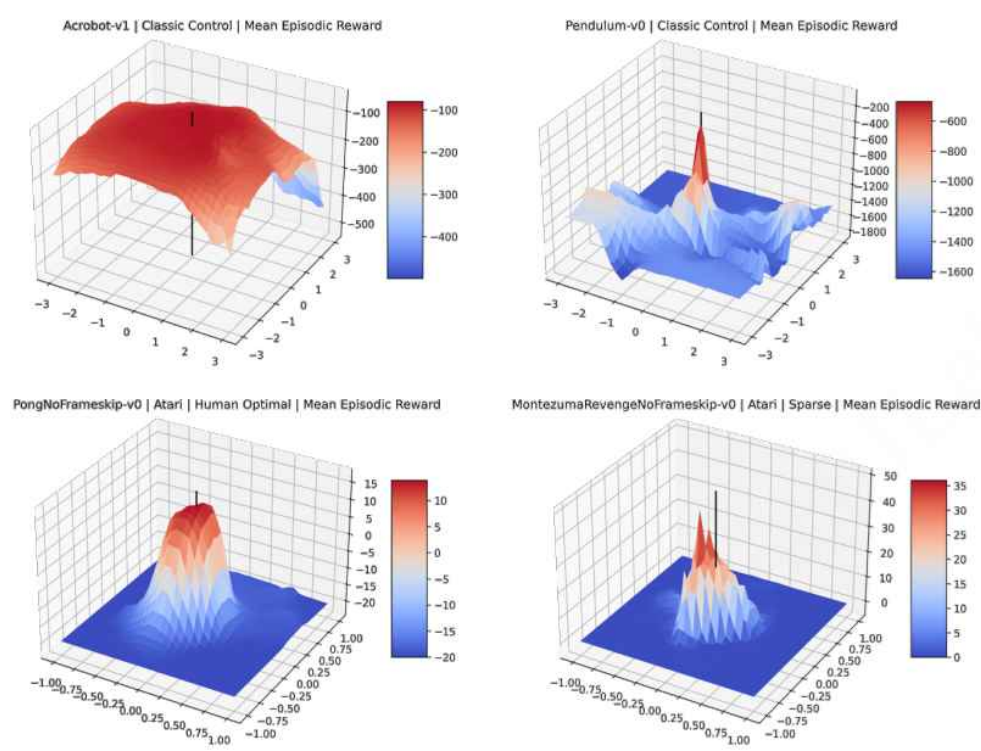
奖励的稀疏性是指：在智能体在和环境交互的过程中，并非每一次交互（每一个时间步）都能得到非零奖励，而是只能在少量关键的时间步能得到非零奖励。例如：

- 在棋类游戏中，最终的奖励只在棋局结束时给出（如获胜为 +1，失败为 -1），其他时间步奖励都为 0
- 在闯关类游戏中，奖励只在某些特定的时间点可能得到（如打败某个 boss、开启某个宝箱等）

这种稀疏奖励的形式非常不利于智能体对未知环境内容的探索。例如，对于闯关击败 boss 的任务，假设一个智能体初期一连串操作都很完美，但是最后却出现了微小的失误，进而导致失败。虽然这一系列动作从人类的角度出发可圈可点，但从智能体所获奖励的角度出发，这一系列操作和随机操作并没

有区别，因为所得到的奖励都是 0。因此，除非智能体在探索时，恰好出现了击败 boss 的完整决策序列，否则智能体就难以继续进步。

为了直观地理解稀疏奖励空间，可以参考下图中对于不同决策环境奖励空间的可视化：



(图 2：奖励空间的各种分布形态示意图。其中包含四种不同的决策环境，不同环境有不同的奖励函数分布，x 轴和 y 轴是降维后的状态与动作信息维度，z 轴是奖励大小维度 [1]。)

在图中，z 轴代表的是奖励的大小。可以看出：

- 对于左上角的环境而言，奖励比较稠密，而且不同状态动作信息下奖励的变化是平滑的。智能体可以循序渐进地利用奖励信号，不断寻找最优的策略；
- 对于其它的环境而言，奖励函数明显呈现阶跃式变化。在这种情况下，如果智能体初始在蓝色区域（即奖励接近为 0 的部分），那么智能体策略很难探索到有正向奖励的红色区域，因此其更新就可能陷入停滞。

上述问题对强化学习智能体的设计提出了更高的要求，需要智能体在探索时能够较好地适应这种稀疏的奖励。

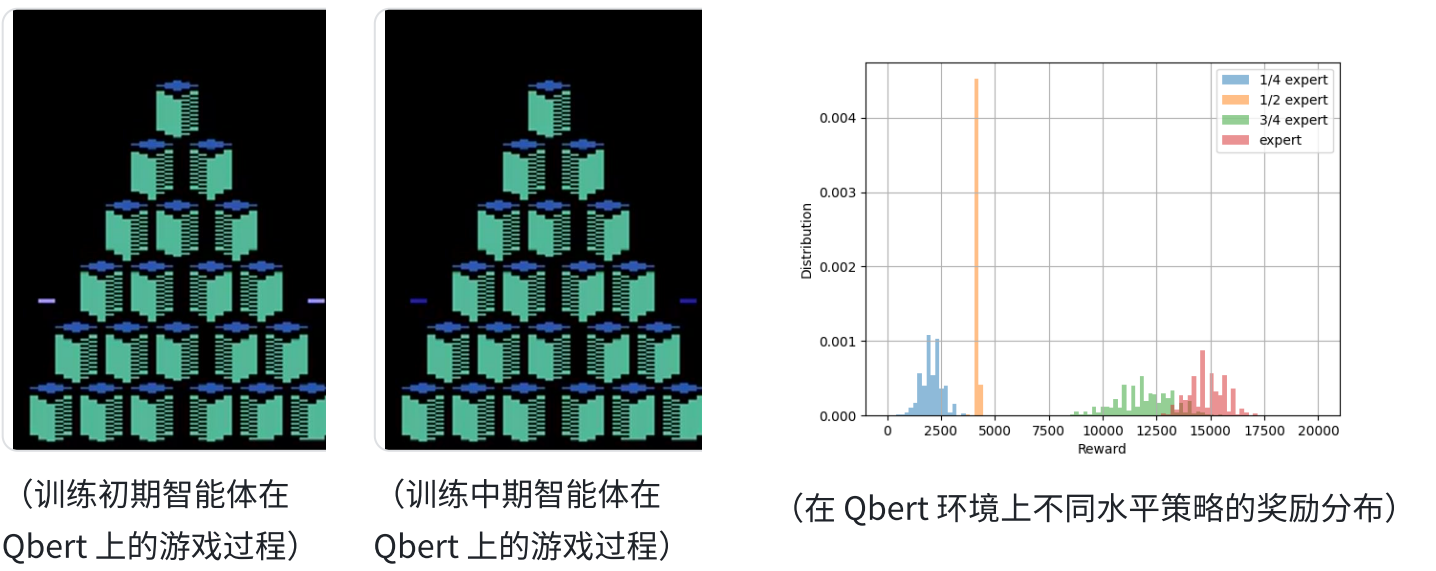
### 4.1.2 奖励的多尺度变化

奖励空间还可能存在多尺度变化的性质，具体来说，可以被概括为两大特点：

- 不同环境奖励取值的分布不同，取决于具体的决策问题。
- 同一环境的不同训练阶段，episode 的长度和奖励的分布都不同。

接下来以 **Atari 中的 Qbert 游戏环境** [2] 为例进行说明。Qbert 是一种在阶梯上跳跃闯关的游戏，智能体需要在躲避敌人的同时跳过所有阶梯才能进入下一关，而不同关卡间的分数差异可以达到数十倍。这就导致在训练初期和训练后期，智能体所获得的奖励在尺度上差别很大。如下左图为训练初期的游

戏过程，中图为训练中期的游戏过程。可以观察到关卡的内容和智能体的表现都有较大不同，训练初期的智能体无法通过第一关，因此分数显著低于 500 分。而训练中期智能体可以到达第二关，此时的分数已经上升到 500 分以上。



(图 3：Qbert 环境的奖励变化详解示意图)

实际上，在强化学习的不同训练阶段，智能体探索到的是不同的环境空间子集，因此智能体遇到的反馈和游戏机制都可能是不同的。具体来说，右图量化地展示了智能体在 Qbert 上不同训练阶段能够达到的奖励分布，其中 1/4 expert 表示能达到专家策略的累积奖励 1/4 水平的智能体。可以看到，接近于专家水平智能体的奖励分布在 15000 分左右，而 1/4 expert 的智能体只能达到 2500 左右的分数。此外，1/2 expert 水平的智能体能够达到的奖励几乎都在 4000-4200 分的区间内，这一部分包含超过总量 80% 的样本。这是因为 Qbert 中不同关卡的分数相差较大，1/2 expert 水平的智能体能够通过第一关却无法通过第二关。这种不同训练阶段之间的奖励分布差异，需要强化学习算法能够适应多尺度变化的奖励分布，从多尺度的奖励中学习知识，从而保持训练的稳定。

### 4.1.3 奖励塑形与模仿学习

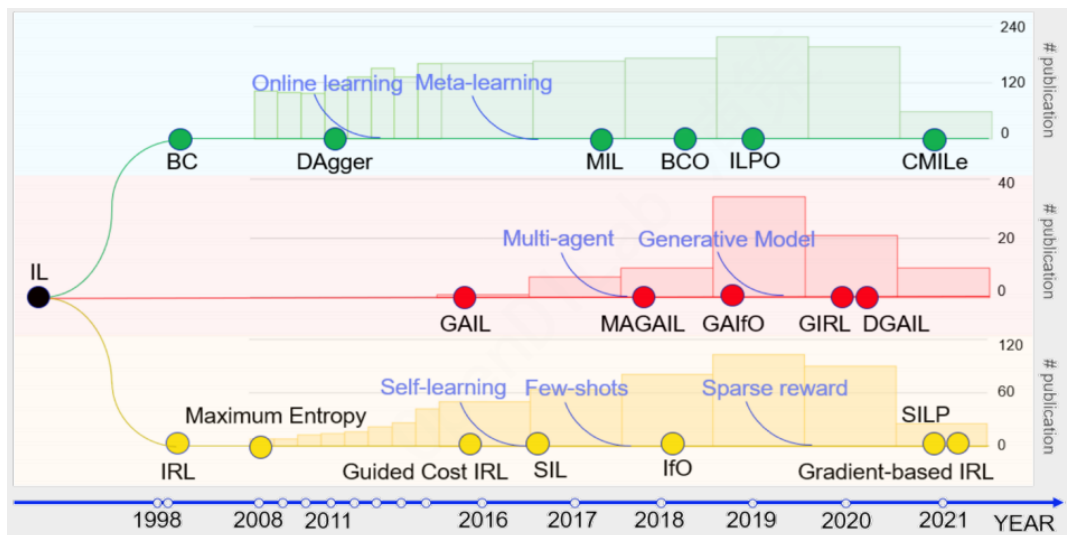
为了应对上述奖励空间的问题，在实践中，研究者们常使用奖励塑性（Reward Shaping）与模仿学习技术。

- 奖励塑性指的是利用特征工程的方法，人为地修改和设置奖励函数。例如在图 4 中展示的是，对于 DOTA2 游戏环境，OpenAI 设计强化学习智能体时所使用的的极为丰富奖励函数的设置。不过，奖励塑形部分工作不仅需要对环境有充分的认识，积累相当的领域专家知识；同时也需要结合实验效果细致调节每个奖励的大小，这样才能得到较好的奖励塑形结果，这个过程无疑是较为繁琐的。

Name	Reward	Heroes	Description
Win	5	Team	
Hero Death	-1	Solo	
Courier Death	-2	Team	
XP Gained	0.002	Solo	
Gold Gained	0.006	Solo	For each unit of gold gained. Reward is not lost when the gold is spent or lost.
Gold Spent	0.0006	Solo	Per unit of gold spent on items without using courier.
Health Changed	2	Solo	Measured as a fraction of hero's max health. <sup>‡</sup>
Mana Changed	0.75	Solo	Measured as a fraction of hero's max mana.
Killed Hero	-0.6	Solo	For killing an enemy hero. The gold and experience reward is very high, so this reduces the total reward for killing enemies.
Last Hit	-0.16	Solo	The gold and experience reward is very high, so this reduces the total reward for last hit to $\sim 0.4$ .
Deny	0.15	Solo	
Gained Aegis	5	Team	
Ancient HP Change	5	Team	Measured as a fraction of ancient's max health.
Megas Unlocked	4	Team	
T1 Tower*	2.25	Team	
T2 Tower*	3	Team	
T3 Tower*	4.5	Team	
T4 Tower*	2.25	Team	
Shrine*	2.25	Team	
Barracks*	6	Team	
Lane Assign <sup>†</sup>	-0.15	Solo	Per second in wrong lane.

(图 4：在 DOTA2 环境中设计智能体时所用的奖励塑形设置 [3]。)

- 另一方面，面对奖励空间的问题也可以采用模仿学习的方法。模仿学习的大体思路是：首先收集一批专家数据，然后让智能体直接或间接地模仿专家数据中的行为，这样就避免了奖励函数的设置难题。概括而言，模仿学习方法可以按照下图的标准进行分为三类：



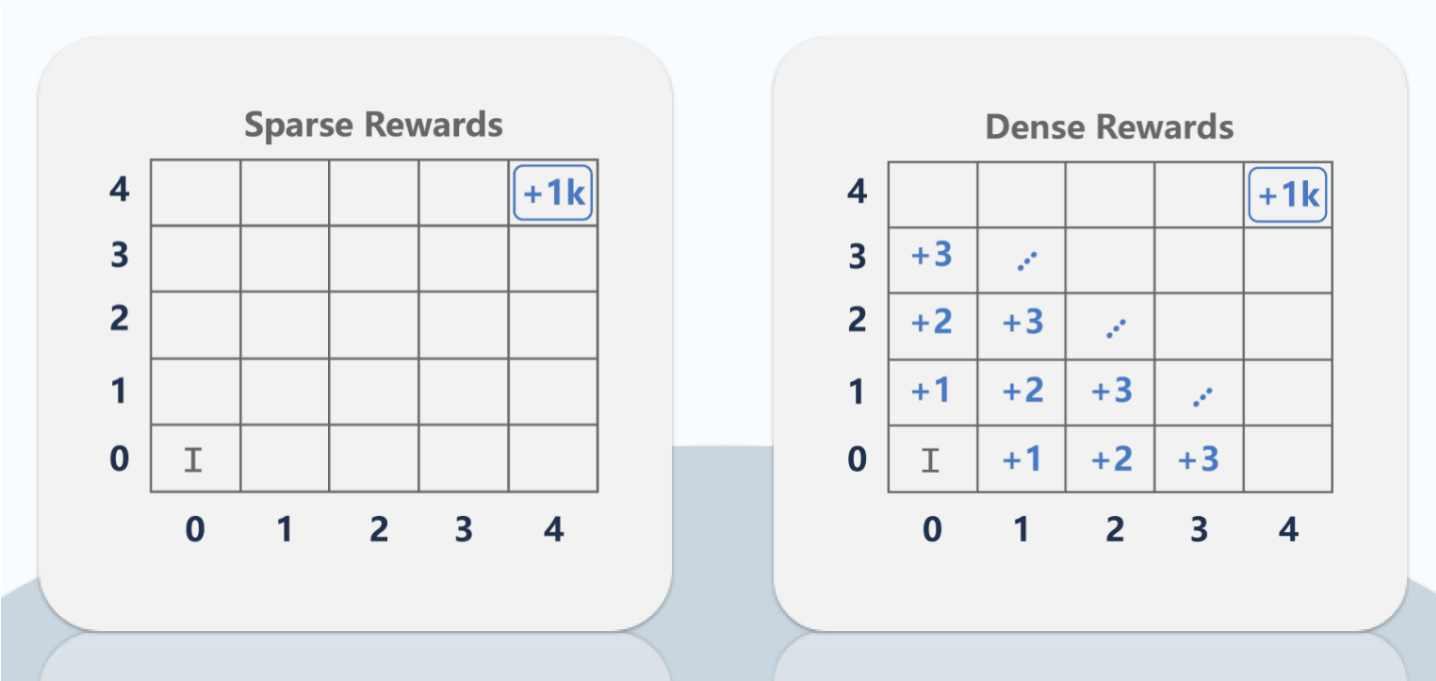
(图 5：模仿学习算法分类示意图 [4]。)

其中，第一类被称作行为克隆算法（Behaviour Cloning，BC），它的目标是直接模仿专家数据中的动作，从而达到学习目的；第二类被称作逆强化学习（Inversed Reinforcement Learning，IRL），它的目标是首先通过分析专家数据，推理出这个环境中的奖励函数，然后利用所学习到的奖励函数进行后续的强化学习；第三类方法则是结合了上述两种思想，提出了对抗模仿学习（Generative Adversarial Imitation Learning，GAIL）及相关的各种算法。

对上述模仿学习算法的各个子分支，更具体的讲解可以参考补充材料：[行为克隆](#)，[逆强化学习](#)。

## 4.2 稀疏奖励

如果没有足够的领域知识去做 Reward Shaping，也没有大量的专家数据去做模仿学习，那应该如何解决奖励空间的稀疏性问题呢？首先，对于稀疏和稠密奖励的区别，本节课使用同一个环境的不同形式的奖励空间来进行说明，具体地，以下图中的格子世界（Grid World）为例。对于左边稀疏奖励的情景，智能体只有到达右上角的目标点才会获得 +1k 的奖励，因此在智能体训练的前期，它只能漫无目的地随机探索，希望能够类似“瞎猫碰上死耗子”，通过不断地随机尝试“撞到”目标点，而在右边稠密奖励的情景，可以定义智能体与右上角目标点之间的距离为奖励，距离越近奖励越大，那么这种稠密的奖励就可以很方便地指导智能体进行学习，从而使整个优化过程变得更加高效。



（图6：格子世界中稀疏和稠密奖励的定义原理图。）

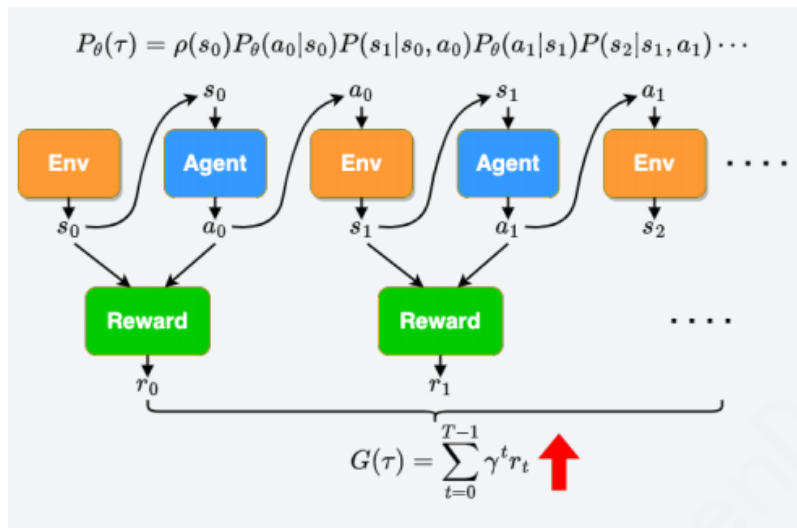
更进一步地，虽然对于格子世界这样的简单环境，可以轻而易举地定义出某种稠密奖励，但是对于更一般的决策问题，该如何定义这种奖励？那么这里就可以引入一种重要的奖励设计机制——好奇心机制来去设计相应的稠密奖励。

### 4.2.1 引言：什么是好奇心驱动的探索方法

总体来说，好奇心机制就是去设计一种新的内在奖励（Intrinsic Reward），去激发智能体的好奇心鼓励探索。那为什么要选择好奇心机制？因为一般情况下，那些智能体没见过的状态和动作，可能蕴含更高的价值或是更可能存在一些新奇未知的内容，所以算法设计时就希望智能体能够更多地朝着这个方向去探索。

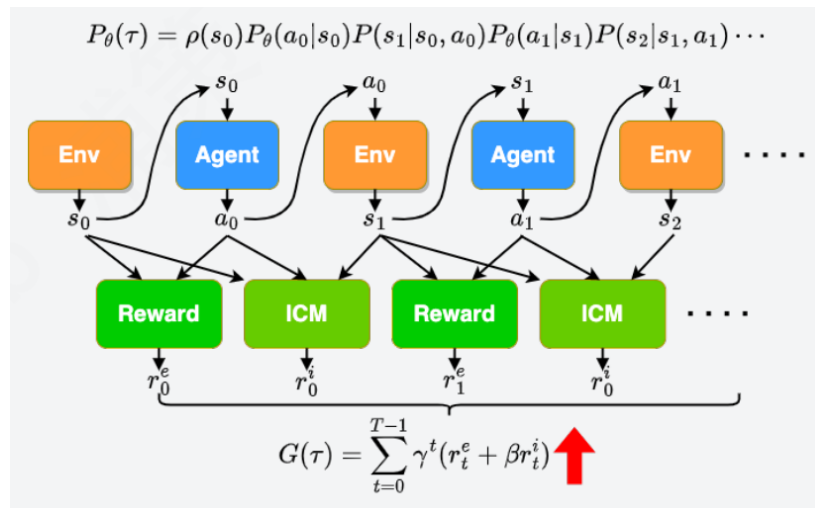
如图7所示，首先给出了经典的策略梯度算法优化时使用的轨迹结构，从轨迹生成的角度来讲，策略梯度算法最终的优化目标为：找到使得整条轨迹的回报函数最大的策略。





(图 7：策略梯度算法优化时使用的轨迹结构和优化目标。)

然而如上所述，对于稀疏奖励的环境，如果智能体探索很困难，长时间无法找到奖励更高的动作，策略的更新就可能陷入停滞。为了解决这一问题，就需要在轨迹中引入内在奖励这一项来激发智能体的“好奇心”，从而促使智能体不断探索新的状态，找到更优的策略。简单而言，内在奖励的目标就是促使智能体探索尽可能多样化的状态和动作空间。对于一个状态-动作对，其越新颖，对应的内在奖励就越大，反之则越小。在添加了内在奖励之后，策略梯度算法的轨迹结构和优化目标就变为如图8所示：



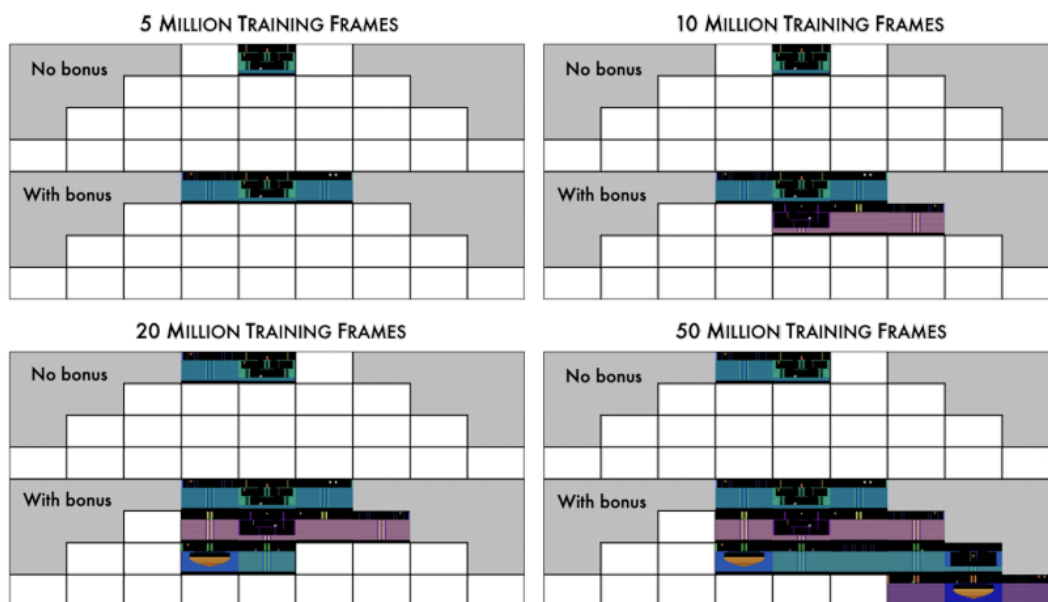
(图 8：添加内在奖励后，策略梯度算法优化时使用的轨迹结构和优化目标。)

实践中，可以使用不同的方法从轨迹信息中提取新的内在奖励。这里图中就使用了接下来要介绍的 ICM (Intrinsic Curiosity Module) [5] 方法作为示例，从  $s_0, a_0, s_1$  中提取出新的内在奖励  $r_0^i$ ，然后将新的内在奖励和原有的外在奖励 (Extrinsic Reward) 通过线性组合的方式结合到一起，作为最终的奖励信号指导强化学习智能体的学习。这样一来，原本的稀疏奖励就变成了稠密的奖励形式，而稠密奖励可以为智能体提供更多的反馈信息，智能体会被促使不断探索新的状态，避免了训练更新停滞不前的情况。

#### 4.2.2 理论：好奇心驱动的经典探索方法 (ICM, RND)

基于好奇心机制的思想，具体应该如何设计算法呢？接下来本节课将首先介绍两种朴素的设计方法：

- **第一种是基于计数 / 伪计数的算法。**这里以下图中展示的《蒙特祖马的复仇》[2] 环境为例：根据游戏机制，该环境共有 24 个房间，那就可以用房间作为基本单位，每个房间统计智能体到达相应状态的数量，计数的数量越大，可以认为智能体对这些状态越熟悉，所以新颖程度应该越低，反之，计数越小，则可以认为这个房间对应的状态是更加新颖，更加值得探索的。除此之外，还可以利用哈希等伪计数方法提升性能，从而使得计数方法能够扩展到连续状态空间的环境中。



(图 9：基于计数的探索算法在《蒙特祖马的复仇》环境上的示意图。在训练初期（如左上角所示），对于探索了三个房间的状态，算法认为是较为新颖的，因此给予内在奖励，而对仅探索了一个房间的状态不施加奖励；对于训练后期（如右下角所示），算法只对探索了非常多房间的状态给予内在奖励，对仅探索少量房间的状态不施加奖励。)

- **第二种是基于神经网络预测误差的算法。**因为神经网络有一个很好的性质：如果神经网络见过一个数据很多次，那么它会倾向于记住这个数据，相应的预测误差就会比较低，而对于那些神经网络没见过的数据预测误差则会比较高，因此这里可以利用这个预测误差作为一种新颖性度量。举例来说，这类算法可以设计一个神经网络，输入当前时刻的状态  $s_t$  和动作  $a_t$ ，去预测下一个状态  $s_{t+1}$ ，预测的状态和真实的状态进行比较计算损失函数。如果神经网络的预测比较准确，就意味着它已经多次见过类似的数据，进而说明该状态的新颖性较差，应当给予较低的内在奖励；反之如果预测不准，说明该状态新颖性较强，应当给予较高的内在奖励。

然而，上述两种方法都是比较朴素的，如果在一些复杂的环境中使用这两种算法，会面临着两个问题：

- **首先，高维表征是难以计数或预测的。**对于高维的状态空间（比如图片状态空间），如果以每一个像素点为基本单位去计数，那么它带来的计算和内存开销是巨大的；而如果使用神经网络去进行预测，则需要设计较为复杂的图片编码网络来特殊处理这种模态的数据，这也是相对比较困难的。
- **其次，这两种方法对状态数据中的噪声都比较敏感。**如下图所示，如果在图片状态中加入噪声，那么显然无论是计数还是预测，都难以针对这种情况给出合理的新颖性度量，这些噪声信息会带来非常严重的副作用。虽然在真实场景中，大概率不会出现如此极端的噪声，但由于实际决策问题的复杂性，在状态-动作对数据中，经常会出现不同程度的干扰信息，所以类似噪声这样的情况仍然是一个重要的、亟待解决的问题。



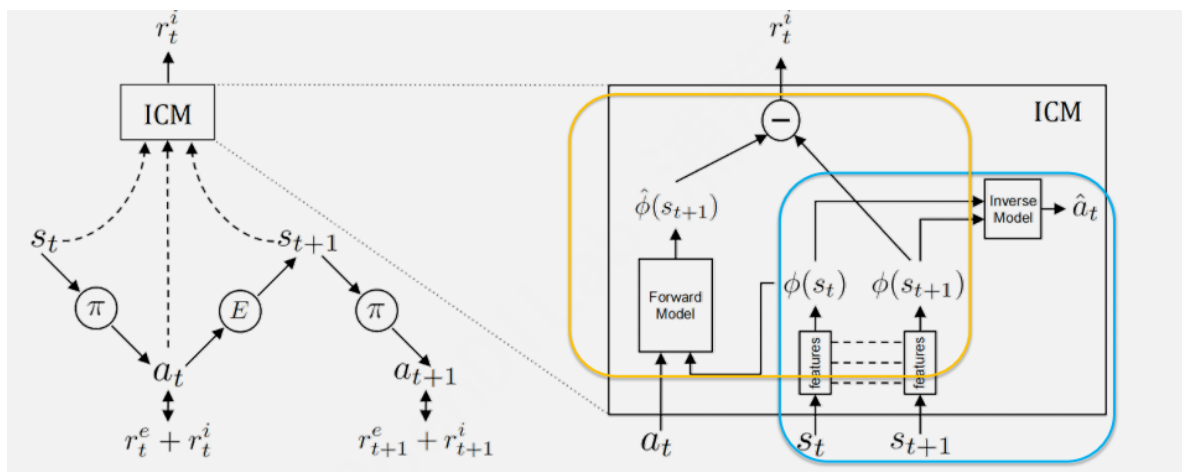
(图 10: VizDoom [6] 环境原始图片状态信息和添加噪声的图片状态信息示意图。)

## ICM

ICM [5] 是第一篇真正将基于好奇心机制的内在奖励，成功定义并应用在强化学习方法中的研究工作。这里算法设计的关键点是：**只有好奇心是不够的，还要知道什么对决策是重要的**。要想更好地设计内在奖励函数，一味追求状态的新颖是不够的，更重要的是要追求对于决策行为有积极意义的新颖状态。总结来说，环境的状态-动作信息可分解为三个部分：

- （需要关注）智能体动作直接控制的内容（例如：智能体开枪射出子弹）。
- （需要关注）虽然不受智能体动作控制，但是会对智能体决策造成影响的内容（例如：怪物的移动）。
- （需要忽略）本质上对决策无效的信息（例如：背景、噪声）。

通过增大对环境信息前两部分的关注，减小甚至忽略第三部分的信息，ICM 能够生成更合理的内在奖励。具体算法原理如下图所示：



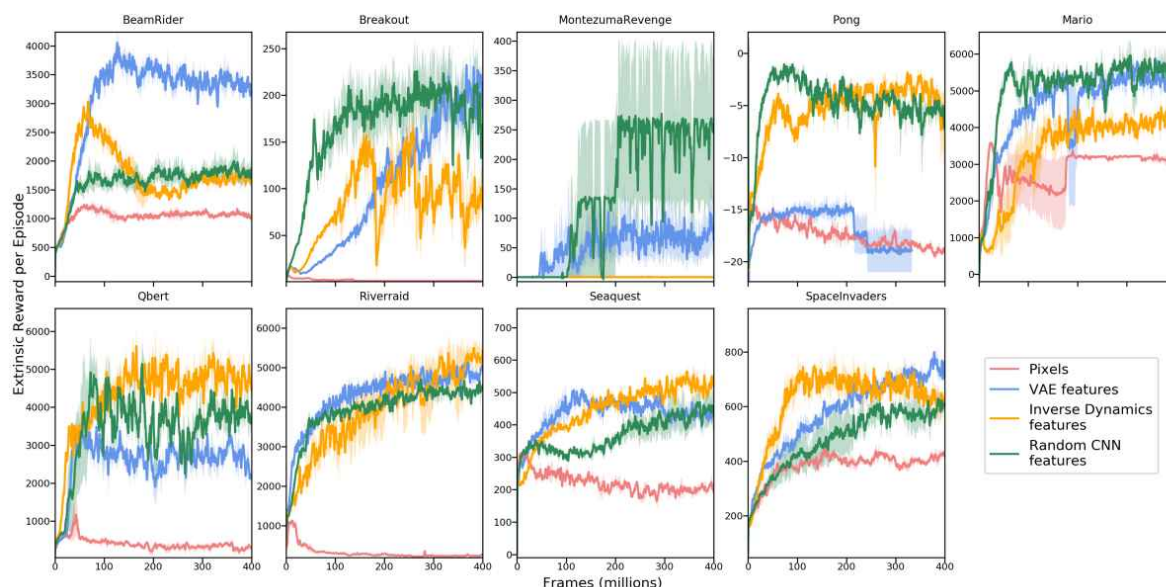
(图 11: ICM 算法原理示意图。)

本质上来讲，好奇心机制系列算法还是要设计一种内在奖励，并把它通过线性组合的方式与原有的外在奖励组合到一起，构成最终的奖励信息用于强化学习训练，即图11的左半部分。本节课将进一步详细阐述图11的右半部分，深入理解 ICM 的工作原理：



- 最朴素的思路，这里可以利用 MDP 的状态转移来设计神经网络预测问题，即输入  $s_t$  和  $a_t$  预测  $s_{t+1}$ ，并将预测值和真实的  $s_{t+1}$  作比较得到预测误差。这个误差越大，就认为智能体之前没有见过这样的状态-动作对，即这种数据更新颖，更加值得探索，于是给予更大的内在奖励，反之，就会给予较小的内在奖励。但这种思路依然无法排除噪声等信息的干扰，于是，这种朴素的方案就需要升级为 ICM 的设计思路。
- 对于黄色框内的设计（又称“前向动力学模型”），ICM 方法与上述介绍的朴素预测方法类似，都是基于  $t$  时刻的状态和动作预测  $t+1$  时刻的状态。不同点在于这里并非直接预测  $s_{t+1}$ ，而是预测它经过一个函数变换后的特征  $\phi(s_{t+1})$ 。而这个函数变换的目的就是尽可能地消除背景、噪声等和决策动作无关的状态部分，提取出一种更高维，更干净的表征用于好奇心机制。
- 为什么  $\phi(s_{t+1})$  可以表示和决策动作无关的状态部分？这里就涉及到蓝色框内的设计（又称“逆向动力学模型”），它的目标是给定  $s_{t+1}$  和  $s_t$ ，预测智能体选取的动作  $a_t$ ，即从相邻的两个状态帧去预测中间实际执行的动作。通过这样的预测任务，就可以使得  $\phi(s_{t+1})$  中提取到更多与决策动作相关的信息，进而一定程度上消除背景、噪声等和决策无关部分信息的影响。
- 通过结合前向和逆向动力学模型，就可以在忽略掉一些无用信息的同时，保证算法对于重要决策信息足够的好奇心，从而使得强化学习算法可以更加高效地探索和优化。

更进一步地，是否存在其他的状态表征  $\phi(s_{t+1})$  适合于基于好奇心机制的探索算法？ICM 中使用了逆向动力学模型（IDM Model）来提取和决策动作相关的信息，但其实还存在很多不同的表征学习方法。ICM 的后继工作 [7]（中文解读[博客](#)）就在 Atari 环境中给出了经验性的实验对比结果，如下图所示：



（图 12：不同的状态表征学习方法对基于好奇心机制的探索算法的影响对比图。）

通过上述实验，概括而言，适合于探索的表征信息应当具有三个特征：

- **紧凑**：特征应该易于神经网络进行学习和建模，最好是低维的特征向量，并且能够尽可能地过滤掉原始状态动作空间中和探索无关的部分。
- **充分**：特征应该包含用于决策的所有重要信息。否则，智能体可能由于缺乏信息而做出错误决策。

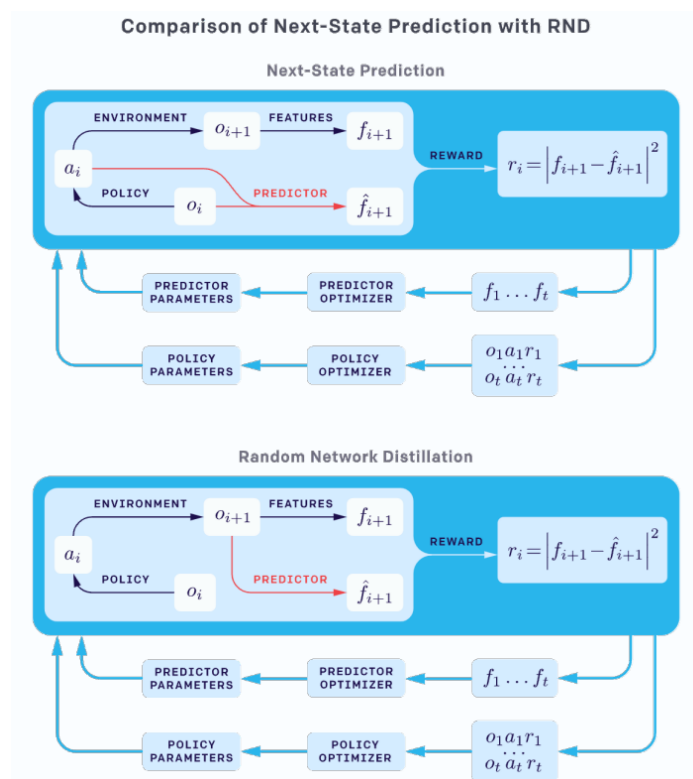
- **稳定**：非平稳（Non-stationary）的奖励会使强化学习训练过程变得更困难。因为强化学习是一个在线训练过程，需要不断在线地收集数据，而内在奖励模型往往也是一个在线训练过程，这会导致内在奖励的取值范围或数据分布也可能会产生显著的变化，从而增大了强化学习算法训练时的非平稳性。因此，好奇心机制相关的探索算法对于超参数会比较敏感，实际使用中往往需要积累一定的调参经验。

## RND

在介绍完 ICM 的设计动机和算法原理之后，本节课将会介绍这类算法进一步的改进空间。具体来说，ICM 系列算法仍然存在某些可能失败的场景，因为 ICM 需要去学习前向和逆向动力学模型，所以 ICM 其实和 MDP 中的状态转移函数非常相关，因此产生了两种主要的失败情形：

- **状态转移非常复杂，而神经网络容量有限**。这样一来，前向动力学模型就难以拟合出合理的状态转移函数，也就无法给出合理的内在奖励值。
- **状态转移不是一个确定的映射，而是一个随机函数**。例如在经典的 noisy-TV 问题中，智能体的目标是走出迷宫，然而迷宫中的某些区域会随机生成一些图片或者视频。由于 ICM 前向动力学模型很难拟合这些随机的画面，所以会对这些位置给出很高的内在奖励，因此智能体就会一直较为关注这些区域，从而沉迷于此无法前进。

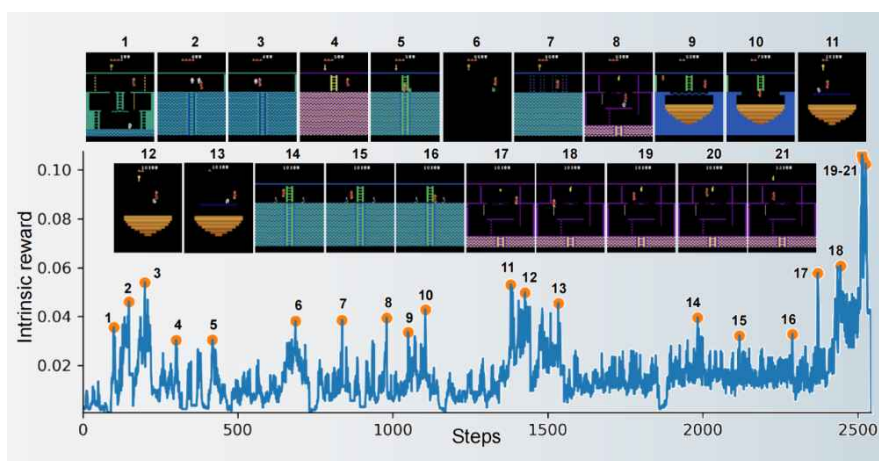
为了延缓甚至解决上述问题，第二类经典的内在奖励探索算法——Random Network Distillation (RND) [8] 算法就应运而生，这类算法的设计原理就是不去构建前向动力学这样的预测问题，而是通过单帧状态信息来构造预测误差，从而得到另一种新颖性度量。理论上讲，这类方法就是激励智能体去探索更多的新颖状态，构建一个只和当前状态相关的随机蒸馏问题，具体的算法框架如下图所示：



(图 13：ICM 和 RND 的算法原理对比图。上半部分是类似 ICM 中前向动力学预测问题的设计，下半部分是 RND 所使用的基于单帧信息的随机蒸馏预测问题设计。)

- 图中上方的部分类似 ICM 算法，使用基于下一状态预测（next state）的算法设计。这种方法的核心在于根据当前状态和动作预测下一帧的状态，即和 MDP 中的状态转移密切相关，使用这个预测误差来作为新颖性度量。
- 图中下方的部分则是 RND 提出的新的算法设计，本质上讲，它是通过新增一个神经网络让当前的状态“自己预测自己”。具体来说，在 RND 算法中有两个神经网络： $\hat{f}$  和  $f$ 。其中第一个网络是随机初始化且不可训练的，一般称作目标网络；第二个网络则是需要训练优化的网络，即预测器神经网络。在训练阶段，对于单帧的状态信息，RND 将其输入目标网络生成固定的目标向量，并同时还将其输入预测器网络得到特征向量，优化目标设定为最小化两个向量的误差  $\|\hat{f}(s_t) - f(s_t)\|^2$ ，即可以看作将随机初始化的目标网络蒸馏到预测器网络中去。而在推理阶段，假如该状态已经多次出现，那么相应的随机蒸馏训练过程也已多次执行，预测误差也就自然比较小，所以这个误差可以作为一种状态新颖性的度量，从而产生内在奖励。
- 为什么使用随机初始化的网络作为目标生成器？这里利用了神经网络的一个性质：一个随机初始化的神经网络可以将输入的状态信息较为均匀地映射到一个特征空间，保证相关信息被充分地提取和使用，另外这个目标是训练过程中保持固定的结果，且不需要任何额外的训练开销。

下图就是 RND 算法在《蒙特祖玛的复仇》这个探索困难环境上给出的内在奖励示意图，遇到不同的房间，RND 模型都会给出新的指引来帮助强化学习。RND 是第一个不依赖人类数据，不使用特殊的数据处理，就能在《蒙特祖玛的复仇》环境上探索出最终胜利结果的算法。




（图14：RND 算法在《蒙特祖玛的复仇》环境的不同房间给出的内在奖励示意图。）

RND 的算法原理看似简单，但其实它的设计动机却隐含着很深刻的算法思想。这里可以通过分析预测误差的影响因素来解密其内核。好奇心机制系列算法希望使用某种神经网络的预测误差来衡量状态的新颖性，而预测误差则主要以下列四个方面构成（中文解读[博客](#)）：

- **训练数据的新颖性。**这里对应到强化学习中未知的状态或状态-动作对，这正是探索算法所期望的部分。
- **状态或目标函数的随机性。**在预测问题中，预测函数的随机性，预测函数的自变量（比如状态、动作）的随机性都会影响最终的预测效果，比如像 ICM 遇到 noisy-TV 问题时，不必要的随机性误差就会相对很大，进而掩盖了环境状态真实的新颖性。而在 RND 中，由于无需预测下一个状态，所以状态转移随机性的影响也就自然消失了。

- **网络模型表达能力不够。**内在奖励本身的训练特性也是非常重要的因素，如果神经网络的表达能力不够就会严重影响最终的结果。但对于 RND 算法，由于预测的目标也是相同结构神经网络提取的随机特征，因此理论上讲，一定可以训练出一个预测器网络完美地拟合所有数据。因此对于 RND 算法，神经网络模型容量也不是问题。
- **模型的学习过程困难。**由于训练所用的数据的动态性，或是优化方法不够好，奖励模型很容易收敛到当前数据的局部最优值，导致具体使用时的预测误差额外偏高。而 RND 方法也能较好地解决这个问题。

综上所述，由于 RND 算法能够减小上述四个方面中后三个方面带来的误差，因此这种预测误差更能反映训练数据本身的新颖性，从而也就更适用于构建内在奖励来帮助强化学习探索。

 一个额外的问题是：如果构建一个自监督学习任务，比如学习一个自编码器（AutoEncoder），将状态映射到隐空间上，然后根据重建误差来给出内在奖励，这种设计会有什么问题？

首先，这个问题比 RND 的预测问题复杂，需要 encoder 和 decoder 两个可训练模块；更重要的是，由于强化学习的数据是在线收集的，由于收集数据的策略一直在变化，因此所得到的数据分布也在不断变化。这样一来，模型优化的目标也就一直随之变化，从而导致了非平稳问题；而对于 RND 而言，优化的目标始终是固定的（即随机初始化的目标网络  $\hat{f}$ ），因此优化时稳定性更好。所以，RND 这种训练方式虽然看起来很简单，但却是一种非常优秀的内在奖励学习方式。

### 4.2.3 代码：奖励模型训练技巧面面观

介绍完 ICM 和 RND 这两种经典的内在奖励训练方法，接下来的部分将会介绍它们如何与 PPO 算法相结合。下图展示了将这两种方法应用到 PPO 算法中的伪代码，奖励模型相关的伪代码为其中蓝色标注的部分，完整的代码示例可以参考[课程仓库](#)以及“[算法-代码](#)”[注解文档](#)：

**Algorithm 1** RND + PPO (RND related is highlighted in blue)

- 1: Input: initialize policy parameters  $\theta$  and value parameters  $\phi$ . Training epochs per collect  $E$ , max number of episodes  $K$ , trajectory length  $T$ , batch size  $B$ . RND: initialize prediction network parameters  $\hat{\chi}$  and fixed target network parameters  $\chi$ , number of prediction optimization steps per collect  $N_{\text{predict}}$ .
- 2: **for**  $k = 0, 1, 2, \dots, K$  **do**
- 3:   Collect a set of trajectories  $\mathcal{D}_k = \{ \{s_t, s_{t+1}, a_t, e_t\} \}$  by running policy  $\pi_{\theta_k}$  interaction with environment.
- 4:   Normalize the observation in  $\mathcal{D}_k$ , then obtain the normalized counterpart:  $\hat{s}_t, \hat{s}_{t+1}$ .
- 5:   **for**  $j = 1$  **to**  $N_{\text{predict}}$  **do**
- 6:     Optimize  $\chi_j$  wrt distillation loss  $\| \hat{f}_{\hat{\chi}}(\hat{s}_t) - f_{\chi}(\hat{s}_t) \|^2$  using Adam.
- 7:   **end for**
- 8:   Calculate intrinsic reward  $i_t = \| \hat{f}_{\hat{\chi}}(\hat{s}_t) - f_{\chi}(\hat{s}_t) \|^2$ . Then normalize the intrinsic reward and obtain the normalized counterpart:  $\hat{i}_t$ .
- 9:   Calculate augmented reward  $r_t = e_t + \beta \hat{i}_t$  and obtain the augmented trajectories  $\hat{\mathcal{D}}_k = \{ \{s_t, s_{t+1}, a_t, r_t\} \}$ .
- 10:   Compute trajectory target return estimates  $\hat{R}_t$  on  $\hat{\mathcal{D}}_k$ . (e.g. n-step TD or other methods)
- 11:   Compute advantage estimates  $\hat{A}^{\theta_k}$  with value  $V_{\phi_k}$  on  $\hat{\mathcal{D}}_k$ . (e.g. GAE or other methods)
- 12:   **for**  $e = 0, 1, \dots, E-1$  **do**
- 13:     **for** minibatch  $b \in \hat{\mathcal{D}}_k$  **do**
- 14:       Update the policy by maximizing the PPO-Clip objective with Adam:
 
$$L_{\theta} = \frac{1}{B \cdot T} \sum_{\tau \in b} \sum_{t=0}^{T-1} \min(r(\theta) \hat{A}^{\theta_k}(s_t, a_t), \text{clip}(r(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}^{\theta_k}(s_t, a_t))$$

$$r(\theta) = \frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_k}(a_t | s_t)}$$
- 15:       Fit the value by regression on mean-squared error with Adam:
 
$$L_{\phi} = \frac{1}{B \cdot T} \sum_{\tau \in b} \sum_{t=0}^{T-1} (V_{\phi}(s_t) - \hat{R}_t)^2$$
- 16:     **end for**
- 17:   **end for**
- 18: **end for**

(图15：PPO + RND 算法训练流程伪代码示意图，其中蓝色标注的部分为奖励模型相关的伪代码。)



另外，在 ICM 和 RND 之后，还衍生出了一系列新的内在奖励相关算法，比如 NGU [9] 和 Agent57 [10]，有兴趣的读者可以参考相关[博客](#)进一步深入了解。

#### 4.2.4 实践：使用 PPO 来探索各种迷宫难题

基于上述的理论和代码知识，本节课在 MiniGrid [11] 这个环境中进行稀疏奖励的实践尝试。

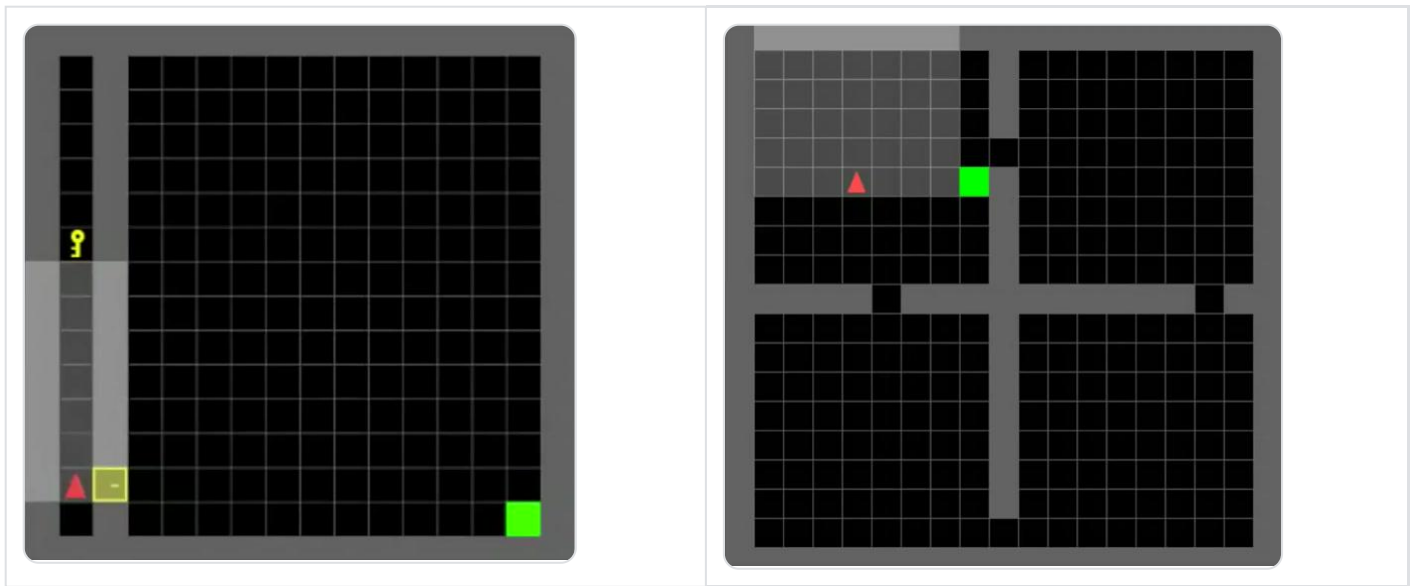


(图 16: MiniGrid 环境的各类子环境示例图。)

MiniGrid 环境中各式各样的子环境类型如上图所示，这些任务可能需要智能体拿到钥匙开启门锁，可能智能体需要跨越一系列障碍等等。其中奖励函数的详细定义如下：

- 智能体（红色箭头）只有在到达目标点（绿色方格）时才获得大于零的奖励，具体的数值由达到目标所用的总步数决定，在这之前奖励都是零。因此，这个环境是一个非常典型的稀疏奖励环境，此外，不同的关卡还需要累积不同的知识来帮助探索。
- 智能体只能观察到部分的迷宫空间，也就是图中明亮的部分。因此智能体需要不断探索迷宫中不可见的部分，进而找到通关方案。

结合 PPO 与本节课学习的 RND 和 ICM 算法，就可以成功地解决这些稀疏奖励问题（如下方视频所示）。不过 ICM 和 RND 算法其实各有千秋，ICM 可以通过逆向动力学模型去忽略部分跟决策动作无关的信息，而 RND 可以更少地受到各类随机性因素的影响，所以在实践中，需要去结合具体 MDP 的性质来选择使用。



(视频1: MiniGrid 环境中训练收敛的智能体行为视频 ([链接](#))。)

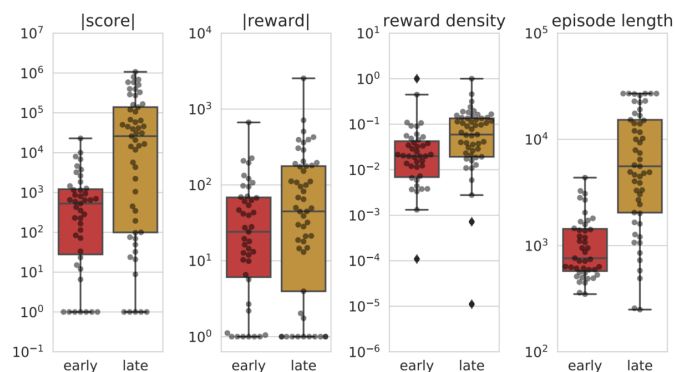
## 4.3 多尺度奖励

在介绍完稀疏奖励的相关问题之后，本节课进入奖励空间所面临的下一个难题——多尺度奖励问题。

### 4.3.1 引言：产生多尺度奖励的原因及其问题

具体来说，通过在 Atari 环境上的定量分析，可以观察到多尺度奖励问题在不同指标上的现象和特点。下图展示了四种指标，分别是游戏得分、游戏得分经过奖励塑形（reward shaping）后得到的奖励函数、奖励函数的密度（reward density）和每局游戏的长度（episode length）。图中的每个黑点代表某个子环境在这一指标下的具体取值，从中可以看到：

- 不同子环境之间的指标差异非常大，甚至有些指标下可以达到 4-5 个数量级。
- 对于同一个环境，由于强化学习主要是在线训练，需要不断地探索和利用，而且 Atari 游戏大多数是闯关形式，因此在训练的前中后期，智能体所面对的游戏内容和游戏机制都差异很大，导致图中前期（early）和后期（late）的指标也出现巨大差异，进而使得强化学习优化变得更为困难。

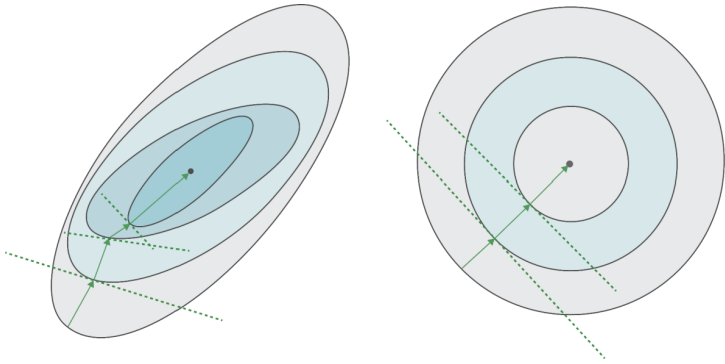


(图 17: Atari 环境中不同游戏的各个奖励指标分布图。)

首先，可以从一下两个角度来理解多尺度奖励为什么会给深度强化学习带来优化上的挑战：

从深度学习角度来说，在更新参数时，如果无法确定损失函数的变化范围，则无法设定一个统一的、合理的学习率，需要在训练过程中根据相关数据分布范围不断精调学习率，稍有不慎就会使网络的优化陷入困境。

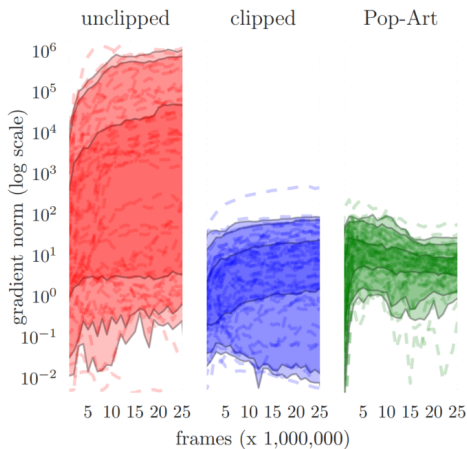
更进一步地，当神经网络反向传播所得梯度矩阵的条件数较高时（ill-conditioned），即在各个方向上变化不规律，就很难用当前的局部信息（也就是梯度）去比较准确地预测最优点所在的位置，只能一步步缓慢的逼近最优点，从而需要更多的迭代次数。如下右图所示，当条件数较低时，各个位置上最优点的方向基本一致，此时较大的梯度依然走在正确的方向上，产生异常结果的可能性较低。然而如下左图所示，当条件数较高时，过大的梯度更新反而会更有可能会远离最优点，陷入某些优化困难的局部。此时为了能够稳定地优化网络，就需要在训练的过程中精调学习率，而这在实际问题中往往是难以实现的。而在强化学习中，例如 PPO 算法，如果奖励函数范围变化很大，值函数（value function）和优势函数（advantage function）都会因此受到影响，进而会导致损失函数比较病态。



（图 18：左图为 ill-conditioned 网络更新过程；右图为非 ill-conditioned 网络更新过程。）

从强化学习角度来说，由于它的在线训练特性，所以智能体往往需要先奖励较低的范围里表现出色，才能抵达奖励较高的范围，此时很难设定全局统一的奖励变化范围和归一化技术。而且强化学习是并行地收集多个环境中的轨迹，导致一个训练批次（batch）中包含了得分高的轨迹和得分低的轨迹。此时得分高的轨迹就会造成更高的损失，进而主导整个训练过程，导致训练的不稳定。

下图展示了不同处理方式下，在整个训练过程中，强化学习所用的神经网络梯度的 L2-norm 的分布范围。可以看到没有奖励裁剪（unclipped）时，不同环境的梯度分布范围差异巨大，甚至达到了 4-5 个数量级。而经过奖励裁剪（clipped）和 PopArt 后，就能将梯度的 L2-norm 控制到合适的范围中。



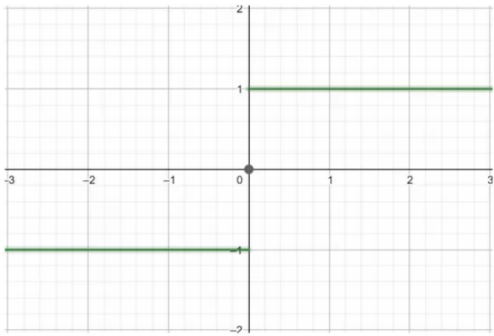
（图 19：不同处理方式下的神经网络梯度的 L2-norm 分布范围。）

### 4.3.2 理论：解决多尺度奖励的三大经典方法（Reward Clip, PopArt, Value Rescale）

#### Reward Clip

2015 年 Double DQN（DDQN）[12] 中提出了 **reward clip** 的概念，直接将奖励用阶跃函数 sign 限制到 1, 0, -1 上，如公式（1）所示。而在 Atari 环境上，由于几乎没有绝对值小于 1 的奖励，此时的 sign 函数相当于将奖励裁剪到范围 [-1, 1] 上。其目的是减弱奖励中的极值带来的影响，提升算法的鲁棒性。

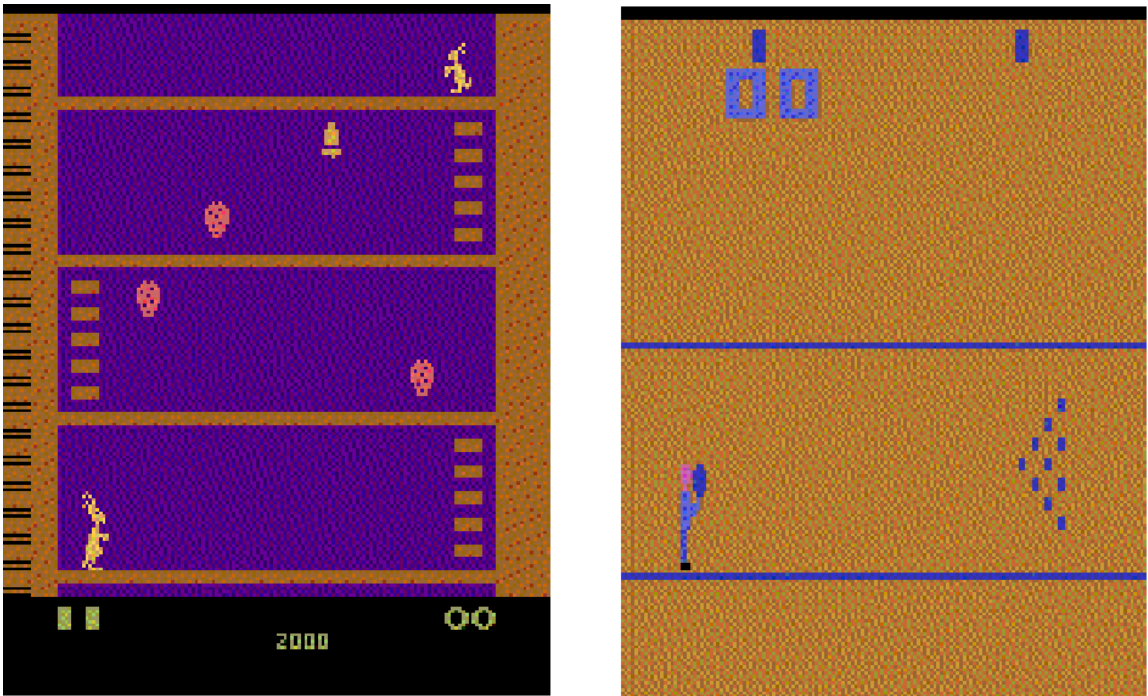
$$\text{sign}(\text{reward}) = \begin{cases} 1 & \text{reward} > 0 \\ 0 & \text{reward} = 0 \\ -1 & \text{reward} < 0 \end{cases} \quad (1)$$



（图 20：sign 函数效果示例图。）

此外，DDQN 中还提出了 **error clip 技巧**，将 TD error 裁剪到 -1 和 1 之间，从而提升算法的稳定性。不过，对于 error 的裁剪实际上就是某种梯度裁剪的特殊形式，本质上还是为了避免反向传播时梯度过大。

然而，暴力的奖励裁剪直接忽略绝对值大于 1 的奖励具有的信息，**可能会改变游戏奖励的含义，使得智能体无法分辨不同水准的行为，导致最优决策行为的改变**。例如，在 Atari 的 Bowling 游戏中，只击中一个球瓶和同时击中 10 个球瓶，经过裁剪后的奖励都为 +1，此时智能体无法区别这两种行为的好坏，最终只能学到每次保证击倒一个瓶子的策略，而很难学习到一次投球全中这样的行为。



（图21：暴力奖励裁剪影响最优决策行为的相关 Atari 环境示意图。）



## PopArt

除了暴力裁剪，标准化也是一种常用的处理这类问题的手段。这在很多监督学习任务中是很常见的做法，因为监督学习往往使用固定的数据集，那就可以从中提取到整个数据集准确的均值和方差。但因为强化学习训练是一个在线过程，没有这样固定的数据集，数据分布是不断变化的，所以均值和方差的统计就很困难。一个进阶的做法是采用指数加权平均（EMA）的方式来动态确定均值和方差，但由于数据的动态性，这样的做法会在强化学习中引入新的非平稳问题，进一步加剧强化学习训练的波动性。



$$\tilde{Y}_t = \Sigma_t^{-1}(Y_t - \mu_t)$$

（图22：金融交易问题中使用指数加权平均来标准化数据的效果图。）

PopArt 算法则是兼顾了上述优点，并避免了引入非平稳问题。具体来说，PopArt [13] 包含 Preserving Outputs Precisely （**POP**）和 Adaptively Rescaling Targets （**ART**）两个阶段。



### 具体的符号定义：

整个神经网络变换记作  $f$ ，均值和方差记为  $\mu$  和  $\Sigma$ ，神经网络最后一个全连接层的权重和偏置记为  $W$  和  $b$ ，神经网络的其他部分参数记为  $\theta$ ，神经网络其他部分代表的变化记作  $h_\theta$

### ART：通过指数加权平均方法（EMA）动态地缩放目标。

对于任意学习目标，其标准化过程可以写为：

$$\tilde{Y}_t = \Sigma_t^{-1}(Y_t - \mu_t) \quad (2)$$

其中  $\mu$  为放缩参数， $\Sigma$  为偏移参数。相对应地，unnormalize 后的输出可以写作：

$$f_{\theta, \Sigma, \mu, W, b}(x) \equiv \Sigma(W h_\theta(x) + b) + \mu \quad (3)$$

例如 PPO 中可以用如上方法动态缩放 value function 和 advantage function。

对于新得到的一批数据，将新数据计算所得的均值和方差，与存储的历史均值与方差进行指数加权平均，其结果则可以用于标准化上述学习目标，其中  $\beta_t$  为更新步长：

$$\mu_t = (1 - \beta_t)\mu_{t-1} + \beta_t Y_t \quad (4)$$

$$v_t = (1 - \beta_t)v_{t-1} + \beta_t Y_t^2 \quad (5)$$

**POP：**在执行 Art 操作之后，调整参数  $W$  和  $b$ ，保持输出结果不变，避免非平稳问题。

当  $\mu$  和  $\Sigma$  发生变化时，POP 的目标是使得变化前后（例如标准化前后）的神经网络的最终输出保持不变，那对于这个原则，由于使用了新的均值和方差，所以需要对应改变神经网络最后一层的参数  $W$  和  $b$ ：

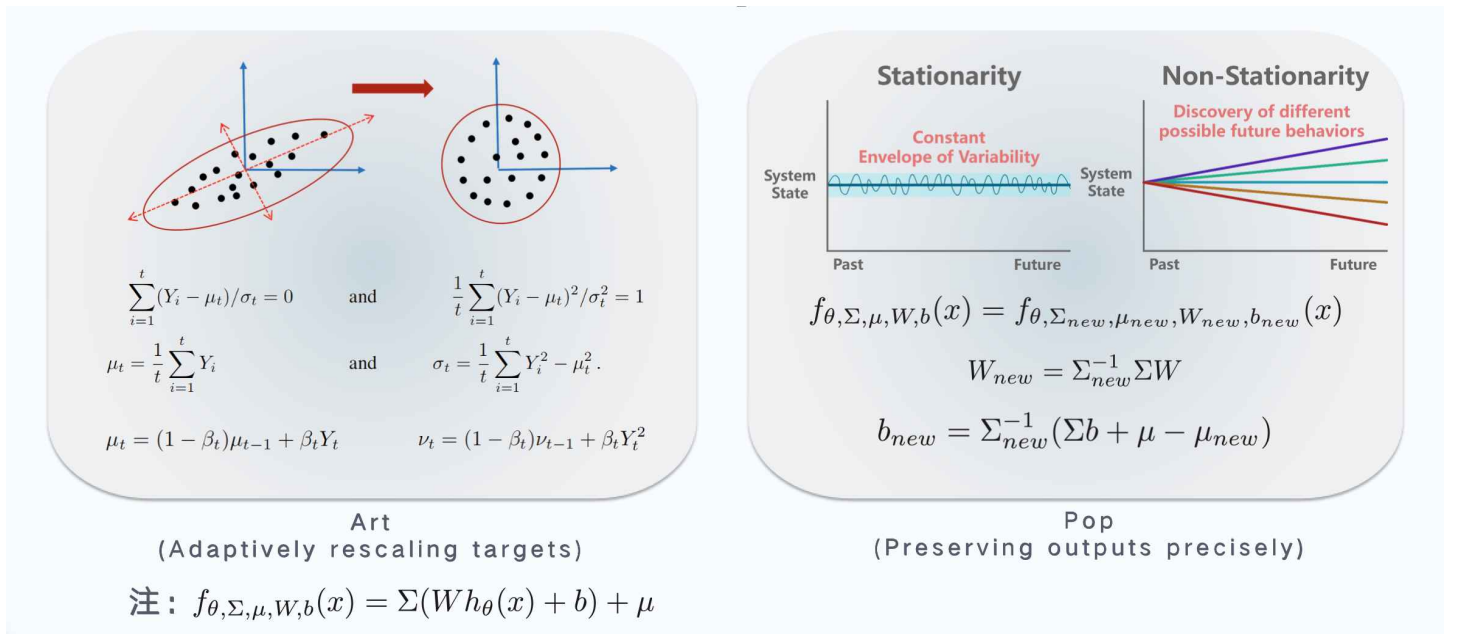
$$f_{\theta, \Sigma, \mu, W, b}(x) = f_{\theta, \Sigma_{new}, \mu_{new}, W_{new}, b_{new}}(x) \quad (6)$$

因此，可以得到神经网络最后一层的参数  $W$  和  $b$  的变化应为：

$$W_{new} = \Sigma_{new}^{-1} \Sigma W \quad (7)$$

$$b_{new} = \Sigma_{new}^{-1} (\Sigma b + \mu - \mu_{new}) \quad (8)$$

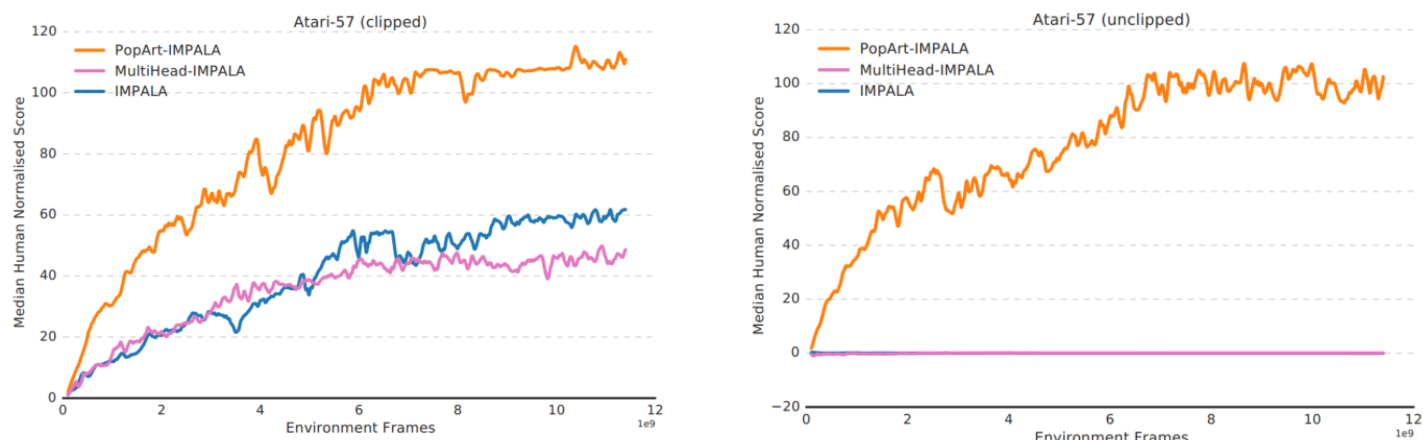
将上述两个步骤组合起来，PopArt 技术就可以利用动态的均值和方差自适应，调节尺度范围，同时避免了引入新的非平稳问题，使得强化学习优化变得更加稳定（整体概述如图23所示）。



(图23：PopArt 算法原理总结示意图。)

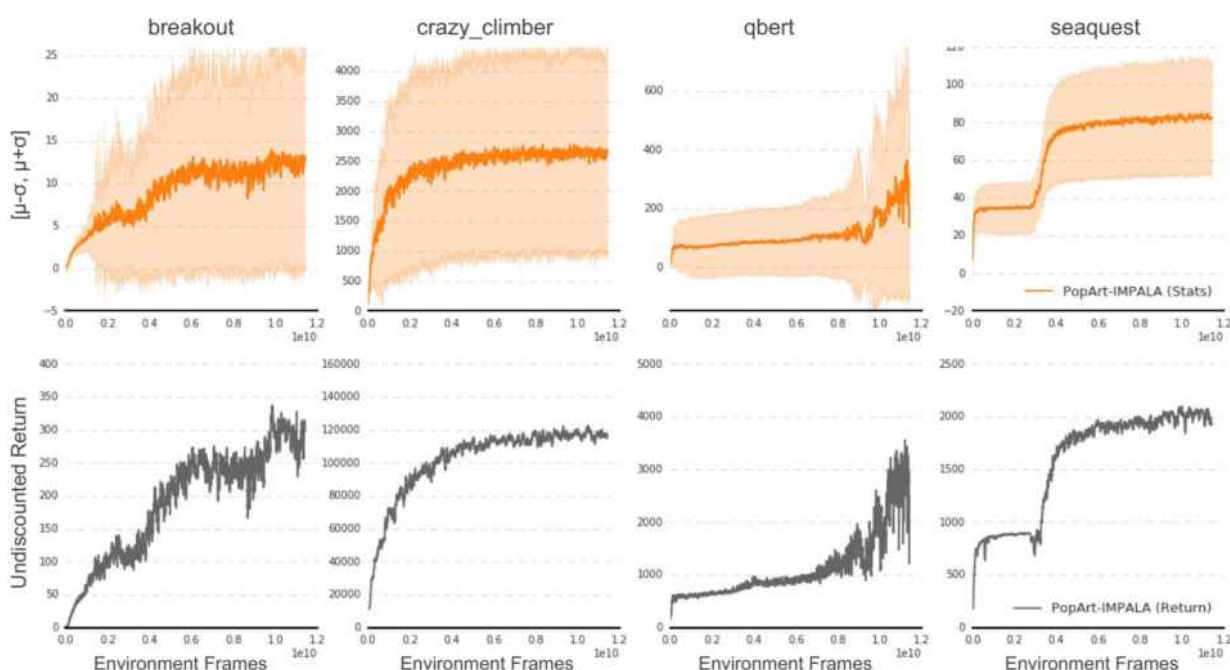
在原始论文 [13] 的实验中，如下图所示，右图展示了在未经过奖励裁剪的情况下，普通的 IMPALA 算法几乎无法学习到好的策略，奖励接近于 0。而经过 PopArt 处理后，奖励范围得到了控制，IMPALA 也能够学习到更好的策略。而左图中展示了经过奖励裁剪的情况，虽然普通的 IMPALA 已经能够达到一定奖励。但是由于在训练后期，episode 的长度不断增加，虽然单步的奖励被裁剪到了  $[-1, 1]$ ，但

整体的 return 尺度会因为 episode length 的变化还在不断变大。而 PopArt 将整体的 return 放缩后，训练的稳定性得到提升，从而能够学习到更优的策略。



(图 24：左图为经过奖励裁剪的情况下，是否使用 PopArt 的 IMPALA 在 Atari-57 上的得分差异；右图为未经过奖励裁剪的情况下，是否使用 PopArt 的 IMPALA 在 Atari-57 上的得分差异。)

下图展示了 PopArt 在不同环境中通过指数加权平均统计得到的实际均值和方差分布范围。其中橙色的线展示了  $[\mu - \sigma, \mu + \sigma]$  的范围。可以观察到，与下方环境中所获得的 return（灰色曲线）相比，橙色曲线的趋势是与灰色曲线保持一致，且数值变化范围更小，因此更便于优化。



(图 25：PopArt 在不同环境中通过指数加权平均统计得到的实际均值和方差分布范围。)

## Value Rescale

Pohlen 等人 [14] 提出了一种应用面更为广泛的方法 **value rescale**，将 value 映射到一定范围内（一般是压缩范围）。这一方法最初应用在 DQN、R2D2 等方法的 Q-value 尺度缩放中，在 PPO 中也可以用于缩放 value function 和 advantage function。其具体方法是通过一个函数将奖励压缩到某个范围，同时保证变化之后 value function 优化的部分性质仍然存在，如压缩映射（contraction）、不动

点相关的性质等。value rescale 所对应的映射一般记为  $h$ ，具体的变换形式如公式 (12) 所示，接下来的部分将会详细介绍相应原理。

基于值函数的方法本质是根据贝尔曼方程更新 Q 值，这里写出贝尔曼算子的完整定义如下：

$$(\mathcal{T}Q) := \mathbb{E}_{x' \sim P(\cdot|x,a)}[R(x,a) + \gamma \max_{a' \in A} Q(x',a')] \quad (9)$$

Contraction 定理 [15]：

对于任一完备的度量空间  $V$ ，假设  $T$  是该空间上的一个  $\gamma$ -contraction，则有：

- $T$  会收敛到唯一一个不动点
- $T$  会以速率  $\gamma$  线性收敛

当贝尔曼算子满足  $\gamma$ -contraction 时，最优的  $Q$  函数为贝尔曼算子的不动点，因此可以通过值函数迭代更新的方式找到最优的  $Q$  函数。所以对  $Q$  函数使用  $h$  进行映射时，也应该使得映射后的算子满足  $\gamma$ -contraction：

$$(\mathcal{T}_h Q) := \mathbb{E}_{x' \sim P(\cdot|x,a)}[h(R(x,a) + \gamma \max_{a' \in A} h^{-1}(Q(x',a')))] \quad (10)$$

事实上，这一性质要求映射函数  $h$  和 MDP 满足如下性质，使得值函数迭代更新的方法依然可以找到最优  $Q$  函数。

- $h$  严格单调递增
- $h$  存在闭式的 (closed form) 逆函数，且逆函数 Lipschitz 连续
- MDP 为确定性
- 衰减因子  $\gamma$  满足：

$$\gamma < \frac{1}{L_h L_{h^{-1}}} \quad (11)$$

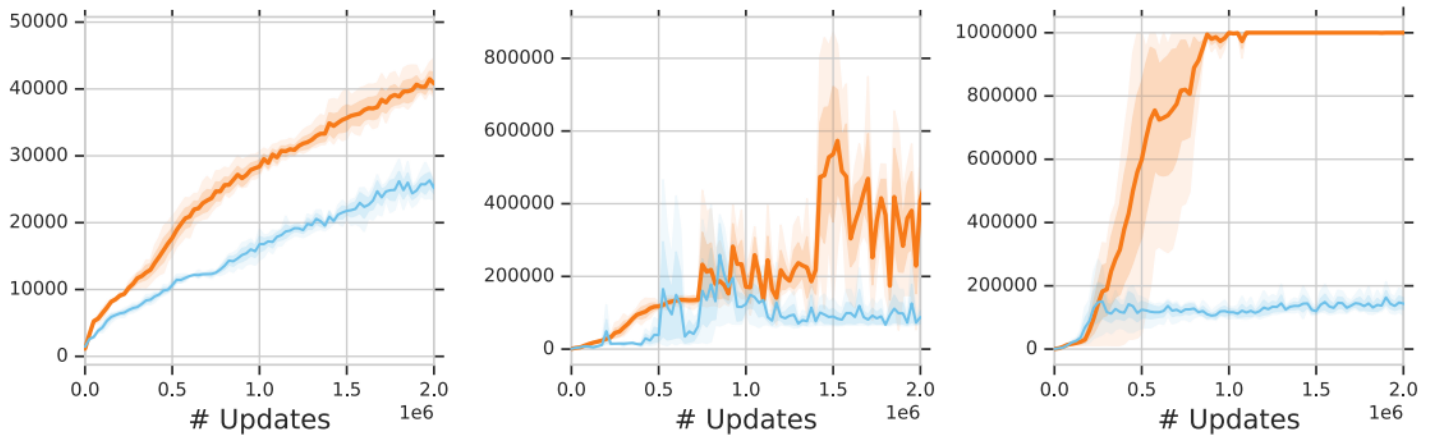
实践中常用的映射函数  $h$  如下：

$$h : z \mapsto \text{sign}(z)(\sqrt{|z| + 1} - 1) + \epsilon z \quad (12)$$

最后正则项的设计目的是保证映射函数的逆函数是 Lipschitz 连续的。

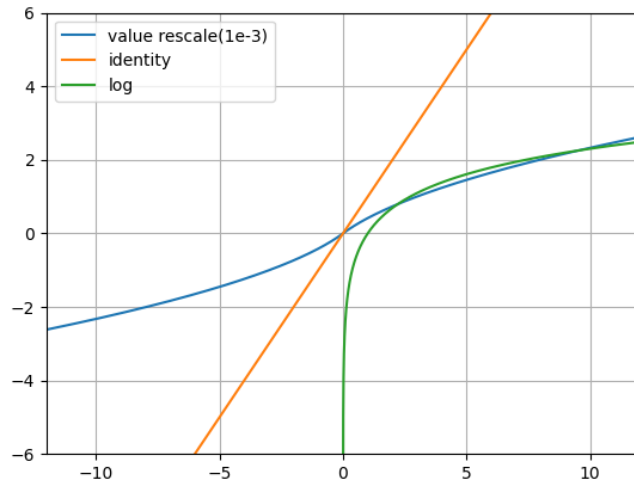
在实验效果方面，下图展示了 value rescale 和奖励裁剪方法在 Atari 的三种游戏上的效果对比，横轴是强化学习智能体更新的次数，纵轴是环境返回的累积回报。其中橙色线是 value rescale 的结果，而蓝色线是奖励裁剪的结果。可以看到 value rescale 下的奖励具有更高的上限和更强的稳定性。





(图 26: Atari 三种游戏 MsPacman、Qbert、SeaQuest 上 value rescale 和奖励裁剪的效果对比。)

图 27 进一步对比 value rescale 的函数值和恒等变换的函数值，可以看到 value rescale 实现了明显的压缩，这一压缩类似于对数函数压缩 (log)，但在数值较大的范围下可以有更明显的压缩效果。而且这一函数是关于原点对称的函数，因此对正向和负向的 reward 都有相应的处理作用。

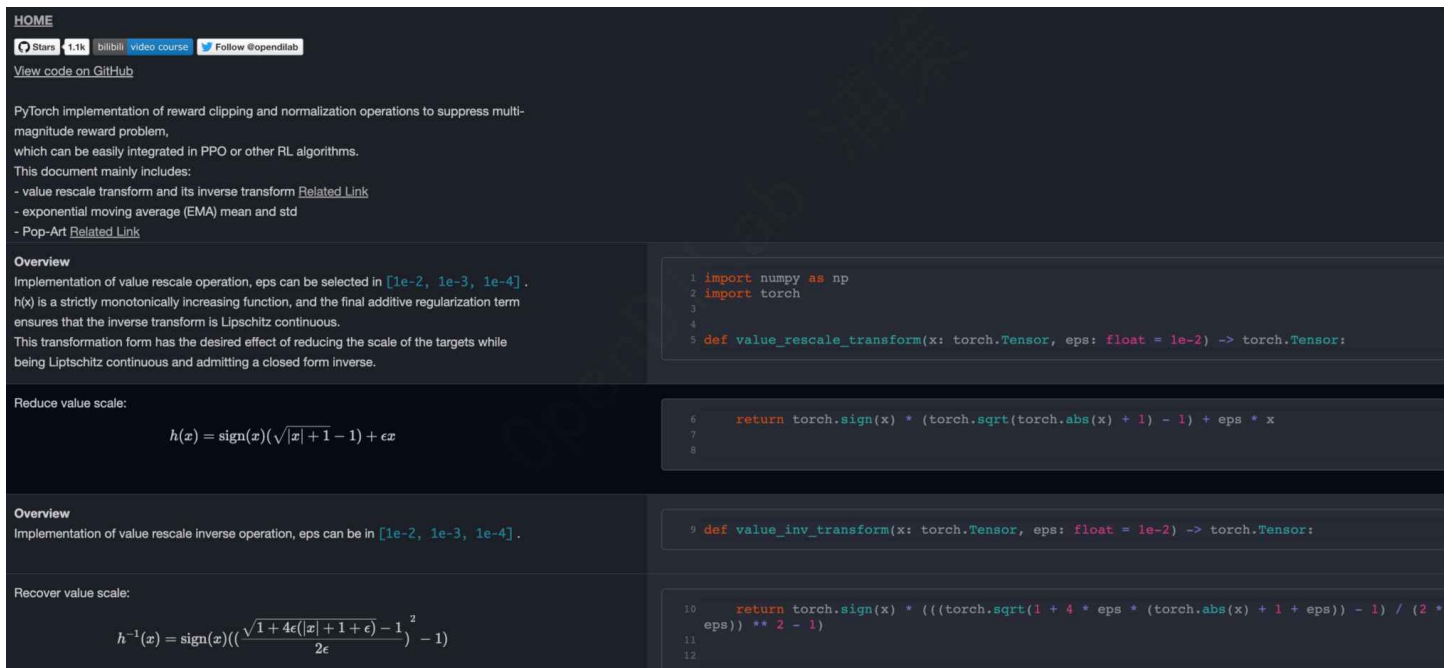


(图 27: value rescale 函数、恒等 (identity) 函数和对数 (log) 函数的曲线比较。)

而在更为一般的随机性 MDP 环境上，value rescale 也可以保留压缩映射性质，但其不动点相关的性质就会有更多变化，具体可以参考原始论文的附录部分。

### 4.3.3 代码：Reward clip/norm 操作的实现

在介绍理论知识之后，本节课也会沿用之前的“算法-代码”注解文档形式讲解具体的代码实现细节。



(图 28: value rescale 代码示例。)

上述代码展示了 value rescale 中的函数映射和逆映射的实现过程。其中映射函数为：

$$h(x) = \text{sign}(x)(\sqrt{|x| + 1} - 1) + \epsilon x,$$

而逆映射为：

$$h^{-1}(x) = \text{sign}(x)((\frac{\sqrt{1 + 4\epsilon(|x| + 1 + \epsilon)} - 1}{2\epsilon})^2 - 1).$$

#### 4.3.4 实践：使用 PPO 来打造自动驾驶智能体

本节课在 [MetaDrive](#) 这一自动驾驶仿真环境中进一步探索多尺度奖励的实践。在 MetaDrive 中，可以构建各种驾驶环境和相应的独特行为，用户可以控制车辆在其中进行丰富多样的决策探索，最终驾驶的目标是控制一辆（或者多辆）汽车安全且按时地从起点开到终点。

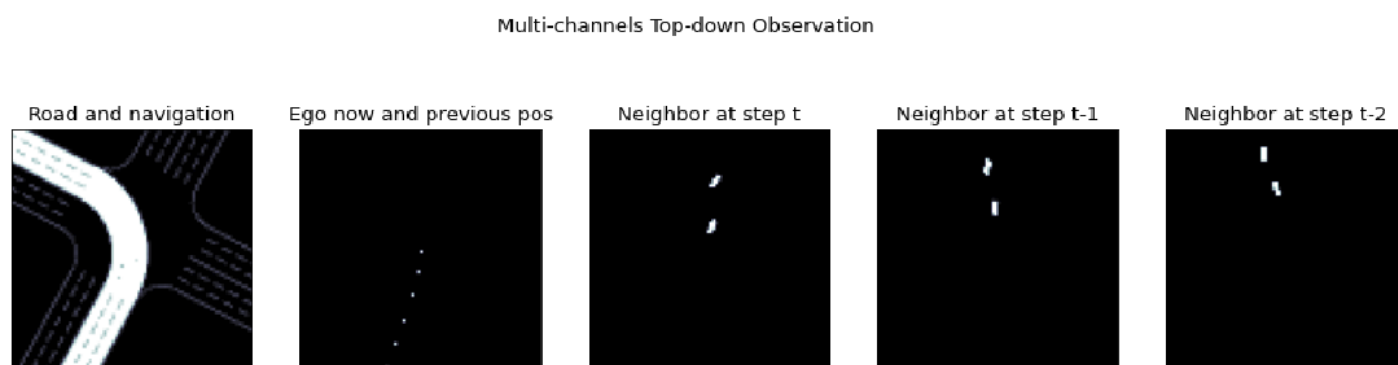


(图 29: meta-drive 观察空间示例。)

原始的观察空间包含三部分内容：车辆当前的状态（例如航向，转向，速度和到边界的相对距离），导航信息和周围的信息。而变换到 RL 环境中，观察空间为 5 通道俯视图。其中五个通道的语义分别为：

- 道路信息和导航信息（Road and Navigation）；
- 自身位置和自身历史位置（Ego now and previous pos）；
- 周围车辆在 t 时刻的俯视图（Neighbor at step t）；
- 周围车辆在 t-1 时刻的俯视图（Neighbor at step t-1）；
- 周围车辆在 t-2 时刻的俯视图（Neighbor at step t-2）。

例如上图中红色车辆正在左转，与对面驶来的蓝色车辆发生交互。此时五通道图像如下：



（图 30：meta-drive 观察空间的五通道图像。）

动作空间是两维连续动作空间，有效范围为  $[-1, 1]$ 。其中第一个维度表示转向角，第二个维度表示加速或刹车。同时，MetaDrive 也允许用户扩展动作空间，引入更多输入操作维度。

MetaDrive 的奖励函数的定义包含四种不同尺度的奖励：

- 驾驶奖励，即在当前车道线开出的纵向距离，是一个密集奖励
- 速度奖励，即当前时刻的速度，速度越大，则奖励越大，也是一个密集奖励
- 横向比例，即自我车辆是否远离当前车道的中心，它提供  $[0, 1]$  范围内的乘数，与驾驶奖励配合使用
- 终止奖励，在 episode 结束时候，其他密集奖励将被禁用，并根据车辆的状态返回一个稀疏的终局奖励，分为到达终点、开出道路、撞到他车、撞到障碍等情况。

当出现如下情况时，判定 episode 结束：

- 车辆成功到达终点；
- 撞到其他车辆或者障碍物；
- 开出道路以外。



（视频2：MetaDrive 环境上随机初始化和训练收敛的智能体行为对比视频（[链接](#)）。）

通过结合本节课介绍的一些 reward scale 的方法，能够训练出效果很好的智能体。左边是随机化时的失败样例，而右边展示了训练收敛后能够平稳驾驶的智能体。

## 4.4 附录（可选阅读）

### 4.4.1 逆强化学习（散度逼近理论）

逆强化学习的本质是：**从观察到的专家数据中逆向推理，提取奖励函数，再以此指导基于奖励函数的强化学习**。然而，逆强化学习对奖励函数和策略的显式约束限制了其通用性，并且，使用基于估计的奖励函数与环境交互也会带来极大的计算损耗。

为了缓解上述问题，生成式对抗模仿学习（GAIL）采用对抗生成网络，提升了模仿策略的学习效率。其中生成式对抗网络的生成器作为策略与环境交互生成新样本，而判别器需要判断数据是来自于专家示范样本还是强化学习探索生成的样本。进一步地，这一方法还可以推广到多模态的模仿学习中。

而从全局角度来看，上述方法从本质上来说都属于**基于散度逼近的模仿学习**，有关这套理论工具的完整定义，以及更多这一框架下的其他模仿学习方法示例可以参考如下补充阅读材料：

[https://github.com/opendilab/PPOxFamily/blob/main/chapter4\\_reward/chapter4\\_supp\\_irl.pdf](https://github.com/opendilab/PPOxFamily/blob/main/chapter4_reward/chapter4_supp_irl.pdf)

### 4.4.2 模仿学习（BC 及各种变体）

行为克隆（Behavior Cloning）是一种纯监督式的模仿学习方法，其本质就是以专家策略生成的动作作为标签，指导策略学习。而 IBC（Implicit Behavior Cloning）则是将模仿学习问题描述为条件能量模型，使得 IBC 在非连续性函数、多分布函数的拟合效果更好，从而提升模型在更多任务中的性能。

然而以上两种 BC 方法都是直接学习从 observation 到 action 的映射，Procedure Cloning（PC）则更希望学习从 observation 到 action 的推理过程。因此 PC 在训练中引入了专家做决策的中间思考结果，得到了更好的泛化性表现。关于上述两个算法的具体介绍，可以参考如下补充阅读材料：

[https://github.com/opendilab/PPOxFamily/blob/main/chapter4\\_reward/chapter4\\_supp\\_bc.pdf](https://github.com/opendilab/PPOxFamily/blob/main/chapter4_reward/chapter4_supp_bc.pdf)



## 参考文献

- [1] Sullivan R, Terry J K, Black B, et al. Cliff Diving: Exploring Reward Surfaces in Reinforcement Learning Environments[J]. arXiv preprint arXiv:2205.07015, 2022.
- [2] Bellemare M G, Naddaf Y, Veness J, et al. The arcade learning environment: An evaluation platform for general agents[J]. Journal of Artificial Intelligence Research, 2013, 47: 253-279.
- [3] <https://openai.com/five/>
- [4] Zheng B, Verma S, Zhou J, et al. Imitation learning: Progress, taxonomies and challenges[J]. IEEE Transactions on Neural Networks and Learning Systems, 2022: 1-16.
- [5] Pathak, Deepak, et al. "Curiosity-driven exploration by self-supervised prediction." *International conference on machine learning*. PMLR, 2017.
- [6] Kempka, Michał, et al. "Vizdoom: A doom-based ai research platform for visual reinforcement learning." *2016 IEEE conference on computational intelligence and games (CIG)*. IEEE, 2016.
- [7] Burda Y, Edwards H, Pathak D, et al. Large-scale study of curiosity-driven learning[J]. arXiv preprint arXiv:1808.04355, 2018.
- [8] Burda Y, Edwards H, Storkey A, et al. Exploration by random network distillation[J]. arXiv preprint arXiv:1810.12894, 2018.
- [9] Badia, Adrià Puigdomènech, et al. "Never give up: Learning directed exploration strategies." *arXiv preprint arXiv:2002.06038* (2020).
- [10] Badia, Adrià Puigdomènech, et al. "Agent57: Outperforming the Atari human benchmark." *International conference on machine learning*. PMLR, 2020.
- [11] <https://github.com/Farama-Foundation/Minigrid>
- [12] Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning[J]. arXiv preprint arXiv:1312.5602, 2013.
- [13] van Hasselt H P, Guez A, Hessel M, et al. Learning values across many orders of magnitude[J]. Advances in neural information processing systems, 2016, 29.
- [14] Pohlen T, Piot B, Hester T, et al. Observe and look further: Achieving consistent performance on atari[J]. arXiv preprint arXiv:1805.11593, 2018.
- [15] Sutton R S, Barto A G. Reinforcement learning: An introduction[M]. MIT press, 2018.