

PPO × Family 第二讲文字稿

本节课将进入 PPO × Family 系列课程的第一个专题：解构复杂动作空间。

如果想将各种各样的深度强化学习算法应用在实际问题中，那么第一步就是需要**将原始的自然决策问题转化为标准的马尔科夫决策过程（MDP）**，具体来说就是需要去定义清楚 MDP 五元组的各个元素，比如状态，动作，奖励等等。而本节课将会从动作空间入手，首先，概述性地讲解动作空间的特点，然后，会选取四种主流的决策动作空间（离散、连续、多维离散、混合）一一展开，最终希望能够使用一个 PPO 算法去解决上述四种动作空间，从“**算法-代码-应用**”三合一的角度来探索各类决策输出的最佳方案，做到真正意义上的决策万能钥匙。



(图1：四类经典的决策动作空间)

2.1 动作空间概述



(图2：一个经典的动作空间例子——键盘动作空间。键盘上的每个按键的按下和松开就代表相应功能的执行还是不执行，一个键就是一个2维的离散动作，整个键盘就是一个巨大的离散动作集合。另外，各个键之间还存在一定的依赖关系，比如 Ctrl + Shift 构成的组合键，这些关系进一步整合起来，就构成了一个非常复杂的动作空间)

动作是智能体 (Agent) 与环境交互时做出的决策行为，即决策算法的输出，动作空间 (Action Space) 是指所有可能动作的集合，是马尔科夫决策过程的五元组元素之一。将原始决策问题的输出端抽象为合适的动作空间，并配合使用相应的决策算法，是通向高效强大决策智能体的必经之路。

动作空间的分类



(图3：动作空间分类概览和样例图)

离散和连续动作是最经典最基本的两类动作空间，在此基础上，又可以推广到更复杂的多维离散动作空间和混合动作空间。在经典的强化学习学术环境中，如图3所示，离散动作空间的例子包括 Super Mario Bros 和 Procgen 环境，相应的，连续动作空间一般则会和机器人更相关，例如图中所示的

DeepMind Control 和 PyBullet 环境，而多维离散 (multi-discrete) 和混合 (hybrid) 动作空间，则更多对应生活实际中的一些决策应用。总结来看，动作空间可以总结出如下几条特点：

- 不同的动作空间常需要不同的算法：**例如处理离散动作空间常用一些 value-based 方法 (DQN) 和策略梯度方法 (PPO)，而连续动作空间则会涉及更多策略梯度方法的变体，例如 PPO、DDPG、SAC。这些算法在设计之初就会为对应的动作空间做一些特殊的设计，进而使得它们在这些场景上会有一定的优势。
- 标准的 RL 算法常需要对特定动作进行适配和定制化：**比如想将标准的 PPO 算法应用到上述四种动作空间上，那么就需要针对具体的动作空间添加一些算法设计和技巧，这也是 PPO × Family 课程设计的重要动机之一。
- 同一个环境可以用不同的动作空间：**基于对决策问题（环境）的领域知识，基于对强化学习算法的使用经验，可以对原有环境的动作空间做一些改造，让改造后的动作空间更适合于算法训练和部署。

动作塑形 (Action Shaping)

上文中提到的动作空间改造，又被称为动作塑形 (Action Shaping)，即对原始的动作空间进行各种预处理和特征工程操作，转化为更适合 RL 的空间，下表就总结了实践中[1] 常用的一些动作塑形操作和变换前后动作空间的类型，其中标明的三种最常用的变换分别为：

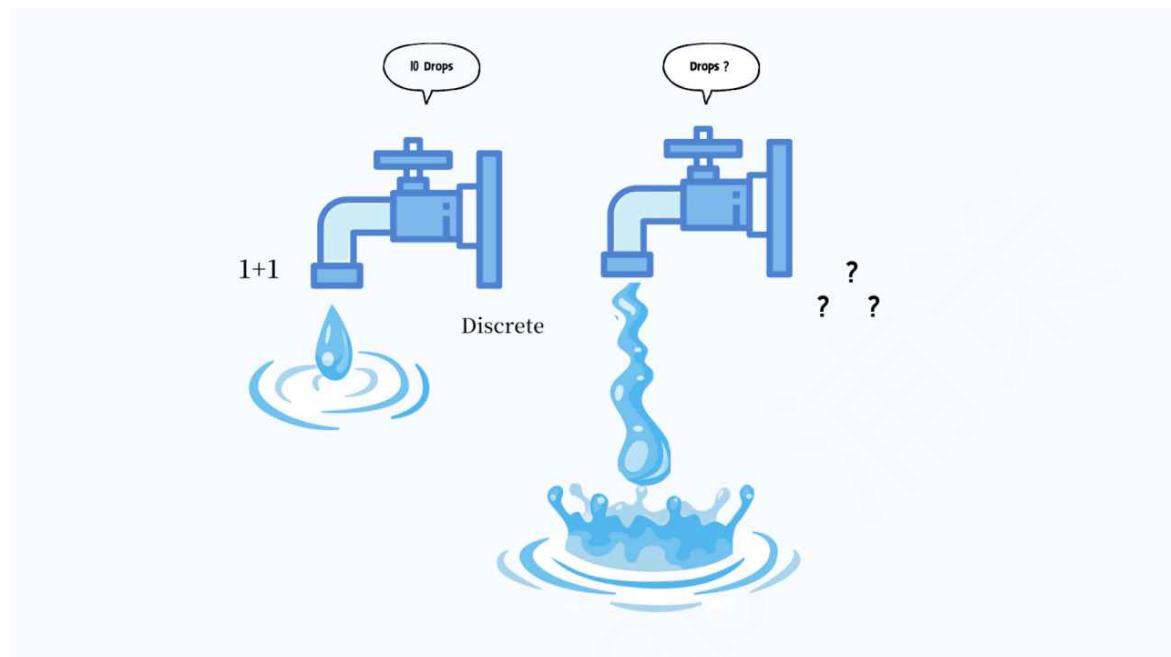
- DC：连续动作空间离散化
- RA：去除部分冗余或无意义的动作
- CMD：将多维离散动作空间转换为离散动作空间

Environment Name	Original action space	Transformation	Transformed action space	Performance		
MineRL	Multi-discrete(2, 2, 2, 2, 2, 2, 2, 7, 8, 5, 8, 3), Continuous(2)	DC	RA	CMD	Discrete(36)	
		DC	RA	CMD	Discrete(10)	1 st place
		DC	RA	CMD	Discrete(216)	2 nd place
		DC	RA		Multi-discrete(2, 2, 3, 3, 7, 8, 5, 8, 3, 40)	3 rd place
		DC	RA		Multi-discrete(2, 2, 2, 5, 8, 3, 8, 7, 3, 3)	5 th place
		DC	RA			
Unity Obstacle Tower Challenge	Multi-discrete(3, 3, 2, 3)	RA	CMD	Discrete(12)	1 st place	
		RA		Discrete(6)	2 nd place	
VizDoom (Doom)	38 binary buttons, 5 continuous	RA	CMD	Discrete(256)	1 st place (Track 2)	
		RA		Discrete(6)	1 st place (Track 1)	
Atari	Discrete(18)	RA		Discrete(4 - 18)		
StarCraft II	Multi-discrete	DC	RA	Multi-discrete		
Dota 2	Multi-discrete	DC	RA	Multi-discrete		
GTA V (car driving only)	Multi-discrete	RA	CMD	Discrete(3)		
Torcs	Multi-discrete	RA	CMD	Discrete(3)		
DMLab (Quake 3)	Multi-discrete(3, 3, 2, 2, 2), Continuous(2)	RA	CMD	Discrete(9)		
Honor of Kings (MOBA)	Multi-Discrete, Continuous	RA	CMD	Multi-discrete, Continuous		
Little Fighter 2 (lf2gym)	Multi-Discrete(2,2,2,2,2,2,2)	CMD		Discrete(8)		

(图4：一些经典的大型决策环境中 Action Shaping 操作一览)

基于上述 Action Shaping 方法的结果，本节课将会关注跟算法设计联系更加紧密的部分，结合 PPO × Family 在不同动作空间上的**算法理论设计、代码实现、衍生应用三方面的相关知识**，详细讲述 PPO 如何解构各类动作空间。

2.2 离散动作空间



(图5：离散动作空间和连续动作空间的对比示意图)

引言：离散动作空间的定义

离散动作空间是最经典、最常规的动作空间，一般是相对连续动作而言的概念，具体由**有限数量**的动作组成，包含**特定任务中所有可用的离散控制指令**，可以类比机器学习中的分类任务。这里举一些离散动作空间的实际例子：

- 石头剪刀布游戏，可选择的动作有三种。
- 红白机游戏，可选择的动作为游戏手柄上的按键（上下左右BA）。

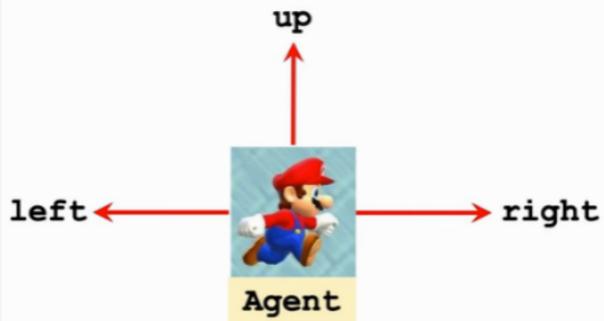
理论：PPO 如何建模离散动作空间

策略定义

如果智能体正在玩超级马里奥游戏，那么将当前游戏这一帧的画面（下图所示）记为图片观察状态 s 。而智能体在观测到状态 s 后，可选择的动作集合则记为 $a \in \{left, right, up\}$ ，即一个标准的3维离散动作空间，每次决策需要从这三个离散动作中选择一个。



图片观察状态 s



离散动作 a

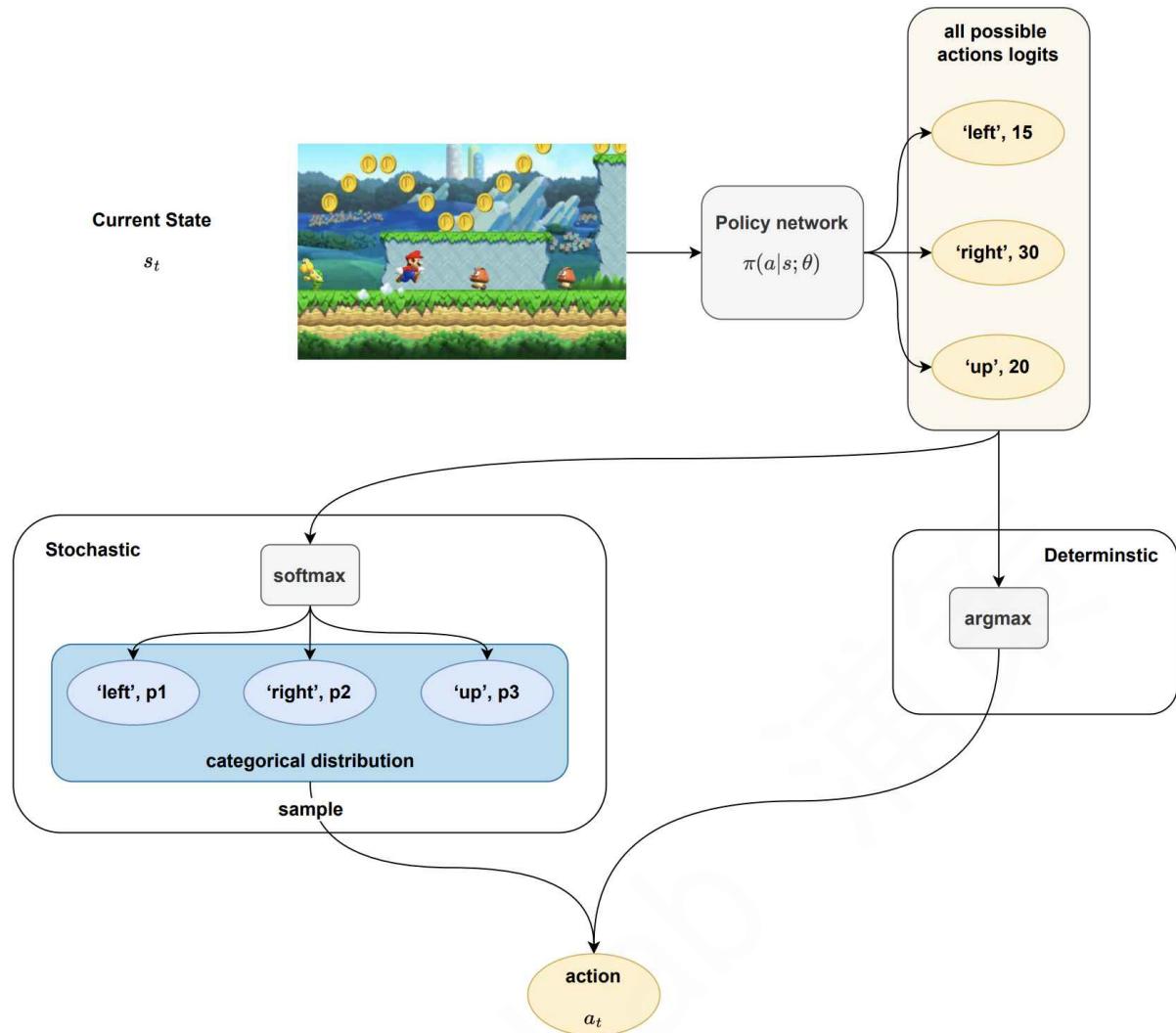
(图6：马里奥游戏中的状态和动作定义示例)

基于上述的状态和动作定义，智能体往往需要从数据中学习到策略 π ，即向策略 π 输入状态，它将输出所选动作，或是选择每个动作的概率，对于 PPO 来说，具体是学习一种随机性策略，即给定一个某时刻 t 马里奥的游戏画面，即图片观察状态 s_t ，PPO 智能体策略输出选择各个离散动作的概率分布 $\pi(a_t|s_t)$ ，一个具体的实例为：

- $\pi(left|s) = 0.2$ 表示选择向左动作的概率是0.2
- $\pi(right|s) = 0.1$ 表示选择向右动作的概率是0.1
- $\pi(up|s) = 0.7$ 表示选择向上动作的概率是0.7

计算图和动作采样

PPO 是一种深度强化学习算法，由于策略要建模的“状态-动作”之间的映射在实际问题中可能比较复杂，所以结合神经网络来实现参数策略函数 $\pi(a|s)$ ，对于这个策略神经网络，输入状态 s ，网络就输出每一维离散动作 a_i 对应的 logit 值，比如上文所示的马里奥游戏就是最终输出3个 logit 值，完整的输入输出数据流可以参考下图：



(图7：使用 PPO 算法处理马里奥游戏，详细的输入输出数据流)

logit 的含义是神经网络 $\pi(a|s; \theta)$ 的原始输出（比如最后一层全连接层的输出，不加激活函数，取实数范围），离散动作空间有 k 个动作，则策略网络输出 k 个动作相应的 logit 值。这些 logit 值经过 softmax 函数即可得到当前策略 $\pi(a|s; \theta)$ 选择每个离散动作的概率。

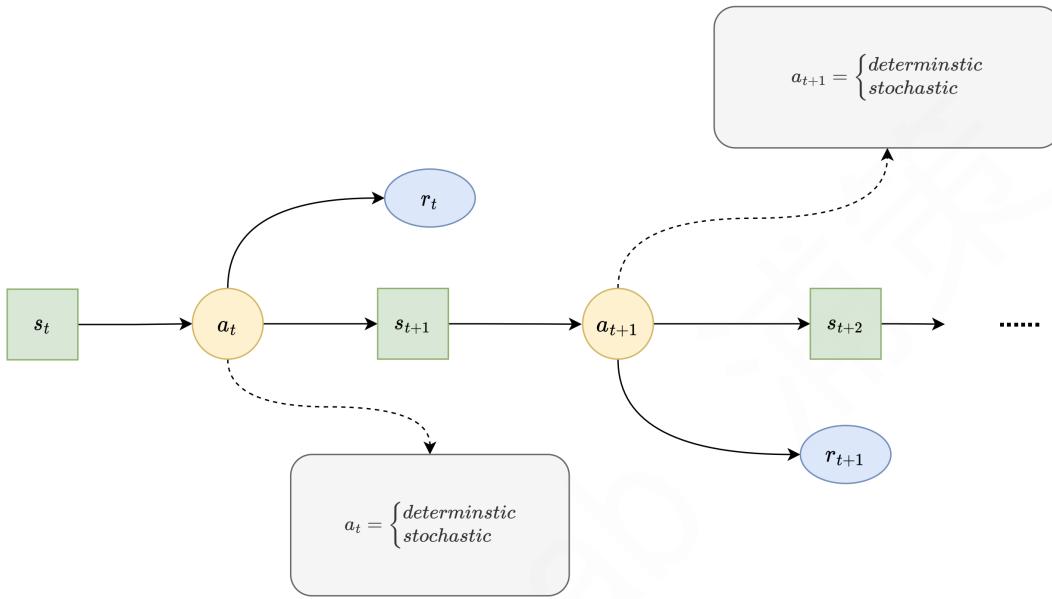
- 关于 logit 的进一步理解可以参考知乎博客：[如何理解深度学习源码里经常出现的logit \[2\]](#)

在获得智能体策略选择各个离散动作的概率之后，一般常有两种采样动作的方式：

- **确定性决策 (deterministic decision)**：直接贪心选择 [\[3\]](#) 概率最大的那一维动作（一般实现中直接选择 logit 最大的，因为 softmax 函数并不改变 logit 的相对大小，即 $\underset{a \in \mathcal{A}}{\operatorname{argmax}} \pi(a|s; \theta)$ ）。
- **随机性决策 (stochastic decision)**：根据之前将 $\pi(a|s)$ 的输出经过 softmax 函数后获得的概率值，构造相应概率分布（离散动作空间即对应玻尔兹曼分布 [\[4\]](#)）进行抽样得到动作。
 - 随机性决策一般用于 PPO 智能体[收集数据](#)的过程，由于强化学习是一个在线训练的过程，所以训练中需要保持足够的探索程度（即概率低的动作也有被采样到的可能性），同时也更多地利用概率高的动作。此外，为了更精细地控制这一点，部分任务中还会在 softmax 处设置专门的温度系数来控制概率分布的平滑程度。

- 另一方面，PPO 智能体训练过程中也会定期进行评测，分析当前智能体的性能。这时则会根据情况选择使用确定性决策还是随机性决策。一般来说，对于环境简单，干扰噪声因素较少的环境（例如超级马里奥），确定性决策往往能直接给出优秀的解。而另外一些过于复杂，环境中随机性较大，最优策略不唯一的环境（例如星际争霸2，局部视野的迷宫），使用随机性决策则可以获得比较高的期望收益。

在确定了智能体策略的计算图和采样动作的方式之后，就可以让智能体和环境不断交互，产生一系列轨迹数据用于训练，整个交互流程如下图所示：



(图8：智能体与环境交互产生轨迹数据的流程示意图)

代码：PPO 中如何实现采样动作 (`torch.distribution`)

PPO 在离散动作空间上的算法理论其实很简单，大部分的算法设计很容易实现，具体可以参考课程第二章的[相关代码](#)。但其中关于[随机性决策采样动作](#)的部分，仍存在一些特殊的地方。关于[如何在深度学习中构建和使用概率分布](#)，TensorFlow 的作者们在2017年提出了关于这部分的标准设计 [5]。本课程中将结合 PyTorch 中具有相似功能与设计的概率分布模块 `torch.distribution` [6] 来进行讲解。

`torch.distribution` 实现了各种概率分布，以及相关用于抽样和计算统计数据的方法。如对于离散动作空间，通常使用 `torch.distribution.Categorical` 来构建策略神经网络输出的离散动作概率分布，决策时直接使用它的函数 `sample()` 即可返回抽样结果。

而要深入了解 `torch.distribution` 的工作原理，就要理解三种类型的 `shape`，即 **Sample shape**、**Batch shape** 和 **Event shape**，它们对于理解 `torch.distribution` 至关重要。

- **Sample shape** 为针对单个分布的采样次数，在使用 `sample(sample_shape)` 时指定，即从这个分布中独立同分布地采样多个值。
- **Batch shape** 为用于采样的（不同）分布的个数，例如对于一批（batch）样本，采样每个样本对应的动作。

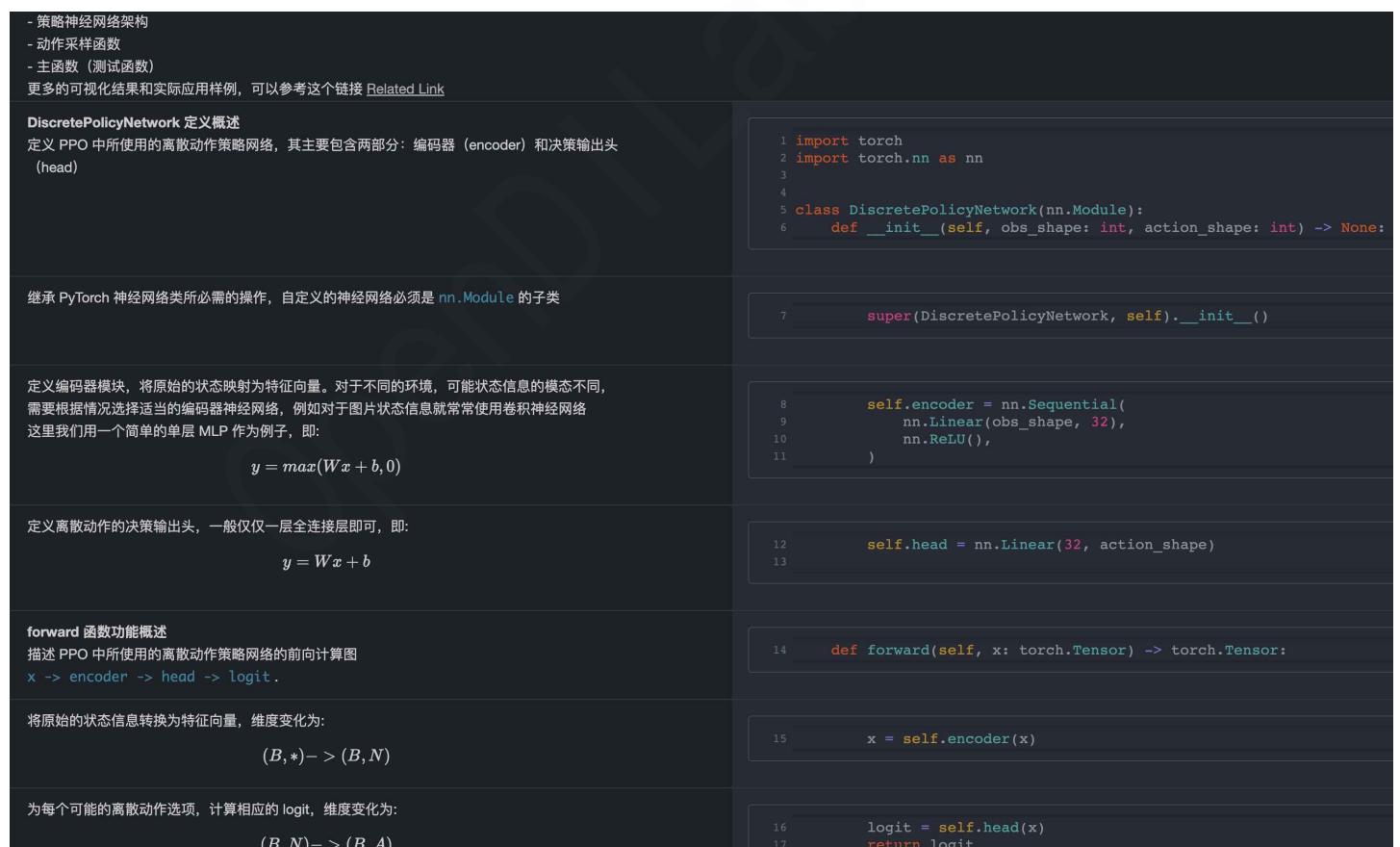
- **Event shape** 为单个分布本身的维度，具体使用时跟概率分布的语义强相关，例如后续更复杂的动作空间，可能需要考虑多重动作之间的关系。

一个更具体的演示图如下所示 [7]：



(图9：三种类型的 `shape` 分布、抽样、维度形式、原理解释的对比图)

完整的“算法-代码”——对比详细讲解，读者可以参考 PPO × Family 课程的[注解文档](#)，一个示意图如下：



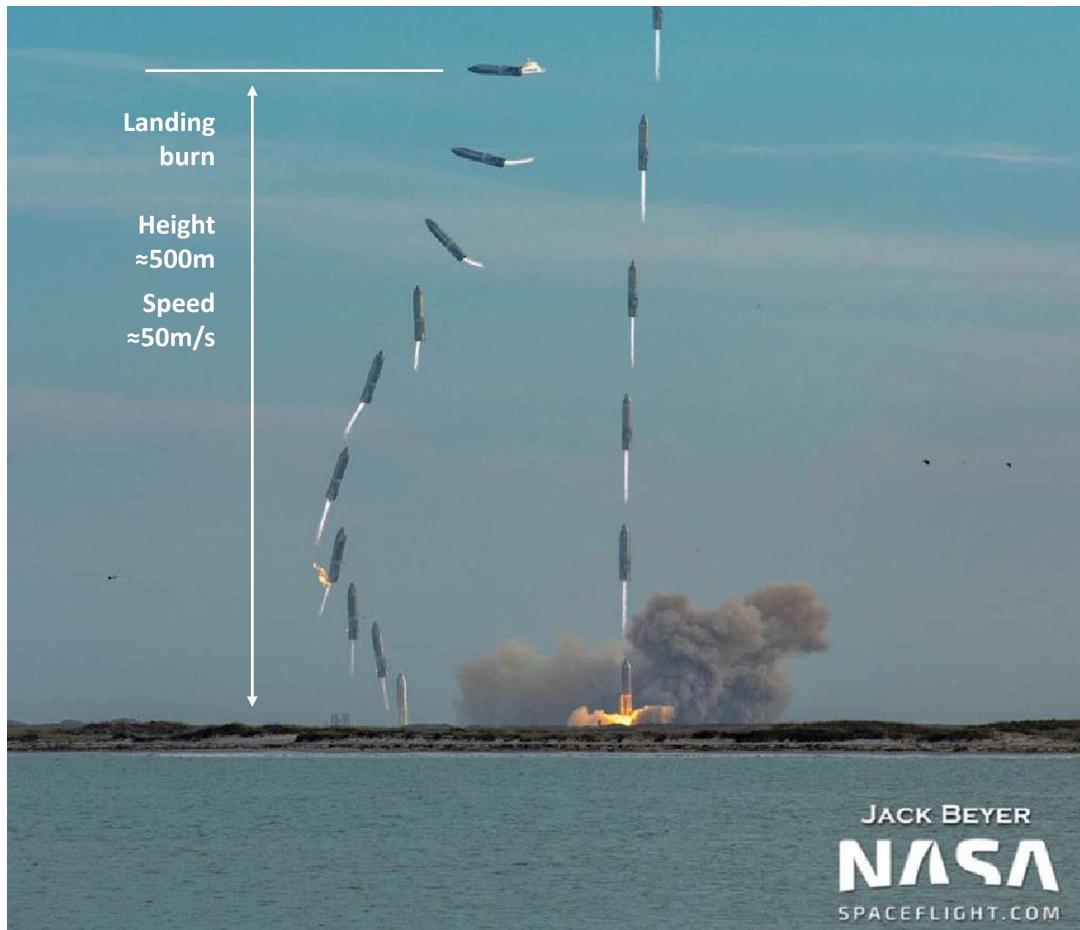
(图10：PPO × Family 课程“算法-代码”注解文档示意图——离散动作空间示例)

实践：使用 PPO 来解决火箭回收中的离散控制

在明确 PPO + 离散动作空间的算法理论和代码实现细节后，本小节尝试来应用相关知识解决相应的实际问题，这里选择简化后的**火箭回收任务中的离散控制问题**。本章节将会介绍环境的各种细节定义以及使用 PPO 的实践经验。

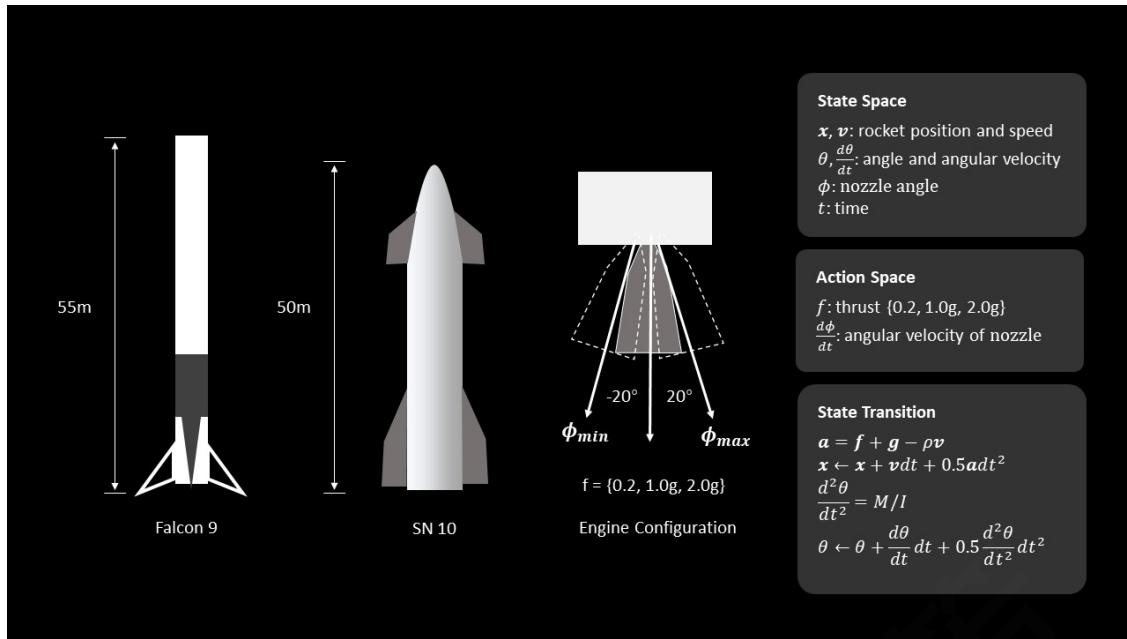
问题背景

2021年3月4日，美国得克萨斯州 Boca Chica 测试基地，SpaceX 的星舰飞船原型 SN10 再次表演了一场空中杂技，实现了星舰高空试飞的首次平稳落地，下图是 SN10 发射和着陆的实际情况 [8]：



(图11：SN10升空和落地过程的叠加合成图[8])

本课程修改并简化了开源代码环境 Rocket-recycling [9] 中模拟 SN10 的回收落地任务，并将其作为 PPO 做离散控制的示例之一，可以在1-2小时内训练收敛。具体来说，这个任务就是要控制火箭从水平的初始状态，不断调整发动机的推力从而控制姿态，最终成功竖直平稳落地。实际的环境实现是一个简化模型，考虑了基本的气缸动力学模型，并假设空气阻力与速度成正比。火箭底部安装了推动力为矢量的发动机，强化学习的目标也就是控制这些发动机的相关参数。具体原理图如下所示，将原始连续的推力控制离散化为三个角度 × 三种推力的九维离散动作空间：



(图12：SN10 火箭仿真结构原理图和动作离散化图)

环境 MDP 设定

Rocket-recycling 仿真环境的 MDP 基础元素定义如下：

初始条件	<ul style="list-style-type: none"> 初始速度设置为 -50m/s。 火箭方向设置为 90° (水平)。 着陆燃烧高度设置为离地 500m。 水平方向初始位置随机
动作空间	<ul style="list-style-type: none"> 推力：推力可调值为 0.2g、1.0g 和 2.0g 喷嘴角速度：可调节为 0、$20^\circ/\text{s}$ 和 $-20^\circ/\text{s}$ 注意也就是只有 9 个组合，动作空间 shape 为 $(9,)$
观察空间	<ul style="list-style-type: none"> 位置 (x, y)：即火箭在画布中的位置，范围分别为 $(-300, 300)$ 和 $(-30, 570)$ 速度 (vx, vy)：即火箭在 x 和 y 方向上的速度 箭身角度 θ：即火箭箭身与竖直方向的角度，范围为 $[0, 360^\circ]$ 箭身角速度 $v\theta$：即火箭箭身旋转角速度 喷嘴角度 ϕ：范围 $(-20^\circ, 20^\circ)$ 时间 t
奖励空间	<p>由两部分组成</p> <pre>reward = dist_reward + pose_reward</pre> <ul style="list-style-type: none"> $dist_reward$ 距离目标着陆位置的度量 $pose_reward$ 偏离正中线的位置 (theta, 正负)

- shape `(1,)` , 类型为 `float`

一般情况下，复杂机械的控制大多都为连续控制，但在精度要求不是特别高的问题中，合理的离散化将会大大提升问题的解决效率 [10]。具体来说，这里将发动机的推力与喷嘴角度进行组合，预设了9个可选的离散动作，即：

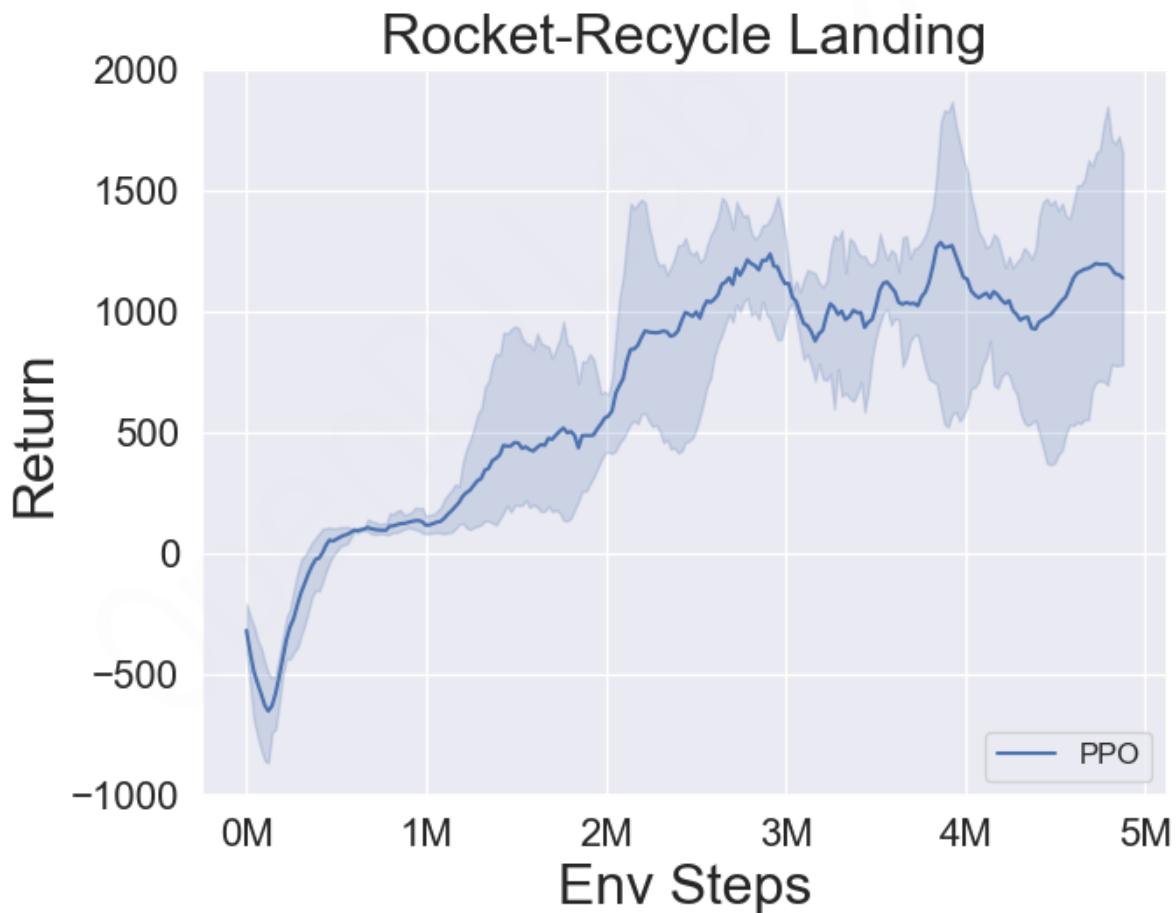
`(0.2g, 0)` , `(0.2g, 20°/s)` , `(0.2g, -20°/s)`

`(1.0g, 0)` , `(1.0g, 20°/s)` , `(1.0g, -20°/s)`

`(2.0g, 0)` , `(2.0g, 20°/s)` , `(2.0g, -20°/s)`

实验结果展示

下方的图13展示了 PPO 智能体控制火箭回收的完整训练过程，纵轴是每局游戏结束（坠毁或者成功降落）智能体获得的累积回报值，横轴是智能体和环境交互的步数，最终 PPO 智能体可以学习到稳定的火箭回收控制策略。



(图13：火箭回收任务 PPO 算法训练曲线，图中5组随机种子平均后的结果，实线为均值，阴影部分为标准差)

由于强化学习是一个需要平衡探索和利用的过程，所以需要不断尝试各种决策行为，并根据奖励反馈学习到最优策略。通过对回放视频以及累积回报值的观察，可以发现 PPO 智能体的学习可以主要分为

以下几个阶段：

1. 乱喷气直接坠落 (0-100k)
2. 先学会前半程的hover (200k-500k)
3. 然后学习轨迹的落点 (大趋势) 尽量靠近目标区域 (不论中间怎么翻滚) (600k-1M)
4. 学会落地减速 (但是落地太想靠近目标区域, 所以临近落地疯狂翻转) (1M-2M)
5. 开始探索较短的轨迹 (反复横跳) (2M-3M)
6. 学习落地保持竖直 (保持后半程的竖直, 一定程度上放弃对目标中心的追求) (3M-5M)

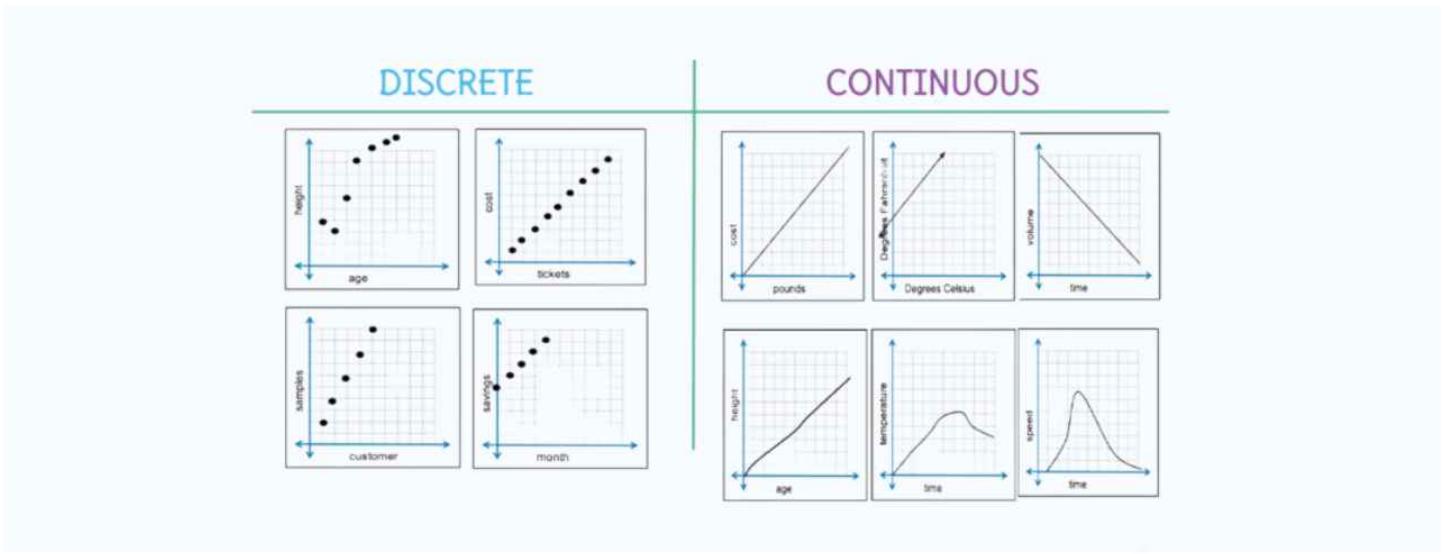
最终训练收敛时获得的策略和随机初始化策略的行为对比视频如下, 完整样例可以参考[官方示例](#)

ISSUE:



(视频1：火箭回收环境中初始随机策略和训练收敛的策略之间的对比)

2.3 连续动作空间



(图14：连续动作空间与离散动作空间对比举例示意图)

在剖析完 PPO 对于离散动作空间的设计之后，本节课第二部分将会聚焦到连续动作空间。离散和连续是决策问题中两种最基本的空间类型。它们之间的关系可以类比于二进制世界中的整型变量与浮点型变量一样，互相独立存在，却又有密切的联系。换个角度来说，在现实宏观世界中，大部分的事物都是连续地在时空中变化的，只是为了人类定义和使用的方便，或是受限于工具与条件，需要将一些连续变量抽象成不同离散化的选项，构造出一些离散的情形。

引言：连续动作空间的定义

通俗点来说，连续空间往往是一个“**充实又稠密**”的空间，即对于空间中任意一个点，在某种距离函数的定义下，该空间中**总存在**一些不同于该点的，但距离无穷小的其它点。这一说法的严格的数学定义如下：

连续空间是一个完备测度向量空间，即指一个拓扑空间 X ，假使对该拓扑空间 X 定义一种度量函数 $F(x, x')$ ，对于拓扑空间 X 中的任意点 x ，该点的任意邻域内都存在一个柯西序列：即对于任意小的正实数 r ，序列中都存在一个序数 M ，使得序列中所有序数大于 M 的点 x_N ， $\forall N > M$ ，与 x 的距离都小于该正实数， $F(x_N, x) < r$

在介绍完定义之后，接下来展示一个很多人熟悉的 MOBA 类游戏的例子，通过举例对比的方式来进一步更直观地理解离散动作空间和连续动作空间：

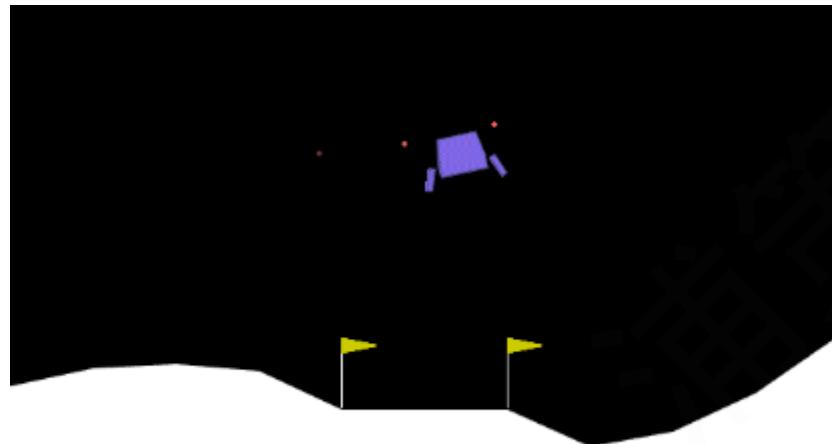


(图15：MOBA类游戏《王者荣耀》中的离散和连续动作空间示例。每一个技能选择是离散动作空间，而确定好技能选择之后，技能的释放范围和方向就可能是连续控制问题)

此外，对于具体的决策问题，也可以根据情况把问题建模成不同的 MDP，例如强化学习的经典环境 Lunarlander（模拟月球着陆器），就提供了离散和连续2种不同的控制方式：

- 离散控制模式：对应着4个离散动作，分别为：什么都不做，启动左方向引擎，启动主引擎，启动右方向引擎（启动引擎的推力为一个固定值）。
- 连续控制模式：对应着2个连续的动作变量，即主引擎和助推器的开启力度。

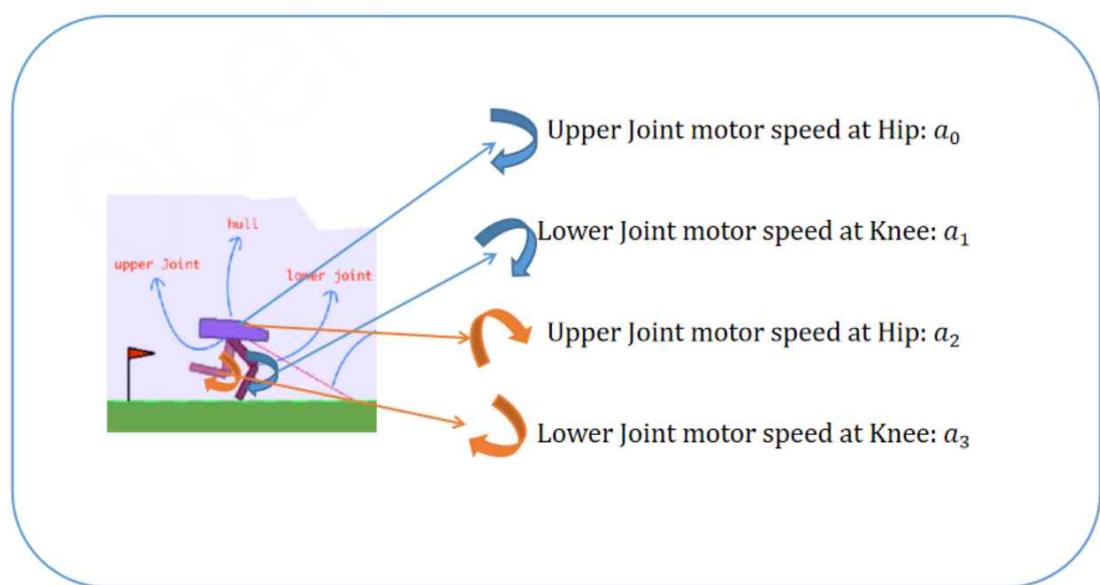
前者训练起来更加简洁高效，而后者可以允许更细粒度的操控。部分强化学习环境中的离散动作空间往往都是将原始连续动作中经过某种归类方式获得，是连续动作的某个特殊形式。



(图16：Lunarlander 环境的成功示例，智能体控制飞船成功平稳降落到两个旗杆之间)

理论：PPO 如何建模连续动作空间

连续动作空间的特点



(图17：bipedalwalker 环境里的四维连续动作实际意义原理图)

更具体地，本章节以经典的学术环境双足机器人行走（bipedalwalker）为例，具体分析连续动作空间的特性。在这个环境中，智能体的任务目标是控制机器人平稳走到右端的终点。如上图所示，具体的决策输出是一个4维的连续变量，这四个维度分别是其机器人位于臀部关节和膝盖关节的铰链的马达转速。对比这样的连续动作空间和上文提到的离散动作空间，可以发现：

- 离散动作空间是非连续的动作的集合（类似分类问题），而连续动作空间是连续的，稠密的，不可数个动作元素组成的连续区间（类似回归问题）。在连续空间中的个体，总能找到许多与它特别相似甚至相同的另一个体；而在离散空间中，不存在完全相同的个体，每个元素自成一派。
- 一般而言，连续动作空间中的变量可以根据具体取值来衡量数值大小和远近关系。比如放置物体时的位置坐标，抛掷物体时的施加力的大小（例如 $1.1N > 1N$ ，那么两者之差 $0.1N$ 能够明确表达两个力之间的数量关系差异）。而离散动作空间中的变量，则需要用一组类别来区分彼此，集合中各个不同的状态点或动作点之间存在着显著的性质上的差异，但他们之间并没有具体的数量关系（比如之前的马里奥游戏中0指代向左2指代跳跃，但两者相减并没有实际意义）。具体实例比如十字路口处的直行左转右转，开关阀门的开启与闭合等等。

总结来说，连续动作空间的特点可以概括为以下三点：

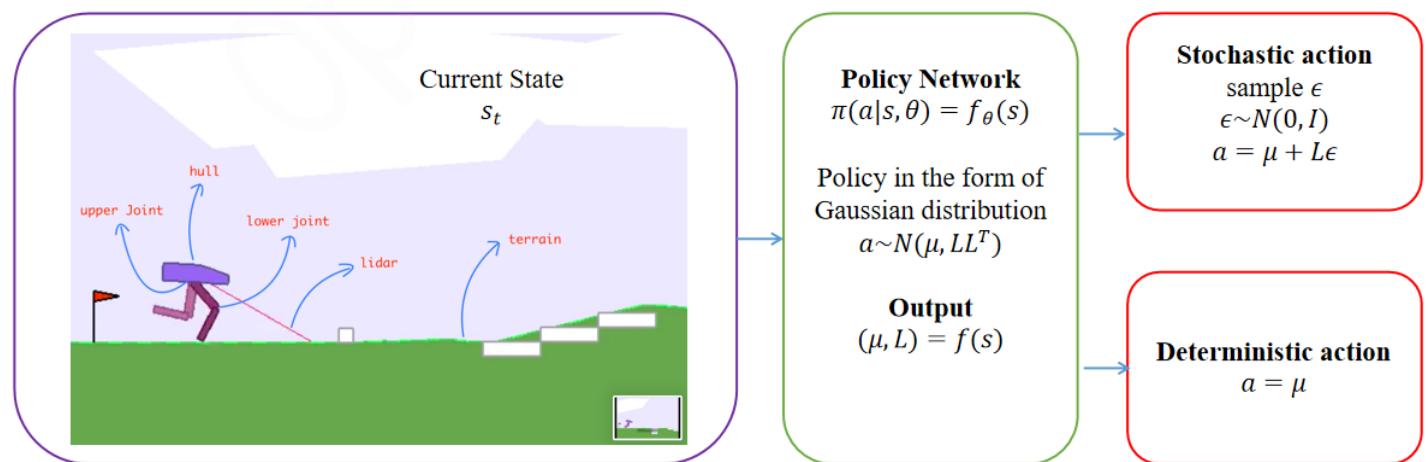
- 连续的，稠密的，不可数个动作元素组成的连续区间（类似回归问题）
- 可以根据具体取值来衡量数值大小和远近关系
- 可以直接回归，也可构建概率分布采样动作（参数化分布）

基于上述分析，关于 PPO 和连续动作空间相关的算法设计，本章节将会从以下两部分展开：

- 如何设计 PPO 算法去输出连续动作
- PPO 和其他连续空间上的决策算法（以 DDPG 为例）的对比

计算图设计（前向和反向计算图）

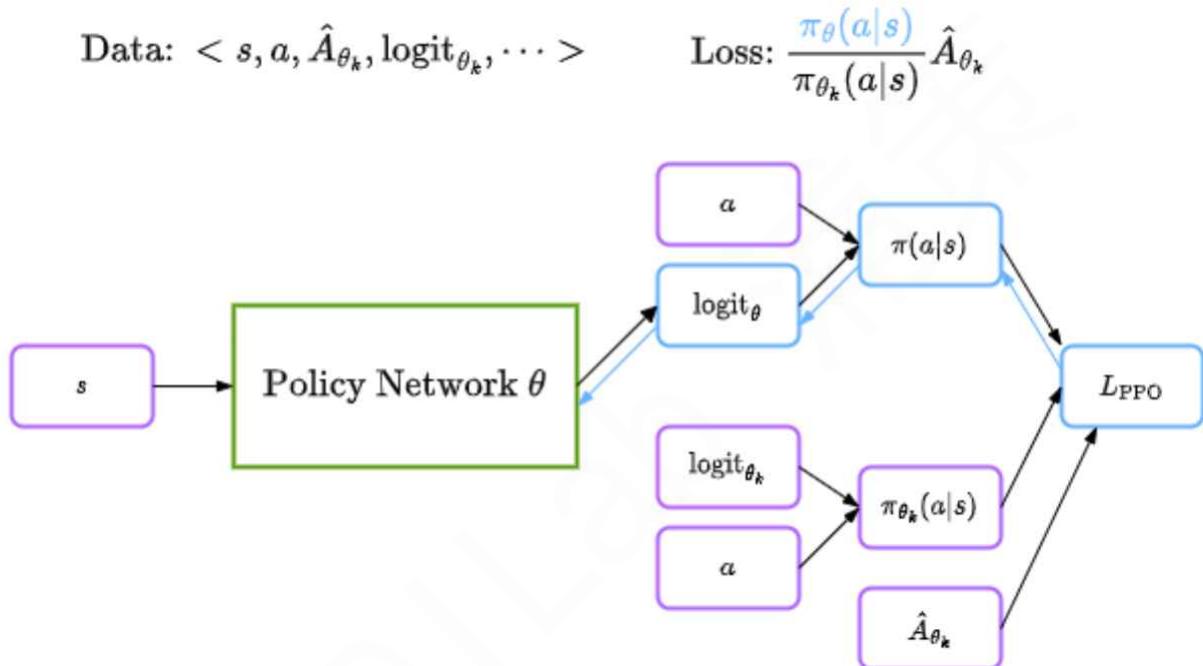
对于 bipedalwalker 环境这样的连续动作空间如何使用 PPO 算法。这里将从前向和反向计算图分别展开：



(图18：bipedalwalker 连续动作环境设计策略并生成动作的前向计算图)

- **前向计算图：**环境内部将机器人各个部件角速度、水平速度、垂直速度、关节位置和关节角速度、腿与地面的接触以及 10 个激光雷达测距仪测量值等信息抽象为一个 24 维的向量观测状态。这个状态信息会送入策略网络 (policy network) ，输出所谓的 “logit” ，但和离散动作空间不同的是，PPO 在连续动作空间采用参数化分布的形式，即针对每一维连续动作输出相应的参数 μ 和 L (均值和标准差) ，然后利用这些参数去构建一个相应的高斯分布，并借助这个高斯分布来进行后续的动作选择。相应的，这里也会存在随机性和确定性决策：

- 前者是从高斯分布中采样获得动作
- 后者是直接将动作分布的均值 μ 作为策略输出的动作，即最大化概率密度的点估计



(图19：连续动作环境使用 PPO 进行优化时的前向和反向计算图。方形代表神经网络，圆角方形代表输入/输出/中间数据节点，只有蓝色注明的数据节点会传递梯度，其他数据节点只是参与计算)

- **反向计算图：**将上文的状态输入和动作输出简记之后，图19得到了 PPO 在连续动作空间上的所有关键数据节点，即经典的深度强化学习训练过程，策略网络输入状态输出 logit 并结合其他数据计算损失函数进行优化。但是在 PPO 的各个优化项中，只有重要性采样 (Importance Sampling) 这一项的分子项回传梯度，分母和优势函数 (Advantage) 都不参与梯度反向传播，只是作为一个计算项参与其中。

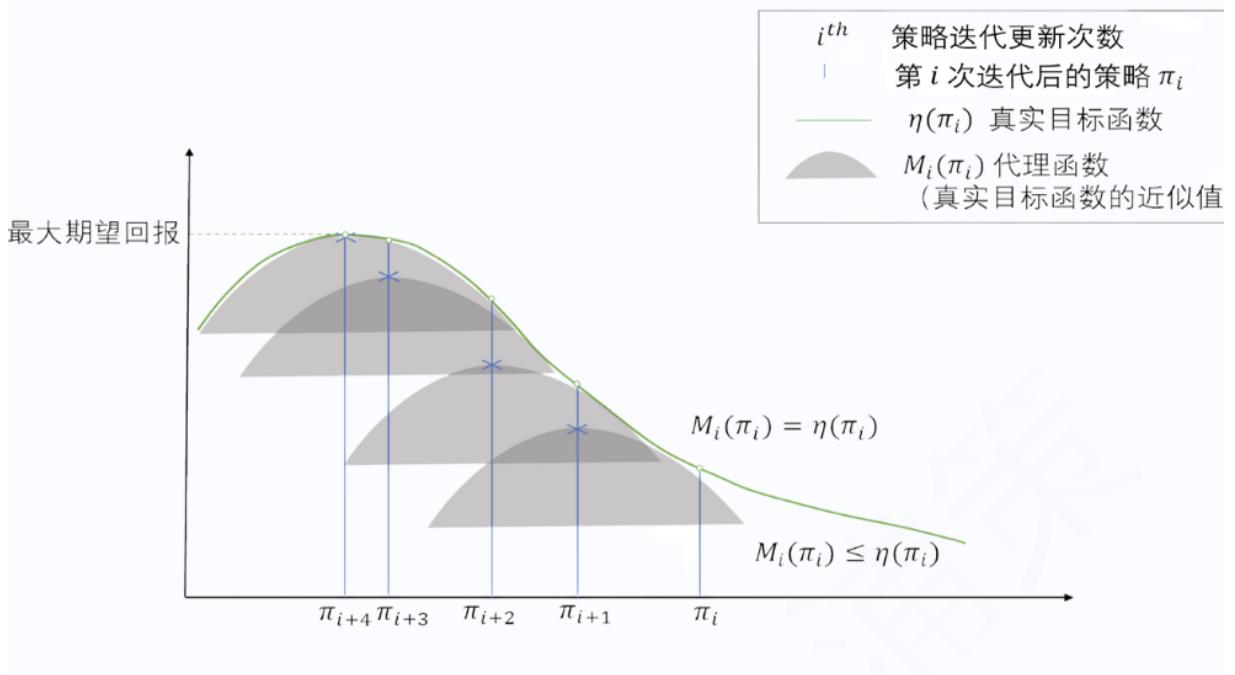
另外，对于连续动作空间，如果某个连续动作的最优解是单峰的（大多数控制任务中很常见），就常常使用高斯分布来建模连续动作。一个高斯分布的形态，只决定于其均值向量 μ 与协方差矩阵 L 。因此仅需让策略神经网络输出高斯分布的均值与方差的数值，随后从该分布中采样即可。除此之外，实际应用中也可以做进一步简化，如果假设不同维度的连续动作是互相独立的分布，那么协方差矩阵可以退化为对角矩阵 $[\sigma_{ii}]$ 。

注：关于强化学习算法和重参数化方法的进一步联系可以参考本节课的[补充材料](#)。

PPO 与 DDPG 算法对比

PPO 和 DDPG 都可以用于经典连续动作空间的决策问题，那么这两个算法相比有什么不同点呢？

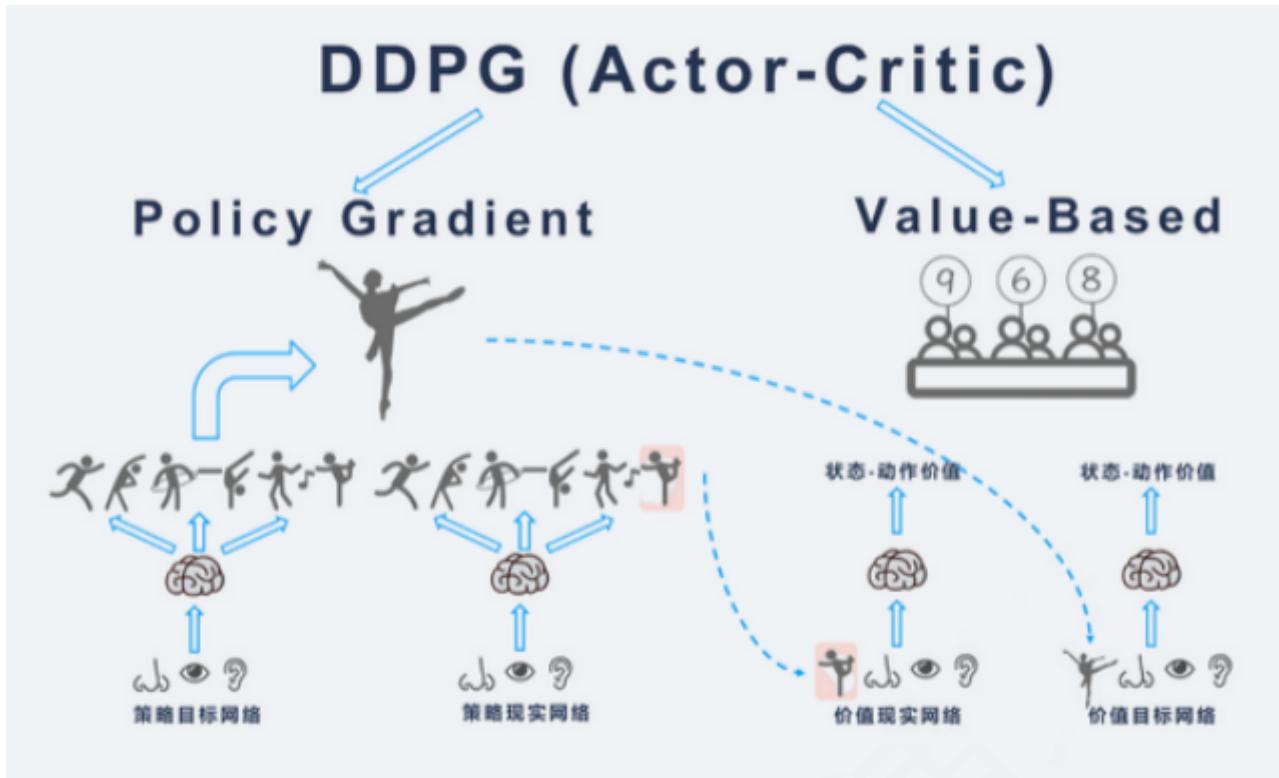
设计理念



(图20：TRPO/PPO 策略提升的迭代过程图)

PPO 算法来自论文《Proximal Policy Optimization Algorithms》，是经典的策略梯度算法的延伸，中文译为近端策略优化算法。顾名思义，PPO 主要是围绕第一节课中提到的策略提升定理展开，也就是 TRPO 算法的相关算法设计和逻辑，核心思想是在一个临近的信赖区域内，让策略朝着优化目标前进（即最大化累计回报），由此推导出的算法是在原始优化目标的基础上，额外增加单次优化前后，对策略变化距离的限制。只是在具体优化实现中，PPO 的前身 TRPO 为了控制优化中策略的变化幅度，在算法中使用了优化前后策略函数的 KL 散度作为约束，进而优化方法较为复杂（具体可参考课程第一讲的[补充材料](#)），而 PPO 继承了 TRPO 的核心理念，采用了更易于计算的 clip 函数替代 KL 散度来限制策略的更新幅度，侧重于保持策略函数的近端优化和渐进单调提升，同时，其价值函数主要起辅助评价作用，只是用来计算优势函数 Advantage，类似一个权重乘到优化项上，并不传递梯度：

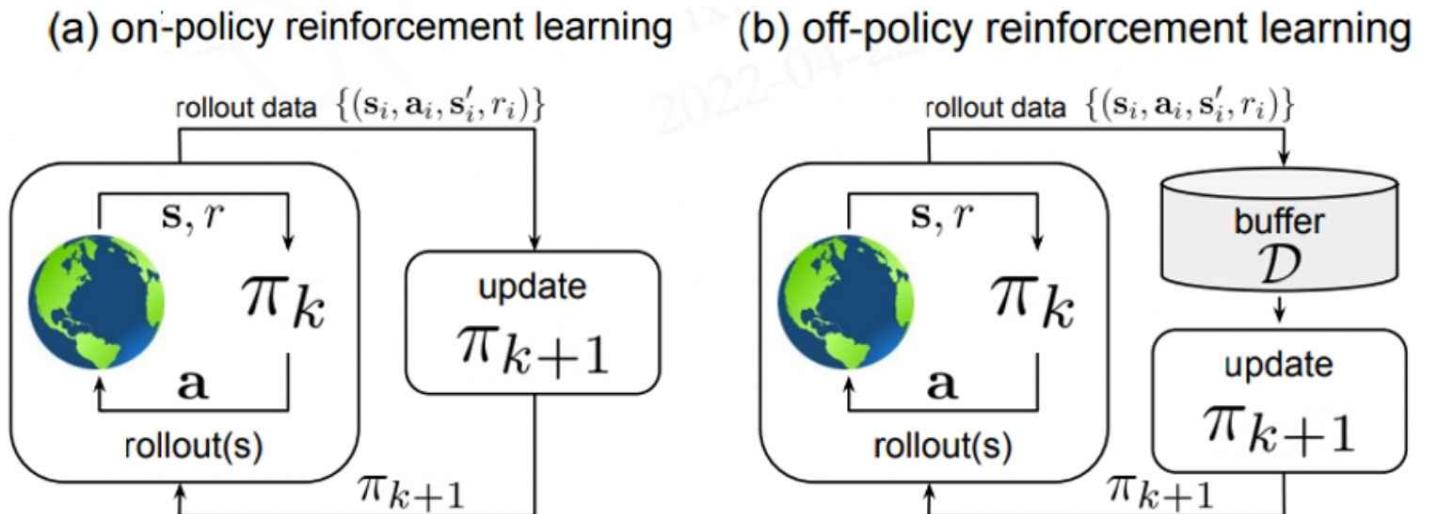
$$J^{\text{CLIP}}(\theta') = \mathbb{E}_{(s_t, a_t) \sim \pi_{\theta'}} [\min\left(\frac{p_\theta(a_t | s_t)}{p_{\theta'}(a_t | s_t)} A^{\theta'}(s_t, a_t), \text{clip}\left(\frac{p_\theta(a_t | s_t)}{p_{\theta'}(a_t | s_t)}, 1 - \epsilon, 1 + \epsilon\right) A^{\theta'}(s_t, a_t)\right)]$$



(图21：DDPG 的算法设计概述，主要围绕优化动作价值函数展开，策略函数用对价值函数求极值的方式引导生成)

相应的，DDPG 算法来自论文《Continuous control with deep reinforcement learning》[11]。DDPG 的全称为 Deep Deterministic Policy Gradient，中文译为使用深度学习的确定性策略梯度算法。虽然和 PPO 一样，它们俩都是 Actor-Critic 系列算法，但两者的设计理念区别很大，DDPG 使用了确定性策略，即一个参数化的策略函数 $\mu(s|\theta_\mu)$ ，而 PPO 是随机性策略，输出一个选择动作的概率分布。DDPG 算法围绕优化动作价值函数 (Q Function) 展开，侧重于最小化价值函数的建模误差，而其策略函数由对价值函数求极值的方式引导生成。如图20所示，就是策略网络输出一个动作，交给价值网络来判断动作的好坏，并通过价值网络回传梯度来指导策略进行更新（特性上更类似于Deep Q-learning [12]），更详细的关于 DDPG 算法设计的细节可以参考原始论文。

数据属性 (On-Policy VS Off-Policy)



(图22：强化学习中不同的数据利用范式示意图)

从数据属性上看：

- PPO 需要尽可能地选用最新 Policy 采样获得的数据轨迹，是一个 on-policy 算法。
- 而 DDPG 则可以使用来自不同的 Policy 生成的数据，对当前的 Policy 进行训练，是一个 off-policy 算法。

PPO 算法需要使用在线数据是由策略提升定理的本质决定的 [13-14]，即使用下述公式进行更新：

$$\eta(\pi') = \eta(\pi) + E_{s,a \sim \pi'}(\Sigma_t \gamma^t A_\pi(s, a)) = \eta(\pi) + \Sigma_s \rho_{\pi'}(\Sigma_a \pi' A_\pi(s, a))$$

这个公式从原理上来讲，就是对于迭代更新前后的两个策略函数 π 与 π' ，如果希望它们的总回报值有稳定的更新提升，即 $\eta(\pi') > \eta(\pi)$ ，那么需要将目标函数设定为此式，并优化 π' 使其最大化累计回报 $\eta(\pi')$ 。但在实际操作中无法获得更新后的轨迹 $\rho \sim \pi'$ 的数据，因而只能选取更新前的轨迹 $\rho \sim \pi$ ，来近似最大化下式：

$$J(\pi') = \eta(\pi) + \Sigma_s \rho_{\pi'}(\Sigma_a \pi' A_\pi(s, a)) \approx \eta(\pi) + \Sigma_s \rho_\pi(\Sigma_a \pi' A_\pi(s, a))$$

相应地，为了使策略提升定理有效，需要保证两个策略接近，它们所采集的轨迹也接近，这样才可以使训练稳定。

而 DDPG 算法则并不依赖在线数据，这是因为 DDPG 的优化思路来源于 DPG，使用最小化均方贝尔曼误差来拟合值函数，随后使用值函数来引导策略函数的生成，并不会对动作求积分，因此可以使用不同策略的数据进行优化：

$$L(\theta_Q) = E_{data}(Q(s, a) - (r + \gamma \max Q_{target}(s', a')))^2$$
$$J(\theta_\mu) = E_{data}(Q(s, \mu(s|\theta_\mu)))$$

从算法性质上讲，DDPG 的优化思路更接近于 value-based 方法，其算法可靠性和值函数本身的建模质量很相关。

注：关于 PPO 和 DDPG 中是否使用重要性采样的详细思考可以参考本节课的[相关补充材料](#)。

代码：如何实现高斯分布的策略函数

关于 PPO 对于连续动作空间神经网络设计的部分，可以参考课程第二章的[相关代码](#)。以下部分将进一步详述如何使用 PyTorch distributions 库的相关函数，实现使用高斯分布来将动作参数化的方法：

$$a|s \sim \mathcal{N}(\mu(s), \Sigma(s))$$
$$\pi(a|s) = \frac{\exp(-\frac{1}{2}(a - \mu)^T \Sigma^{-1}(a - \mu))}{\sqrt{(2\pi)^k |\Sigma|^{-1}}}$$

独立分布的动作

对于独立分布的动作，可以使用多个独立分布的单维高斯分布来实现参数化。因为一个单维高斯分布由均值 μ 和方差 σ^2 构成，具体地，对于 PyTorch distributions Normal 类就需要提供均值 μ 和标准差 σ 。

由于标准差是非负数，代码实现中可以使用对数来构造一个无界的標準差的参数，然后再用指数操作来还原它。

如果环境有动作区间的限制，除了通过环境中适配截断动作这个处理办法之外，也可以通过函数变换，来让无界的高斯分布，限定至一个有界的区间内。

```
1 import torch
2 from torch import nn
3 from torch.distributions import Independent, Normal, TransformedDistribution, Ta
4
5 # Get mean vector
6 mean = nn.Parameter(torch.zeros(action_dim))
7 # Get log_sigma to ensure sigma is positive
8 log_sigma = nn.Parameter(torch.zeros(action_dim))
9 # usually they are model output
10 mean, log_sigma = model(obs)
11
12 # Define Independent Normal distribution with 1 Batch size
13 sigma=torch.exp(log_sigma)
14 # Get normal distribution and reinterpret 1 batch dim to be event dim
15 distribution = Independent(Normal(output_mean, output_std), reinterpreted_batch_
16 # Transform distribution into a bounded interval by functions such as Tanh into
17 distribution = TransformedDistribution(base_distribution=distribution, transform
18
19 # Sample action from Distribution with Autograd to it
20 action=distribution.rsample()
21 # Sample action from Distribution without Autograd to it
22 action=distribution.sample()
```

非独立分布的动作

对于存在协方差的非独立的动作分布，可以使用一个多维高斯分布来实现参数化。多维高斯分布由均值向量 μ 和协方差矩阵 $\Sigma = LL^T$ 构成。

对于 PyTorch distributions MultivariateNormal 类就可以提供均值 μ 和协方差矩阵 Σ 来构造它。

因为协方差矩阵是一个对称的(半)正定矩阵，而且需要符合几个特殊要求：

- 各维度标准差为非负。
- 协方差项的绝对值，小于两个标准差项之积。等价于相关系数位于 $[-1, 1]$ 之间。

但是实践中一般无法直接使用一个矩阵来编码协方差矩阵 Σ ，那样参数自由度太大。而其矩阵分解的下三角矩阵 L 虽然满足自由度的数目，其所需满足的边界要求也不易编码。

因此，这里需要借助容易编码边界的相关系数矩阵 C ，和标准差向量 σ ，来构造协方差矩阵 $\Sigma = [\sigma_i c_{ij} \sigma_j]$ 。

矩阵 C 自由度为 $\frac{(D-1)(D)}{2}$ ，标准差向量 σ 自由度为 D ，共计 $\frac{(D+1)(D)}{2}$ 个参数变量。

```

1 import torch
2 from torch import nn
3 from torch.distributions import Independent, MultivariateNormal, TransformedDist
4
5 # Get mean vector
6 mean = nn.Parameter(torch.zeros(action_dim))
7 # Get log_sigma to ensure sigma is positive
8 log_sigma = nn.Parameter(torch.zeros(action_dim))
9 # Get correlation_param to ensure sigma is positive
10 correlation_param = nn.Parameter(torch.zeros(action_dim*(action_dim-1)//2))
11 # usually they are model output
12 mean, log_sigma, correlation_param = model(obs)
13
14 # Define Independent Normal distribution with 1 Batch size
15 sigma=torch.exp(log_sigma)
16 # Get low triangle decomposition matrix of correlation matrix
17 low_tri = CorrCholeskyTransform()(correlation_param)
18 # Get covariance matrix
19 corvar_matrix=torch.mul(torch.einsum('bij,bkj->bik', low_tri, low_tri), torch.ei
20 # Get multivariate normal distribution and reinterpret 1 batch dim to be event a
21 distribution = Independent(MultivariateNormal(loc=mean, covariance_matrix=corvar
22 # Transform distribution into a bounded interval by functions such as Tanh into
23 distribution = TransformedDistribution(base_distribution=distribution, transform
24
25 # Sample action from Distribution with Autograd to it
26 action=distribution.rsample()
27 # Sample action from Distribution without Autograd to it
28 action=distribution.sample()

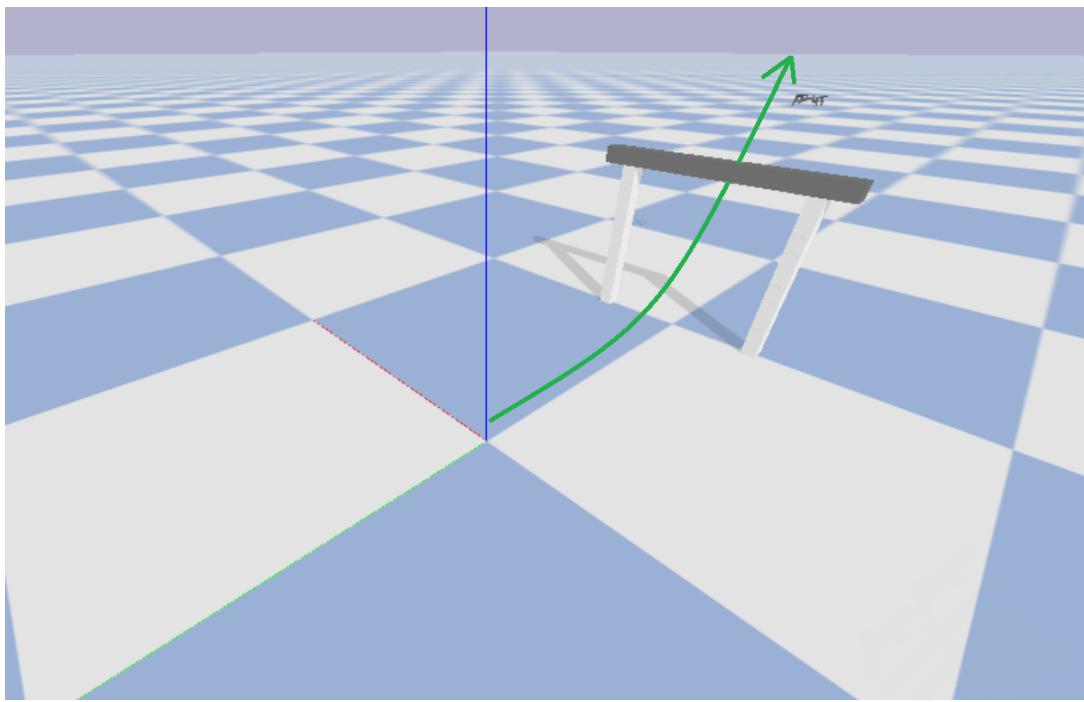
```

实践：使用 PPO 来解决无人机姿态控制问题

在明确 PPO + 连续动作空间的算法理论和代码实现细节后，本小节尝试来应用相关知识解决相应实际问题，这里选择一个简明的**无人机姿态连续控制**问题。本章节将会介绍环境的各种细节定义以及使用 PPO 的实践经验。

问题背景

近年来，多旋翼无人机技术发展迅速，该技术已经能够赋予很多传统行业一些新的技术方法和商业解决方案，比如物流业、农林业、旅游业等。本节课以一个开源的无人机模拟环境——gym-pybullet-drones [15-16] 为例介绍 PPO 在连续动作空间上的应用。研究对象是一台性能指标开源的四旋翼小无人机，bitcrazeflie [17]。这里选定了一个简单的任务 Flythroughgate，目标是让四旋翼无人机从原点出发，飞行穿过一座门框。



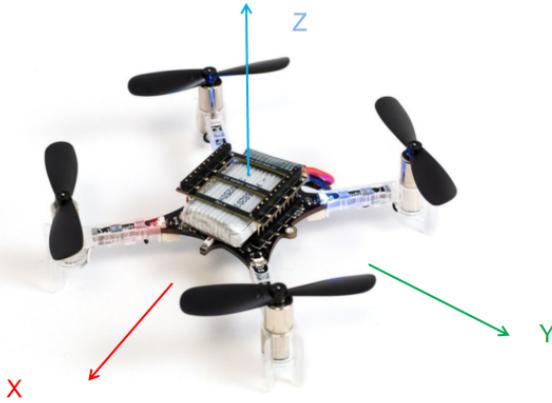
(图23：Flythroughgate 无人机任务环境示意图)

环境 MDP 设定

状态空间：该任务的状态空间为一个12维度的连续空间，按顺序分别是：

变量	性质	区间	备注
x, y, z,	无人机在世界坐标中的位置	[−1, 1]	经过clip截断函数和scale范围缩放处理，对于z坐标，被限制在 [0, 1] 之内
y, p, r	r~roll滚转角, p~pitch俯仰角, y~yaw偏航角	[−1, 1]	r~roll滚转角与p~pitch俯仰角，经过clip截断和scale范围缩放处理。 y~yaw偏航角仅经过scale范围缩放处理。
vx, vy, vz	无人机在世界坐标中的速度	[−1, 1]	经过clip截断函数和scale范围缩放处理
wx, wy, wz	无人机在世界坐标中的角速度	[−1, 1]	经过clip截断函数和scale范围缩放处理

动作空间：该任务提供了多种控制无人机的模式，包括PID控制模式，转速控制模式等，对应着不同的动作空间。



(图24：四旋翼无人机示意图)

1. 直接控制四个旋翼的转速模式，4维连续动作空间：

$$\left(\frac{20 * \Delta RPM_0}{RMP_{hover}}, \frac{20 * \Delta RPM_1}{RMP_{hover}}, \frac{20 * \Delta RPM_2}{RMP_{hover}}, \frac{20 * \Delta RPM_3}{RMP_{hover}} \right)$$

上式中的分子是各个旋翼转速与悬停转速的差值，分母是悬停转速，整体取值范围为 $[-1, 1]$

2. PID 算法修正 position 模式，三维连续空间：

$$(10 * \Delta x, 10 * \Delta y, 10 * \Delta z)$$

分别是各个方向上所期望无人机运动的位置修正量的10倍大小，单位为米，取值范围为 $[-1, 1]$ 。即智能体只需要给出下一步期望到达的空间运动量，PID 算法将根据当前位置和期望运动量计算得出实际的操作控制信号。

3. PID 算法修正velocity 模式，4维连续空间：

$$(V_x/|V|, V_y/|V|, V_z/|V|, |V|/|V_{max}|)$$

分别是各个方向上的速度分量的相对大小，以及速度相比系统最大允许速度的相对大小。

采用第一种方式，直接控制无人机四个旋翼各自的转速，将其作为智能体的动作输出，这在物理概念上十分直观。但这种模式在实际训练中会使得无人机的状态十分不稳定，常常倾斜起飞或翻转，而无法达成目标。这是为什么呢？举例来说，如果想让无人机学习到竖直原地起飞这样的动作，就需要它控制四个旋翼转速保持一致，这件事对于人类来说是很简单的概念，但对于智能体来讲是庞大的连续动作空间中的一个点，因此它很难在没有任何辅助信息的情况下探索到这件事。在实际运行中，还会可以发现四个旋翼之间的配合度往往会有比较差。因为转速的变化在带来升力的同时也会带来扭矩，扭矩会传递给整体，带来整体的姿态变化，而姿态的变化会影响力的方向，进而让无人机进入负面的状态而产生损失，强化学习算法因此也难以训练。

为了解决第一种动作空间的问题，可以借助课程一开始讲到的 Action Shaping 的思路，具体来说就是将强化学习和经典控制理论中的 PID 结合起来使用，即对应到第二和第三种控制方式。实践中，采用 PID 算法可以很大程度上协调和统一四个旋翼之间的转速和配合。因此智能体在这类模式下，训练较为

平稳，无论是 PID-position 还是 PID-velocity 模式下都有较好的效果，课程中为了简便主要使用 PID-position 模式。

奖励空间：四旋翼无人机的每个时刻都会基于其位置而获得奖励，计算方法如下：

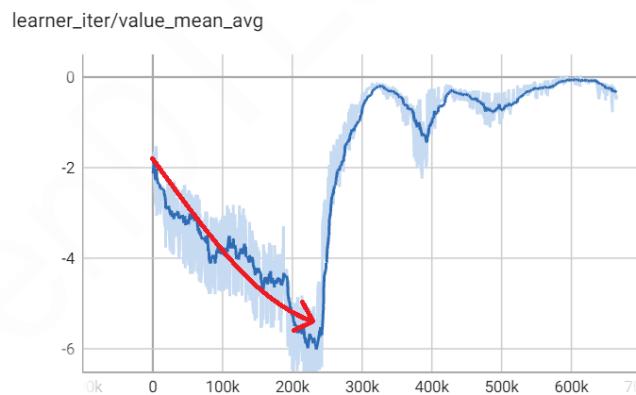
$$r(x, y, z) = -10 * ([x, y, z] - [0, -2 * \frac{t}{T}, 0.75])^2$$

通过奖励函数的组成，可以看出，该任务将激励无人机往y值减小，且z值增大至某一定高度的轨迹上飞行。此外，由于还会受到不可见的障碍物的影响（门框与地面），存在损失和错过奖励的风险。因此，在该任务中无人机需要学习成功起飞，飞离地面，通过试错的方式，感知和躲避障碍。

实验结果展示

接下来的部分将介绍 PPO 智能体完整的训练过程，具体分析使用 PPO 控制 Flythroughgate 的各个训练阶段：

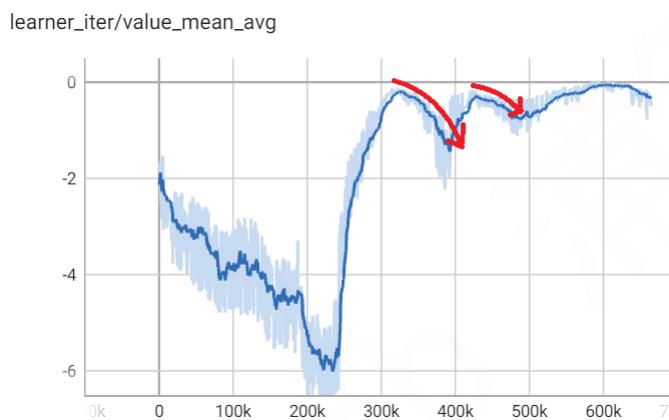
- 在初始阶段（训练迭代0~200K），从视频中可以看到，四旋翼无人机驻留在原地，还未学会起飞。这是因为根据强化学习环境的奖励的设计，此时无人机会在每一帧收到一个比较差的 reward。这个阶段 PPO 算法的 Actor 的策略网络显然还无法输出好的决策，这是因为从 Critic（价值网络）是随机初始化的，无法给出正确引导，它需要拟合现状，学习到停留在原地是坏的决策（这从 Critic 输出的均值逐渐下降可以看出，即下图红色箭头）：



（图25：四旋翼无人机训练各迭代的轨迹的动作价值的平均值统计结果，注：横坐标为训练迭代次数，纵坐标为各迭代的轨迹的动作价值的平均值，红色箭头为调整阶段标注（训练迭代 0~200K））

- 在前期阶段（训练迭代200K~300K），四旋翼无人机的 Critic 网络已经逐渐趋向成熟，它开始“意识”到，起飞提升高度，是一件会被鼓励的事情。无人机的高度越高，奖励也越大。因此在这个阶段，从视频中可以看到，无人机在 100K 训练迭代内，迅速地学到了快速起飞这件事情。图中可以看到价值网络的输出明显快速地变好。

- 虽然在刚开始的时候，无人机的飞行方向是反向远离目标方向的（比如训练迭代为220K时）。这说明此时 Actor 网络还没有学习到起飞之后的合理策略。这很正常，因为之前无人机并没有成功飞离地面的经验，没有相关的历史数据，自然 Critic 网络也没有对飞行方向有合理的经验评价，所以 Actor 网络也不会太成熟。但只要无人机离开地面，无人机就会慢慢学到，朝着目标方向飞行才是对的。截至训练迭代300K时，无人机已经大致学会了选择合适的方向。
- 在调整阶段（训练迭代300K~400K），这个阶段四旋翼无人机开始微调策略网络。虽然环境中存在大门这个物理实体，但无人机的观测空间中并没有关于这个实体的描述，因此无法直接感知到它的存在。这使得在训练迭代310K所对应的视频里，无人机为了尝试追求更高的奖励，采纳的飞行路线正好和大门的横梁发生了撞击而掉落，使得奖励整体偶尔会变差。
 - 此类的事故导致的训练不稳定现象，在价值网络的平均输出的曲线中也可以清晰的看到：

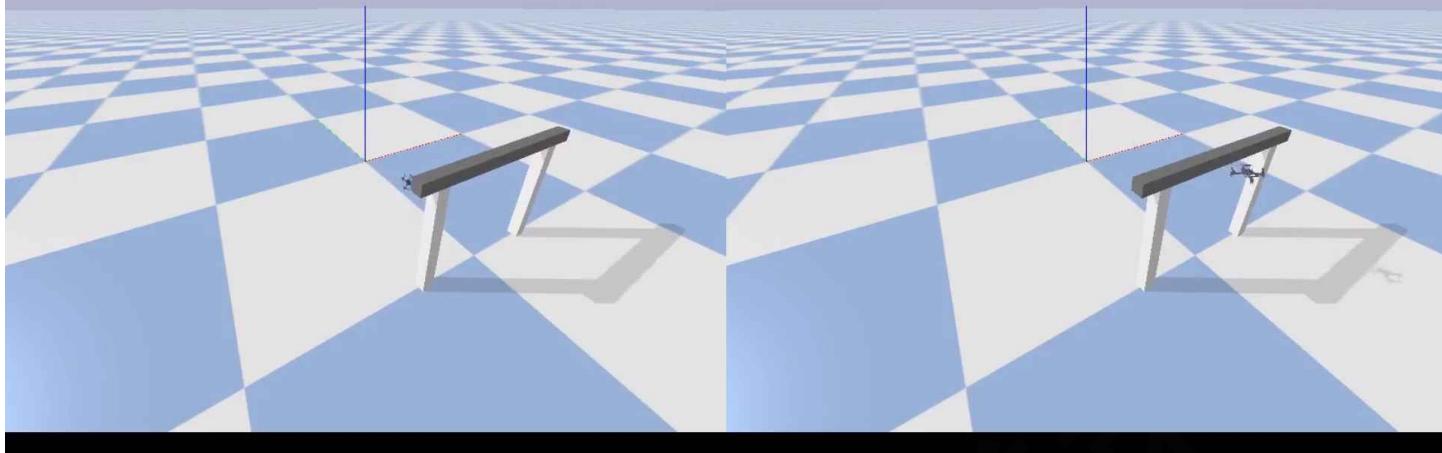


（图26：四旋翼无人机训练各迭代的轨迹的动作价值的平均值统计结果，注：横坐标为训练迭代次数，纵坐标为各迭代的轨迹的动作价值的平均值，红色箭头为调整阶段标注（训练迭代300K~500K））

- 这也就是强化学习中，所谓 Trial-Error 模式的一个特征，通过尝试和犯错，这样智能体的评价器会学习到这种惩罚，从而“感知”到横梁与立柱的存在。
- 最终，四旋翼无人机逐渐调整策略网络，找到了一条，既可以最大化奖励，又可以规避惩罚的最优路线，训练收敛时和随机初始化时无人机完整的轨迹对比视频如下（完整样例也可以参考[官方示例 ISSUE](#)）：

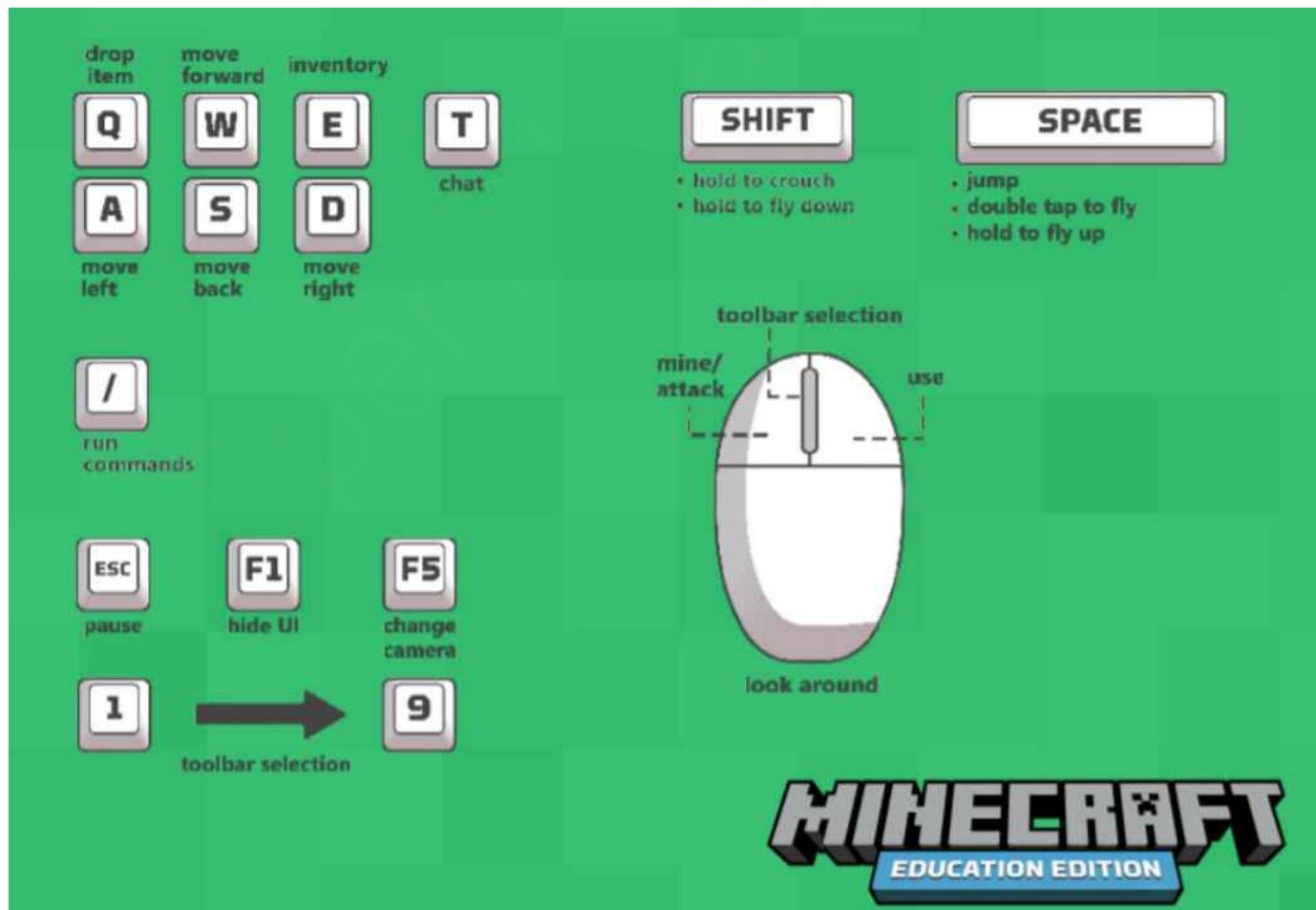
random agent

trained agent



(视频2：Flythroughgate 环境成功案例示例视频，四旋翼无人机从起点出发，成功飞行穿过一座门框)

2.4 多维离散 (multi-discrete) 动作空间



(图27：《MineCraft》游戏中的键盘鼠标操作示意图，这就是一种经典的多维离散 (multi-discrete) 动作空间。每一个操作按键，鼠标的各种点击操作，都是一个单独的离散动作空间，整体合

起来就构成了多维离散动作空间。)

引言：多维离散动作空间的定义

在介绍完最经典的两种动作空间——离散和连续动作空间之后，本节课将以这它们为基石，去构造真实决策问题中更多复杂的动作空间。而在本小节，首先将讨论一个相对复杂的动作空间：**多维离散(multi-discrete) 动作空间。**

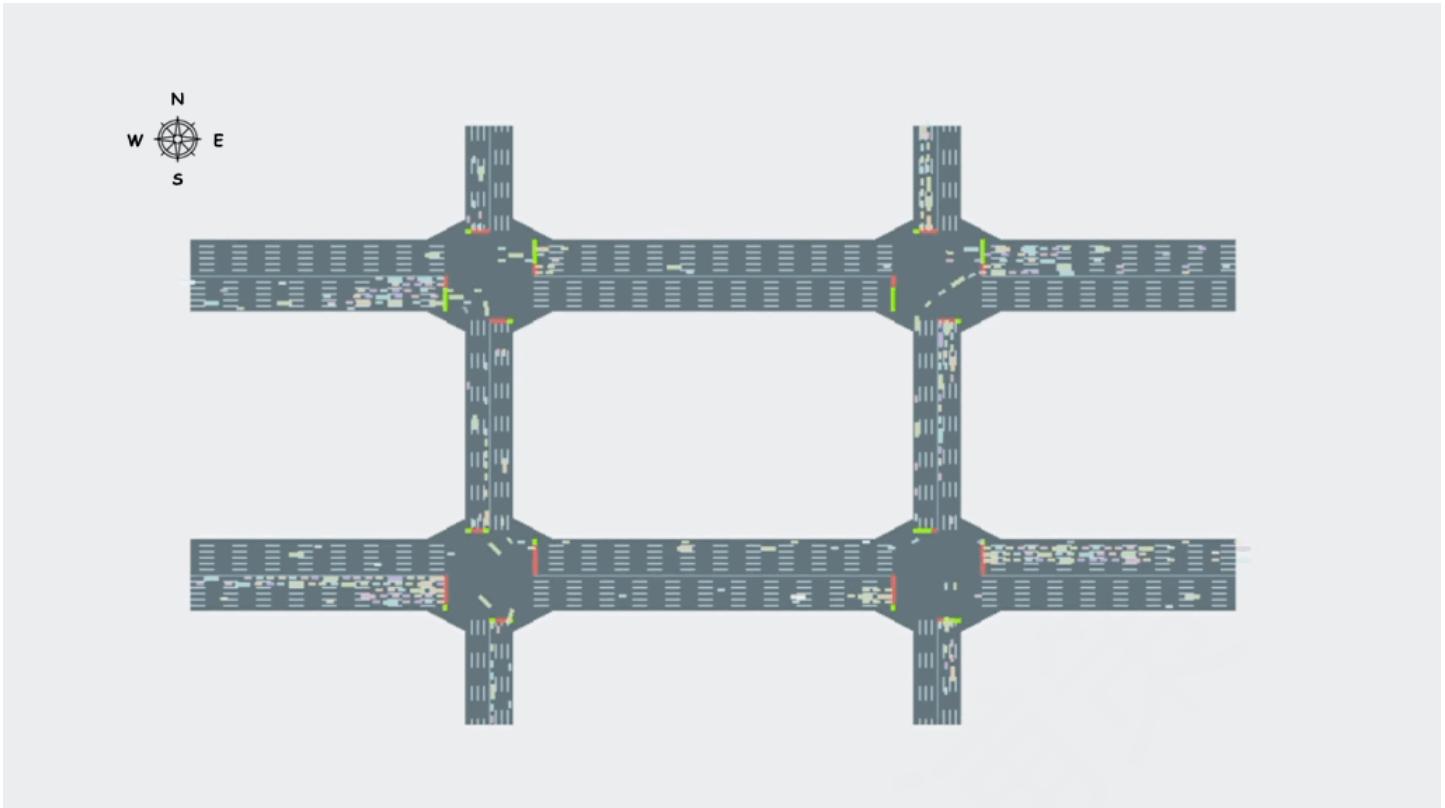
首先，这里来明晰一下 multi-discrete 动作空间的定义：顾名思义，它指的是具有多个维度的离散动作空间，换句话说，可以理解为将多个不同的离散动作空间组合在了一起，形成了一个新的动作空间。为了便于理解，以下列举几个多维离散动作空间的例子：

- 驾驶汽车的时候，既要控制汽车的档位（第一个离散动作空间），又要控制是踩油门还是刹车（第二个离散动作空间），因此是一个多维离散动作空间；
- 使用电脑的时候，既要选择敲击键盘的哪一个按键（第一个离散动作空间），又要控制点击鼠标的左键或者右键（第二个离散动作空间），这也构成了多维离散动作空间。

从上述例子可以看出多维离散动作空间在实际决策问题中是很常见的。而接下来将介绍其严格的数学定义：

multi-discrete 动作空间的数学定义为：若 \mathbf{a} 为多重离散动作空间中的一个动作，则它是一个长度为 N 的向量， N 是离散空间的个数，其中第 i 个离散空间的大小为 N_i 。这个向量中第 i 个元素 a_i 表示在对应的离散空间中采取的动作，而 a_i 可选择的动作取值集合为 $\{0, 1, \dots, N_i - 1\}$ 。

特别地，当智能体策略需要对**若干个相对独立的个体同时进行控制时**，multi-discrete 动作空间就是实际问题中最常用的建模方式。例如交通信号控制相关决策问题，如下图所示是一个 2×2 的十字路口路网图，其中每个十字路口有四种相位，即四种通行方式（南北直行，南北左转，东西直行，东西左转），那么其实每个路口都对应一个4维的离散动作空间。但当决策行为需要扩大到整个路网系统时，就需要将这4个十字路口的离散动作空间合在一起，因此构成了一个 multi-discrete 动作空间，它是4个4维离散动作空间的组合。



(图28：交通信号控制 2×2 路网示例图，包含车流和交通信号灯信息。)

总结来说，对于多维离散动作空间，可以总结出以下三个特点：

- 多个不同离散空间的组合（比如键盘，多路口信控）。
- 离散动作的个数可能非常大。比如抛开上述的 2×2 路网，倘若需要处理更大的区域信控问题，那就需要操控几十甚至上百个路口，这时总的离散动作空间的个数就会变得非常之“多”了。
- 各个离散动作之间存在一些较弱的联系。比如上游的交通路口发生事故或者拥堵现象，那么这势必会对下游路口的决策也带来一定的影响。

理论：PPO 如何建模多维离散动作空间

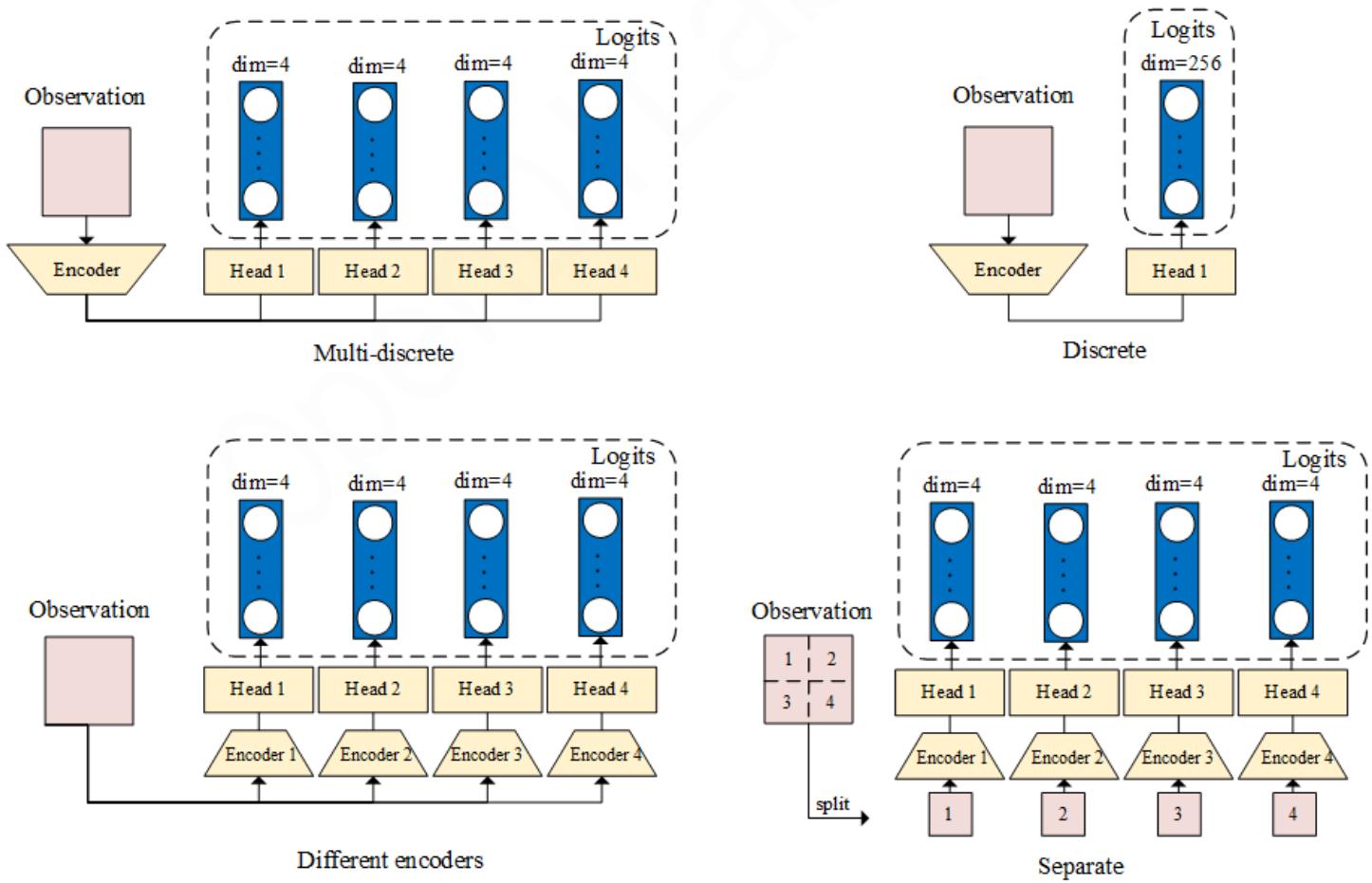
对于具体建模多维离散动作空间的方法，其实只需要继承离散动作空间的基本优化思路，并在网络结构设计上做一些相应的处理即可，这里有很多种不同的处理方法，具体如下所示（也可参考下方的图表理解）：

- multi-discrete：输入全局所有路口的观察信息，分别输出四个路口各自的决策动作，即使用一个观察空间为176维的 encoder + 4个4维的 head（各个路口的信息合在一起共用 encoder，独立训练 head）。
- discrete：先将多维离散动作空间转换成离散动作空间，新的离散动作空间是用原来各个离散部分的笛卡尔积进行表示。而在网络结构方面，编码部分依然使用 176 维输入的 encoder，输出部分则是一个 256 维的 head。
- different-encoders：使用4个观察空间为 176 维的 encoder + 4个4维的 head（即各个路口 head 和 encoder 都是独立训练的）。这种结构和 multi-discrete 较为类似，只是没有共享 encoder。

- separate: 使用4个观察空间为 44 维的 encoder (即只输入路口自身局部的 observation) + 4个4维的 head。这种结构相当于训练了四个完全独立的智能体，各个路口的 head 和 encoder 都是独立训练，每个智能体只根据局部的观察空间信息进行决策。

实验名称	每个encoder输入 观察空间的维度	每个head输出 logit的维度	encoder的数量	head的数量
multi-discrete	176	4	1	4
discrete	176	256	1	1
different- encoders	176	4	4	4
separate	44	4	4	4

(表1：多维离散动作空间四种网络结构设计的关键信息对比表。discrete这样的情形 head 的输出维度会变得非常大，进而可能存在一些优化困难问题，尤其当维度很多时就会出现组合爆炸现象。而最后两种方法则会需要更多的网络参数，计算开销会相对较大。)



(图29：多维离散动作空间四种网络结构设计的对比图。)

另外，上述的不同网络结构设计，优化时所用的方法都非常简单，沿用 PPO + 离散动作空间的设计即可，只是多个 head 的损失函数需要加起来，然后一起反向传播计算梯度。

实践：使用 PPO 来解决交通信号控制问题

问题背景

对于多维离散动作空间，本节课选择 CityFlow [18-19] 仿真器来模拟经典的交通信号控制问题。具体来说，基于 CityFlow 构建了一个 2×2 路网地图，一共包含了4个十字路口。每个路口有四种通行方式，因此每个路口是4维的离散动作空间。相应地，决策任务的目标是寻找一个合适的智能体策略，使得路口车辆吞吐量实现最大化。



为什么每个路口的离散动作空间是4维的？

这里就涉及到交通信号控制理论中“相位”的概念。可以假想一下：如果当前所在的车道允许直行，同时迎面而来的车流也允许直行放行，那么这两股车流显然不会互相进行干扰，这是一个合理放行规则。正式一点来说，这种使得通行的所有车流都不会互相造成干扰的一组信号灯状态就被称作一个“相位”。为了通行的高效，自然希望在每个相位中，放行的车流数量越多越好、相位总数越少越好。那么最优的相位设置是怎样的呢？信号控制相关的研究指出，对于一个十字路口而言，最少需要4个相位（即前文中提到的南北直行等相位）才能保证所有车道正常通行。因此，在本课程的动作空间设计中，每个路口的信号灯状态设置为四个相位之一，进而动作空间就是4维的。

环境 MDP 设定

明白了这个应用问题的机理，这里来看看如何设计这个路网的 MDP 基础元素：

观察空间：每个路口用一个 44 维的特征向量表示，其中包含路口各车道的车辆分布信息，而四个路口总的观察空间维度为 $4 \times 44=176$ 。

动作空间：

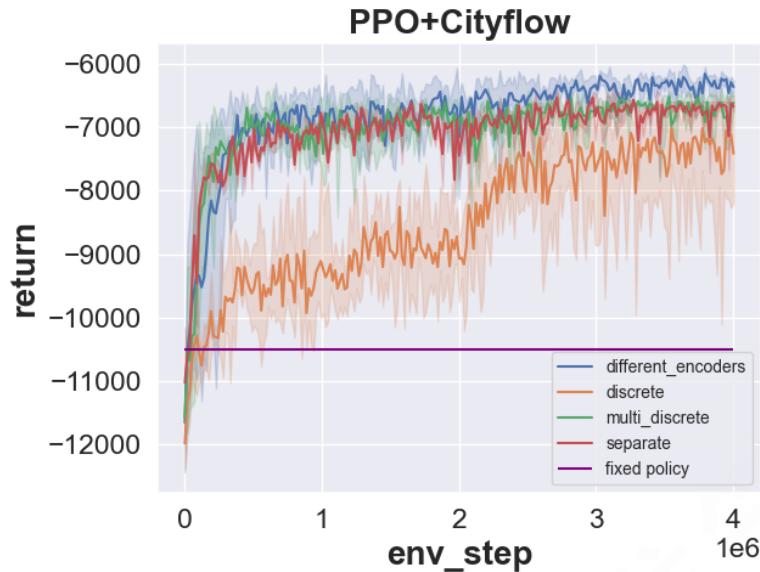
- 如果采用暴力离散化的形式，转换后的动作空间的维度应当为： $4^4 = 256$
- 而如果采用 multi discrete 的形式，则把每个路口选择的动作作为一个单独的4维 logit，每个 logit 单独计算损失函数，最终损失函数加起来一起反向传播，那么最后 logit 的总维度可以看作是： $4 + 4 + 4 + 4 = 16$ 。

奖励空间：每辆在进入路口时需要等待红灯的车都会产生负的奖励，同时等待时间越长，负奖励越大；每辆行驶出路口的车则会产生另一种正的奖励。两部分相结合就促使智能体学习到最大化路口吞吐量的策略。

实验结果展示

核心对比实验

将上文中定义的四种网络结构结合 PPO 算法，应用到 2×2 路网中进行实验，并对比了传统信号控制算法的结果（即 fixed policy），最终的实验结果如下图所示（每组取5组种子的平均值）：



(图30：交通信号控制 2×2 路网中，四种网络结构结合 PPO 算法实验结果对比图。强化学习优化的四组实验，在历经一定的优化部分后，都可以表现出比固定规则算法明显更好的性能。而对于 discrete 实验组（橙色线）则可以明显看出，在收敛速度和最终性能方面，相对其他三组结果存在明显劣势。different-encoders 实验组拥有最高的性能，这也是多智能体协作算法设计的基本形式，但其明显使用了更多倍的神经网络参数。综合计算开销和算法性能，multi-discrete 方式是推荐的最优选择，且能更适用于多路口的控制情形。)

那么强化学习算法究竟学到了什么样的策略来使得路网局面更优呢，下方的图31可视化了传统信号控制算法和 multi-discrete + PPO 算法具体给出的决策行为，即在一个 episode 内车辆等待时间的分布直方图。其中横轴是车辆通过红绿灯路口消耗的等待时间，纵轴指的是对应等待时间的车辆数量。PPO policy 可以很明显地优化掉那些等待时间非常长的情形，从而获得更高的 episode return。



(图31：固定规则策略和强化学习策略，相同控制周期内车辆等待时间的分布直方图。由于大多数车辆都可以在 $\text{waiting time} = 1$ 的时间内通过路口，因此为了对比更清晰，这里呈现的结果中去掉了

waiting time=1 的情况。)

总结通过上面的曲线和分析，可以总结出下列实验结论：

- 由于车流分布波动明显，且十字路口中两条道路的车道数量差距较大，所以采用传统算法控制信号灯很容易导致拥堵，进而使得路网内车辆的等待时间偏高。而强化学习的各种结果相比传统算法都有明显的提升，显著提高道路通行顺畅程度。
- different-encoders 这组实验类似多智能体协作的处理方法，建模了路口之间的协作关系，不过这种方法需要的神经网络参数最多，在路口比较多时计算开销很大。
- separate 方法可以取得不错的效果，但是这种方法不共享神经网络的 encoder 部分，当路口比较多时计算开销就会很大，而且只获取路口自身的信息，对于更复杂的车流无法很好地做到路口控制策略之间的配合。
- discrete 方法，由于暴力离散化方法的缺陷，整体训练过程波动很大，而且收敛速度会较慢，最终的 episode return 也相对低一些。
- multi-discrete 方法是综合性能和计算开销的最优选择。

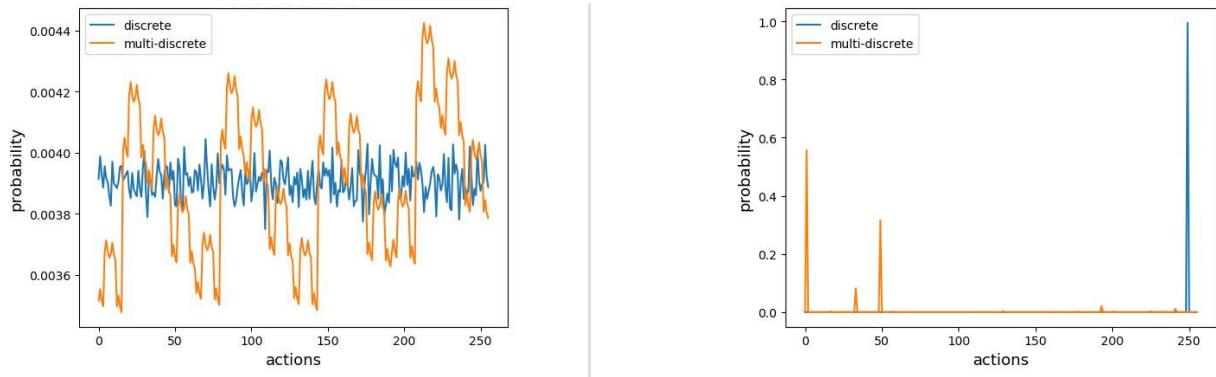
为什么 multi-discrete 在这个场景里强于 discrete

从上面的实验结果可以看出，multi-discrete PPO 相比于 discrete PPO 在交通信号控制环境上取得了明显更好的效果。但是这是为什么呢？本节课将从探索和利用两个角度对此进行探究。希望能给读者带来一些分析方法上的启发。

探索角度：相比于 discrete，multi-discrete 的探索更有**针对性**。具体来说，假设 multi-discrete 网络结构下的策略认为第一个路口应该左转，那么该策略就可以针对第一个路口左转进行针对性地探索，去看看当第一个路口左转时其他的路口怎么操作，这样就提高了探索的效率。为了验证上述说法的正确性，这里可视化出了在不同训练阶段，两种策略输出的概率，可以从中看出：

- 当训练前期，即 Iteration=1000 时，discrete PPO 只能暴力地去搜整个空间，仍然是在整个动作空间里比较“均匀”的遍历探索各种可能性，这种探索是平庸且效率极慢的。而 multi-discrete PPO 的概率明显出现了**周期性**，这说明它发现了某个路口的一个较好的动作，并对最终所有包含这一动作的整体动作都给出了较高的概率，这恰好符合之前上文“针对性探索”的假设，也就验证了 multi-discrete 的形式更能反映动作空间的内在结构。
- 当训练中期，即 Iteration=10000 时，multi-discrete PPO 的输出出现一种多峰的结构，更加便于探索。

Iteration 1000	Iteration 10000



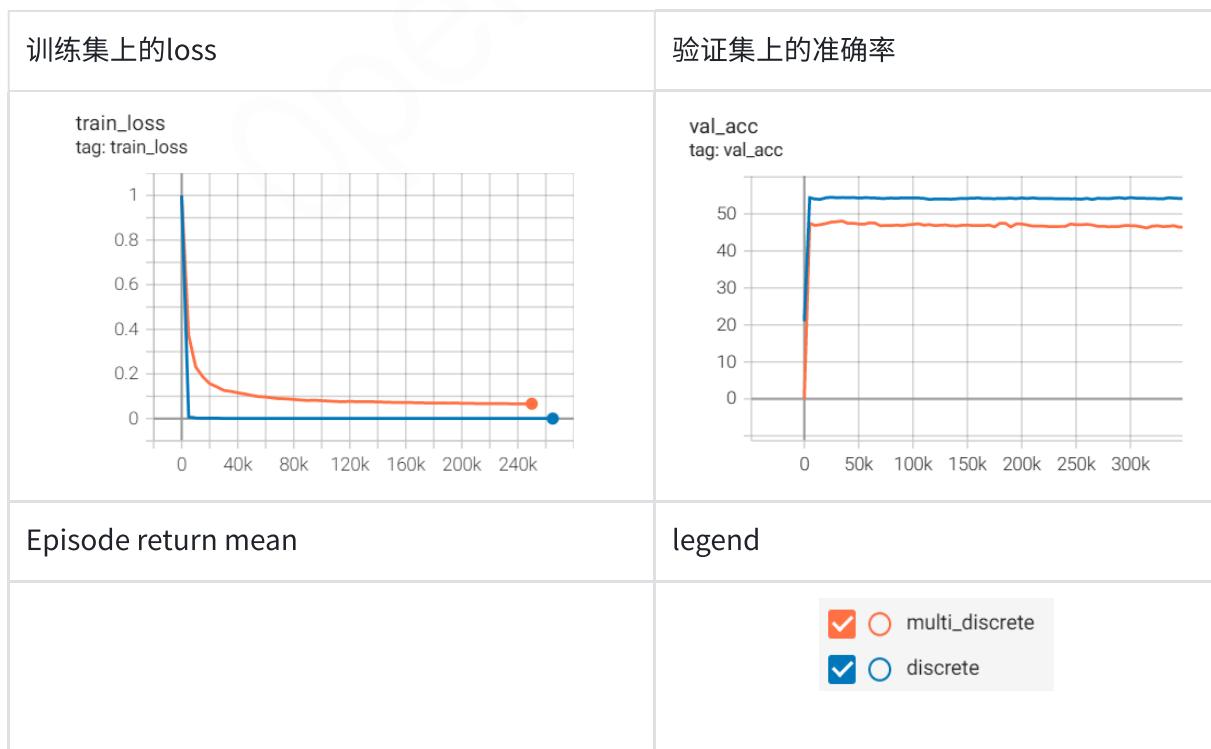
(图32：两种网络结构设计下的 PPO 智能体不同迭代数下的策略输出分布对比图。其中横坐标指的是不同的动作索引，暴力离散化后为 256 维，纵坐标为策略输出对应动作的概率。)

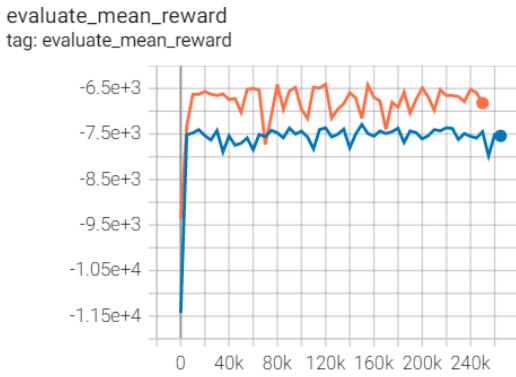
利用角度：第二方面，本节课尝试从利用的角度出发，并猜想这个方面 multi-discrete 相比 discrete 同样有优势，即：在收集同样多数据的情况下，multi-discrete PPO 可以获得比 discrete PPO 更好的性能，也就是说前者能够更好地利用已收集到的数据，学习到其中知识。

不过，因为强化学习是一个在线训练过程，训练数据分布是不断动态变化的，一般很难分析相应的数据利用能力和学习能力，但这里也可以简化问题，从在线的强化学习训练流程中抽取出为一个离线的监督学习部分。为了验证上述猜想，这里使用训练好的 PPO 收集了一组包含 2000 个状态转移对的专家数据。并分别使用从零初始化的 multi-discrete PPO 和 discrete PPO 模型，套用行为克隆（Behavior Cloning，简写为 BC）方法进行训练。

所谓的行为克隆，顾名思义，指的是需要收集“专家”在遇到某一状态 s 时，采取的动作 a ，然后以此作为监督学习的训练样本对策略进行拟合。用数学语言来说，就是收集一个包含大量专家状态-动作对的数据集 D ，然后训练一个策略，使得：

$$\operatorname{argmin}_{\pi} \mathbf{L}(\pi) = -\frac{1}{N} \sum_{k=1}^N \log \pi(a_k | s_k), \quad \text{for } s_k, a_k \in D$$

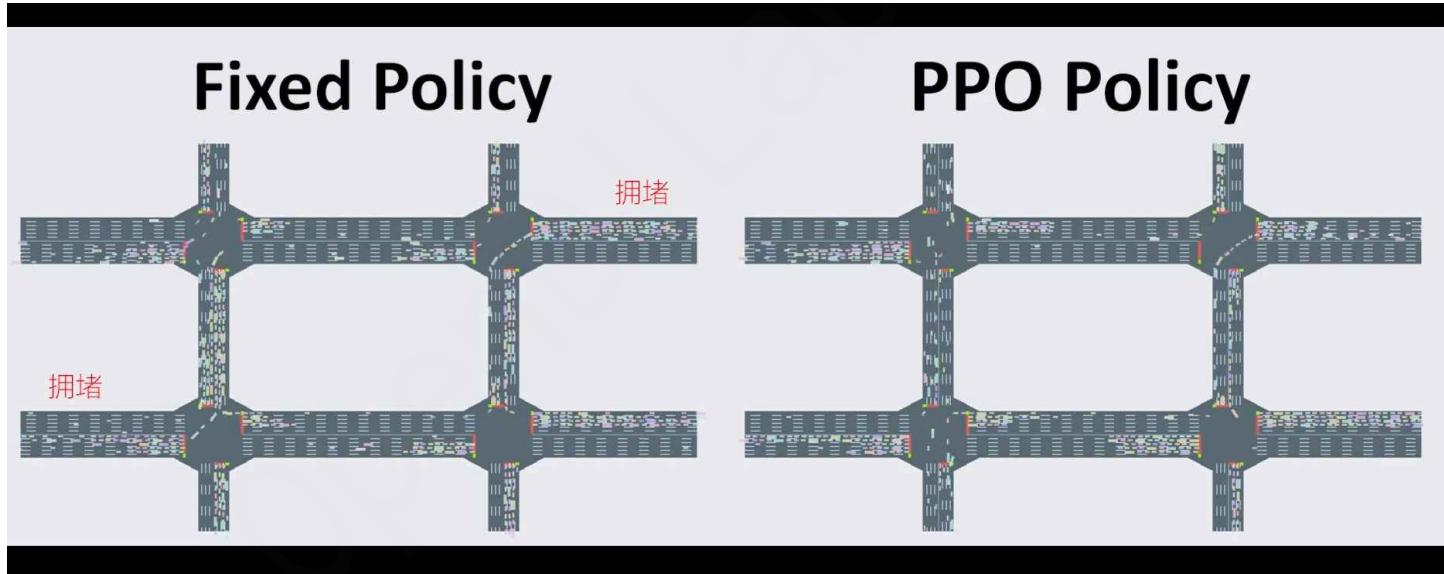




(图33：两种网络结构（橙线是 multid-discrete，蓝线是 discrete）使用行为克隆方法在专家数据集上训练的对比曲线。横轴是训练中智能体和环境交互的帧数 env step，纵轴分别是训练集上的损失函数值，验证集上的准确率（accuracy），测试环境中的平均 episode return。）

从图33可以看出，相比于 multi discrete，虽然 discrete 实验组的训练 loss 下降更快（见如上左图），但严重过拟合训练数据集；在验证集上的准确率低于 multi discrete 8 个百分点（见如上中图）；而在最终测试时，discrete 实验组获得的 episode return 也更低（见如上右图），这也就印证了上文中的观点。

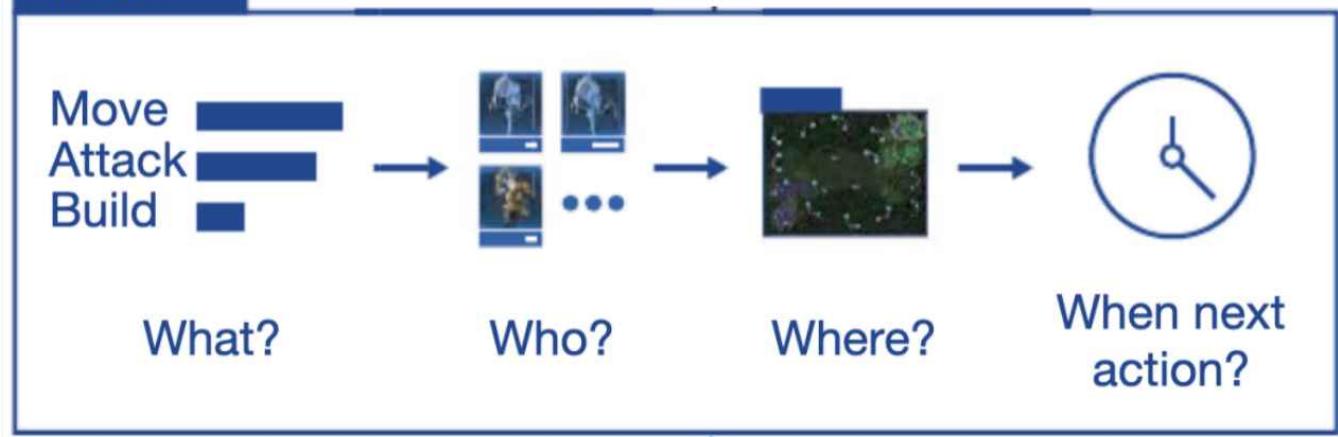
最终训练收敛时获得的策略和随机初始化策略的行为对比视频如下，完整样例可以参考[官方示例](#)
ISSUE：



(视频3：CityFlow 2 x 2 路网中，训练后的 PPO 智能体更加灵活地控制路网交通信号，从而减少拥堵增加吞吐。)

2.5 混合动作空间

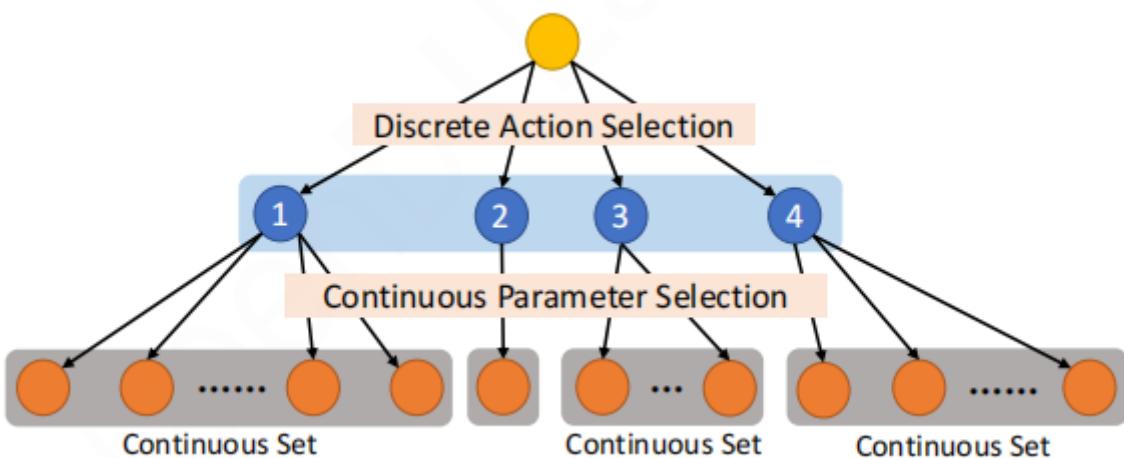
Action



(图34：《星际争霸2》中的复杂混合动作空间。每次决策需要确定做什么动作（What），即移动、攻击还是建造建筑等，哪个单位来执行这个动作（Who），攻击目标或是建造位置在哪里（Where），以及什么时候执行下一个动作（When next action），即用于控制合理的操作频率APM。)

引言：混合动作空间的定义

基于上述各种动作空间，可以组合构建出更加复杂的混合动作空间。而在很多大型决策问题里，混合动作空间才是最普适的情形，需要处理各种不同的动作子部分，并建模各个动作部分之间的复杂关系。



(图35：参数化动作空间的示意图，由一个离散动作类型和相应的若干个动作参数组成（零个，一个或多个），整体结构可以类似图中的树形图示来表达。)

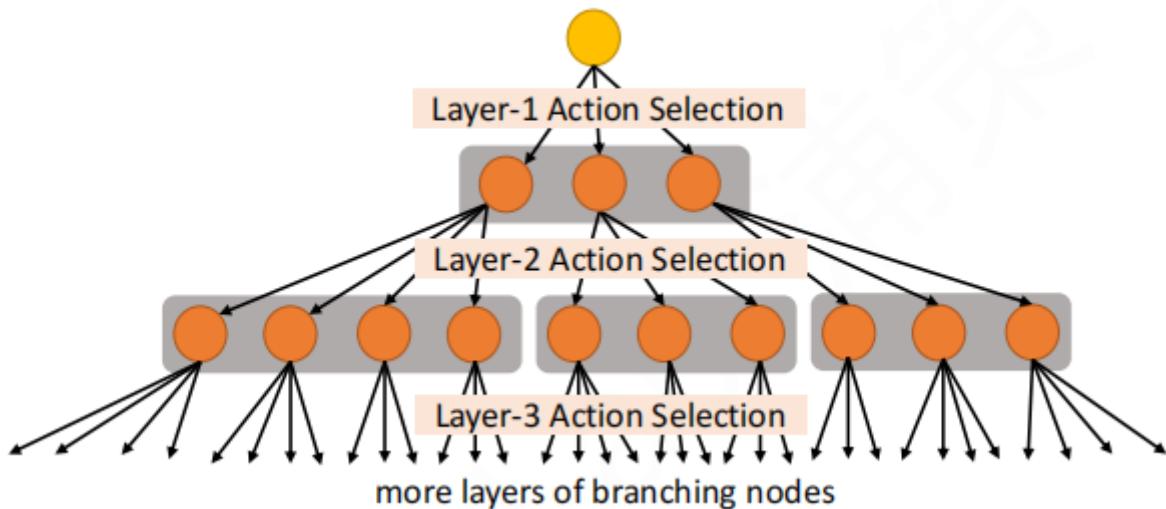
首先，这里来介绍最简单的一种混合动作空间：**参数化动作空间 (parameterized action space)**，具体来说：

- 智能体不仅需要选择一个**离散**的动作类型，还要根据动作类型去选择相应的**连续**参数。
- 动作类型和参数之间一般有着非常强的依赖关系。
- 不同的离散动作对应着不同的连续参数。

具体的示例，比如导航任务中可以选择加速和转向两种离散动作，但还需要确定加速度和转向角两个连续参数。或是王者荣耀中，操纵者需要确定下一个动作选取什么类型（移动、攻击、技能1、技能2还是技能3）这是离散的，如果你选择了技能3，比如后羿的大招，那么控制这个技能则需要一个连续变量（0-360度的方向）。

具体的数学定义如下 [PDQN]：离散动作从一个有限集合 $\mathcal{A}_d = \{a_1, a_2, \dots, a_k\}$ 中选择，每个 $a \in \mathcal{A}_d$ 都有一组连续实数参数 $\mathcal{X}_a \subseteq \mathbb{R}^{m_a}$ ，所以在混合动作空间中，一个动作需要用一个元组 (a, x) 来表示， $a \in \mathcal{A}_d$ 是选择的离散动作， $x \in \mathcal{X}_a$ 是执行动作 a 的参数。整个动作空间 \mathcal{A} 是每个离散动作与该动作的所有可能参数的并集：

$$\mathcal{A} = \bigcup_{a \in \mathcal{A}_d} \{(a, x) | x \in \mathcal{X}_a\}$$



（图36：层次结构化动作空间的示意图，会由多层离散动作类型选择和若干个动作参数组成。）

一个离散动作辅以一个连续参数只是最简单的情况，实际的决策问题中，还有很多更加复杂多层嵌套组合的动作空间，例如在《星际争霸2》中，智能体的一个动作包括动作类型，所选单位，目标单位，目标位置等多个子动作，对于这种情况，可以形式化定义为**层次结构化动作空间 (Hierarchically Structured Action Space)**，并用更大更深更广的树形结构来表示它。如图36所示，一个层次结构化动作空间可以看成一棵树：

- 自根节点向子树节点进行动作选择，中间的节点可以是各种离散动作空间，而每个叶节点参数则均是一个独立的离散动作空间或连续动作空间，那么上文中的参数化动作空间则可以看做是层数为2的层次结构化动作空间。
- 各个子树之间的动作性质相差很大，可能存在非常多多样化的情形。比如某颗子树表达作战攻击相关的操作，另一颗子树表达科技研究，或是视野侦查相关的决策行为等等。

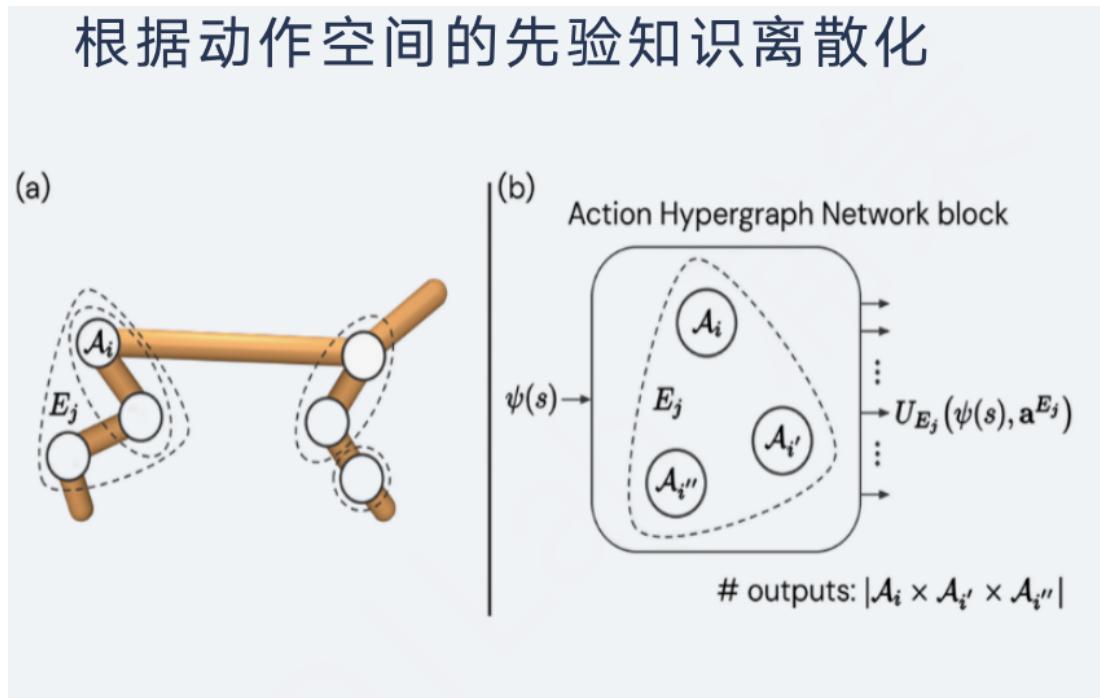
理论：PPO 如何建模复杂的混合动作空间

动作空间预处理

对于混合动作空间，一种很直接的思路就是对动作空间进行某种预处理，即本节课一开始讲到的 action shaping 操作，将复杂的混合动作空间转化为更加常用且熟悉的离散或连续动作空间，具体可

以分为两类做法：

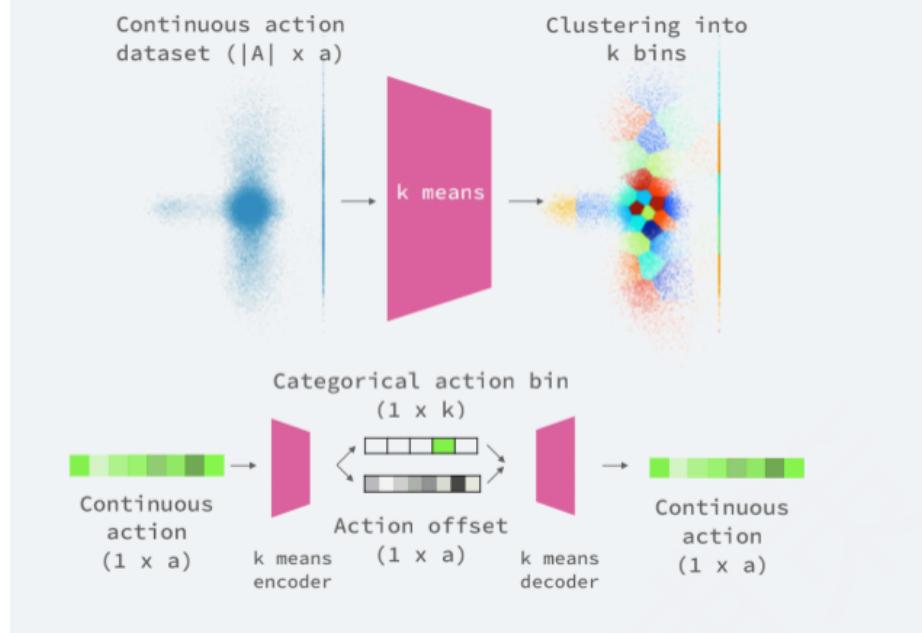
- **离散近似**：将混合动作空间的连续部分进行离散化，整个动作空间转化为若干个的离散集合
 - 优点：方法简单，将问题转化为离散动作空间或多维离散动作空间。
 - 缺点：失去对细粒度控制的能力，且合理的离散化需要对问题有足够深刻的理解。
 - 离散化方法一：动作空间先验离散化。如图37所示，如果对这样的机器人控制问题有一定的先验知识，知道它的某些关节之间存在强依赖关系，那么就可以根据这种强依赖关系来设计一些变化，更好地离散化原来的动作空间，得到一个更加简洁的动作空间表示，然后结合经典的PPO算法即可。



(图37：动作空间先验离散化示意图，来源于HGQN [20]。)

- 离散化方法二：数据挖掘离散化。如图38所示，倘若没有相应的先验知识，但是有相应的专家数据，即在该决策环境上拥有较好性能的策略的行为数据，那就可以结合一些经典的机器学习方法（比如聚类方法），自动化地挖掘离散化的结果。

根据数据特性自动挖掘离散化



(图38：数据挖掘离散化示意图，来源于 Behavior Transformer [21]。)

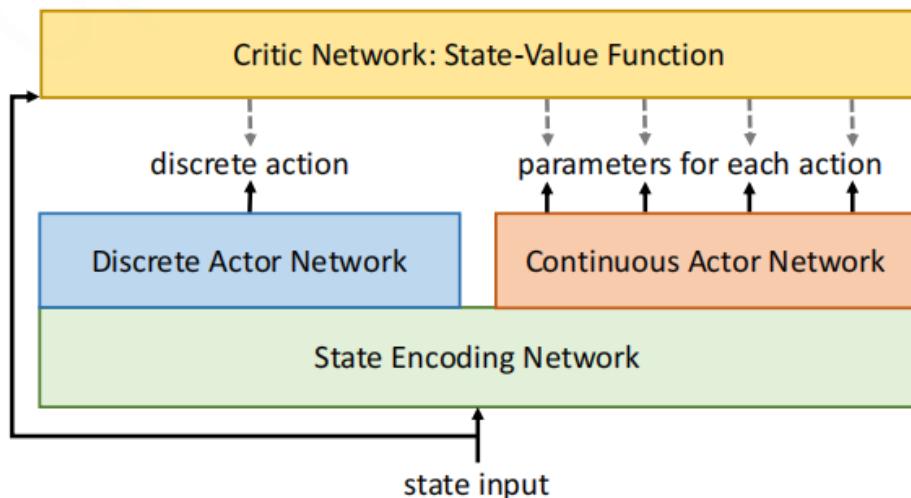
- **连续松弛：**将动作空间的离散部分转化为一个连续的空间，例如下列定义：

$$\tilde{\mathcal{A}} = \{(f_{1:K}, x_{1:K}) | f_k \in \mathcal{F}_k, x_k \in \mathcal{X}_k, \forall k \in [K]\}$$

- 但一般情况下这种操作会显著增加动作空间的复杂性，优化效果并不好

建模方法：Hybrid Proximal Policy Optimization (H-PPO)

如果想去直接使用 PPO 去处理混合动作空间，那该怎样建模呢？Hybrid PPO (H-PPO) [22] 是 PPO 用于解决混合动作空间问题的衍生算法，一言以蔽之，就是沿用 PPO 的 Actor-Critic 架构，但各个不同的动作部分使用独立的 Actor Head 网络结构，各自独立优化，离散部分使用离散 PPO，连续部分使用连续 PPO，不过，总体各部分都使用同一个 Critic 提供的 Advantage 信号指导 PPO 本身的优化。完整的网络结构示意图如下：



(图39：H-PPO 算法整体架构图。首先是一个共享的状态编码网络，在提取到相应的特征向量后，离散和连续部分使用各自独立的网络模块（即Actor）来输出相应的动作。整体用一个统一的价值网络（即Critic）来指导优化。)

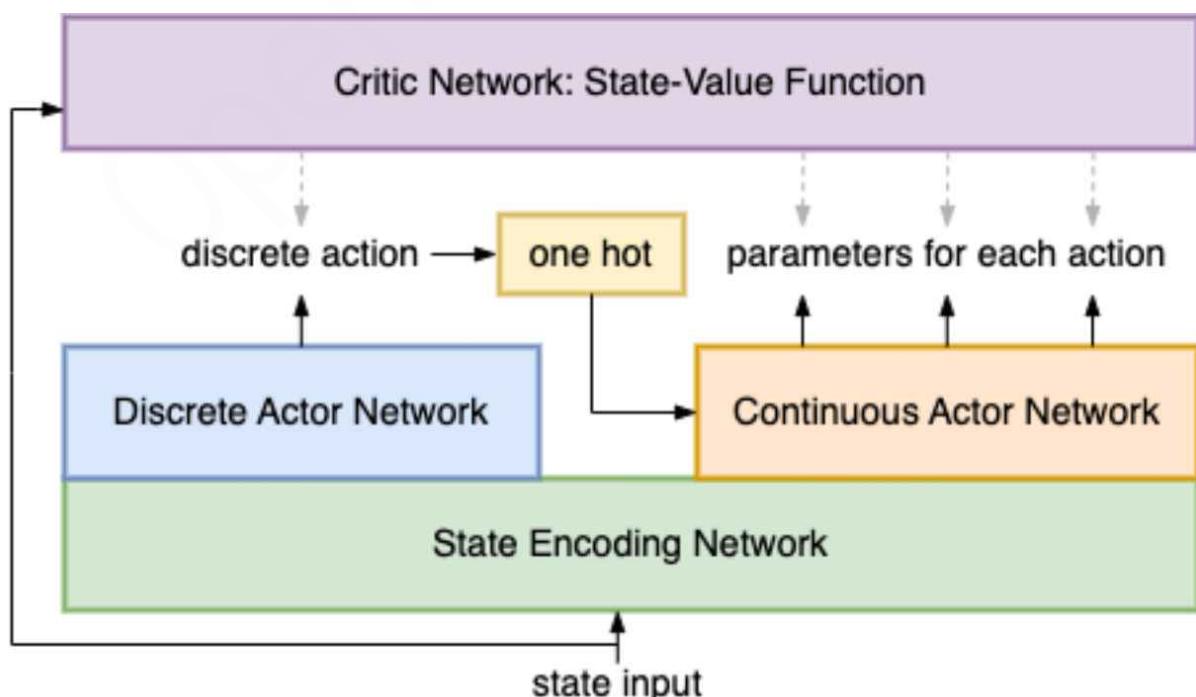
更详细的解释说明如下，H-PPO 是对上面两部分策略 π_{θ_d} （离散部分）和 π_{θ_c} （连续部分）进行优化

- 离散部分 (Discrete Actor Network)
 - 输入状态信息
 - 输出 p 个离散动作的 logit
 - 构建玻尔兹曼分布进行采样选择动作
- 连续部分 (Continuous Actor Network)
 - 输入状态信息
 - 输出 q 个连续动作的高斯分布参数 μ, σ
 - 构建高斯分布进行采样选择动作

总的来说，H-PPO 是一种实现简洁且扩展性很高的算法，更复杂的动作空间只需不断新增相应的 Actor Network 即可，但是这里也存在一个大问题，H-PPO 并没有去显式建模动作各部分之间的依赖，这也就产生了一些变种方法。

实践中的 H-PPO 改进方法一：自回归 (autoregressive)

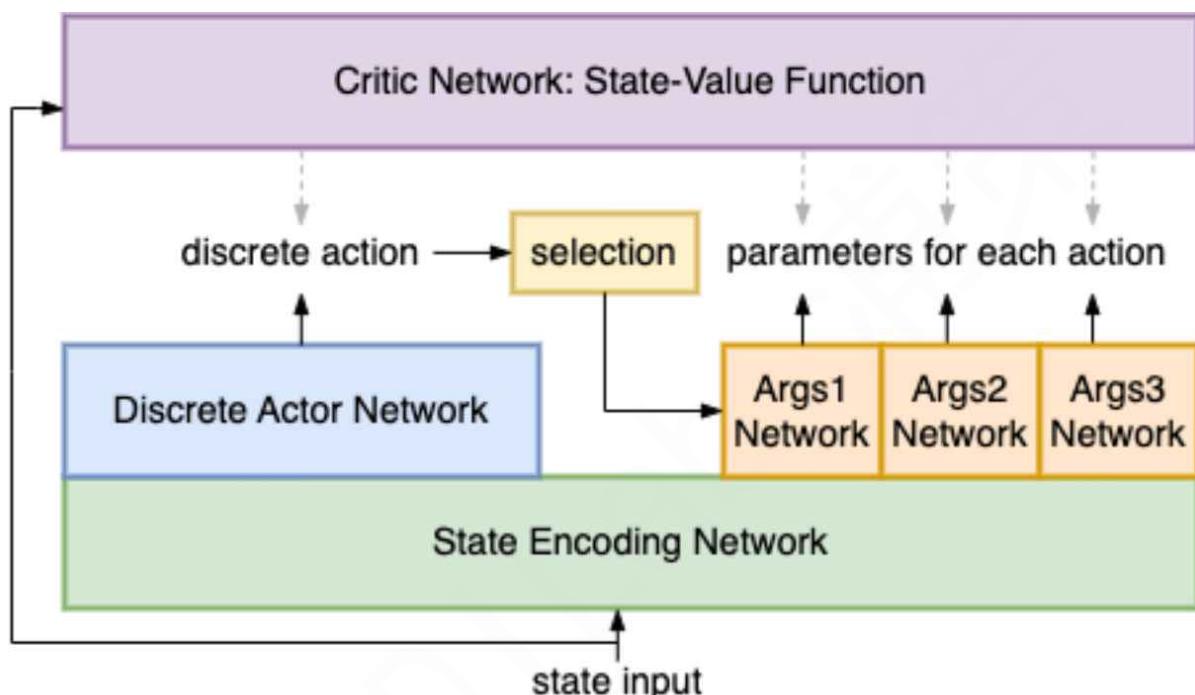
对于动作各部分之间的强依赖关系，如果已经存在一些先验知识，比如动作部分A可能很受动作部分B的影响，那么可以采用自回归 (autoregressive) 的方式来建模这些依赖关系。例如：离散的动作类型决定相应的连续参数（转向和转向角），那就可以将选出的离散动作进行 one-hot 编码，再和提取到的状态特征拼到一起，去预测连续参数。



(图40：H-PPO 自回归变体算法整体架构图。根据动作之间的依赖的关系，如果 continuous 部分依赖于 discrete 部分，那么需要将离散部分的输出动作进行 one-hot 编码，并把离散动作编码后的结果和 state 编码后的结果拼在一起，作为 continuous 网络的输入，从而让后面部分的决策输出可以根据前面部分的结果来自适应地进行改变。)

实践中的 H-PPO 改进方法二：分离 head

另外，还有一种对应关系更直接的建模方法，即分离 head 法。具体来说，就是对于每个离散动作的连续参数，单独设置一个连续策略网络，根据选出的离散动作，选择其中对应的一个连续网络来输出相应的连续参数，从而保证各部分之间的一一对应，多个连续网络之间各做各的事情，不会存在任何干扰和混用的情形。不过这种方法仅限于离散动作数量较少的情形，否则多个网络的计算开销过大。



(图41：H-PPO 分离 head 变体算法整体架构图。根据离散部分输出的动作类型，选择为其专门对应设置的连续网络来输出相应的连续参数，隐含了动作部分之间的依赖关系。)

此外，在最近两年的学术研究中，还有一些更新颖的方法来建模混合动作，去抽取统一的动作表征来帮助决策，详情可以参考附录2.6.3部分。

代码：使用 TreeTensor 来处理结构化数据

由于混合动作空间数据结构上的复杂性，实际编程中常常遇到各种困难，这种结构化的数据（nested data）跟深度学习中常用的标准 Tensor 是很不兼容的，因此，本节课的代码部分也介绍一种专为结构化数据设计的数据容器——TreeTensor [23]。借助这种新的数据容器，可以像使用 Tensor 一样便捷地操作各种结构体。完整的代码可以参考课程第二讲的相关代码样例和算法注解文档。

TensorTree on Nested Data

```
import tensortree.torch as torch

# create a nested data tensor
t = torch.randn({
    'a': (6, 2, 3),
    'b': {'x': (6, 3), 'y': (6, 1, 4)},
})

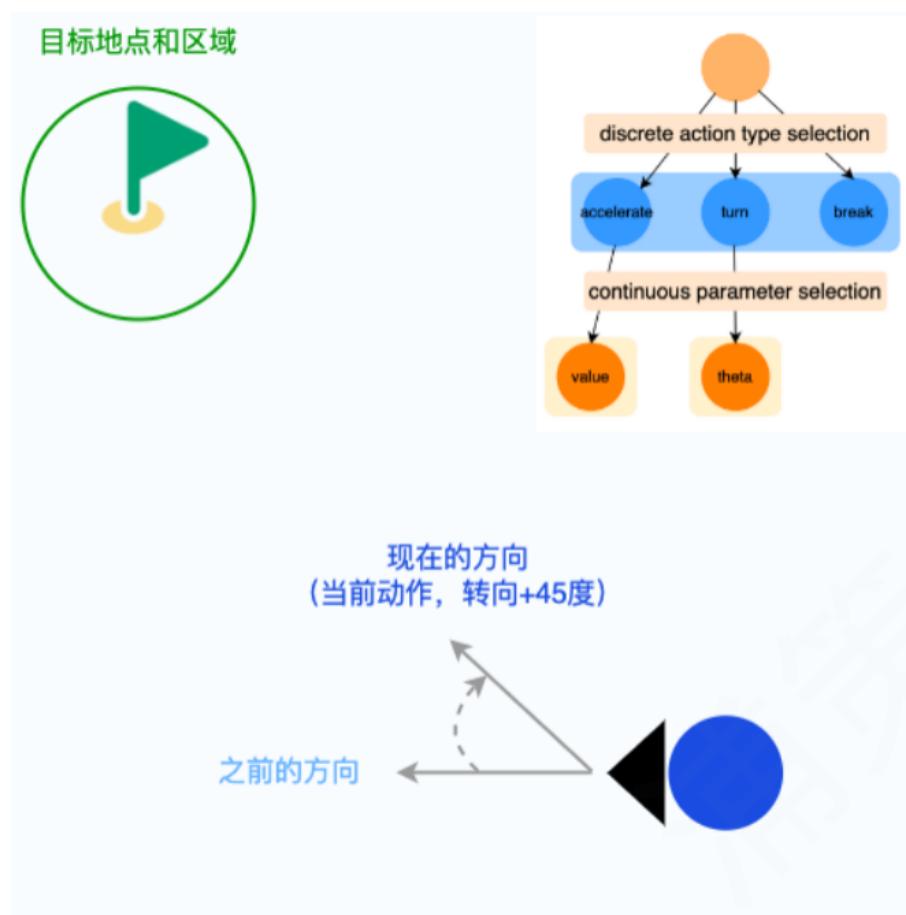
# structural operations
print(torch.stack([t, t]))
print(torch.split(t, (1, 2, 3)))
# math calculations
print(t ** 2)
print(torch.sin(t).cos())
# access like attribute
print(t.b.y)
```

(图42：TreeTensor [23] 代码演示示意图。)

实践：使用 PPO 来解决导航控制问题

问题背景

对于混合动作空间，本节课选择导航控制问题 gym-hybrid [24] 环境作为示例。在这个环境中，智能体需要在边长为 2 的正方形框内，从随机出发点启动，通过加速（Accelerate）、转向（Turn）或刹车（Break）等一系列控制操作，最终希望停留在红色目标区域（一个半径为 0.1 的圆），如下图所示：



(图43：gym-hybrid 环境示意讲解图。)

环境 MDP 设定

状态空间：该环境的状态空间是一个有 10 个元素的数组，描述了当前智能体的状态，包含智能体当前的坐标，速度，朝向角度的正余弦值，目标坐标，距离目标的距离，是否到达目标，当前相对步数。

动作空间：该环境属于参数化动作混合空间，有3个离散动作：`Accelerate`，`Turn`，`Break`，其中动作 `Accelerate`，`Turn` 需要给出对应的 1 维连续参数。

- `Accelerate (Acceleration value)` : 表示让agent以 `acceleration value` 的大小加速。`Acceleration value` 的取值范围是 `[0, 1]` 。数值类型为 `float32`。
- `Turn (Rotation value)` : 表示让智能体朝 `rotation value` 的方向转身。`Rotation value` 的取值范围是 `[-1, 1]` 。数值类型为 `float32`。
- `Break ()` : 表示停止。

奖励空间：每一步的奖励设置为，智能体上一个 step 执行动作后到目标的距离，减去当前 step 执行动作后到目标的距离，即 `dist_t-1 - dist_t`。另外，算法内置了一个惩罚项 `penalty` 来激励智能体更快的达到目标。当 episode 结束时，如果智能体在目标区域停下来，就会获得额外的奖励，值为 1；如果智能体出界或是超过 episode 最大步数，则不会获得额外奖励。奖励的伪代码实现如下：

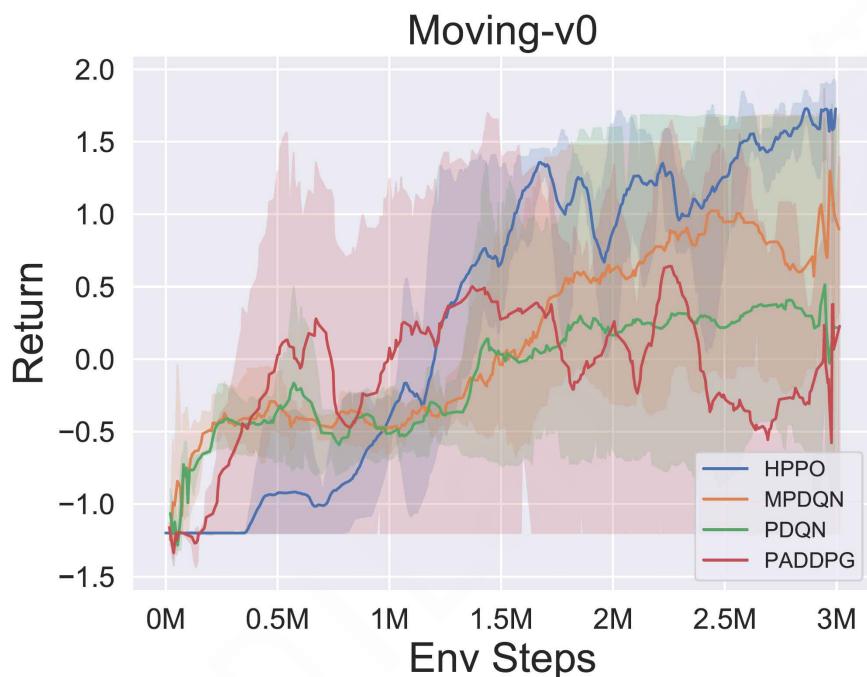
```
1 reward = last_distance - distance - penalty + (1 if goal else 0)
```

终止条件：遇到以下任何一种情况，则环境会认为当前 episode 终止：

- 智能体成功进入目标区域
- 智能体出界，触及边缘
- 达到 episode 的最大上限步数（默认设置为200）

实验结果展示

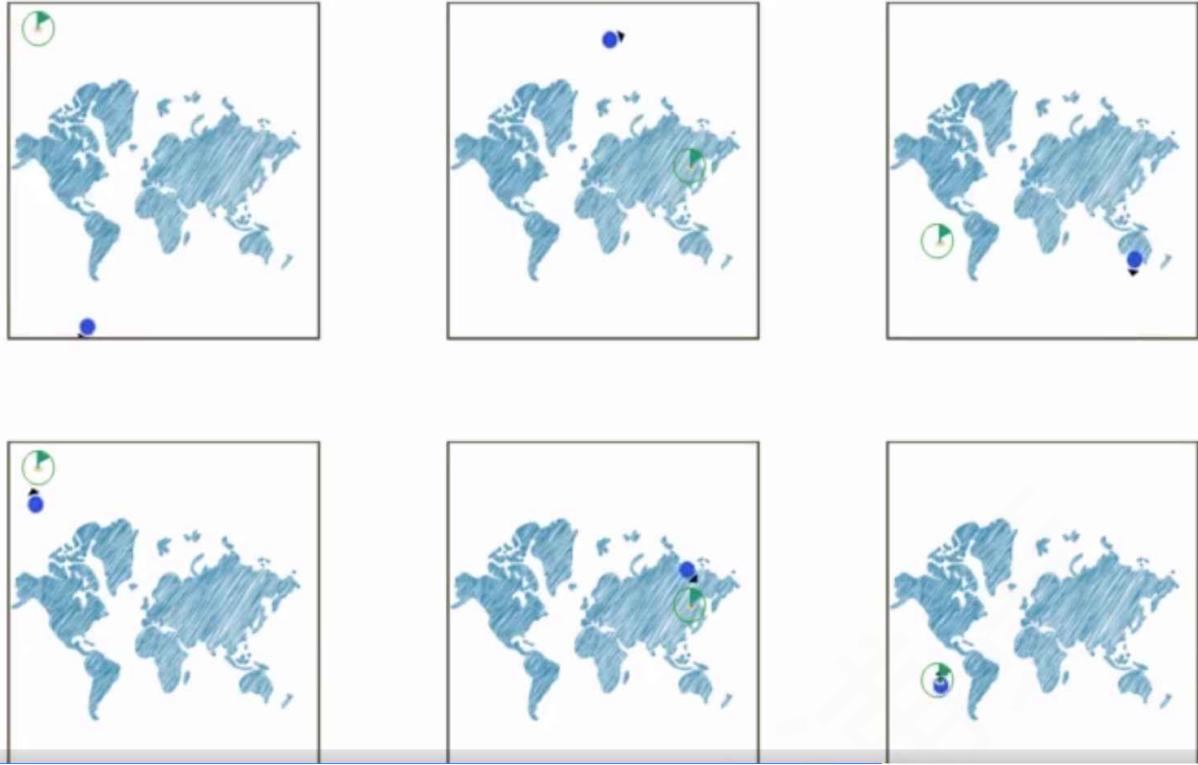
本节课在 gym-hybrid 的 moving-v0 子环境上对比了 H-PPO 和其他几种混合动作空间相关的常用算法（详情可以参考[中文讲解博客](#)），H-PPO 表现出了更高的收敛性能和更好的稳定性，具体如图44所示：



（图44：HPPO 和其他混合动作空间算法在 Moving-v0 上的训练曲线图。实线表示各个算法在5个种子上的测试局对应 return 的平均值，阴影部分表示5个种子上的标准差，在每个种子的每个测试点上一共评估8局。横坐标为训练时与环境交互的步数。episode return \geq 1.5 视为一次成功的测试。）

最终训练收敛时获得的策略和随机初始化策略的行为对比视频如下，完整样例可以参考[官方示例](#)

ISSUE:



(视频4：不同随机设置下的 gym-hybrid 环境，训练后的 PPO 智能体可以准确地找到目标，快速导航抵达终点。）

2.6 附录（可选阅读）

2.6.1 重参数化与强化学习

2.6.2 为什么 PPO 需要 重要性采样，而 DDPG 这个 off-policy 算法不需要

2.6.3 混合动作空间表征学习方法介绍（HyAR）

参考文献

- [1] Kanervisto, Anssi, Christian Scheller, and Ville Hautamäki. "Action space shaping in deep reinforcement learning." *2020 IEEE Conference on Games (CoG)*. IEEE, 2020.
- [2] <https://www.zhihu.com/question/60751553/answer/1986650670>
- [3] https://en.wikipedia.org/wiki/Greedy_algorithm
- [4] https://en.wikipedia.org/wiki/Boltzmann_distribution
- [5] Dillon J V, Langmore I, Tran D, et al. Tensorflow distributions[J]. arXiv preprint arXiv:1711.10604, 2017.
- [6] <https://pytorch.org/docs/stable/distributions.html>
- [7] <https://ericmjl.github.io/blog/2019/5/29/reasoning-about-shapes-and-probability-distributions/>

- [8] <https://twitter.com/thejackbeyer/status/1367364251233497095>
- [9] <https://github.com/jiupinjia/rocket-recycling>
- [10] Dadashi, Robert, et al. "Continuous Control with Action Quantization from Demonstrations." *arXiv preprint arXiv:2110.10149* (2021).
- [11] Lillicrap T P, Hunt J J, Pritzel A, et al. Continuous control with deep reinforcement learning[J]. *arXiv preprint arXiv:1509.02971*, 2015.
- [12] Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning[J]. *arXiv preprint arXiv:1312.5602*, 2013.
- [13] Sutton R S, McAllester D, Singh S, et al. Policy gradient methods for reinforcement learning with function approximation[J]. *Advances in neural information processing systems*, 1999, 12.
- [14] Schulman J, Levine S, Abbeel P, et al. Trust region policy optimization[C]//International conference on machine learning. PMLR, 2015: 1889-1897.
- [15] <https://github.com/utiasDSL/gym-pybullet-drones>
- [16] Panerati J, Zheng H, Zhou S Q, et al. Learning to fly—a gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control[C]//2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2021: 7512-7519.
- [17] <https://www.bitcraze.io/>
- [18] <https://github.com/cityflow-project/CityFlow>
- [19] Tang Z, Naphade M, Liu M Y, et al. Cityflow: A city-scale benchmark for multi-target multi-camera vehicle tracking and re-identification[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019: 8797-8806.
- [20] Tavakoli, Arash, Mehdi Fatemi, and Petar Kormushev. "Learning to represent action values as a hypergraph on the action vertices." *arXiv preprint arXiv:2010.14680* (2020).
- [21] Shafiullah, Nur Muhammad Mahi, et al. "Behavior Transformers: Cloning \$ k \$ modes with one stone." *arXiv preprint arXiv:2206.11251* (2022).
- [22] Fan Z, Su R, Zhang W, et al. Hybrid actor-critic reinforcement learning in parameterized action space[J]. *arXiv preprint arXiv:1903.01344*, 2019.
- [23] <https://github.com/opendilab/DI-treetensor>
- [24] <https://github.com/thomashirtz/gym-hybrid>