

PPO × Family 第三讲文字稿

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算法技术——深度强化学习，并深入浅出地介绍了最强大通用的算法——PPO。第二讲（解构复杂动作空间）从决策输出设计的角度展开，介绍了 PPO 算法在四种动作空间上的各类技巧。而第三节课——表征多模态观察空间，将会着眼于表征建模问题，从深度学习的角度进行拓展，介绍观察空间的三部曲及衍生的“算法-代码-实践”知识。



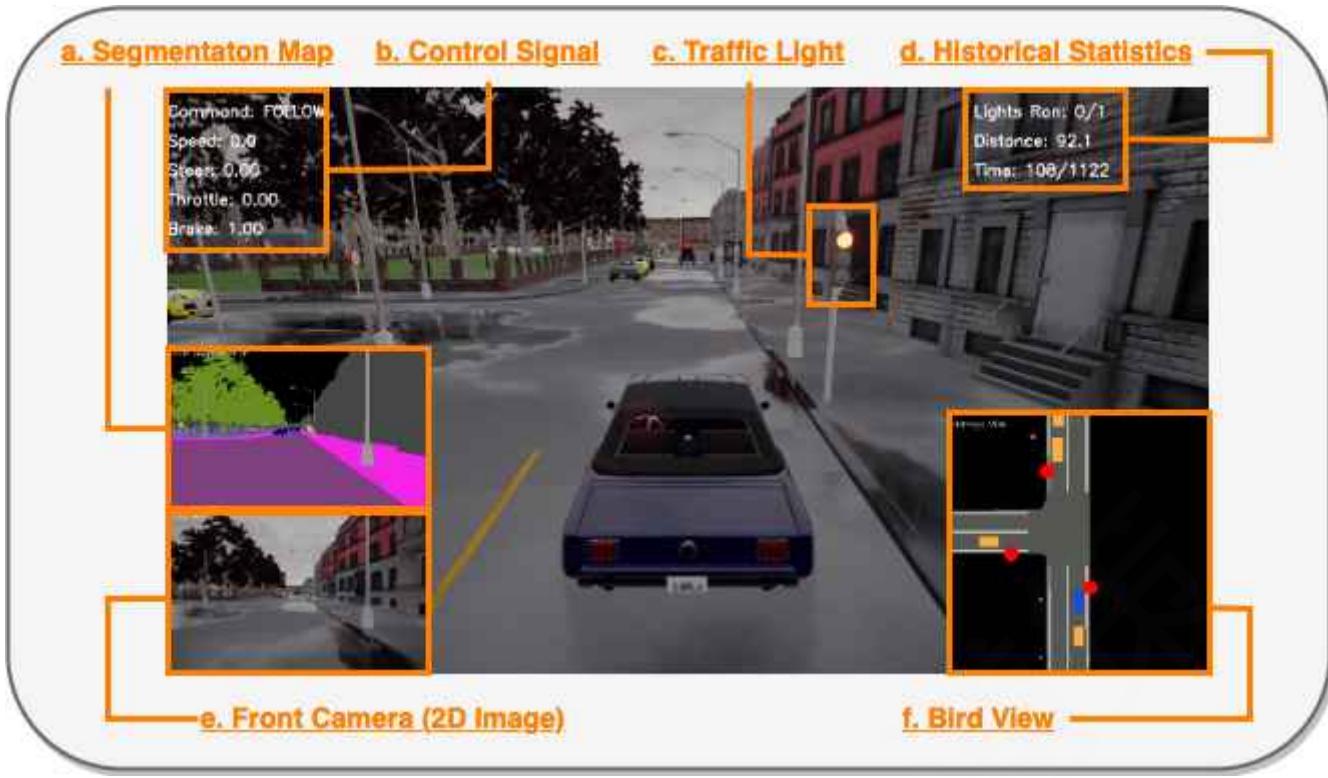
(图1：观察空间三部曲。)

第一步，本节课会讲解读者最熟悉的观察空间——向量观察空间。这种观察空间通常会应用于机器人控制等类型的问题。第二步，读者将会接触到一个更高维的观察空间，即在很多红白机游戏（比如超级马里奥），或是经典的学术环境 Atari/Procgen 中应用的图片观察空间，具体来说，本节课会结合一些计算机视觉相关的知识进行拓展。第三步则会推广到更复杂更一般的场景，这部分将会讲解所谓的结构化观察空间，时间应用方面则会涉及到像《星际争霸2》[\[1\]](#)，《DOTA2》[\[2\]](#)这样的复杂决策问题，以及各式各样的实际决策应用场景（例如自动驾驶），如图2所示。在介绍完三种经典观察空间之后，本节课将介绍一些通用的训练方法，帮助读者扩充知识面和技术栈。

3.1 观察空间概述

观察空间作为 MDP (Markov Decision Process) 五元组中的重要元素，究竟有什么特点呢？首先从一个例子开始展开。

3.1.1 常见的观察空间类型



(图2：自动驾驶的观察空间示意图。)

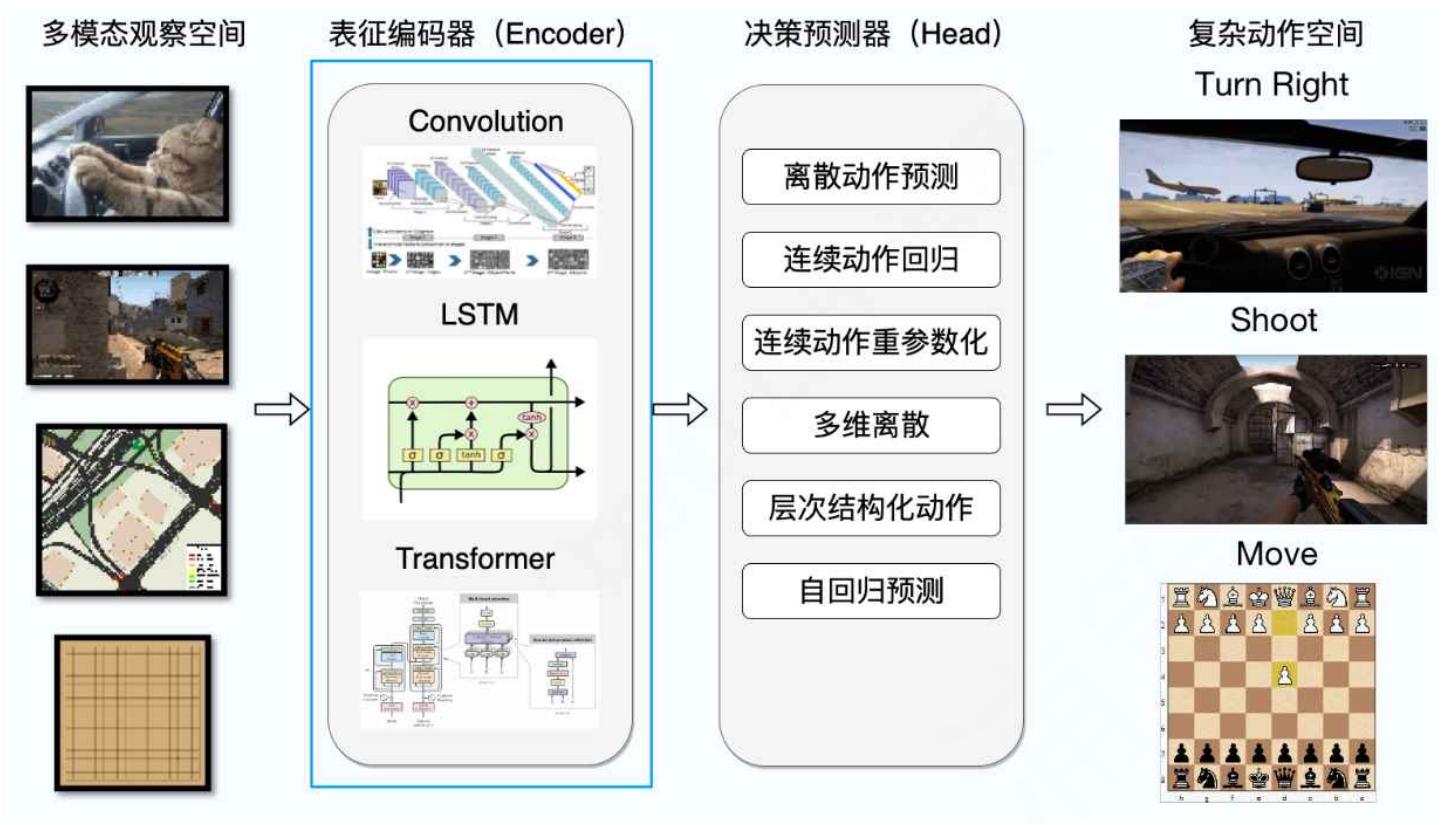
上图展示的是在自动驾驶问题中，决策智能体通常可以获得的观察信息类型。在最上面一排，可以获得例如控制信号、交通灯信号以及历史车辆行驶轨迹等等这样的一些标量信息，并将它们转化为类似于向量观察空间的形式。其他部分，则可以获得诸如这种语义分割的结果图，以及前置摄像头拍出来的RGB图像等等信息，其中可能会包含天气信息或者是路面的积水情况等，此外，也少不了在自动驾驶中最常用的鸟瞰图信息（Bird View），用于更全面地掌控车辆行驶情况。

如果想要做好自动驾驶中的决策问题，那么算法设计部分就需要去充分运用这些观察空间信息。那么总结来说，其中的关键点可以分为三个部分：

- 关键的特征预处理可以事半功倍：**很多自然的决策问题，其实都是不太适合直接应用神经网络，或是直接运用强化学习方法去进行训练优化的。实践中，经常需要将原始的决策问题的观察空间，转化为适合于神经网络和强化学习训练的特征向量，从而提升整体的数据利用效率和训练稳定性。
- 高维的观察信息更需要专门设计的神经网络架构，需要运用与数据相适配的表征提取器。**例如像图片这样的高维的观察信息，更需要一些专门设计的神经网络架构，比如对于图片观察空间常常借助计算机视觉领域的经典知识，通过借助这些特定领域的先验来帮助决策智能体更好地提取表征、获取信息。
- 复杂的决策问题往往包含多种模态的观察信息，需要综合运用多种模态的建模方法和融合技巧。**例如像自动驾驶这样复杂的决策问题，往往包含多种模态的观察信息，所以设计决策智能体时，要能够做到将多种不同的表征建模方法融合在一起，更好地为最终的决策行为服务。

3.1.2 编码器

了解完观察空间的基本概念和特点之后，接下来的部分将介绍观察信息在智能体训练 pipeline 中的地位。具体来说，决策智能体的整体训练 pipeline 其实如下方这幅图所示，输入端是多种模态的观察信息，而输出端就是在第二节课中讲到的离散、连续等等各种复杂的决策动作。



(图3：决策算法整体 pipeline 概述。)

如图3所示，决策智能体算法设计中的深度学习模块可以分为两大部分：

1. 根据观察空间，设计表征提取编码器（Encoder）。
2. 基于决策类型（动作空间），设计决策预测器（Head）。

课程第二讲重点讲解了 Head 部分的设计思想和算法技巧。而第三讲将主要在编码器部分做文章。具体包括：利用卷积神经网络去建模图片信息，结合 LSTM [3] 去处理相应的时序建模，以及运用 Transformer [4] 去建模多个单位之间的相关性等问题，希望能够通过结合神经网络的经典知识和决策智能算法的需求，从而打造出最强大的决策智能体。

3.2 向量观察空间

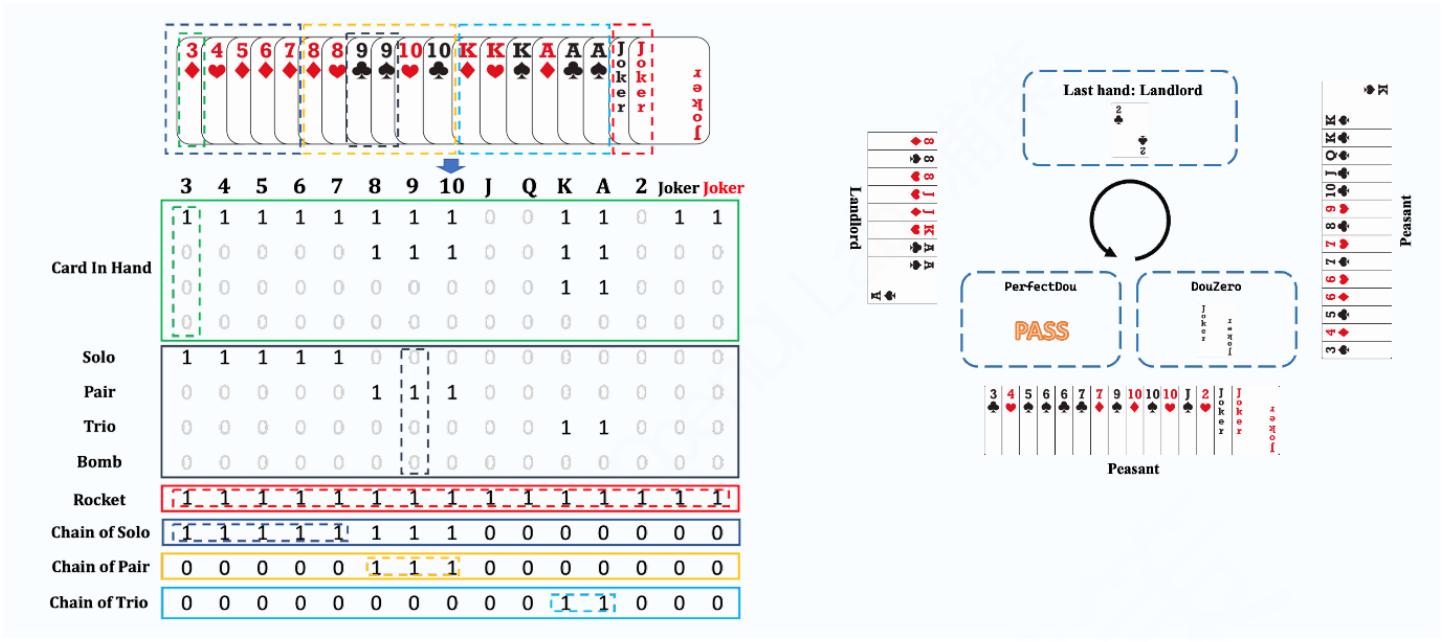
3.2.1 定义：什么是向量观察空间

向量观察空间，顾名思义，就是需要去把原本决策问题中的观察信息编码成一个适合于神经网络训练的特征向量。

3.2.2 引言：哪些场景常用到向量观察空间

在强化学习环境中，关于当前状态的描述信息，往往是以简单的数值或是字符串的形式表达的。使用合适的方法和技巧，将原始数据编码成相应的观测特征向量，对提高强化学习算法的训练效果有很大帮助。

这里以经典的斗地主游戏为例：假如试图设计一个斗地主 AI，该如何做状态编码呢？



(图4：左图：将牌面编码成一个 15×12 的矩阵，15 指的是玩家手中所能持的最大牌数，12 代表了每一张牌所能够对应的相应的牌型。右图：斗地主游戏示意图。具体参考 PerfectDou [5]。)

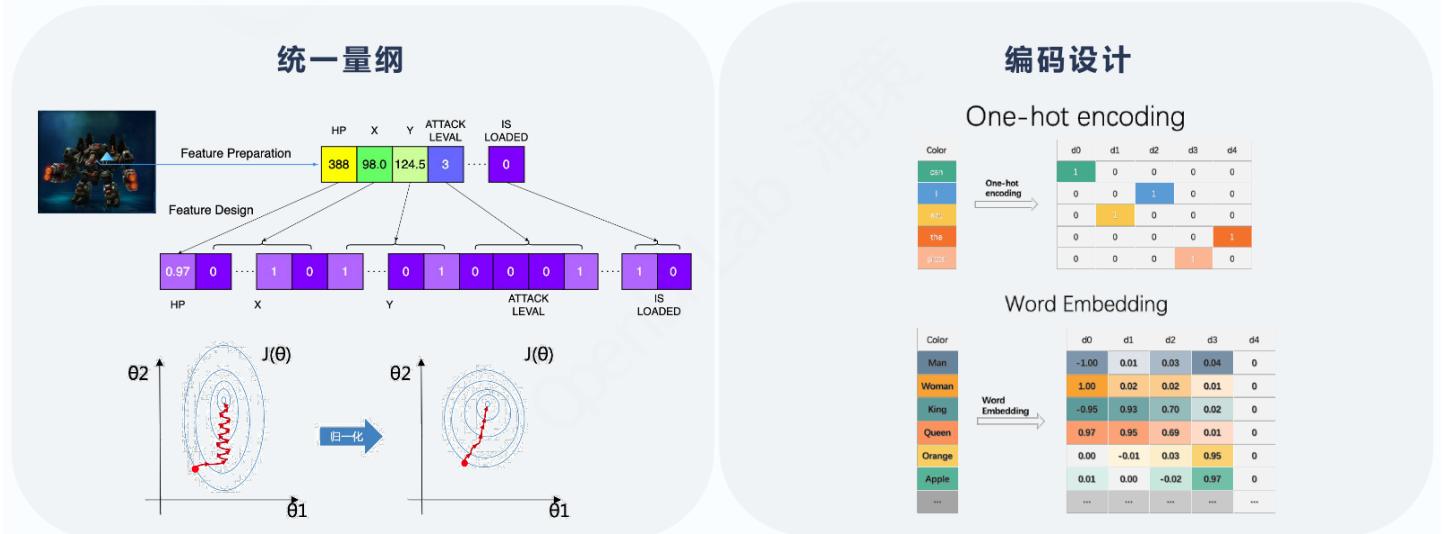
斗地主中牌面组合有很多种方式，朴素的编码方式很难应对，但是有一种很高效的方式：将牌面编码成一个 $15 * 12$ 的矩阵，15 指的是玩家手中所能持的最大牌数，12 代表了每一张牌所能够对应的相应的牌型，比如说这张牌是单走、组成对子还是三带一，根据规则和场上形势填充完这些牌面组合信息之后，再把这个 $15 * 12$ 的矩阵拉平（Flatten），最终构成一个长向量，就可以作为决策算法的输入。

3.2.3 理论：PPO + 向量观察

接下来，本节课将系统性地介绍向量观察空间的相关知识，主要分为**预处理操作**和**不变性建模**两部分。

预处理

预处理操作部分需要灵活应用一些传统机器学习的算法知识。第一点是**统一量纲**。以《星际争霸2》[\[1\]](#)中的重型作战单位——雷神为例，如果要做这款游戏中的AI，该怎么设计它的特征编码呢？一个很朴素的想法是：把雷神这个单位的所有跟做决策可能相关的信息都拼起来，组合成一个向量，比如说包括它的血量、位置、攻防等级，是否可以被运输机装载等等。



(图5：左图：《星际争霸2》中雷神的血量、位置、攻防等级、是否可以被运输机装载编码成一个长向量。右图：One-hot encoding 适用于类别信息，而 Word Embedding 适用于隐含一些语义信息的自然语言。)

这个朴素的方法粗看起来没什么问题，但是如图5中的颜色区分所示，实践中会发现这种向量的元素的数值范围差异非常大，不同的特征之间量纲完全不同。假如说直接将这种特征作为输入交给神经网络，训练优化时会出现很多不稳定的现象，因此，通常会使用一系列编码设计方法来把它们转化为更好的形式。比如对于血量，把它归一化到 $[0, 1]$ 之间，对于 x 、 y 这样的位置信息，使用 binary encoding 变成二进制的编码。那么对于攻防等级和是否被运输机装载等信息，他们更适合使用 one-hot encoding 来变成类别信息所对应的编码。

对于上述两种不同的编码方式，它们的作用机理可以借助目标函数优化曲面的可视化图来进行对比理解，原始的编码信息更偏向于一个椭圆，优化过程可能有很多的不稳定，即优化轨迹出现所谓的“盘旋”状态。而编码完善后的轨迹会更加稳定，且能够比较一致的朝着优化方向去前进，从而在效率和最终性能上都能获得更好的结果。

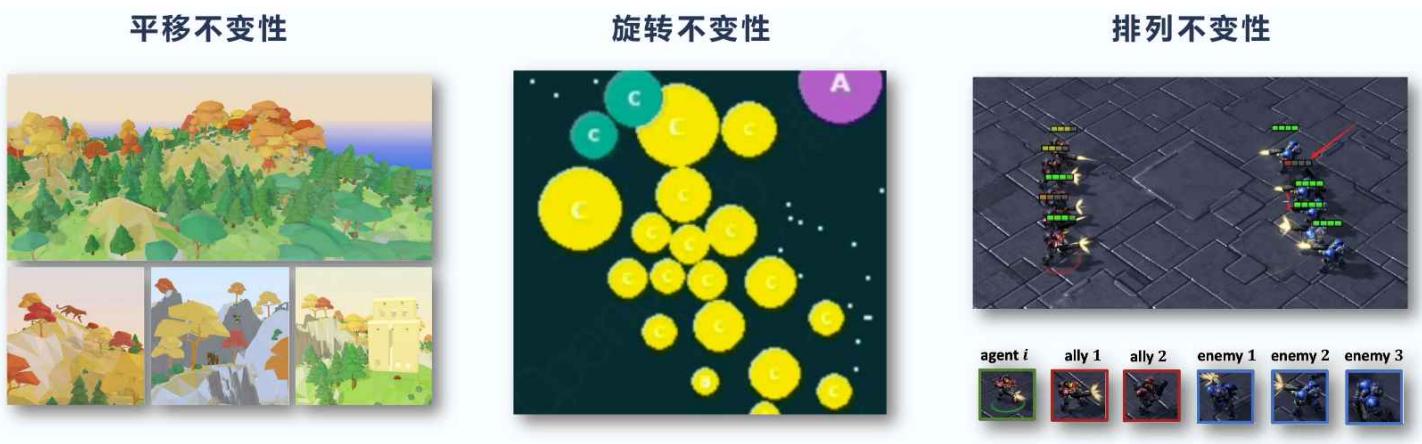
更进一步，具体的编码设计该如何操作？这里要根据不同的数据类型去选择相应的编码设计方法，每种编码设计方法会包含对应的特殊设计思路，适用于某种类型的数据，这里以两种方式举例：

- 第一种方式是 One-hot encoding，一般常用于类别信息数据（Categorical Data）。为什么对于类别信息，不用简单的数字编码 1、2、3、4，而是要把它转换成 one-hot 形式？因为像类别信息，比如说猫、狗、狮子、老虎，它们之间其实都是平等的，因为这些信息都是单纯的类别，所以它们两两之间的距离应该是相等的。但是如果给它们赋予数字的语义，如给猫1，狗2，狮子3，那么 2-1 和 3-1 的距离是完全不一样的，这样会给它们强加上某种不应该存在的数值关系，而 one-hot 编码会保证它们两两之间距离相等，所以更适合编码这种特征信息。
- 第二种方式是 Word Embedding，这种方法则是在自然语言数据中更加常用。例如将人类的自然语言转化为一个长向量，向量的不同维度可能隐含一些相应的语义信息，比如图5中 d_0 这个维度，对于男性和女性这两个词给出 -1 和 1 两个截然相反的值，那么这就意味着这一维可能隐含着性别相关的语义，而 d_3 这个维度则可能隐含着是否可食用的语义，可以看到橘子和苹果这些食物在这一维特征上的取值比较高，其他的词则会比较低。

不变性

讲完了预处理部分，其实还有一个很重要的宏观原则，就是表征建模的不变性。当然，不变性适用于各种类型的观察空间，这里只是引出这个概念来进行讲解。首先，本节课先给出不变性和等变性的定义（各种观察空间都适用）：

- 预设 F 指代表征建模函数（即特征编码和神经网络）， T 表示相应的变换，比如平移旋转、改变顺序等等。
- 不变性指在变换前后，比如平移旋转前后，函数的输出都是一致的，即 $F(X) = F(T(X))$ 。
- 等变性则是说先做变换，再做函数 F 和先做函数 F 再做变换，最终结果是一样的，即 $T(F(X)) = F(T(X))$ 。



(图6：左图：平移不变性示例：相同的信息出现在图片的不同位置都能提取到。中图：旋转不变性示例：多智能体的博弈环境 GoBigger 示例：大球吃小球，每一个球是一个单位，不管旋转什么样的角度，最优决策是不变的。右图：排列不变性示例：避免多个智能体编码的顺序关系跟决策行为强依赖。具体细节参考 GoBigger [6]，ACE + SMAC [7]。)

不变性和等变性这两种性质在不同的场景里各有相应的用处，具体例子如图6所示：

1. 第一个例子是平移不变性，这在图片数据中更成为常见，具体是指相同的事物出现在不同的位置，编码器都能够很好地提取到它，并且把它转化为相同的输出结果。例如无论猫出现在图片左上角还是右下角，算法都能够成功识别出相应的类别。
2. 第二个例子是旋转不变性，这里用经典的多智能体博弈环境 GoBigger 来举例说明。这个环境的规则很简单，就是大球吃小球，每一个单位是一个中心对称的球。所以不管旋转什么样的角度，智能体的最优决策应该是不变的，不应该因为智能体的相对位置在旋转角上发生了变化而导致它的最优决策也发生变化。因此，在设计这个环境的观察空间编码时，如果能考虑到旋转不变性，那么智能体学习策略的过程会变得非常简单，否则智能体就要去处理各种角度上的策略，那这个时候决策问题就变得更为困难。
3. 第三个例子是排列不变性，这个话题主要在多智能体领域中会涉及到。因为多智能体领域需要去控制多个智能体，需要去编码多个智能体的信息，这个时候如果智能体的顺序信息或顺序关系，跟决策行为存在很紧密的依赖关系，那么只要顺序关系发生轻微的变化，决策行为也会发生天翻地覆的变化，这种情况也是实践中极力希望避免的。所以如果特征设计之中包含一定的排列不变性处理，那对于多智能体协作会有很大益处。

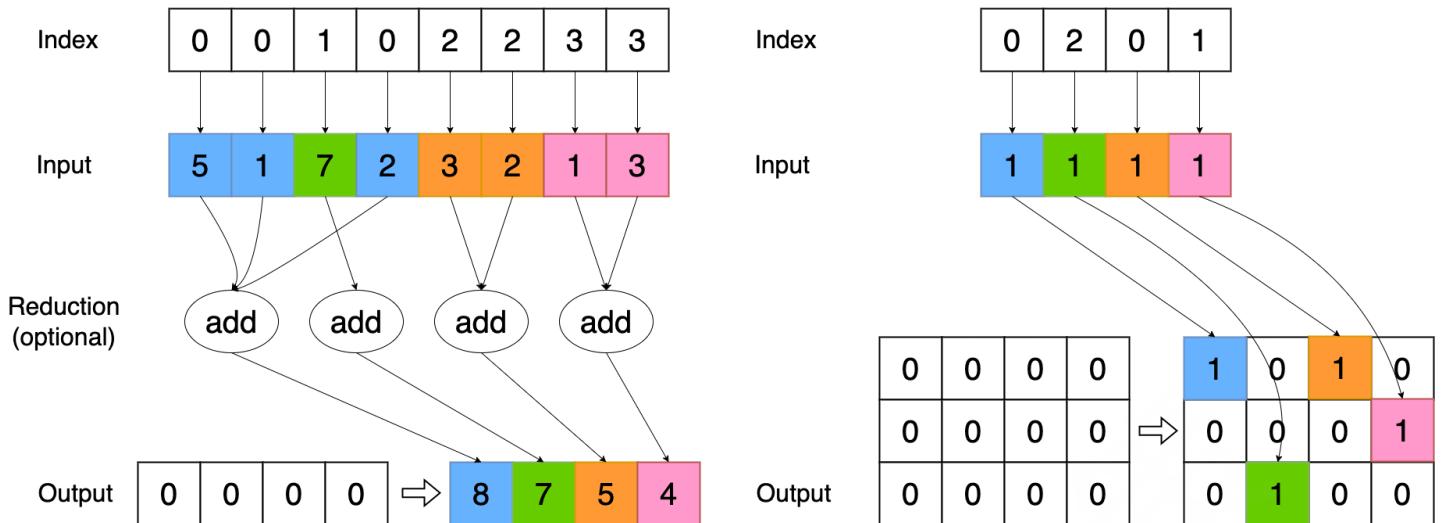
3.2.4 代码：特征工程中的张量操作技巧

在了解完向量观察空间相关的算法原理知识之后，本节课会继续讲解代码层面需要注意的地方。可能很多初学者下意识会觉得这些特征工程相关的操作只是简单地调用 API 而已，其实并不然。这部分首先需要能够**快速地上手掌握**这些 API，掌握函数功能和接口，了解其中的算法原理，然后在了解这些基本机制之后，能够**灵活地**将这些功能应用到实际问题中快速构建决策智能体。

具体来说，本节课通过一个常见的 API (`torch.scatter`) 和对应衍生的 one-hot 编码来进行介绍。顾名思义，这个操作就是将数据发射或者散播出去，即通过 index 的指引，将 input 所对应的值通过某种 reduction 操作，最后规约到一个相应的输出位置上去。

$$\text{Scatter: } \text{out}_i = \text{out}_i + \sum_j \text{in}_j$$

One-Hot Encoding By Scatter



(图7：左图：scatter 操作将 input 中根据 index 对应的值通过加法操作合并到一起，最后作为 index 位置上对应的输出。右图：将 scatter 操作应用在 one-hot 编码中，通过把类别信息表示为 index，然后通过把一个全 1 向量发射或者散播到 index 的位置这种方式来产生 one-hot 编码。)

可以看到如图7 左半部分所示，0 所对应的这些位置点，会从 input 中选取到这些蓝色的值，并将它们通过加法操作合并到一起，最后作为相应位置上对应的输出。而在实际应用中，scatter 可以快速构造一些网络结构设计里面的实现，一个经典的应用就是使用它去做 one-hot 编码，具体来说，就是将类别信息表示为 index，其中 0201 就代表它可能是第 0 类、第 2 类、第 0 类和第 1 类，然后通过这些类别 index 的指引把一个全 1 向量发射或者散播到相应的位置上去，最后产生 one-hot 编码。

希望读者可以通过这样的过程快速上手掌握相应的 API 和功能函数，并且能够了解到一些这些功能函数在实际问题中的相应实践应用。完整的示例可以参考课程的“算法-代码”注解文档，致力于帮助读者获取到一些在编程和代码实践上的相关经验。

3.2.5 实践：PPO+ 软体机器人

Evolution Gym (Evogym) [8] 是第一个用于共同优化软体机器人设计和控制的大规模基准。其实读者更可能熟悉的是刚体机器人，如 MuJoCo 或者 DeepMind Control 等经典控制任务。软体机器人会有一些不同的特点，每个机器人都由不同类型的体素（如软体、硬体、执行器）组成，从而形成了一个模块化和富有表现力的机器人设计空间。该环境跨越了广泛的任务范围，包括在各种类型的地形上进行运动和操纵。

观察空间

- 观察空间包括机器人的状态信息，一个包括每个四边形体素的顶点相对于机器人 $(2N)$ 质心的相对位置的矢量 $(2N + 3)$ ，以及质心的速度和方向 (3) 。数据类型是 `float32`。
- 为了处理复杂的任务，特别是那些具有不同地形类型的任务，提供了一个包括地形信息的额外观察向量。
- 此外，提供与目标相关的信息，以告知控制器当前任务的执行状态。

- 例如，在操纵任务中，机器人与一些物体 O 互动时，获得了方向和速度，以及 O 的质心相对于机器人的位置。

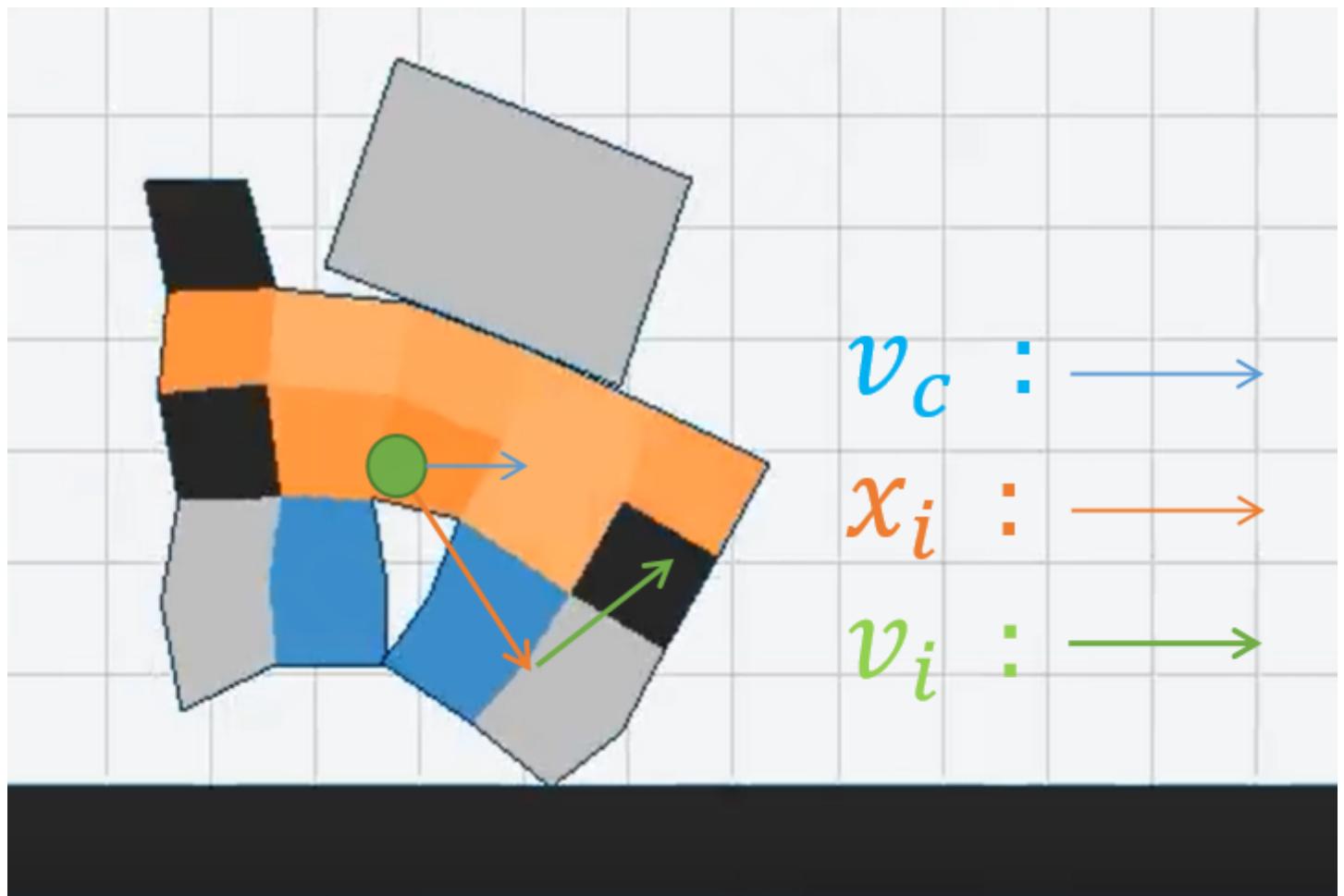
动作空间

- 动作空间是大小为 N 的连续动作空间。动作向量的每个分量都与机器人的一个执行器体素（水平或垂直）相关联，并指示该体素的变形目标，将体素的大小从 60% 压缩或拉伸到 160%。数据类型是 `float`。
- 具体来说，行动值 `u` 是在 `[0.6, 1.6]` 范围内，相当于该执行器逐渐膨胀/收缩到其静止长度的 `u` 倍。

奖励空间

- 每个任务都配备了一个奖励函数，衡量机器人当前的控制行动的性能。一般来说，它是一个 `float` 值。

具体细节可以参考：https://di-engine-docs.readthedocs.io/zh_CN/latest/13_envs/Evogym_zh.html

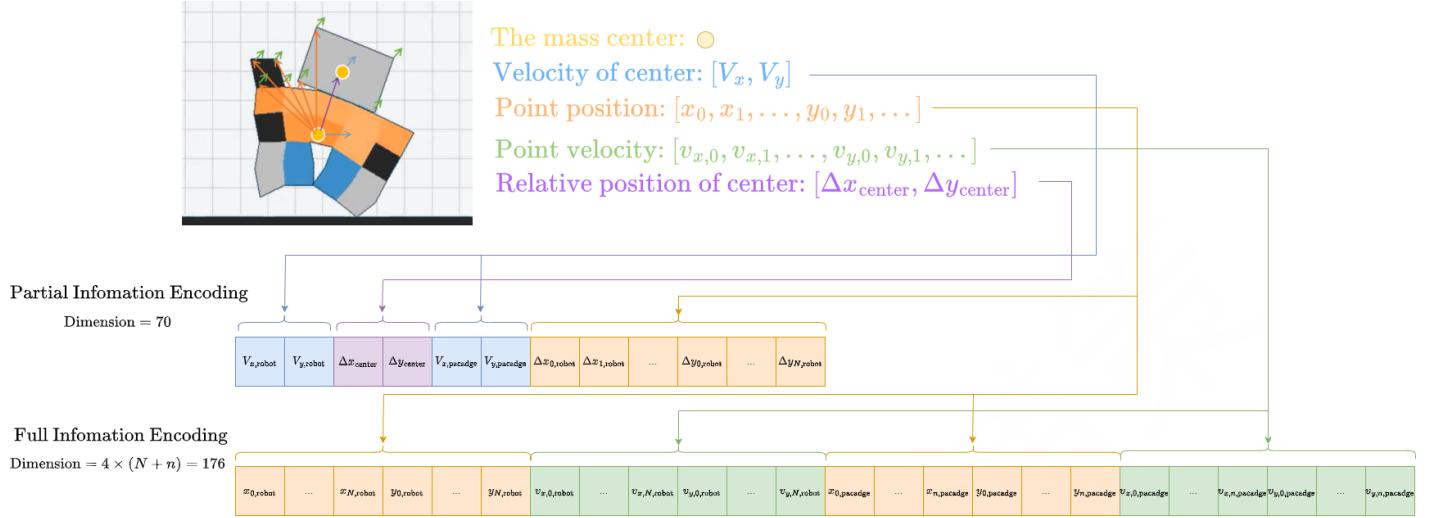


(图8：Evogym 软体机器人。)

在软体机器人环境中，机器人整体是由很多个小块（质点）构成的，每一个质点的速度和方向和它整体的运动速度和方向其实是不尽相同的，这就会带来很多在观察信息编码以及实际决策控制上的问题。那么一个最笨的办法就是去将所有质点的位置和速度都编码成为一个特征向量，那么这种方式虽然信息很全，但是当机器人的尺寸变得很大的时候，相应的编码信息会非常庞大，所以在计算效率上

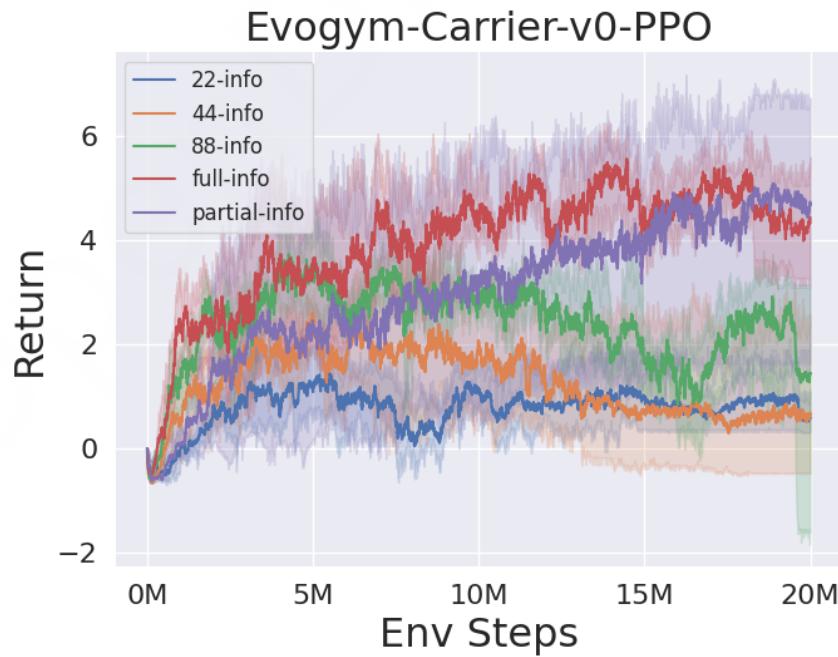
会存在问题。那么在实践中，一种常用的方式就是用所谓的 partial information encoding，就是编码信息由两部分构成

- 第一部分依然是所有质点的位置。
- 第二部分只用全局速度的一些统计量。



(图9：机器人全局和局部信息的编码。partial information encoding 用了所有质点的位置和全局速度，而 full information encoding 用了所有质点的位置和速度。)

那么从图10可以看到，partial information 和 full information 其实在最终性能上是相近的，full information 在前期收敛速度上会快一点，但是它因为编码信息更大，所以如果在一些比较复杂的情形，它消耗的计算资源和计算项也会更多。



(图10：partial information、full information和只保留软体机器人22、44、88个点的对比，可以看到 partial information 相比保留22、44、88个点能够保留一定的最终收敛效果。)

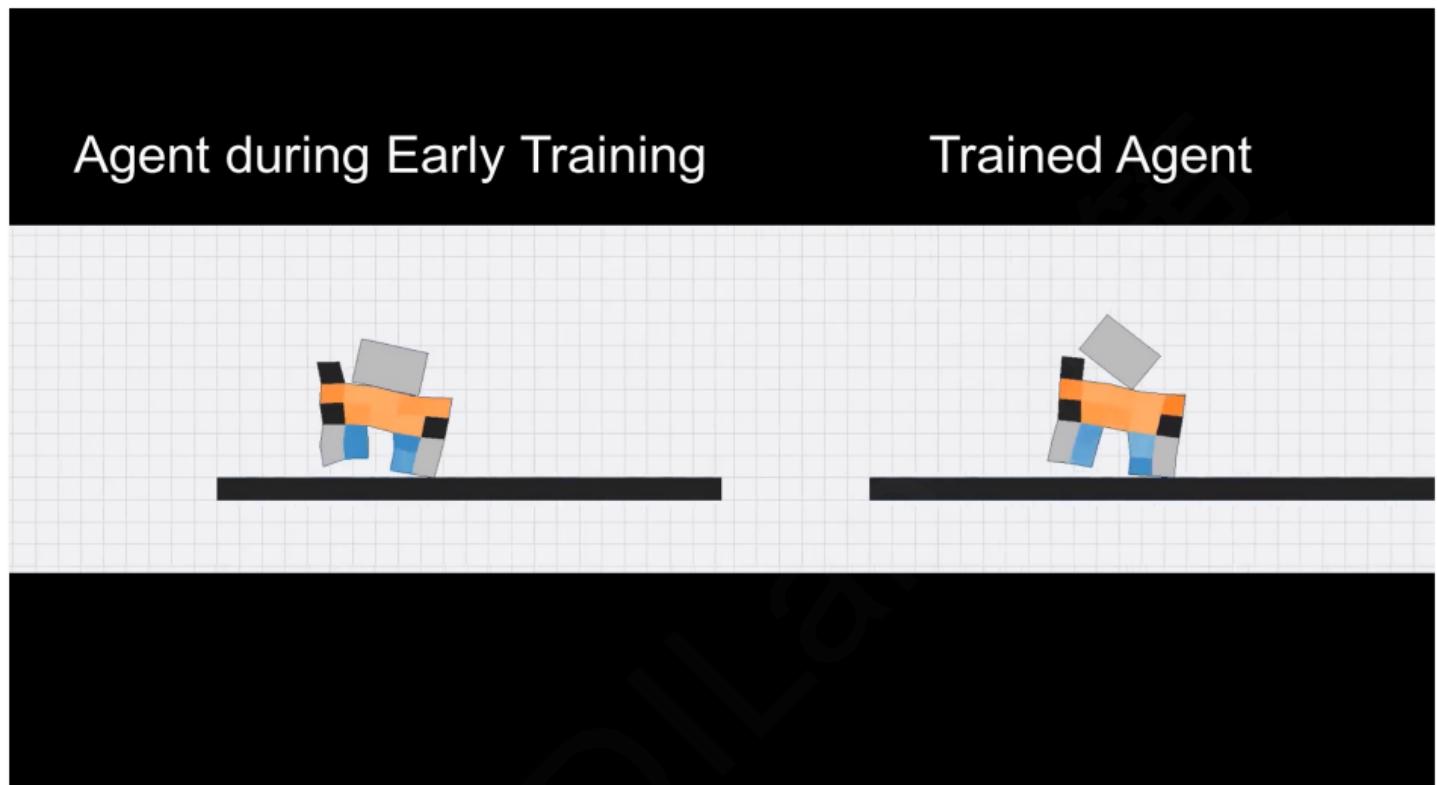
而当用另外的一些方式减小 observation 大小，比如很简单暴力地去丢弃其中的一部分点，那么只剩下 88 个质点，44 个质点和 22 个质点。可以看到丢弃的 observation 越多，它的性能会明显的下降，

而像 partial information 这种合理的设计方式则能够让它保留一定的最终收敛效果。

最后通过下方的对比视频来观察软体机器人在训练过程中以及到达收敛之后有什么样的表现。可以看到，虽然在训练早期和训练完成之后都可以向右运动，但是前期，整个机器人各个部分的控制仍然是存在一些问题，它的步履比较蹒跚。右边的这个训练好的智能体其实已经可以大步向前走。

完整的视频样例以相应的训练日志可以参考：

<https://github.com/opendilab/PPOxFamily/issues/8>。

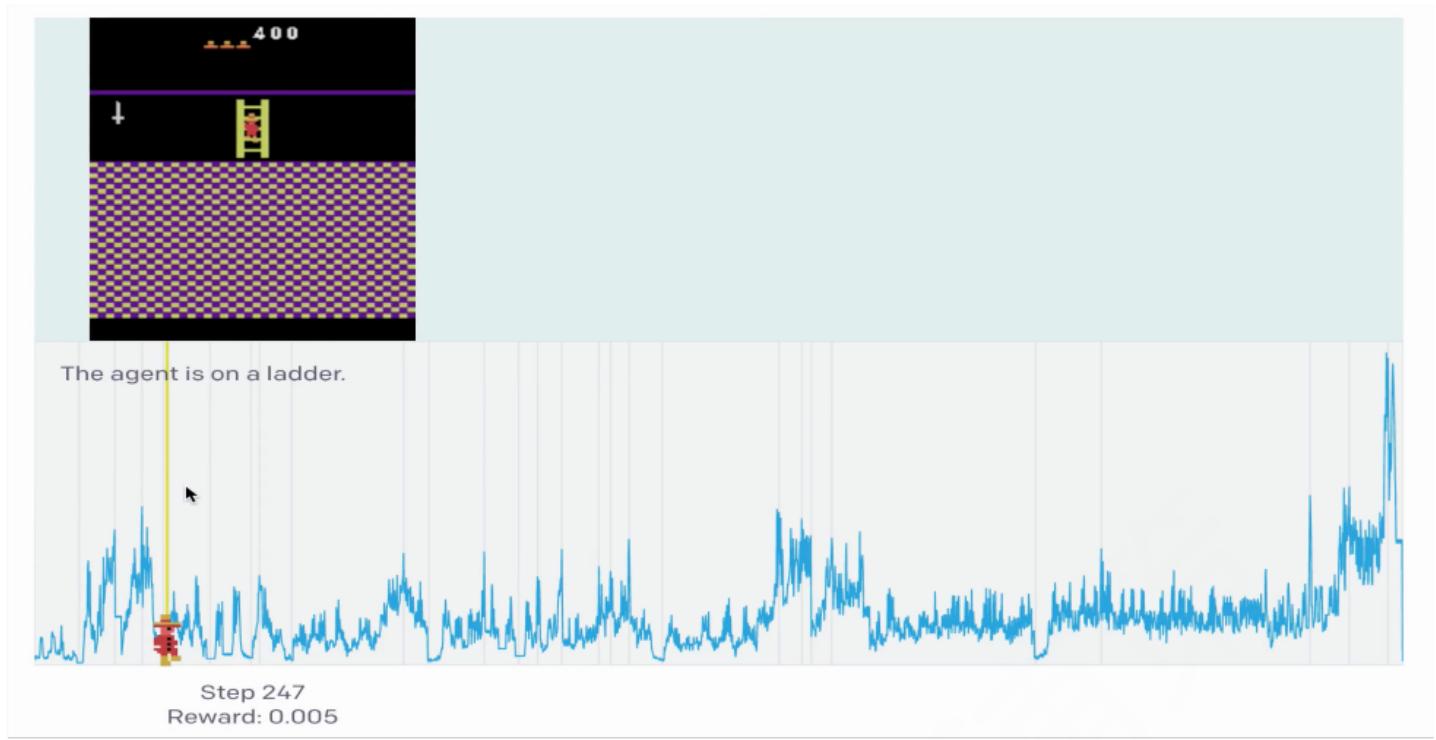


(图11：软体机器人训练过程。)

3.3 图片观察空间

3.3.1 引言：图片观察空间的特点和应用场景

接下来，本节课进入到一个更高维的观察空间——图片观察空间。人眼可以从各种各样的所见所感中提取到视觉信息来帮助人类决策，那么，强化学习智能体又是如何看待图片视频这些视觉信息的呢？



(图12：Atari——蒙特祖玛的复仇：一个完整 episode 内智能体遇到的观察状态和获得的奖励。)

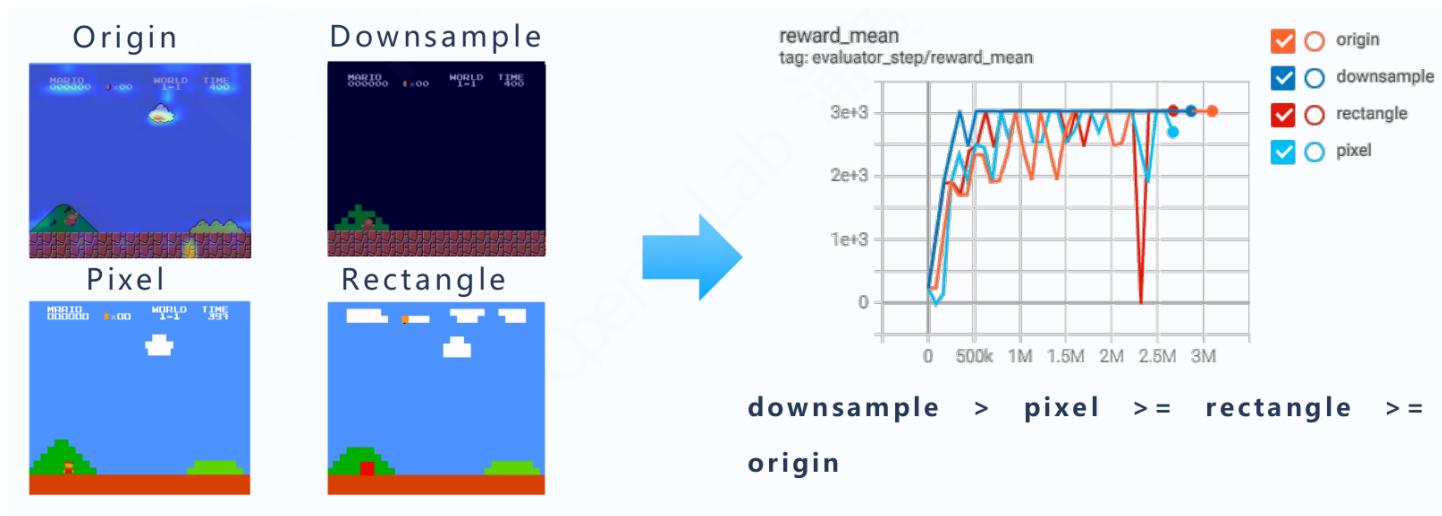
对于上面这个问题，可以用经典的学术环境 Atari 里面的一个非常困难的子环境——蒙特祖玛的复仇为例来说明。上图展示了一个完整 episode 内智能体遇到的观察状态和获得的奖励。从中可以发现，智能体在这个游戏的前中后期，需要去处理不同的观察信息，需要去借助这些信息来建模和处理游戏中的关键决策。

接下来，本节课将介绍具体对于 PPO 算法，如何让一个 RL 智能体更好地去建模图片观察空间。

3.3.2 理论：PPO + 图片观察

预处理

原始的图片观察空间最大的问题就是**图片的信息密度很低**，虽然维度很高，但是有很多冗余，这里使用经典的《超级马里奥》[9] 环境来举例。



(图13：左图：4种图像预处理方法。右图：三种改进方式都会相对于原图有更高更快的收敛速度和更稳定的效果，相比之下downsample > pixel >= rectangle >= origin。)

从图中可以看到，这个环境的决策任务就是要控制这个左下方的智能体小人，不断地向右闯关，最终到达城堡。但是，图片中的大部分内容，比如蓝天、白云和草丛都是跟游戏无关的装饰性道具，这些对于智能体决策其实是毫无意义的，但是这些信息本身依然存在于整个画面之中，所以智能体如果接收到的都是这样的信息，自然训练时数据利用效率较为低下。那么有什么样的预处理方法呢？接下来本节课将涉及一些简单具体的处理方法，即 downsample，pixel 和 rectangle 三种方法。

如图13 左半部分所示，downsample 方法就是对原始的图片信息去做降采样处理，并且把其中的蓝天、白云和草丛，一些冗余的跟游戏无关的东西全部去掉，而下方的 pixel 和 rectangle 方法其实是对原图本身去做一些像素化或者矩阵化处理，希望把这个信息变得更加简单低维。

算上原始的观察空间，这4种观察空间运用到 PPO 算法上会得到什么样的效果？最终的实验效果如右半部分所示，可以看到三种改进方式都会相对于原图有更高更快的收敛速度和更稳定的效果。尤其是第二种 downsample 方式，即将冗余信息都去掉的方式，它的收敛速度最快，而且在整个训练后续都非常稳定，智能体可以保持很高的通关率。

更深入地，可以通过 CAM (Class Activation Map) 这样的可视化工具来去分析智能体在不同的观察空间下看到了什么。如下面两个动图所示，左边这个是原图的 CAM 可视化，右边为 Downsample 处理后的 CAM 的可视化。图片中越亮的部分代表着这个智能体越关注这部分。可以看到，左边智能体需要学到去观察或者说注意到输入信息中的各种部分，而右边智能体会更加专注于自己本身的运动信息及跟游戏非常相关的道具；左边可能对于蓝天白云目不暇接，但是右边只是一心向右闯关。

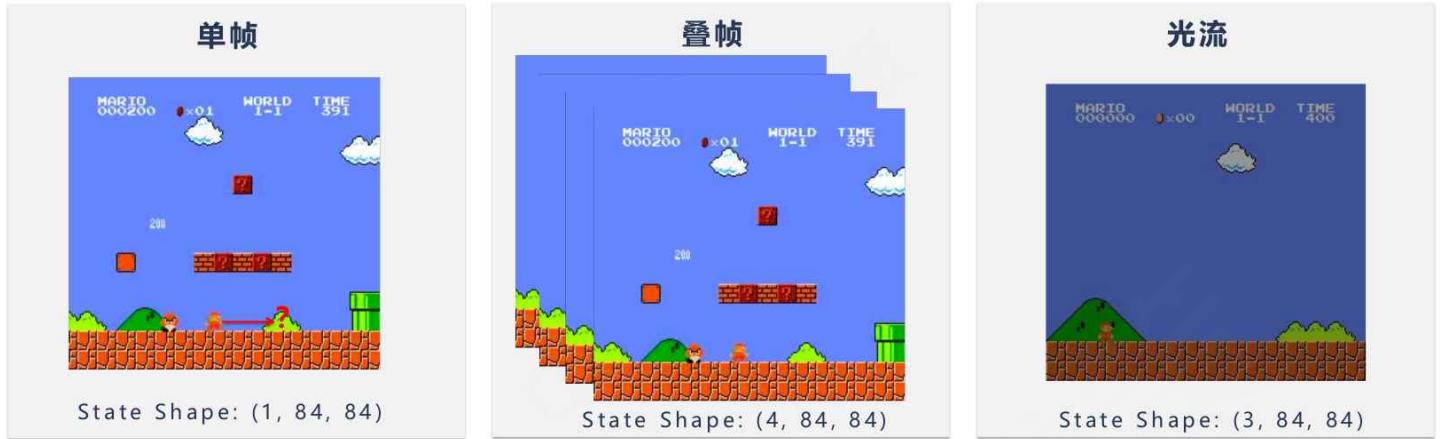
Tip：关于 CAM 可视化工具相关的更多的资料，读者可以参考相应的 GitHub 链接：GradCAM
<https://github.com/frgfm/torch-cam>。



(图14：利用 CAM 对智能体在不同的观察空间的关注区域进行可视化分析。CAM全称Class Activation Mapping，既类别激活映射图，也被称为类别热力图、显著性图等。可以理解为对预测输出的贡献分布，分数越高的地方表示原始图片对应区域对网络的响应越高、贡献越大。左图：智能体在原始游戏图像观察空间下的 CAM 可视化，可以看到智能体对画面中的蓝天白云等无关元素也进行了

关注。右图：智能体在Downsample 处理后的图像观察空间下的 CAM 可视化，智能体更专注于专注于自己本身的运动信息及跟游戏非常相关的道具。)

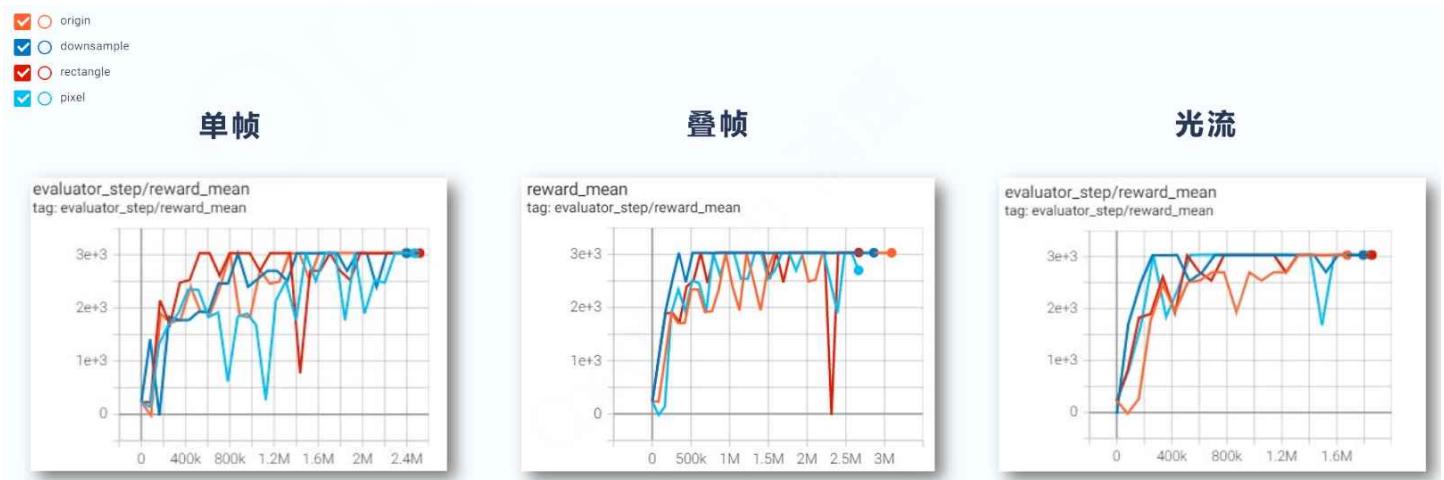
运动信息



(图15：左图：单帧图片无法表达运动信息。中图：将连续 4 个环境帧堆叠成一个动作帧。右图：提取 x 方向和 y 方向相应的光流信息并把这个信息和原始图片叠到一起。)

另一方面，运动信息的建模是很重要的，为什么要考虑运动信息的建模？主要还是单帧图片是无法表达运动信息的，比如左边的这个单帧图片，智能体仅仅看到这一帧，无法知道图中的小人它是向左还是向右，小人的速度是多少，那么像方向速度这些运动信息其实都是无法从单帧图片中获得的，所以做决策时就会少了一些必要的信息。那么怎么解决这个问题呢？实践中有两种很经典的思路。

1. 第一种思路就是叠帧，即将连续 4 个环境帧堆叠成一个动作帧，然后将它们整体输入给智能体，给出决策行为。
2. 第二种方式可以利用计算机视觉里面的经典方法--光流法，即使用光流去提取相应的运动信息。这里可以提取 x 方向和 y 方向相应的光流信息，并把这个信息和原始的图片叠到一起，然后输入给 PPO 智能体进行决策。右边的图片展示的就是光流提取出来的效果，可以非常准确地掌控整个环境中物体的运动情况。



(图16：单帧、叠帧和光流效果图。运用单帧信息的智能体训练过程的波动非常大，应用叠帧和光流后，智能体的收敛稳定性会好很多。)

上图给出了单帧、叠帧和光流这三种方式在马里奥环境上的效果，从中可以看到运用单帧信息的智能体，它训练过程的波动会非常大，因为少了必要的速度和方向这些信息，所以它在很多时候的探索和决策其实都是盲目的，会有一些相应的数据缺失和非平稳（non-stationary）的问题。但是叠帧和光流的设置下，当补足这个运动信息之后，智能体的收敛稳定性就会好很多，并且可以发现光流其实是更稳定、更高效的一种选择。

网络架构

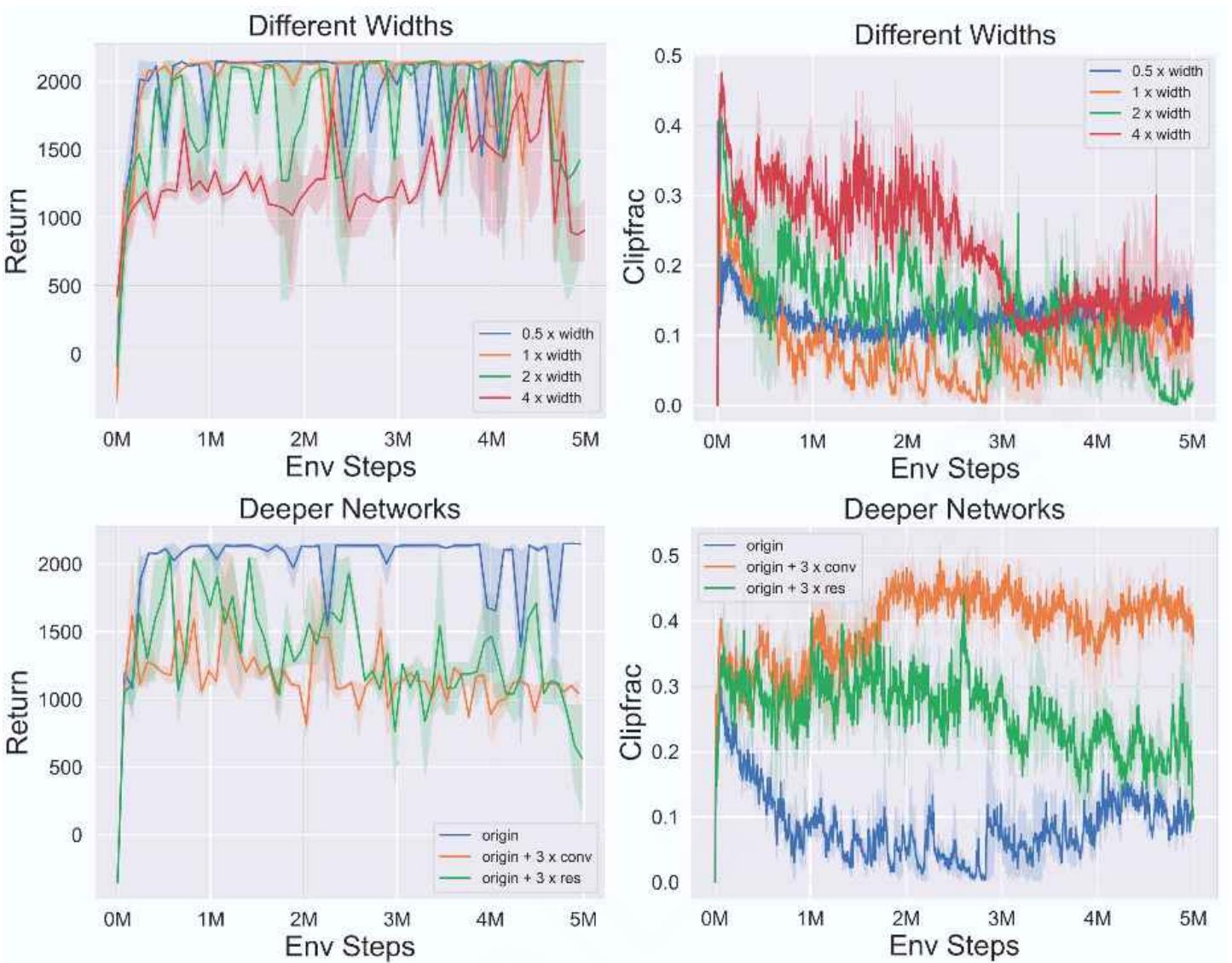
因为决策 AI 算法设计使用深度强化学习，那么**深度学习和强化学习其实是并重的**，概述课和动作空间专题更多提到了强化学习算法本身的设计，而对于观察空间更重要的就是关于神经网络和其训练方法的相关设计。

- 深度强化学习中网络设计的问题

在强化学习中使用神经网络，一个很重要的事实是：**单纯的加深或者加宽网络其实并不总是有效的**。在计算机视觉或者自然语言处理里面可能都会信奉一个理念，就是所谓的大力出奇迹，那么越大的网络、越大的数据一定能够带来更多的提升。但是在强化学习中，因为是在线训练的过程，智能体不断地跟环境交互，要平衡探索和利用去产生在线数据进行训练，而且训练的过程中并没有直接的监督信号，还是通过这种延迟的间接性的奖励进行学习，所以强化学习训练过程包含非常多不稳定和方差很大的情况，简单地增大网络并不一定有效，需要配合其他算法模块。

进一步理解，简单地扩大神经网络，虽然扩大了神经网络表征的表征能力（capacity），但是并不一定能够让网络学到这种信息。可能优化的上限虽然提高了，但是优化的难度可能也一并提高了，那么实际的训练过程就很难达到网络所对应的上限。图16给出了在更宽（channel 数变多）和更深（block 数变多）两种设定下，智能体在超级马里奥环境上的训练效果，简单的变宽和变深都会使智能体训练效果下降。而且从 PPO 特殊的 clip fraction 指标来看扩大神经网络的实验组训练不稳定性也更大。

- clip fraction 代表 PPO 算法设计中裁剪掉的部分。通常来讲，优化过程中希望 π_{old} 和 π_{new} 比较接近，保持优化过程比较接近 on-policy，从而使得训练曲线比较稳定。那么 clip fraction 越大，说明算法要必须通过 clip 去掉的部分就越大，优化的不稳定性也越大。



(图17：左上：简单加宽神经网络使得 return 下降。右上：简单加宽神经网络使得 clip fraction 上升。

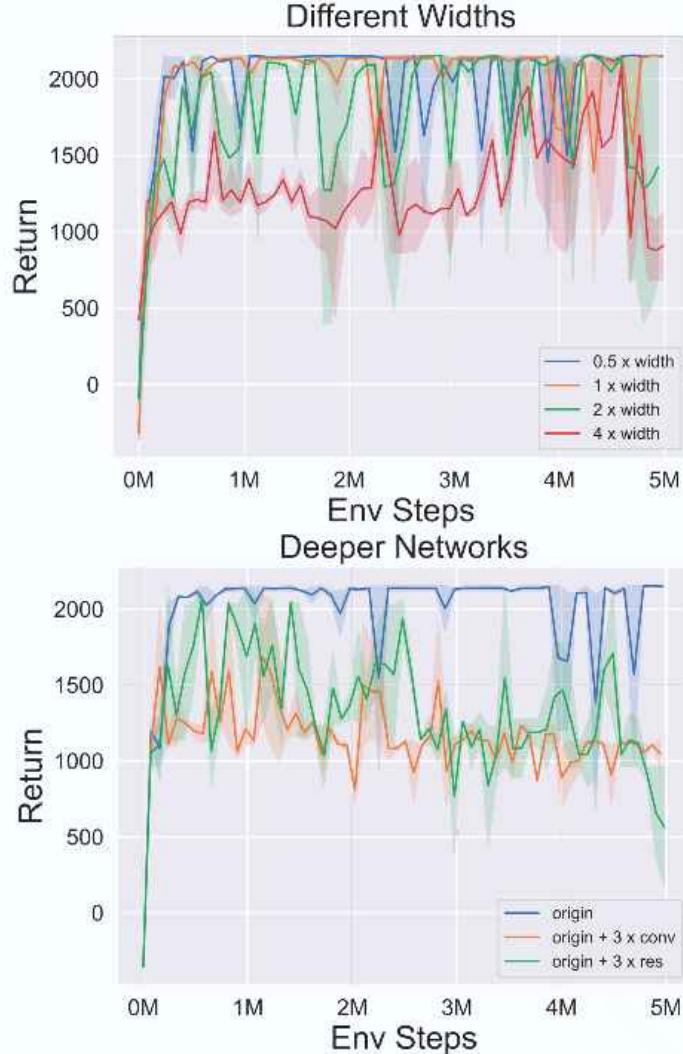
左下：简单加深神经网络使得 return 下降。右下：简单加深神经网络使得 clip fraction 上升。)

- 使用 LN 和 residual connection 来稳定优化

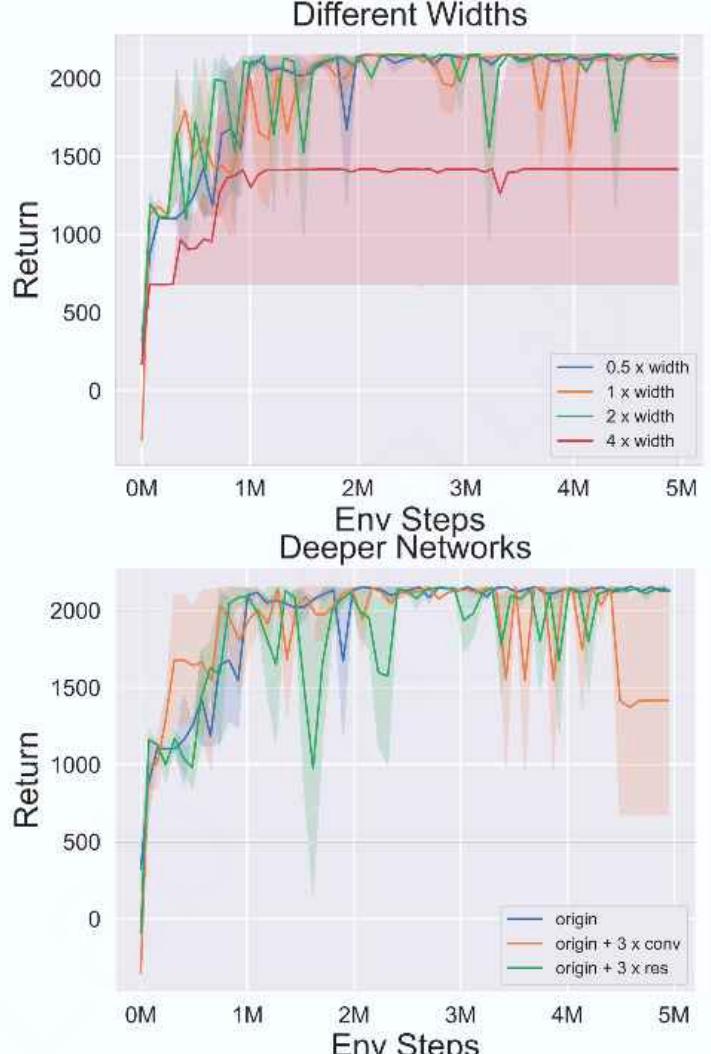
那么如何把神经网络的能力上限真正用起来？这里可以借助一些的神经网络的经典结构设计和训练方法，比如说在强化学习之中，就可以在网络中使用像 Layer Norm (LN) 这样的归一化方式。要注意，这里不能使用经典的计算机视觉设计中常用的 Batch Norm (BN)。因为在线强化学习是收集一个 episode 或者说一整条轨迹的数据，这样收集的数据肯定不满足独立同分布的假设。直接使用 Batch Norm 会遇到这种数据相关性的问题，不满足 BN 算法设计的相应假设，所以更推荐使用 LN 去做。比如说对于图片信息，就是将它的 C, H, W 这三个维度合起来进行 norm。对于上文中这样网络变宽和变深之后发现性能明显下降的情况，在添加了LN 之后，那么训练稳定性和效果都有一定的提升，并且从 clip fraction 上看也会变得更加稳定。

那么关于 LN 更详细的实现以及数学理论上的理解，可以参考[作业题](#)中的相关内容。

without LN



with LN



(图18：左图：没加 LN 加宽加深神经网络对训练效果影响很大。右图：添加 LN 后再加宽加深神经网络，训练稳定性和效果都有一定提升。)

3.3.3 代码：巧用 Env Wrapper

观察空间中有很多经典预处理操作，比如上面讲到的添加光流、叠帧、图片降采样等等操作。而这些操作其实在经典的强化学习框架里面已经有非常多、非常好用的实现。本节课对于各种预处理操作的代码实现，抽象为 Env Wrapper 这种代码结构，读者能够通过简单的代码来一键使用这些 Env Wrapper，从而对原始的强化学习环境快速构建一系列预处理操作，让它变得更加适合于强化学习训练。

wrapped_mario_env 函数功能概述
使用叠帧包裹器，以及多种其它包裹器，对马里奥环境进行包裹。

```

42 def wrapped_mario_env():
43     env = gym_super_mario_bros.make("SuperMarioBros-1-1-v0")
44     return DingEnvWrapper(
45         JoypadSpace(env, [[["right"], ["right", "A"]]]),
46         cfg={
47             'env_wrapper': [
48                 lambda env: MaxAndSkipWrapper(env, skip=4),
49                 lambda env: WarpFrameWrapper(env, size=84),
50                 lambda env: ScaledFloatFrameWrapper(env),
51                 lambda env: FrameStackWrapper(env, n_frames=4),
52             ],
53         },
54     )
55     lambda env: EvalEpisodeReturnEnv(env),
56

```

基础马里奥环境初始化。

限制动作空间为“向右”、“向右且起跳”。

返回相邻四帧的最大值。这个包裹器的作用类似于在时间维度上进行了一次最大池化。

使用双线性插值对图像的尺寸进行修改，变成 84×84 并且从三通道的 RGB 转变为单通道的灰度图像。

将值归一化，具体使用的公式是：

$$scaled_x = \frac{x - \min(x)}{\max(x) - \min(x)}$$

将相邻四帧叠在一起形成一个 obs。

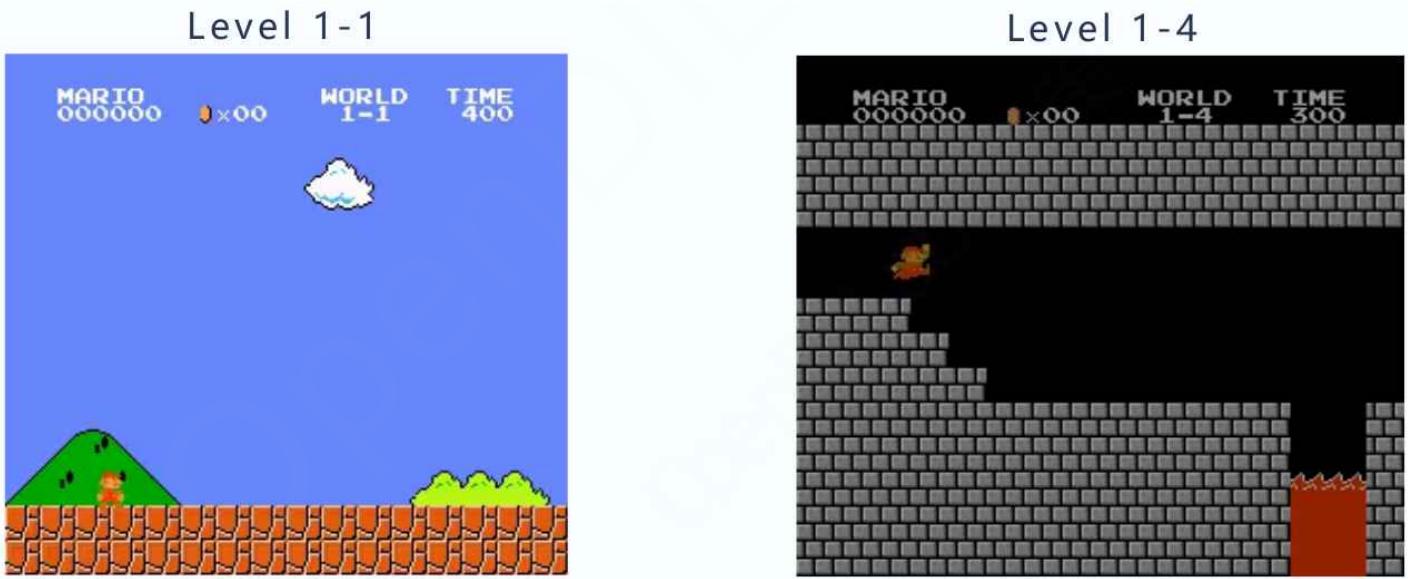
在 evaluate 的时候计算最终的累计回报。

(图19: Env Wrapper 的代码讲解示例展示。)

Env Wrapper 完整的例子可以参考课程的“算法-代码”[注解文档](#)。

3.3.4 实践：PPO + 超级马里奥

超级马里奥环境介绍



(图20: 左图: level 1- 1。右图: level 1- 4。)

智能体经过训练可以快速完成通过 1-1 关卡，并且，如果额外添加了一些金币奖励，智能体在通关的过程中还可以做到快速三连跳吃掉金币等等操作。而在右边 1-4 这个环境中，由于关卡难度的提升，智能体则需要去更加聪明地躲避各种障碍和危险，所需的训练时间也越长。

这里给出了运动光流加游戏画面本身的图像。其实 1-4 本身为了保持快速通关需要不断向右走，但是如果走的太快，又会遇到一些相应的问题。比如下图中面对最后三个关卡道具，智能体会有一个简单的急刹车住恰当的位置保持距离，然后快速通关。



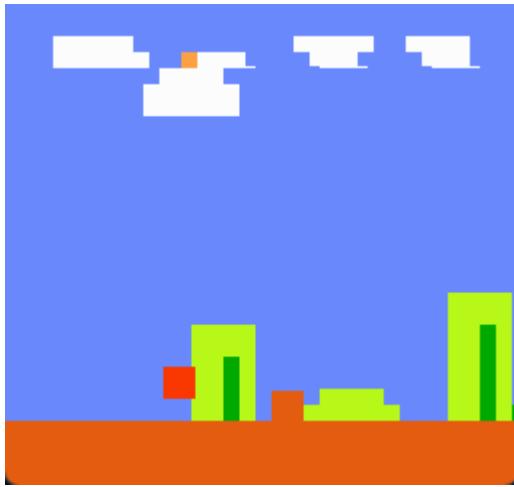
(图21：运动光流加马里奥游戏本身画面的图像。可以看到在面对最后三个需要“时机把握”的障碍物时，智能体利用光流信息学习到了这里“刹车”等待时机的策略。)

状态空间

gym-super-mario-bros 的状态空间输入是图像信息，及三维的张量矩阵 (datatype=uint8)。此外，游戏的不同版本对应的图像分辨率 `240*256*3` 相同，但版本越高，图像越简略（像素块化），具体如下所示：

```
1 >>> # 查看观测空间
2 >>> gym_super_mario_bros.make('SuperMarioBros-v3').observation_space
3 Box([[[0 0 0] [0 0 0] [0 0 0]
4 ...
5 [0 0 0] [0 0 0] [0 0 0]]], [[[255 255 255] [255 255 255] [255 255 255]
6 ...
7 [255 255 255] [255 255 255] [255 255 255]]], (240, 256, 3), uint8)
```

SuperMarioBros-v3 对应的游戏截图如下，用到了上文中说到的 rectangle 处理方法：



(图22：SuperMarioBros-v3 对应的游戏截图)

动作空间

gym-super-mario-bros 的动作空间默认包含任天堂红白机全部的 256 个离散动作。为了压缩这个大小（利于智能体学习），环境默认提供了动作 wrapper `JoypadSpace` 来降低动作维度：可选的动作集合及其含义如下：

```
1 # actions for the simple run right environment
2 RIGHT_ONLY = [['NOOP'], ['right'], ['right', 'A'], ['right', 'B'], ['right', 'A',
   'B'], ,]
3 # actions for very simple movement
4 SIMPLE_MOVEMENT = [['NOOP'], ['right'], ['right', 'A'], ['right', 'B'], ['right',
   'A', 'B'], ['A'], ['left'], ,]
5 # actions for more complex movement
6 COMPLEX_MOVEMENT = [['NOOP'], ['right'], ['right', 'A'], ['right', 'B'], ['right',
   'A', 'B'], ['A'], ['left'], ['left', 'A'], ['left', 'B'], ['left', 'A', 'B'],
   ['down'], ['up'], ,]
```

例如：

```
1 env = gym_super_mario_bros.make('SuperMarioBros-v0')
2 # 使用 SIMPLE_MOVEMENT
3 env = JoypadSpace(env, SIMPLE_MOVEMENT)
4
5 # 或者自己设置动作空间为只有向右和向右跳
6 env = JoypadSpace(env, [{"right"}, {"right", "A"}])
```

对于 SIMPLE_MOVEMENT 所代表的 7 维离散动作空间，使用gym环境空间定义则可表示为：

```
1 action_space = gym.spaces.Discrete(7)
```

奖励空间

游戏规则希望马里奥能更多地**向右移动、更快地抵达终点而不会死亡**，因此每一帧的奖励设置由如下三部分组成：

1. v ：代表连续的两帧之间，马里奥的x坐标之差（可以理解为向右的速度），有正有负；
2. c ：每一帧的用时，简单理解为每一帧都有一个负的reward，用来push智能体更快到达终点；
3. d ：死亡的惩罚，如果马里奥死亡，给与 -15 的高额惩罚；

总的奖励 $r = v + c + d$

奖励被 clip 到 $(-15, 15)$

终止条件

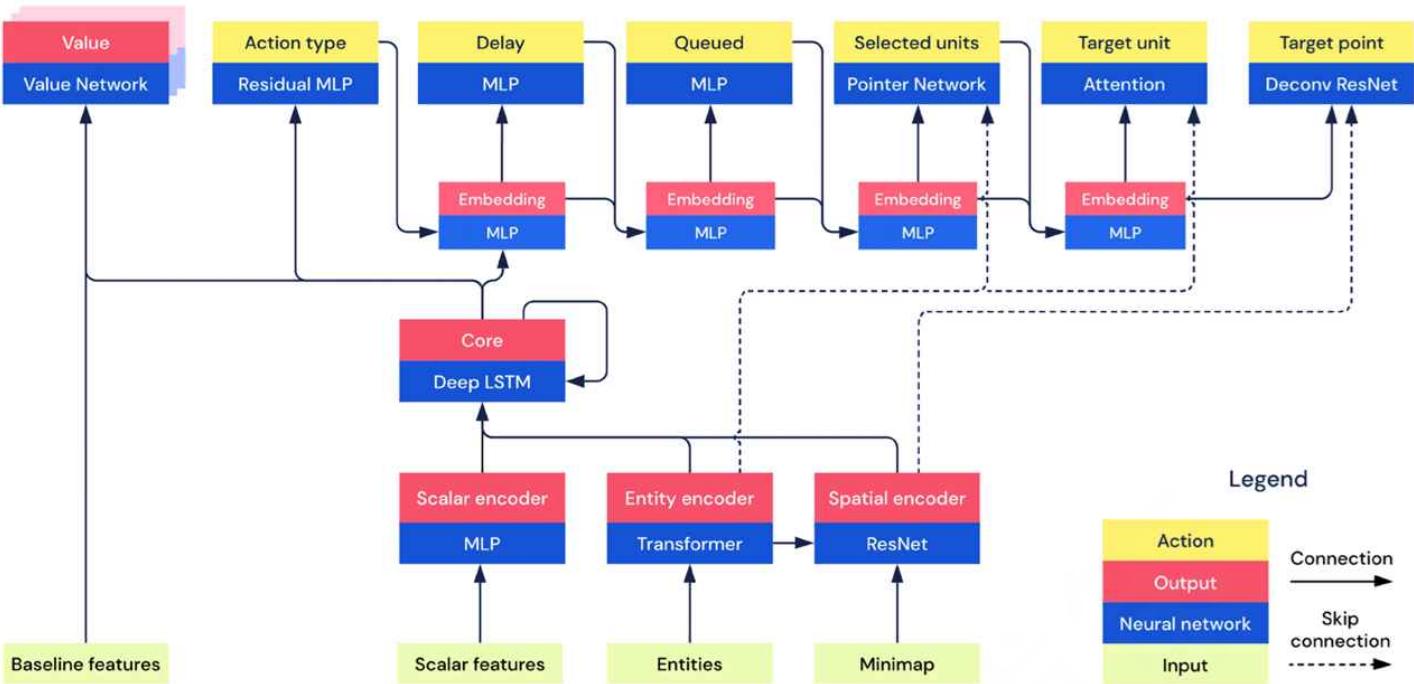
gym-super-mario-bros 环境每个 episode 的终止条件是遇到以下任何一种情况：马里奥成功通关、马里奥死亡、倒计时结束。

具体文档参考：https://di-engine-docs.readthedocs.io/zh_CN/latest/13_envs/gym_super_mario_bros_zh.html

3.4 复杂结构化观察空间

3.4.1 理论：用 Transformer 建模元素间相关关系

本节课的第三部分将介绍实践中更加常用，也是更具有挑战性的观察空间——复杂结构化观察空间。在大规模复杂决策问题里（如《DOTA2》、《星际争霸2》）经常遇到这样的观察空间。下图是 DeepMind 设计 AlphaStar[10] “星际争霸2” 智能体时所使用的网络结构。可以看到，其中包含各式各样的神经网络模块以及复杂的连接关系。



(图23：Deepmind 在 AlphaStar[10] 中为“星际争霸2” 观察空间所设计的网络结构。)

这个复杂方案主要有三个关键点：

- **第一部分：不同模态的数据需要结合对应领域经典的建模方法和网络结构设计。** 比如像《星际争霸2》的地图信息，其实就可以视作为一种图片观察空间，使用计算机视觉里面的经典结构 U-Net[11] 来进行建模；而对于《星际争霸2》中单位之间的协作和配合，就可以使用 Transformer 这种网络结构建模各个单位之间的相关关系；此外，由于《星际争霸2》是一个长期规划决策博弈的即时战略类游戏，所以时序建模就是一个很重要的话题，自然少不了 LSTM 等循环神经网络的使用。
- **第二部分：一些重要信息需要利用跳远连接 (skip connection) 来进行补充。** 这里着重指表征输入部分和决策预测器输出部分之间所需要特殊补充的一些跳远连接。比如对于预动作类型 (action type) 的预测，如果智能体希望连续输出多个重复的动作，则可以把上一帧的动作类型 (last action type) 作为一个关键的输入信息补充过去。在处理复杂结构化的观察空间时，网络会很庞大很复杂，其中很多信息可能存在遗忘或丢失的现象，所以会采用这种特殊的 skip connection 进行信息的补充。另一个例子，在预测技能的释放位置时，会将提取到的多尺度 (multi-scale) 观察信息一起送入最终的技能位置 (target location) 预测器，以帮助智能体更精确的预测技能的释放位置。
- **第三部分：要根据决策任务的特点灵活组合各种网络结构设计方法。** 《星际争霸2》是一款即时战略类游戏，相关的先验知识可以启发设计更恰当的网络结构。对于不同的决策问题，不能生搬硬套相应的神经网络方法，而是应该结合决策问题的特性综合设计。比如现在需要建模一个3D视觉输入的问题，并不能简单地将2D的卷积改成3D的卷积，这样会引入巨量的额外计算，而是需要采用一些3D建模的经典方法，比如 Occupation Network[12]。或是例如在《星际争霸2》中利用 Transformer 来建模智能体之间的配合，但如果现在需要得到股票买入卖出的最佳策略，问题就从单位之间的相关性转换到股票之间的排序和比较，此时 Transformer 并不一定是一个最好的选择。

3.4.2 实践：使用 PPO 玩转羊了个羊

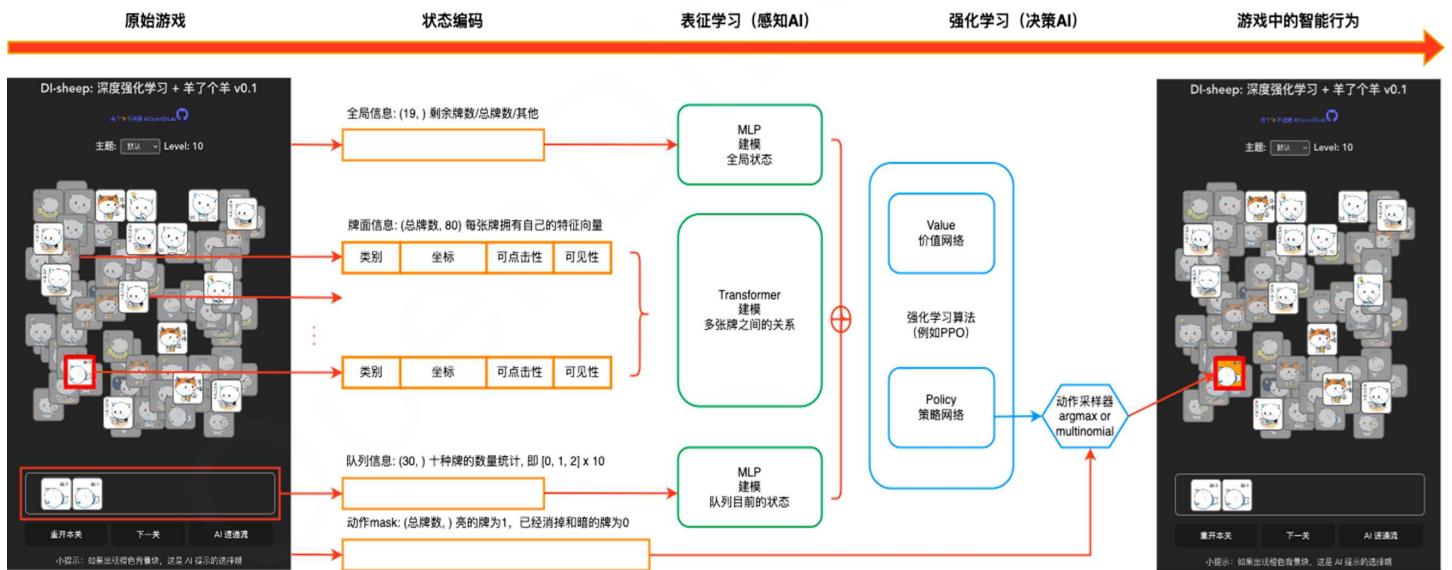
羊了个羊环境介绍

接下来将介绍一个有趣的结构化观察空间实践应用，PPO 算法结合2022年很火的一个小游戏“羊了个羊”。这里简单搭建了一个“羊了个羊”的仿真环境，尝试探索 PPO 强化学习算法如何处理这样的问题。“羊了个羊”其实是一种很简单的三消类小游戏，如图24左边画面所示，首先需要处理状态编码和表征建模部分，需要编码的信息可以分为三部分：

- 全局信息：包含剩余牌数和总牌数等宏观指标。
- 牌面信息：每一张牌的类型、是否可点击、是否可看见等属性。
- 队列信息：在队列中的各种类型牌的数量信息。

对于不同模态的数据应选择不同的网络结构进行处理，例如，这个游戏中牌面信息的建模与《星际争霸2》单位选择很类似，需要建模牌面之间的关系，例如牌的位置关系，牌的类别是否可以进行各类组合或配合，此时 Transformer 就成了一个很好的选择。最终，对于这三种模态的数据使用相应的神经网络结构进行特征向量提取，并将这些特征向量合并起来，即为最终决策预测器所需的特征向量。在决策预测器部分，则构造相应的 policy head 和 value head 分别输出离散策略概率和价值函数。完成设计之后，整个神经网络部分使用 PPO 算法进行优化，最终训练得到能够玩转“羊了个羊”的智能体。

Tip：详细的网络结构设计和详细的代码可见 [DI-sheep 开源项目](#)。



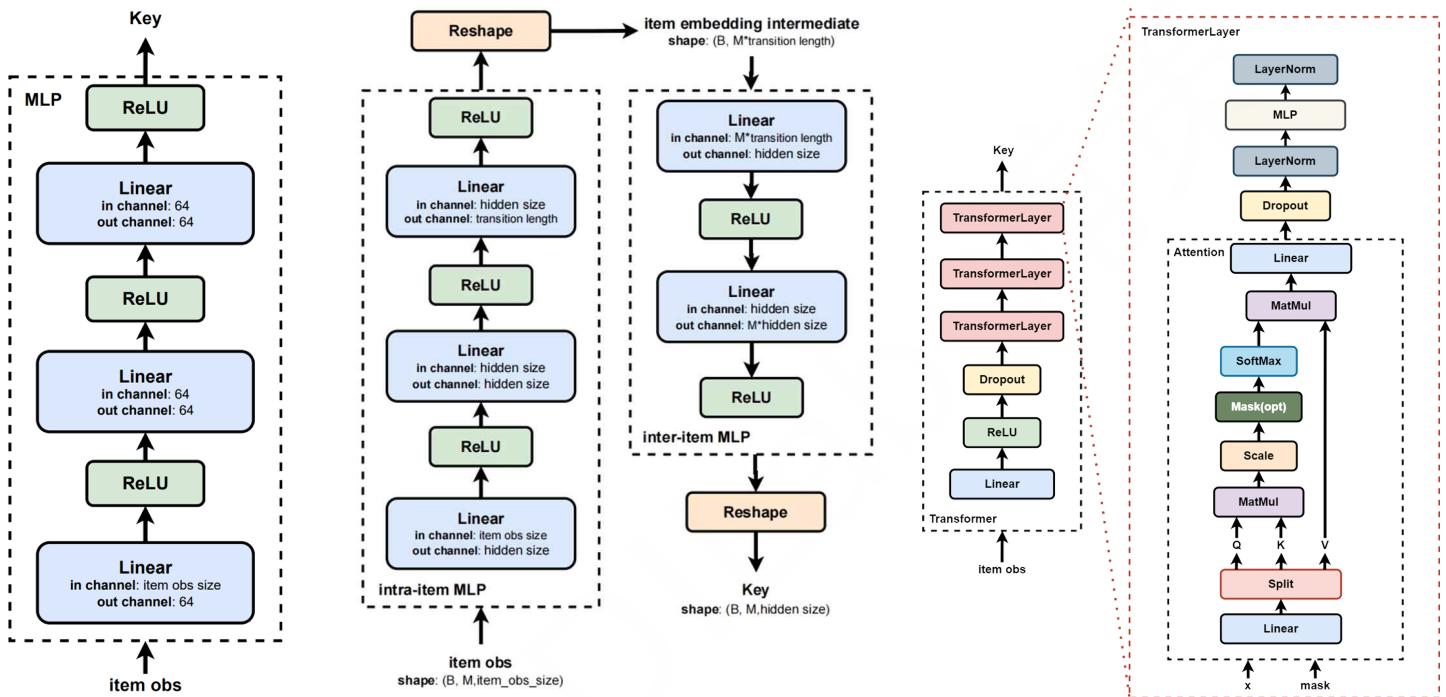
(图24：DI-sheep 算法原理解析，包含输入信息，状态编码，网络结构，输出信息四大部分。)

Transformer 和 MLP 的对比分析

更具体地，为了分析使用 Transformer 建模牌面关系的优势。本节课基于“羊了个羊”样例做了一个更深入的对比实验，对比了三种不同的牌面信息编码器。

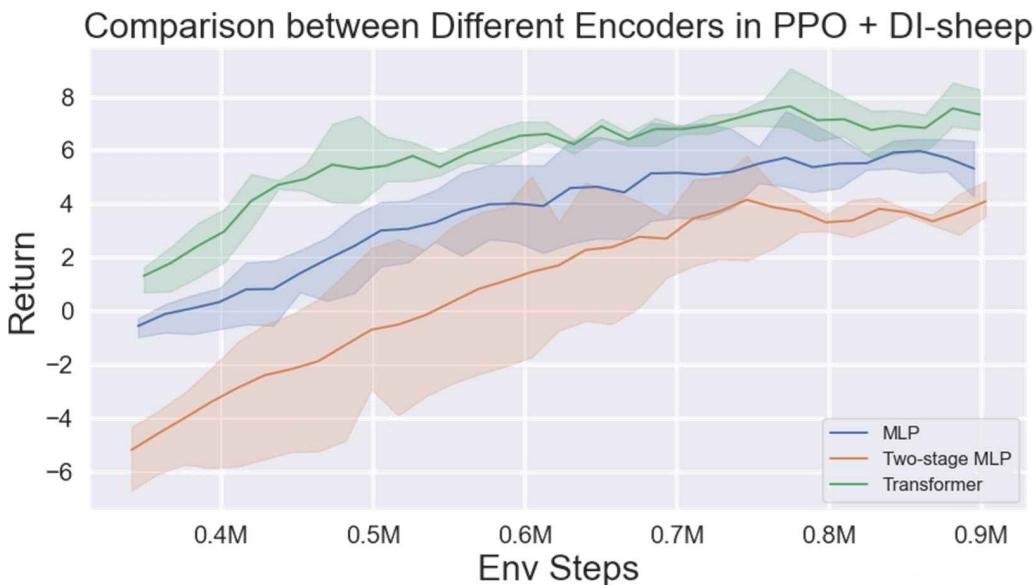
- 第一种采用最简单的多层感知机 MLP (Multi-Layer Perceptron)，即将所有牌的信息拼到一起，然后一起并行进入这个 MLP。这种方式下所有的牌都是并行通过，神经网络中对牌与牌之间的关系并没有做任何的信息交互，只是对牌内部的属性了一定的处理和建模。

- 第二种方式采用了两阶段的 MLP，两阶段各自负责建模牌内部属性和牌之间关系。第一阶段依然让所有的牌都并行通过 MLP 部分，此时是对牌的类别、牌的可见性、可点击性等信息进行处理并压缩成一个更小的特征向量，然后将它们拍平（Flatten）或者 Reshape 成一个 $(B, M^*length)$ 的向量，将这个向量输入到另外一个 MLP，在这个第二阶段的 MLP 中即可发生牌之间的信息交互。在输出完最终结果之后，再把结果 Reshape 回原来的形状，最终就得到每一张牌本身的特征表示。理论上来讲，这种两阶段设计是可以建模牌与牌之间的联系和关系的。
- 第三种方式采用 Transformer 建模牌面信息，优点在于它可以更好地到牌与牌之间的组合，因为 Transformer 中的多头注意力机制（Multi-Head Attention）就是在建模各个牌之间是否有相关关系，并且多个 Head 可以建模多种类型的联系，此时模型的表征能力就会很强大。



（图56：针对 DI-sheep 设计的三种牌面信息编码器。（左）：采用一层最简单的多层感知机 MLP (Multi-Layer Perceptron) 作为牌面信息的编码器，这种方式对牌与牌之间的关系并没有做任何的信息交互，只是对牌内部的属性了一定的处理和建模。（中）：采用两阶段的 MLP 作为牌面信息编码器，希望在第一阶段的 MLP 中对牌内部的属性进行处理建模，在第二阶段的 MLP 中进行牌与牌之间的关系关系建模。（右）：采用 Transformer 作为牌面信息编码器，多头注意力机制（Multi-Head Attention）使之可以建模牌间关系，且多个 Head 可以建模多类型联系。）

似乎第二种和第三种方式都可以建模牌之间的关系，那么建模牌之间的关系究竟重不重要，使用这两种网络结构会有怎样的差别？“羊了个羊”游戏 Demo 上的对比实验揭露了答案，结果如图26所示。可以看到：建模牌之间的关系非常重要，Transformer 方法相比蓝色线所示的 MLP 方法有更高的收敛速度和最终的性能；而且进一步对比发现，两阶段的MLP 理论上是可以建模牌与牌之间关系，但实际上优化出来的结果非常之差，甚至不如简单的一阶段 MLP。



(图26：三种牌面信息编码器的学习效果对比。可以看到 Transformer 方法收敛速度和最终性能都最好，其次是一层 MLP 编码器，两阶段 MLP 编码器的效果最差。)

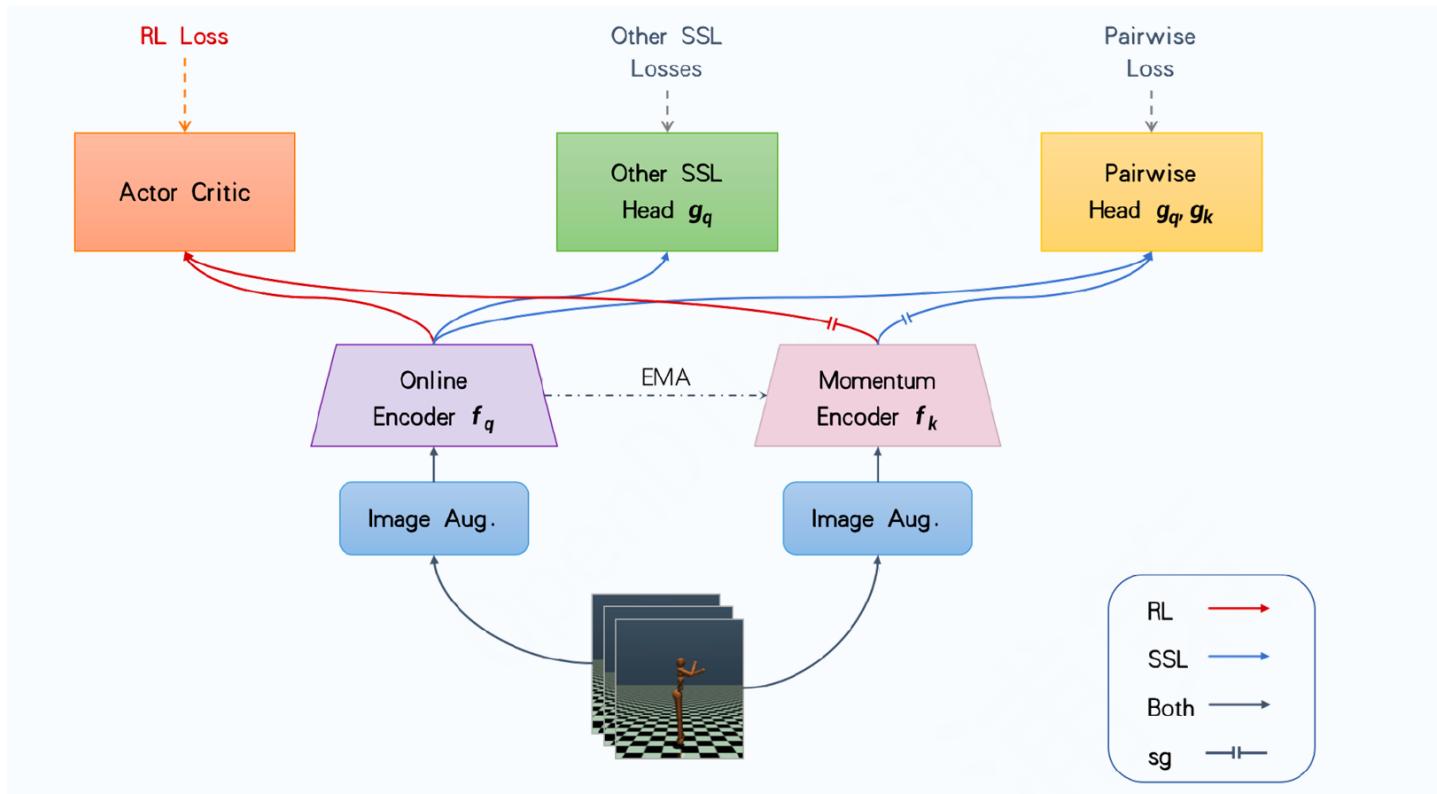
这个对比实验可以揭露一些关键事实：

- 第一点，从环境角度而言，多张牌之间的关系建模对此类游戏非常重要。
- 第二点，从计算的角度来讲，神经网络参数量的大小并不一定很重要，合理利用参数进行计算才是最重要的事情，这3种网络结构在神经网络参数几乎保持相等相近的情况下，Transformer 会有更密集的计算模式，这种计算模式其实是它真正具有强大表征能力的重要的原因之一。
- 第三点，从算法角度上来讲，一个网络结构的设计代表问题的上限。在这个例子里，网络结构设计中的信息交互和相应的映射关系，确定了这个问题的优化上限空间。但是正如两阶段 MLP 实验结果所反映的，虽然说这种设计有建模的能力，但是在实际优化中这种方式其实是比较“笨重”的，一般的训练方法很难训出期望的结果，所以网络的训练方式决定问题的下限。

更多关于 PPO + 羊了个羊的实验细节可以参考：

- 实验细节：<https://github.com/opendilab/PPOxFamily/issues/8>
- 科普文章：<https://mp.weixin.qq.com/s/4Z3WtkcWRp6x4x60RVELfQ>

3.5 通用观察空间训练方法



(图27：通用观察空间训练方法之一：强化学习和表征学习联合训练。在强化学习的经典流程之外，额外定义一些辅助的表征训练模块，帮助强化学习提取更好的表征，从而让决策变得更加容易，主要采用自监督学习方法（Self Supervised Learning, SSL），例如基于正负样本对的对比学习，利用数据增强进行正则化，类似 MAE (Masked Auto-Encoder) 的上下文补全的辅助任务。在不同环境或不同决策问题中，不同辅助任务往往有不同的表现和特点，如何获得一致提升的联合训练方法是一个待研究的问题。关于此方面的研究请参考[观察空间表征学习补充材料](#)。图片来源于论文 [13]。)

3.5.1 理论：Actor 和 Critic 之间是否共享编码器

本节将从强化学习本身特点的角度思考 Actor-Critic 类的强化学习算法中一个关键问题：**Actor 模型与 Critic 模型是否共享编码器。**

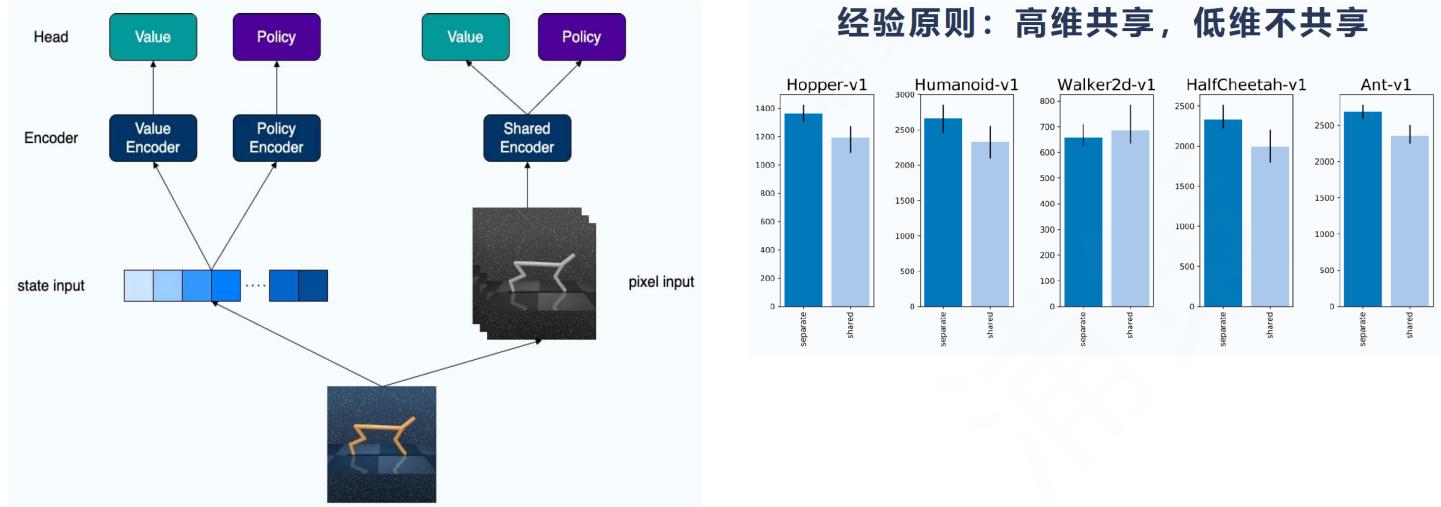
强化学习中的编码器 encoder

对于包括 PPO 在内的 Actor-Critic 算法，一般由两个不同功能的组件构成，即 Actor 模型与 Critic 模型。前者负责针对观测状态，生成策略动作。后者负责针对观测状态和动作，判定动作价值。

在强化学习的环境中，观测值往往是复杂的，尤其是对于包含视觉输入的场景而言。对于复杂环境，一般观测状态是高维且抽象的输入，无法直接编码成向量，让 Actor 或 Critic 模型直接处理这个高维输入。此时，算法中需要设计一些特征工程的方法，但人为手工设计的编码器往往只能适用于特定的问题和环境，迁移性泛化性较差。机器学习领域往往使用设计的神经网络编码器（Encoder），将高维输入中的有效信息映射为较低维度但具备较高信息密度的特征（Feature）。

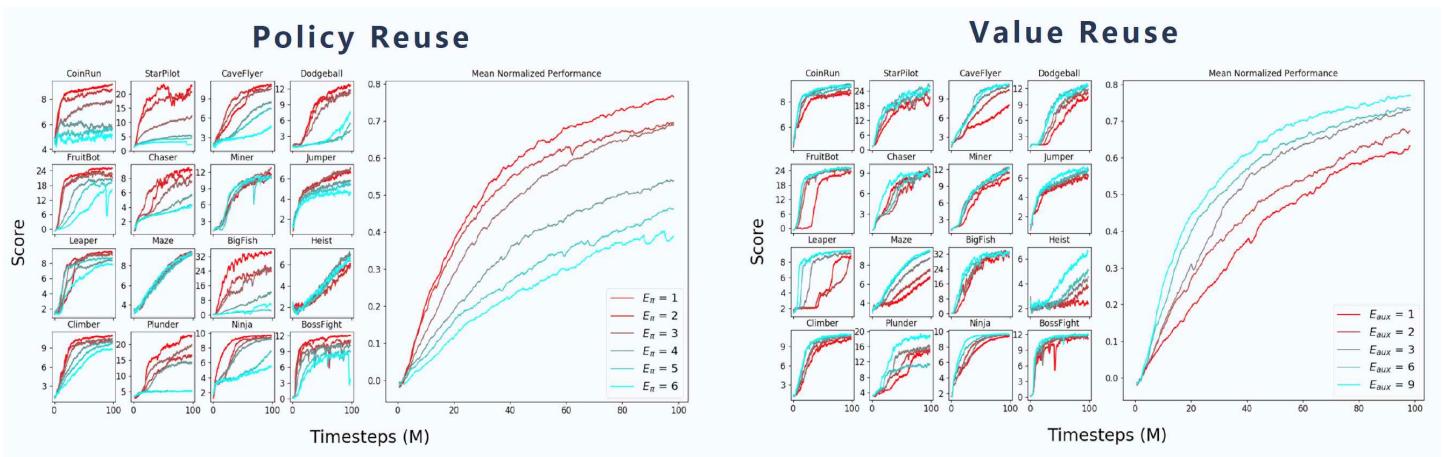
是否共享编码特征以及如何学习

- 编码器的学习是强化学习算法中比较重要的问题。在 Actor-Critic 算法中，需要将观测的原始信息传入一个编码器，然后将编码后生成的特征向量再传入 Actor 模型与 Critic 模型。在训练方面是通过优化 Actor 模型和 Critic 模型的目标函数来学习 encoder 的参数的。此时，会遇到一个设计上的抉择，即需要为 Actor 模型与 Critic 模型设计两个独立的编码器，还是让 Actor 模型与 Critic 模型共享同一个编码器。例如，对于 PPO 算法，policy 网络和 value 网络是否要共享主干 encoder。一般有一个经验原则，即“高维共享，低维不共享”，如下图所示。



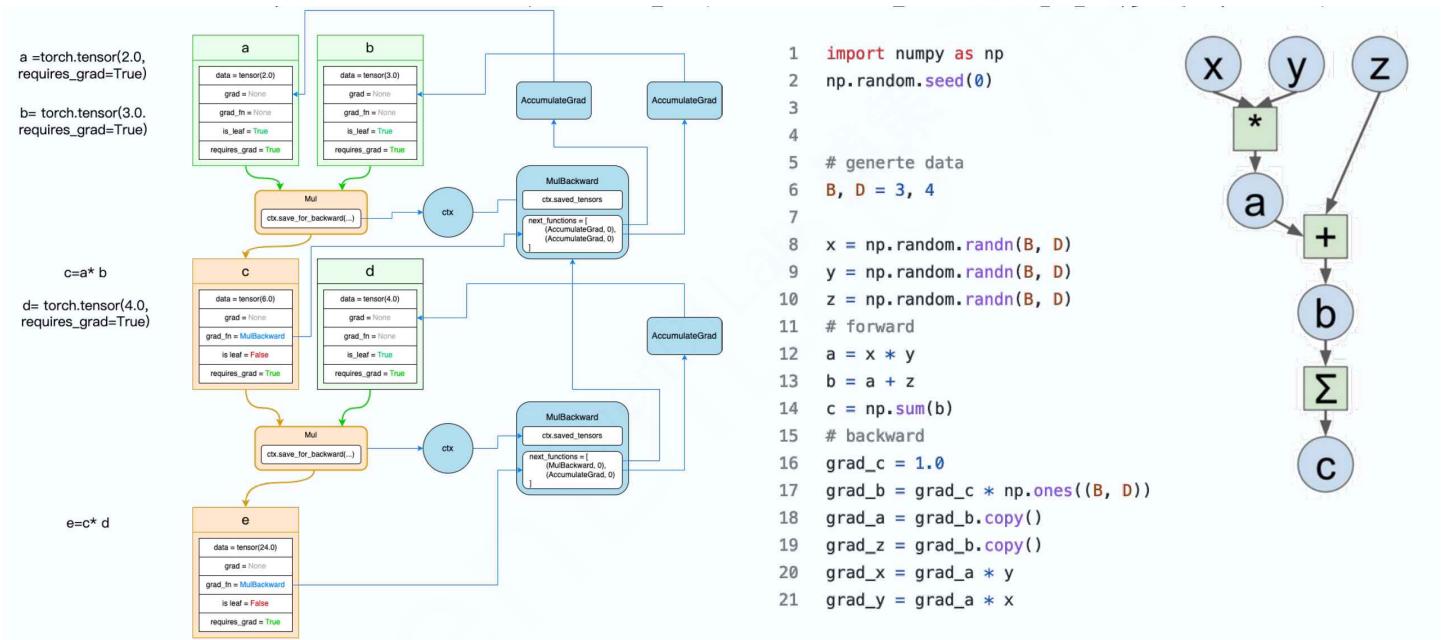
(图28：左图：DeepMind Control 环境的输出既可以是关于物理参数信息的向量输入 (state input)，也可以是渲染得到的图像输入(pixel input)。右图（来源于论文[14]）：对于 state input 的 PPO 算法，如果共享 encoder 会使收敛速度和最终性能有明显下降，但对于 pixel input 的 PPO 算法，如果共享 encoder 会使收敛速度和最终性能有明显上升。）

- 为什么会有这样的经验原则呢？简单分析，原因在于：
 - 不同任务联合训练，自然会有他们之间的梯度竞争和干扰，在低维的观测空间上，这种干扰是占主导地位的，所以共享 encoder 会造成性能下降。而在高维图片观测空间上，希望 encoder 能够把原始图片映射为一个低维的紧凑的表征向量，此时 actor 和 critic 这2个训练任务，对于提取低维表征向量是有利的，从而让共享 encoder 在总体上看来是更加有效的选择。
- 但是进一步思考，actor 和 critic 的训练特性是完全不同的，这里以 PPO 的后续改进工作 PPG[15] (Phasic Policy Gradient) 为例进行介绍。在该论文中，首先着重分析了 policy 和 value 对于数据复用次数的不同特性，以 Procgen 这样一个经典的 benchmark 上进行实验的。策略函数对数据新旧程度和复用次数很敏感，尽可能每个数据只用一次，但是价值函数需要更高的数据多样性，可以通过复用数据来提升性能。
- 那如果既想通过共享 encoder 来加速训练，又希望利用 policy 和 value 在训练时的不同特性，究竟该如何设计呢？PPG 提出了一个综合这2点的算法，通过新加入一个额外的值网络，以及增加辅助训练阶段来实现上述的想法，完整的论文笔记请参考 [PPG 详解补充材料](#)。



(图29：左图：策略函数对数据新旧程度和复用次数很敏感，尽可能每个数据只用一次。右图：价值函数需要更高的数据多样性，可以通过复用数据来提升性能。来源于论文 [15]。)

3.5.2 代码：如何控制计算图中的梯度流动



(图30：左：PyTorch 的前向计算图与反向计算图的构建过程，图中每一步详细展示了计算图的展开和中间变量的记录过程。右：通过 numpy 代码手动实现的前向和反向计算公式。)

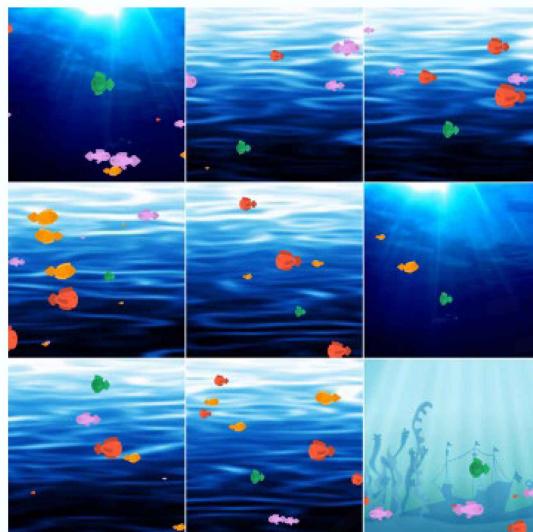
- PyTorch 的前向计算图与反向计算图的构建过程，可以参考 PyTorch 官方文档 [Autograd](#)。
- 完整的代码可以参考课程第三讲的相关[代码样例](#)和[算法注解](#)文档。

3.5.3 实践：使用 PPO 来解决大鱼吃小鱼问题

Procgen 环境集

- Procgen Benchmark[16] 是 OpenAI 发布的一组利用 16 种利用程序随机生成的环境 (CoinRun, StarPilot, CaveFlyer, Dodgeball, FruitBot, Chaser, Miner, Jumper, Leaper, Maze, BigFish, Heist, Climber, Plunder, Ninja 和 BossFight) 。 procgen 的全称是 Procedural Generation，表示程序化生成。对于 procgen 环境，它可以生成同一难度但是采用不同地图的游戏，也可以生成采用同一地图但是不同难度的游戏，可以用来衡量模型学习通用技能的速度，从而判断算法对于环境的泛化能力。具体参见 [DI-engine + Procgen Doc](#)。

Bigfish 环境

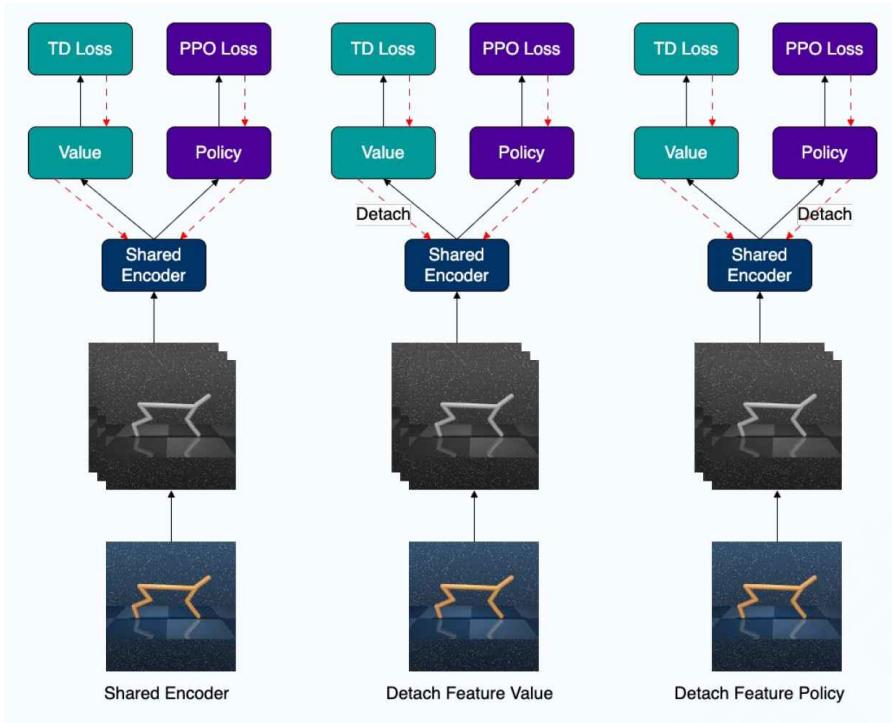


(图31：左图：Bigfish 环境示意图。玩家开始时是一条小鱼，通过吃掉其他鱼而变大。玩家只能吃比自己小的鱼，如果玩家接触到一条更大的鱼，玩家就会被吃掉并且这一局结束。玩家会因为吃到较小的鱼而获得小的奖励，奖励值的大小与鱼的大小成正比，时间耗尽或者玩家撞到了墙壁游戏也会结束。)

- 观察空间 (Observation)：观测为 RGB 三通道图片，具体尺寸为 $(64, 3, 3)$ ，数据类型为 `float32`。这些图像表示了游戏场景中的不同层次，包括背景、墙壁、水草和鱼群。观测值被标准化为范围 $[0,1]$ 之间的浮点数。
- 动作空间 (Action)：由一个整数表示，表示 agent 应该采取的行动。在 BigFish 中，动作空间的大小为6，代表了6个可能的方向：向左、向右、向上、向下、不动和吃鱼。
- 奖励空间 (Reward)：奖励根据 agent 的行为而变化。当 agent 吃到鱼时，它会获得一个正的奖励，奖励值的大小与鱼的大小成正比。(TODO: 如果 agent 撞到了墙壁或者没有吃到鱼，则它会受到一个负的奖励，奖励值为-1。如果 agent 没有做出任何动作，则它会受到一个微小的负奖励，奖励值为-0.1)。

Policy 和 Value 梯度对共享编码器的不同影响

实验设计

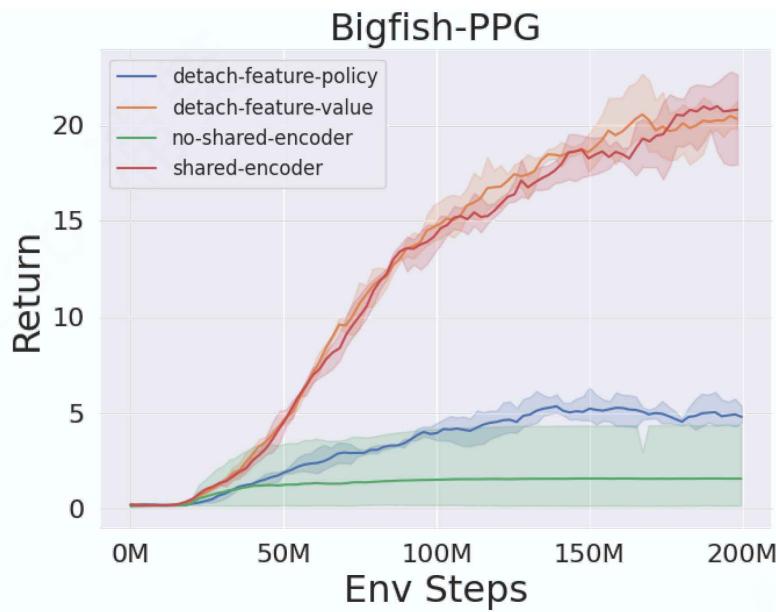


(图32：在共享编码器的情况下，Policy 和 Value 的不同梯度训练设计。左图：通常的共享编码器实验设计 (shared-encoder)，policy 和 value 的梯度都用于更新 Shared Encoder。中图：detach-feature-value，只是 policy 的梯度都用于更新 Shared Encoder。右图：detach-feature-policy 通常的共享编码器实验设计，只是 value 的梯度都用于更新 Shared Encoder。）

对于高维图片观察空间，Policy 和 Value 共享编码器是非常重要的，在训练前中期有利于帮助更快的提取高维图片的地维表征，但 Policy 和 Value 的梯度对于编码器具体有什么样的影响关系，二者是缺一不可，还是各有不同的作用？为此，参考图 3.5.3，设计了下面的4组对比实验：

- no-shared-encoder：通常的非共享编码器实验设计，即 policy 和 value 使用独立的 Encoder。
- shared-encoder：通常的共享编码器实验设计，即 policy 和 value 使用同一个 Encoder，而且 policy loss 和 value loss 的梯度都用于更新 Shared Encoder。
- detach-feature-value：policy 和 value 使用同一个 Encoder，但只是 policy loss 的梯度都用于更新 Shared Encoder。
- detach-feature-policy：policy 和 value 使用同一个 Encoder，但只是 value loss 的梯度都用于更新 Shared Encoder。

实验结果



(图33：Policy 和 Value 梯度对共享编码器的不同影响。detach-feature-policy, detach-feature-value, no-shared-encoder, shared-encoder 的学习曲线图。可以观察到，detach-feature-value 具有和 shared-encoder 类似的性能，而 detach-feature-policy 和 no-shared-encoder 的设置类似，学习缓慢几乎完全不能学到一个好的策略。)

- 这里在 Procgen 的 Bigfish 子环境上对比了以上4种设置 (detach-feature-policy, detach-feature-value, no-shared-encoder, shared-encoder) 的学习曲线图。可以观察到，detach-feature-value 具有和 shared-encoder 类似的性能，而 detach-feature-policy 和 no-shared-encoder 的设置类似，学习缓慢几乎完全不能学到一个好的策略。
- 关于 detach-feature-value 和 detach-feature-policy 性能差异巨大的原因，一个猜想是，由于 policy 和 value 的学习本质上是2个不同的任务，但是归根到底，学习到的 value 是为了 policy 的更新提升的，在 detach-feature-policy 的设定下，value loss 的梯度可以用于更新 shared-encoder，因此可以让 shared-encoder 学习到的表征让 value loss 降的很低，即让 value 预测的很准，但是如果没有匹配到当前的策略上，也是无法提升最终算法性能的。
- 最终训练收敛时获得的 PPO 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](#)：



(图34：最终训练收敛获得的 PPO 策略和随机初始化策略的行为对比视频。)

3.6 附录（可选阅读）

3.6.1 观察空间表征学习的其他方法

3.6.2 Actor 和 Critic 在表征学习方面的不同点：PPG

3.6.3 表征空间建模中的不变性与等变性

参考文献

- [1] Vinyals, Oriol, et al. "Grandmaster level in StarCraft II using multi-agent reinforcement learning." *Nature* 575.7782 (2019): 350-354.
- [2] Berner, Christopher, et al. "Dota 2 with large scale deep reinforcement learning." *arXiv preprint arXiv:1912.06680* (2019).
- [3] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9.8 (1997): 1735-1780.
- [4] Ashish Vaswani, et al. "Attention Is All You Need." *Advances in Neural Information Processing Systems 2017*.
- [5] Yang, Guan, et al. "Perfectdou: Dominating doudizhu with perfect information distillation." *Advances in Neural Information Processing Systems 35* (2022): 34954-34965.

- [6] Ming Zhang, et al. "GoBigger: A Scalable Platform for Cooperative-Competitive Multi-Agent Interactive Simulation" *International conference on learning representations*. 2023.
- [7] Chuming Li, Jie Liu, Yinmin Zhang, Yuhong Wei, Yazhe Niu, Yaodong Yang, Yu Liu, Wanli Ouyang. "ACE: Cooperative Multi-agent Q-learning with Bidirectional Action-Dependency", arXiv:2211.16068
- [8] <https://evolutiongym.github.io/>
- [9] <https://github.com/Kautenja/gym-super-mario-bros>
- [10] Deepmind. AlphaStar. <https://github.com/deepmind/alphastar>. 2022
- [11] Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5–9, 2015, Proceedings, Part III* 18. Springer International Publishing, 2015.
- [12] Mescheder, Lars, et al. "Occupancy networks: Learning 3d reconstruction in function space." *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019.
- [13] Li, Xiang, et al. "Does self-supervised learning really improve reinforcement learning from pixels?." *Advances in Neural Information Processing Systems* 35 (2022): 30865-30881
- [14] Andrychowicz, Marcin, et al. "What matters for on-policy deep actor-critic methods? a large-scale study." *International conference on learning representations*. 2021.
- [15] Cobbe, Karl W., et al. "Phasic policy gradient." *International Conference on Machine Learning*. PMLR, 2021.
- [16] Cobbe, Karl, et al. "Leveraging procedural generation to benchmark reinforcement learning." *International conference on machine learning*. PMLR, 2020.