

PPO × Family 第一讲文字稿

注：文字稿主要是对课程 PPT 内容的补充和细粒度讲解

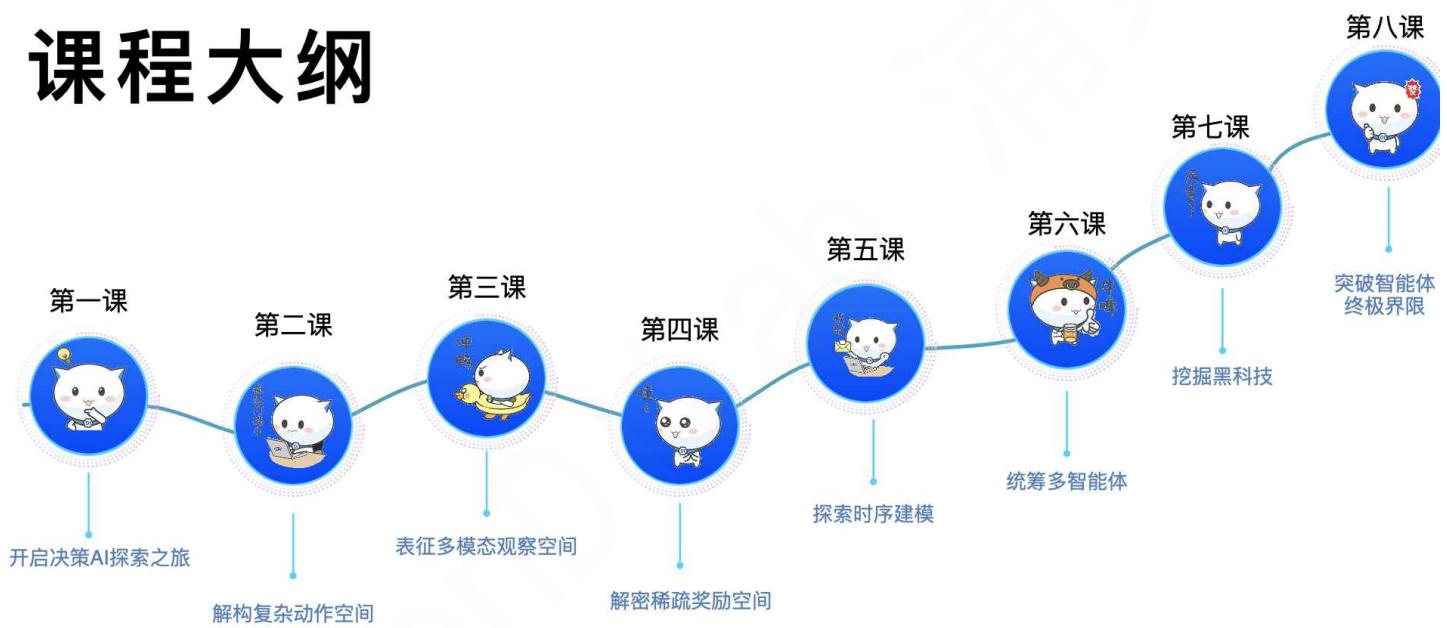
1.0 课程简介

课程目标

- 从应用出发，自底向上地，真正用一个 PPO 算法解决各种各样的决策 AI 问题

课程大纲

课程大纲



八节线上公开课，每节课40-45分钟

- 深度强化学习简介 + 原生 PPO 剖析（第1课时）
- PPO × Family 各个子专题的技术和应用（第2-7课时）
- PPO × Family 的集大成之作（第8课时）

考核机制

★完成每节课后小作业即可获得"通过课程学习"认证证书

关于学习证书

★完成实训大作业，成绩突出者可获得"优秀"认证证书

答疑机制

大家可以在哪里提出疑问

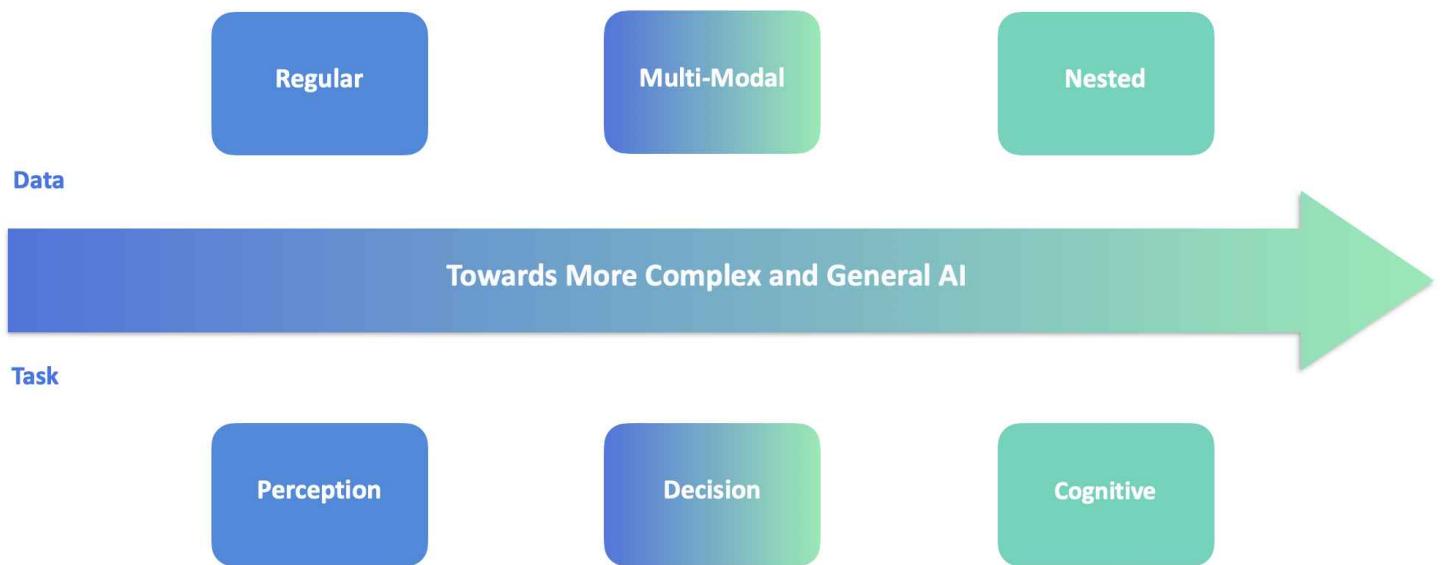
★在课程Github Issues专区提出问题，会有技术小哥回复哦～

★加入官方课程Slack频道，和我们一起讨论

★填写DI小助手发布的答疑问卷，我们会根据大家提出的问题组织讨论会，欢迎大家参加！

- 小助手微信号：OpenDILab
- GitHub：<https://github.com/opendilab/PPOxFamily>
- Slack：https://join.slack.com/t/opendilab/shared_invite/zt-v9tmv4fp-nUBAQEH1_Kuyu_q4plBssQ

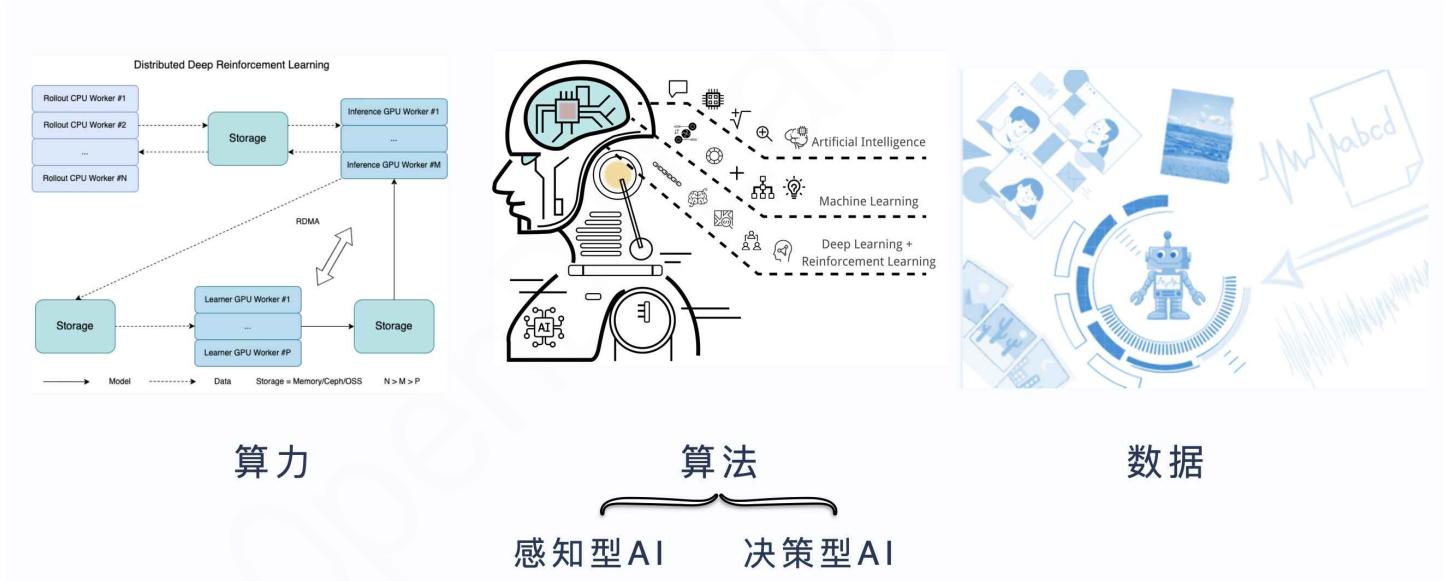
1.1 引言：从感知 AI 到决策 AI



1.1.1 决策 AI 的特点

人工智能（AI）被视作是产生新一代技术革命的重要爆发点之一，近十年来，AI 在人脸识别，语音助手、自动驾驶等领域已经取得了媲美甚至超过人类的成果。

一般来说，人工智能的成功离不开下面的三个要素：**算力**，**算法**和**数据**。而其中的算法部分，又可以**感知型 AI 和决策型 AI** 两大类。



那究竟感知型和决策型AI的关系和分界线是什么呢，这里通过一个例子来了解：

感知

语言/语音/图像



<https://openai.com/dall-e-2/>



<https://github.com/opendilab/DI-star>

规划/推理

决策

- **感知型AI：**更多强调对于多模态（语言/语言/图像）数据的表征和建模。例如OpenAI发布的文图生成算法DALL·E 2 [1]，让AI拥有根据一段话生成对应图像内容的能力。
- **决策型AI：**基于感知型AI的分析和建模结果，更多关注复杂的推理决策，长期规划，多智能体博弈等问题。例如OpenDILab开源的星际争霸2智能体DI-star，在最复杂的即时战略游戏中让AI达到媲美人类职业玩家的水准。

当然，感知和决策是相辅相成的关系，要想让人工智能真正应用到实际产业中，两者缺一不可。接下来，这里通过游戏AI和自动驾驶中的两个例子来展示二者如何协作：

第一个例子是OpenAI打造的DOTA2决策智能体，需要从原始游戏中提取到小地图，物品栏，技能栏等多种模态信息，然后整合这些信息做出一系列决策行为。

感知+决策



感知：多种模态信息的提取和融合



决策：选择目标单位



决策：选择目标位置

第二个例子是仿真器中的自动驾驶问题，智能体需要处理多种传感器和视角传来的数据，感知交通标志，行人和其他车辆等等，并控制做出加减速，打方向变道等动作，保持平稳高效的驾驶。

感知+决策



感知：交通标志、路障和车道线的检测



决策：控制车距



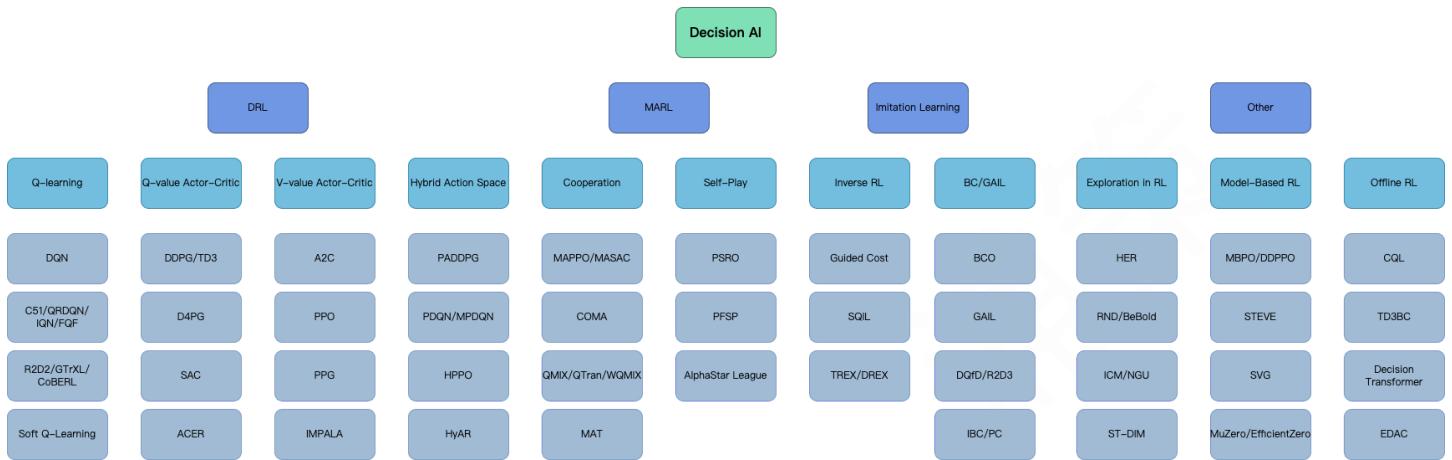
决策：车辆变道

1.1.2 如何设计决策 AI

过去10年，感知AI已经让机器具备了从“看清”到“看懂”的能力，而决策AI将进一步推动人工智能向推理、决策、规划等方向发展，在未来10年将为自动驾驶、智慧城市等领域带来颠覆性创新。然

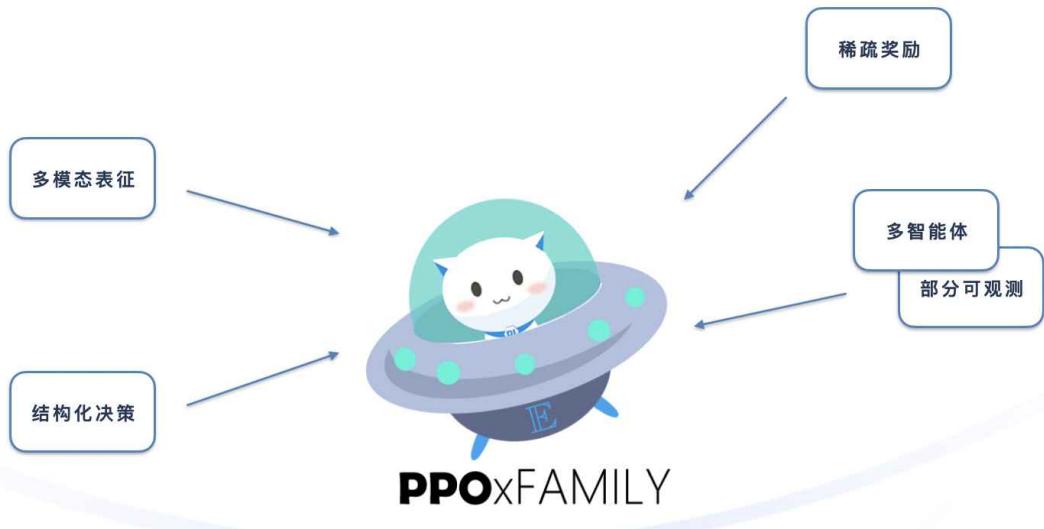
而，相较于感知识别任务，决策类问题因涉及**多模态数据空间、跨尺度计算逻辑、多领域算法融合**等问题，所以标准化难度高，一直没有通用的解决方案。

而在现今主流的各类决策 AI 算法技术中，深度强化学习（Deep Reinforcement Learning, DRL）[\[2\]](#)是重中之重。现今的深度强化学习技术可谓是百花齐放，为了解决不同类型的决策问题，衍生出了一系列风格迥异的相关算法。例如，解决机器人控制问题，往往需要结合模仿学习（GAIL [\[3\]](#) / BCO [\[4\]](#)）和连续控制强化学习算法（DDPG [\[5\]](#) / SAC [\[6\]](#)），提高数据利用效率。在多智能体领域，为了更好的处理智能体之间的合作和竞争，有需要像 QMIX [\[7\]](#) 和 PSRO [\[8\]](#) 这样的新算法加持。每一个细分的研究领域都会衍生出大量的研究课题，这也正说明了决策 AI 设计的复杂性。



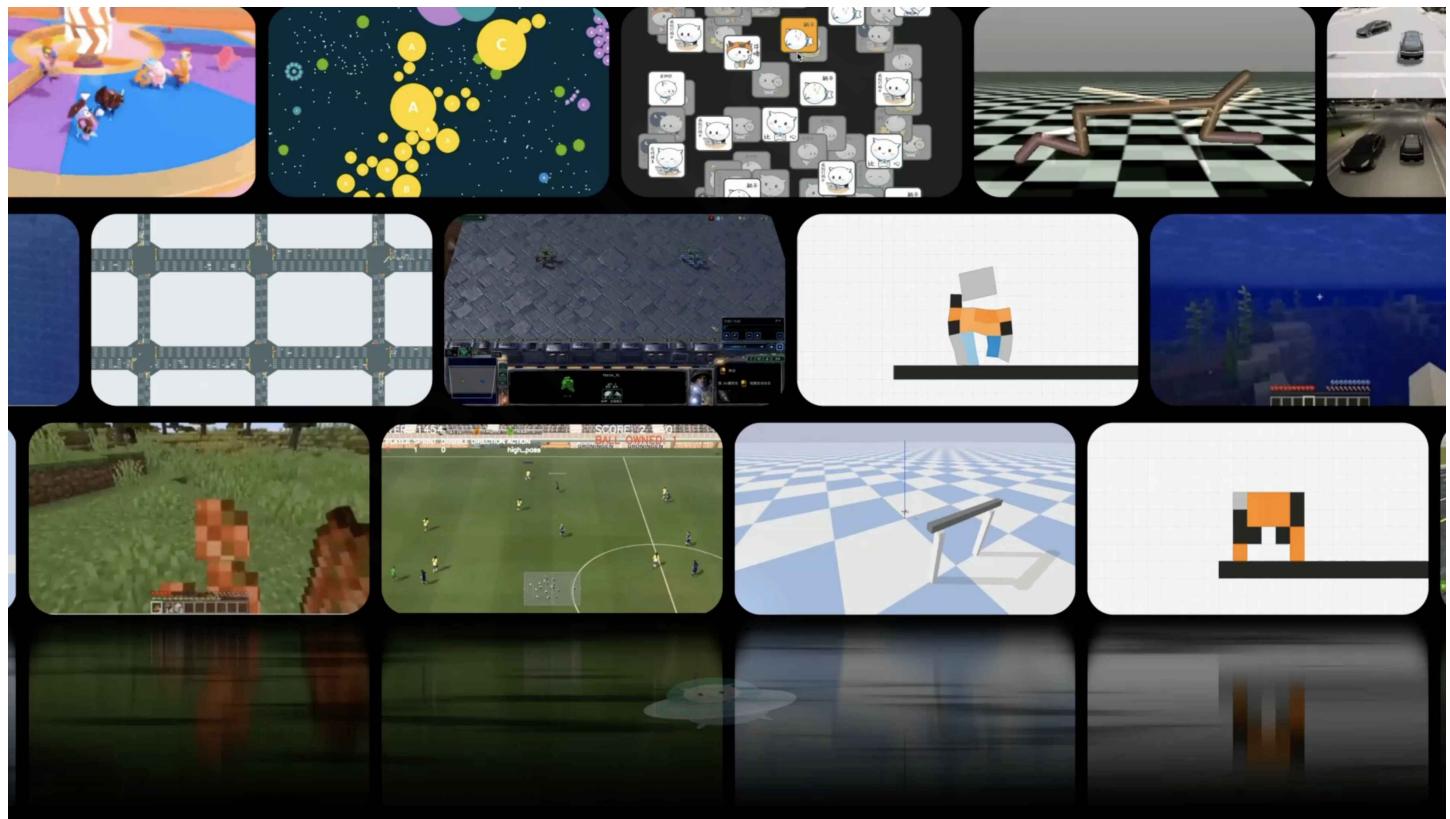
虽然从图中看起来决策 AI 相关算法研究的发展前景十分广阔，但如此庞大的算法族也给技术的进一步传播和应用带来了挑战，想要入门，想要运用深度强化学习技术，需要付出大量的学习成本。而对真正大多数想要了解深度强化学习技术的人，**深度强化学习领域的算法多而庞杂，新手一时难以分辨现阶段到底应该学习什么算法**。而 OpenDILab 希望能总结整合出一种最通用强大的算法，融会贯通之后便可以灵活应用于各种所需场景。由此，OpenDILab 从应用出发，推出了 PPO × Family 系列课程。现今已有的经典的强化学习课程，主要是自顶向下地去介绍强化学习算法发展的方方面面。而 PPO × Family 系列课程则选择自底向上的视角，希望运用一个（**Proximal Policy Optimization**）PPO 算法[\[9\]](#) 解决几乎所有常见的决策智能应用，帮助一切对于深度强化学习技术有好奇心的人，轻便且高效地制作应用原型，了解和学习最强大最易用的 PPO × Family。

注：课程 GitHub 传送门：<https://github.com/opendilab/PPOxFamily>



集中一点，登峰造极 深入理解一种算法 PPO，灵活应对各种决策场景

所谓天下决策，唯“PPO”不破，究竟 PPO × Family 课程能解决怎样的决策问题，一起来看下面的 PPO × Family 决策智能应用混剪视频：



(PPO x Family 决策智能应用混剪视频)

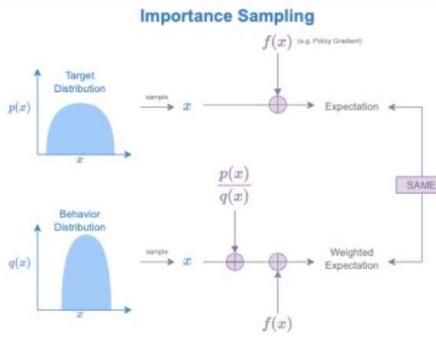
那如何掌握这样强大且通用的算法技术呢，这里就要搬出上手决策 AI 的核心公式：

上手AI = 盘清算法理论 + 理顺代码逻辑 + 玩转应用实践

课程的每一个子专题，会从具体的决策 AI 应用入手，深入探讨要解决这类问题需要的算法理论知识和代码实现原理，做到“算法-代码-应用”一一对应，让学习这门课程的技术爱好者，可以知其所以然（了解算法），知其然（了解代码实现），并能学以致用（将所学知识运用到具体的决策智能实际问题中去），一些可以公开的 demo 效果如下：

- 算法，代码，应用三合一

PPO x Family

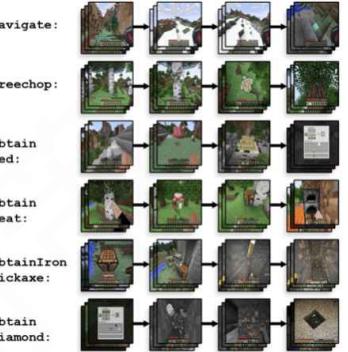


```
def ppo_policy_error(data: namedtuple,
                     clip_ratio: float = 0.2,
                     dual_clip: Optional[Tuple] = None) -> Tuple[namedtuple, namedtuple]:
    logit_new, logit_old, action, adv, weight = data
    if weight is None:
        weight = torch.ones_like(adv)
    dist_new = torch.distributions.categorical.Categorical(logits=logit_new)
    dist_old = torch.distributions.categorical.Categorical(logits=logit_old)
    logp_new = dist_new.log_prob(action)
    logp_old = dist_old.log_prob(action)
    dist_new_entropy = dist_new.entropy()
    if dist_new_entropy.shape != weight.shape:
        dist_new_entropy = dist_new_entropy.unsqueeze(-1).expand_as(weight)
    entropy_loss = (dist_new_entropy + weight).mean()
    logp_ratio = (logp_new - logp_old) / weight
    # policy loss
    ratio = torch.exp(logp_new - logp_old)
    if ratio.dim() == adv.dim():
        ratio = ratio.unsqueeze(dim=1)
    surr1 = ratio * adv
    surr2 = ratio.clamp(1 - clip_ratio, 1 + clip_ratio) * adv
    if dual_clip is not None:
        clip1 = torch.min(surr1, surr2)
        clip2 = torch.max(clip1, dual_clip * adv)
        # only use dual_clip when adv < 0
        policy_loss = -(torch.where(adv < 0, clip2, clip1) * weight).mean()
    else:
        policy_loss = (1 - torch.where(adv < 0, surr1, surr2) * weight).mean()
    with torch.no_grad():
        approx_kl = (logp_old - logp_new).mean().item()
        clipped = ratio.gt(1 - clip_ratio) | ratio.lt(1 + clip_ratio)
        clipfrac = torch.sum(clipped) / torch.sum(~clipped).item()
    return approx_kl, policy_loss, entropy_loss, clipfrac
```

盘清算法理论

理顺代码逻辑

玩转应用实践



- 算法与代码一一对应

Importance sampling weight:

$$r(\theta) = \frac{\pi_{new}(a|s)}{\pi_{old}(a|s)}$$

Original surrogate objective:

$$r(\theta) A^{\pi_{old}}(s, a)$$

Clipped surrogate objective:

$$\text{clip}(r(\theta), 1 - \epsilon, 1 + \epsilon) A^{\pi_{old}}(s, a)$$

Dual clip proposed by <https://arxiv.org/abs/1912.09729>.

Only use dual_clip when adv < 0.

```
22     ratio = torch.exp(logp_new - logp_old)
```

```
23     surr1 = ratio * adv
```

```
24     surr2 = ratio.clamp(1 - clip_ratio, 1 + clip_ratio) * adv
```

```
25     if dual_clip is not None:
26         clip1 = torch.min(surr1, surr2)
27         clip2 = torch.max(clip1, dual_clip * adv)
28         policy_loss = -(torch.where(adv < 0, clip2, clip1) * weight).mean()
```

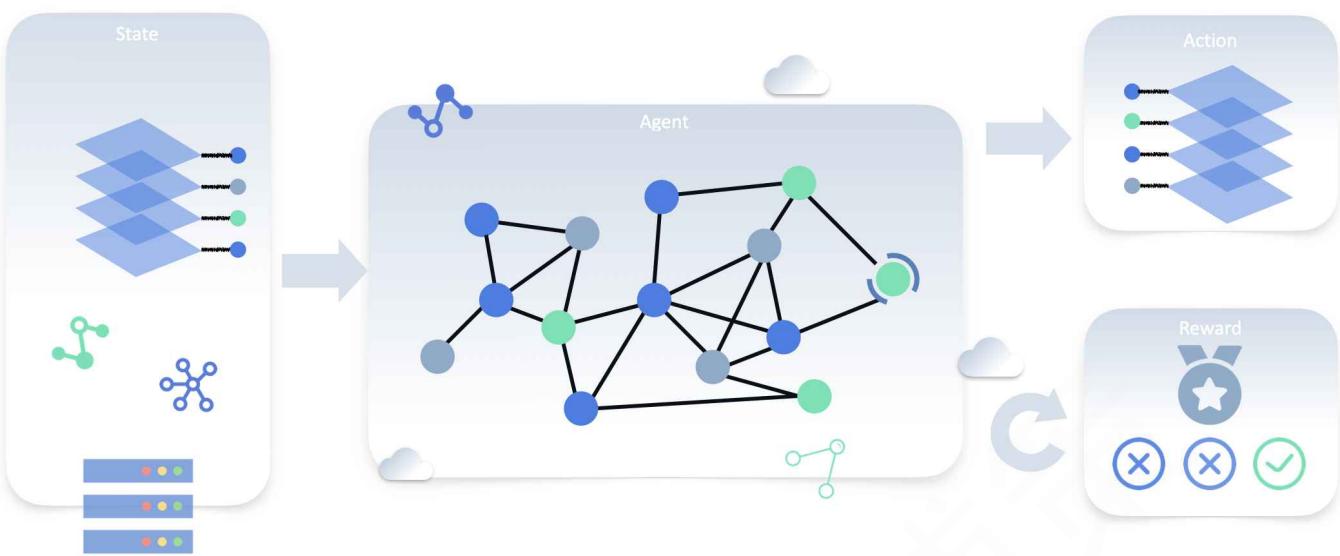
PPO-Clipped Loss:

$$\min(r(\theta) A^{\pi_{old}}(s, a), \text{clip}(r(\theta), 1 - \epsilon, 1 + \epsilon) A^{\pi_{old}}(s, a))$$

```
29     else:
30         policy_loss = (-torch.min(surr1, surr2) * weight).mean()
```

Multiply sample-wise weight and reduce mean in batch dimension.

1.2 为什么用深度强化学习来解决决策问题

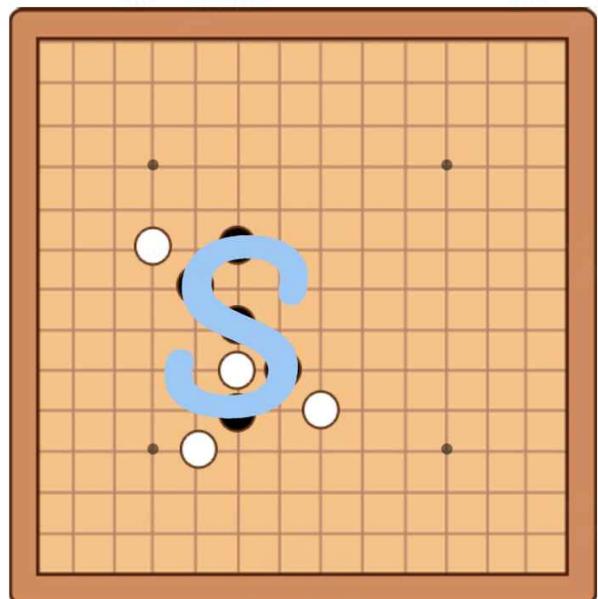
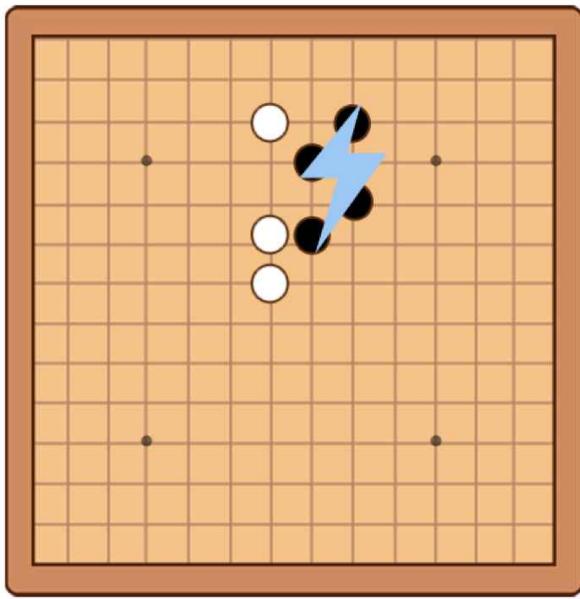


1.2.1 搜索最优解的不同方式

决策可以看做是一种**搜索最优解的智能行为**，即在某种情况下，做出能获得长期最优的选择。那么如何选择最优解呢，这里可以来对比思考下人寻找最优解的方法。其中主要有“从模仿中学习”和“从试错中学习”两大类方法，他们各自又有相应的特点。



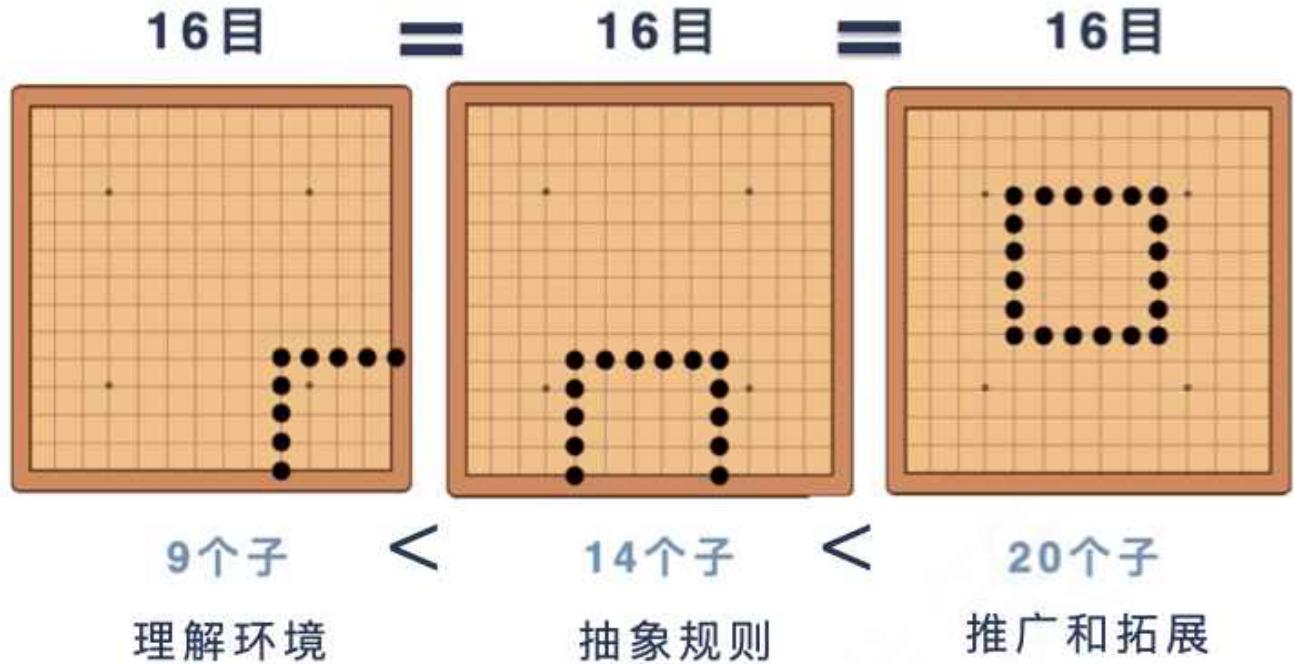
- **从模仿中学习：**一个很经典的例子就是人类学习下五子棋和围棋，在这一轮 AI 风潮兴起之前，很多人都是通过棋谱学棋，模仿其中的行为，逐渐掌握更多样的方式和方法，这种方法的**优点是简洁又直观**，具体例子比如下图中五子棋的各种所谓“必胜”阵法。



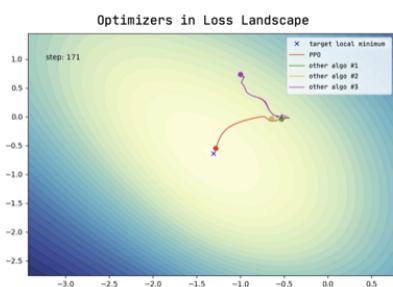
但模仿有一个天然的劣势，需要预先准备好大量“情况和选择”成对的例子（**数据要求高**），在棋类游戏中有人类几千年累积下来的棋谱和套路，但是这在很多决策问题中是几乎不可能的。而且，模仿不能是机械式的“鹦鹉学舌”，否则对于例子之外的情况将毫无应对能力（**可迁移性差**），比如从 19x19 围棋换到 9x9 围棋，从五子棋到四子棋，相应的最优套路也会发生变化。

- 从试错中学习：除了模仿，另一个重要的角度就是从试错中学习，对于未知的事情，人会选择适当地探索新事物，根据所得到的反馈信息来决定是继续还是退出。还是以棋为例，与自己水平相近的对手博弈往往能获得显著的棋力提升，**通过在不断的尝试中总结和创造相应的下棋策略，逐渐丰富对于最优解的认知和判断，不断提升与强化**。这个过程可以进一步分解为：

- 理解环境：即理解环境的定义和规则，比如五子棋五子相连为胜，围棋中计算目的逻辑。
- 抽象规则：基于对环境的理解，抽象出一些高级语义的规则，比如围棋中的“三三”和“打吃”。
- 推广：对于更复杂的问题，基于环境信息和规则的组合，提炼出更加宏观的战术和策略，比如围棋中经典的“金角银边草肚皮”，就是从“围空的效率”出发分析得到的更高阶智能逻辑。



那如果用设计人工智能来从试错中学习，该如何着手呢。一个最简单的方法就是“暴力搜索”，列举所有可能出现的下棋方法，找出里面获胜的情形，但这在例如 19×19 围棋这样恐怖的搜索空间复杂度面前根本不现实。进而人们想出了各种更高效的搜索方法，通过设计“记忆化搜索”和各种各样的剪枝方法，一定程度上缩小了决策空间，但想要获得好的结果仍然需要搜索很长的路径，这仍然会带来庞大的计算代价，所以，**搜索效率**是设计算法的重中之重。



搜索效率



随机性



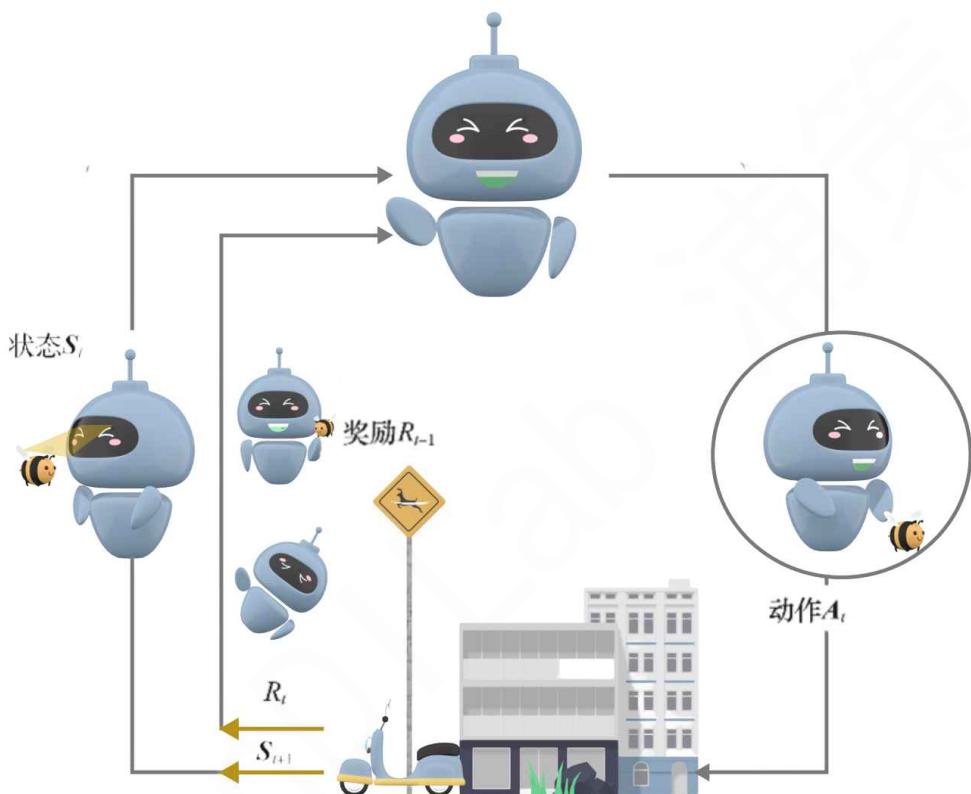
非完全信息

同时，一个重要的事实是，棋类游戏其实是拥有很多良好性质的决策问题，双方对弈的信息全都是已知的，但其他类型的决策问题可能有诸多难以处理的性质。例如经典的 2048 游戏，每一步新的方块“2”和“4”都是在场上空余位置随机生成的，这种**随机性**对于一些经典搜索算法就会造成很大负面影响。又比如另一种经典游戏——麻将，就需要去考虑**非完全信息博弈**的问题，思考牌面下的暗流涌动，如果换成更动态的决策环境（不像棋牌类有一成不变的环境规则），那么智能体的每一个行为都有可能引发环境本质的变化，这也大大加深了决策的复杂性和不确定性。

为了解决上述决策方法的局限性，就需要一种**更强大更通用更稳定的**搜索方法，在浩瀚无垠的决策空间找到更高效的最优解，于是，**深度强化学习**就呼之欲出。

1.2.2 用强化学习来寻找最优解

本章节将会先明晰强化学习（Reinforcement Learning, RL）的基本设定，首先，需要将原始的决策问题**抽象和标准化为强化学习环境**。而 AI 智能体的目标就是找到这个环境中的最优解。具体操作中，智能体（Agent）通过和问题环境（Env）的交互来不断进行在线学习，不断强化自身，最终智能体持续进化直到找到这一环境下的决策任务的最优解。具体来说，每个时刻，智能体从环境中得到观察信息（状态 S ），智能体通过一系列思考给出决策（动作 A ），这个决策行为被传递给环境，环境给以反馈信息（即奖励 R ）和下一帧的观察信息，如此循环不断往复。



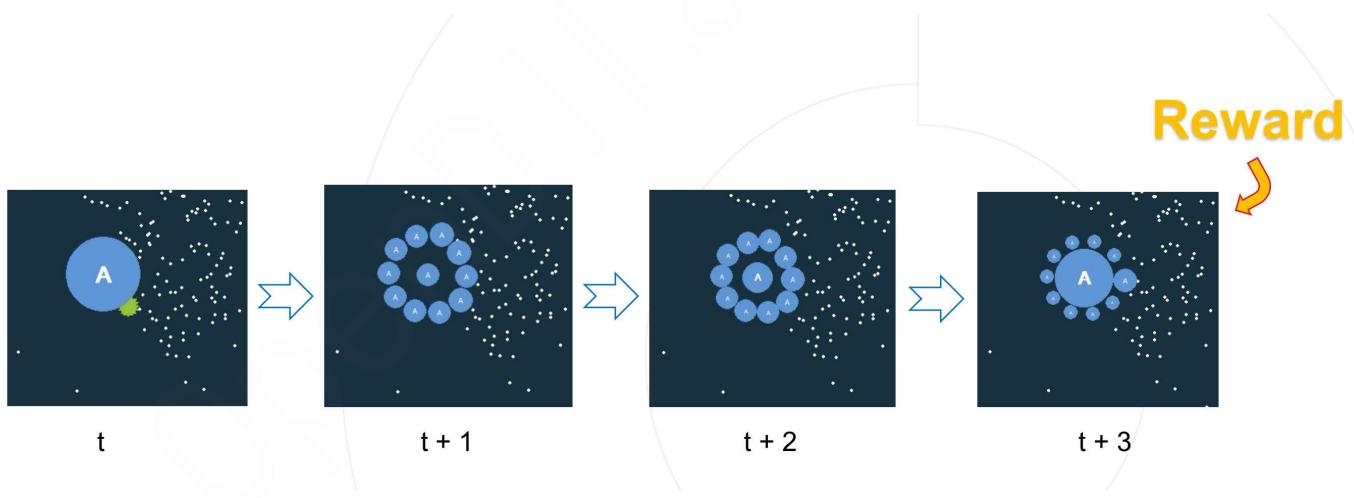
那么强化学习做决策具体有什么特点呢，这里通过三个对比来剖析：

- 对比传统搜索方法，可以**建模环境的未知性和不确定性，自主学习到更抽象的搜索策略**。环境未知性即非完全信息下的决策，例如斗地主中不知道对方的牌面。而环境不确定性可表现为状态和奖励的各种不确定性，例如随机生成的游戏道具和关卡等等。



(强化学习对棋类环境中决策行为的预测示意图)

- 对比监督学习，需要从延迟性的，间接的奖励中进行学习。这里的延迟性是指当前决策行为的奖励可能时间上滞后很久才能获得，例如，前人栽树，后人乘凉。而间接是相对于监督学习来讲，前者是直接通过标签来监督，而强化学习使用的奖励信息是间接的。



(GoBigger 环境高级操作“分裂->移动->中吐”示意图)

换句话说，**监督学习是固定标签数据，希望最小化损失函数的值。强化学习是固定回报函数，希望找到能最大化回报的数据轨迹对应的策略**。如果有在某个状态下执行动作的标签，就会变成很接近监督学习的形式，但是通常情况下都是只有更间接的奖励信息。另外，在部分环境中，奖励会延迟性的，即智能体需要完成一连串的长序列决策之后，才能获得奖励，在最后一步完成之前都没有相应的奖励信号。例如上图中展示的 GoBigger 决策环境，这个游戏很简单，规则就是大球吃小球，但是小球拥有更快的移动速度，因此智能体常常需要执行“分裂->移动->中吐”的连续操作，先分裂成多个小球加速追赶敌人，然后中吐合成一个大球消灭对手。但是这样一系列操作只有在执行完

成且成功吃掉对手球才有奖励，即是一种延迟性的奖励，所以这种动作序列对于强化学习智能体来说一般很难学习到。

- 对比离线学习，需要**平衡探索和利用**，探索（Exploration）是指找到新的有价值的数据和信息，例如探索迷宫中的未知部分，而利用（Exploitation）则是根据已有数据挖掘出知识内涵，例如对迷宫中已知部分进行分析。



另外，强化学习还往往需要处理**非独立同分布**（常包含时序信息）的数据

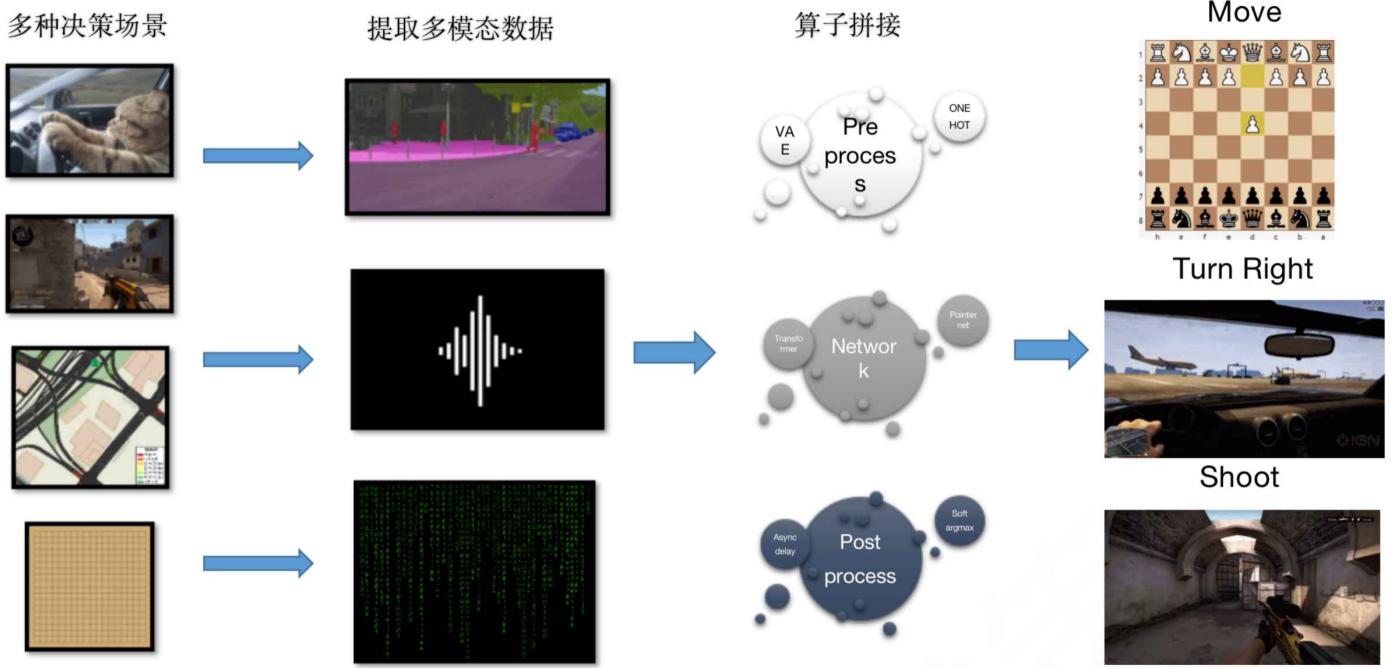
注：关于非独立同分布数据的问题定义可以参考相关[链接](#)

一言以蔽之，强化学习就是一种研究如何从试错中学习的优化方法，核心关键点在于利用和探索，即如何利用过去的交互经验学习到优秀的策略（奖励有收益的行为，惩罚带来危害的行为），如何探索未来新的可能性，两个过程不断交织反复，让策略不断进化升级。

1.2.3 用深度学习来建模各类非线性单元

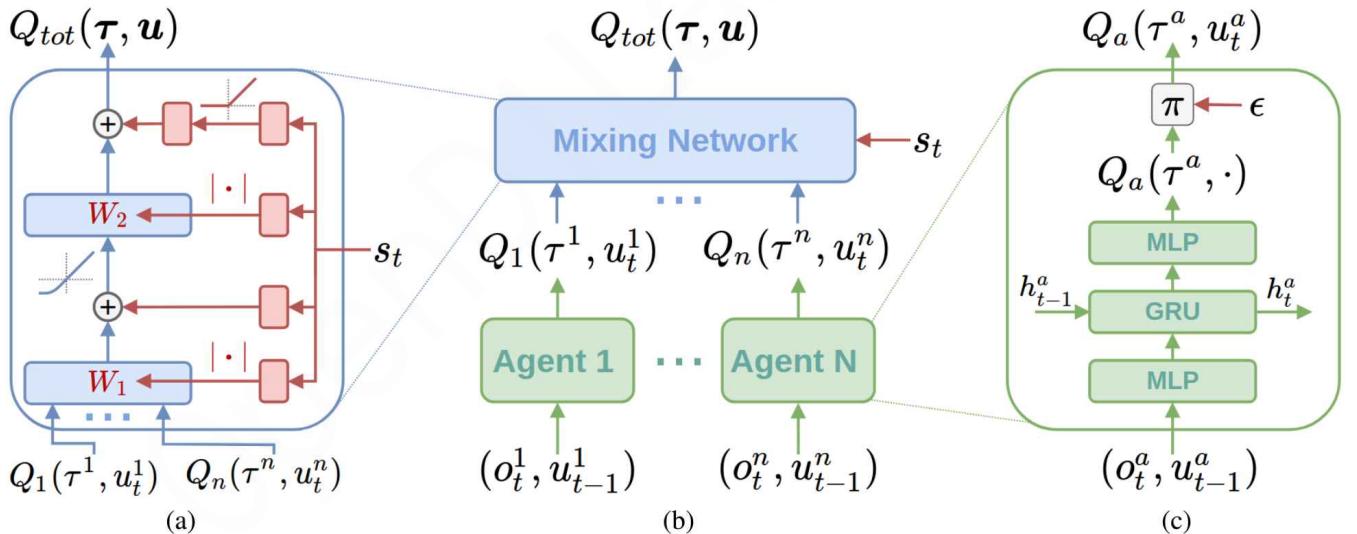
其实强化学习经典理论在几十年前就已经有很多成果，但为什么直到近十年来才在各类实际问题中广为应用呢，其中的关键因素就是深度学习的兴起。很多以前传统强化学习处理不了的复杂问题，在深度学习兴起后，借助深度学习强大的表征建模能力支持下就可以更高效地找到最优解。更具体地，通过对整个深度强化学习领域的分析，可以总结出，在整个算法的设计中，主要有两类使用深度学习的场景：

- 一是处理多种决策场景的复杂输入和输出（即多模态观察空间和动作空间问题）



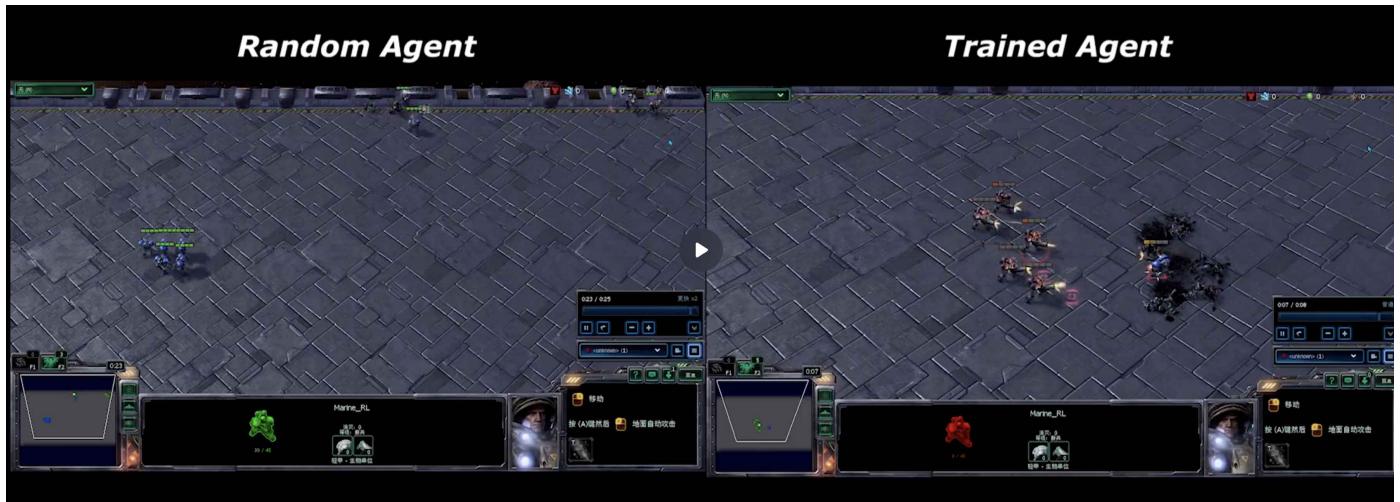
(多种复杂决策场景的输入输出空间示意图)

- 二是建模强化学习算法中独有的一些算法概念，最经典的就是用过参数化的神经网络来充当价值函数和策略函数。又或者是，在多智能体强化学习中，用深度神经网络来建模智能体之间的协作关系 QMIX [7]，即用神经网络表征每个智能体在团队协作中的贡献比例：



(QMIX 算法神经网络设计架构图)

注：关于多智能体协作智能体的演示视频可以参考 [Agent Demo List](#) 中的 SMAC 5m VS 6m

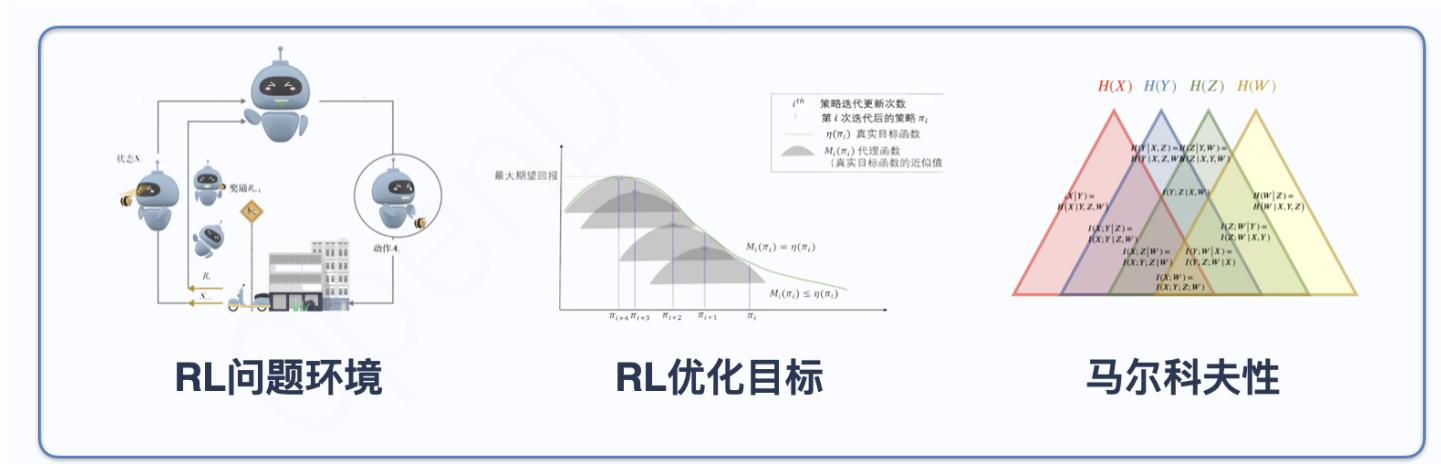


要在强化学习中用好神经网络，则需要明确两个核心目标：

- 神经网络的输入和输出，训练神经网络的目标（损失函数）是什么，这决定了使用神经网络对决策问题效果的**上限**。
- 神经网络的结构设计和优化方法，这决定了使用神经网络对问题效果的**下限**。

1.2.4 标准化研究问题的形式

为了**系统性地分析**要解决的决策问题和使用的算法，**量化且标准化**所研究的问题，本章节开始引入一些强化学习的基本概念和数学定义，更具体地，设计决策AI需要将原本的决策问题（例如自动驾驶，多智能体博弈）转化为一个马尔科夫决策过程（Markov Decision Process, MDP），强化学习算法的方法设计都是以MDP为基础的，相关的核心定义主要分为下面三部分：



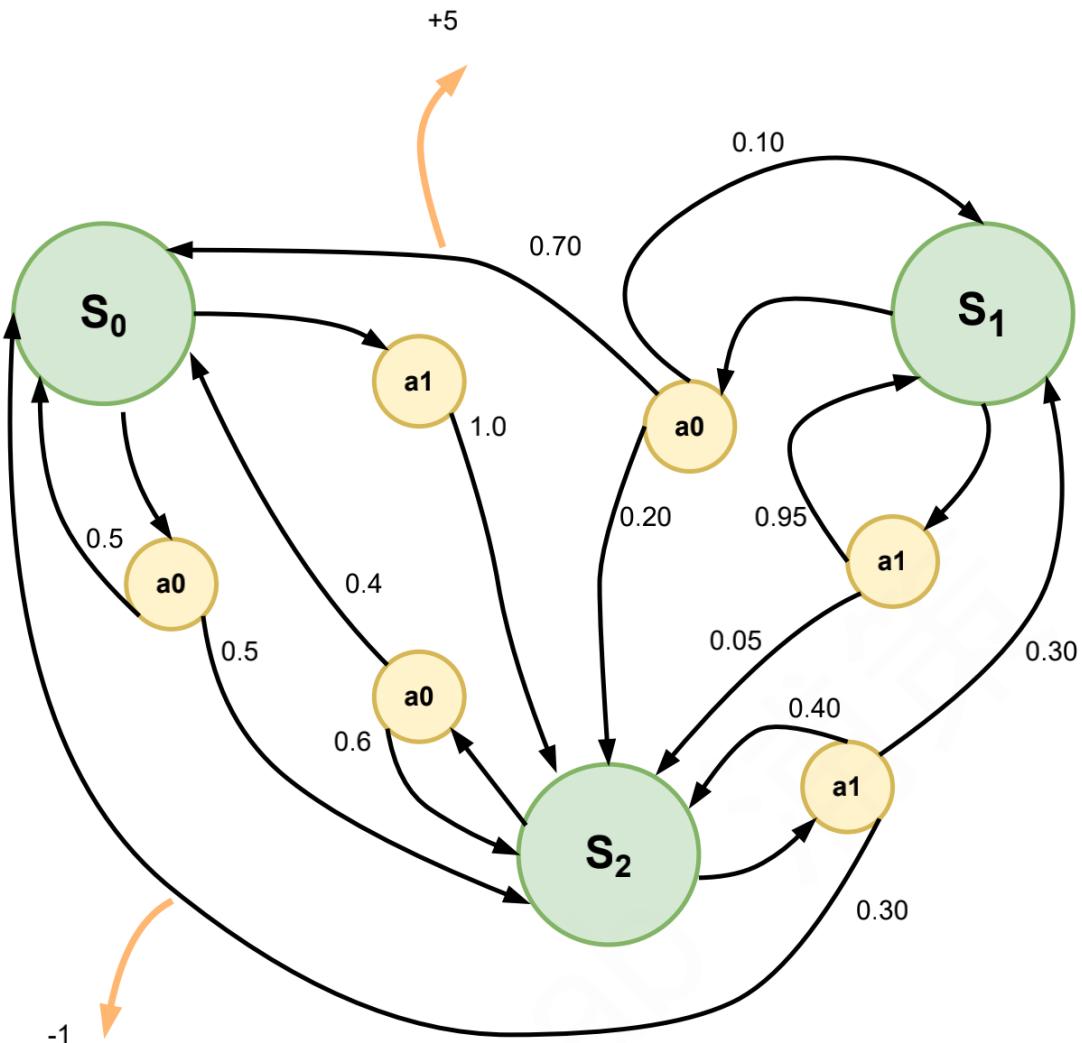
- **强化学习的问题定义（环境）**：一般来说，常常用五个元素（五元组） $\langle S, A, P, R, \rho_0 \rangle$ 来描述环境（下文括号中是自动驾驶场景中对应的例子），用带下标时间的小写字母 s_t, a_t, r_t 来指代某一时刻 t 的状态、动作、奖励值。
 - 状态的集合 S （路况，车速）
 - 动作的集合 A （油门，刹车，方向盘）
 - 在状态之间转移的规则，即转移概率 $P(s_{t+1}|s_t, a_t)$ ，其中 $s_t \in S, a_t \in A$ ， t 表示一个episode内的一步（转移概率的例子：行驶中急刹车让车停下来）

- 状态转移后，获得”即时奖励“的规则 R ，即 $r_t = R(s_t, a_t, s_{t+1})$ （比如人为设定行驶到目标地点获得高额奖励）
- 初始状态分布 ρ_0 ，即 $s_0 \sim \rho_0$ （比如驾驶任务的出发点情况）
- **强化学习的优化目标：**而基于上文的环境定义，强化学习的优化目标可以描述为，寻找可以最大化回报 G_t 的策略 π ，这之中涉及的关键概念有：
 - 智能体策略 π ，如果是确定性策略则为函数 $a_t = \pi(s_t)$ ，如果是随机性策略则为条件概率分布 $\pi(a_t | s_t)$ ，如果将表示策略的参数记为 θ ，策略通常又写作 π_θ
 - 轨迹 τ ，策略 π 和环境进行交互，产生的状态动作序列（按照交互的时间先后顺序），假设有 T 步决策（ T 由环境本身决定），即 $\tau = (s_0, a_0, s_1, a_1, \dots)$
 - episode，一般称智能体和环境交互的一段完整轨迹为一个 episode，例如围棋中的一局
 - 回报 G_t ，一般情况下，强化学习希望能够最大化整个 episode 能够获得的总奖励。并且，一般常使用折扣因子 $\gamma \in (0, 1)$ 来平衡短期和长期的奖励，于是，可以得到回报的定义为
$$G_t(\tau) = \sum_{k=0}^{T-1} \gamma^k r_{t+k}$$
- 那么强化学习就可以定义为如下的优化问题，寻找可以最大化期望回报的策略：

$$\operatorname{argmax}_{\pi_\theta} \mathbb{E}_{\tau \sim \pi_\theta} [G_t(\tau)]$$

- **马尔科夫性：**那么为什么要称这个定义为马尔科夫决策过程呢，就是因为它满足马尔科夫性质，即 **下一个状态只取决于当前状态，而不会受到过去状态的影响**，用数学公式表达则为 $P(s_{t+1} | s_t, s_{t-1}, \dots, s_0) = P(s_{t+1} | s_t)$ ，那么就可以仅用当前的状态信息决定未来，而不需要考虑过去的历史信息了，这会大大简化问题的求解过程。当然，如果是不满足马尔科夫性质的决策环境，其实还有很多扩展的 MDP 定义及解决方法，这将在后续课程中一一展开。

一个简单的 MDP 例子如下图所示，有三种状态和两种动作，例如在状态 S_1 下，可以执行 a_0 和 a_1 两种动作，其中执行 a_0 动作后，会有 10% 的概率转移回 S_1 ，20% 的概率转移到 S_2 ，70% 的概率转移到 S_0 ，且只有转移到 S_0 时会获得 +5 的奖励，其他情况下奖励都为 0：



(简易马尔科夫决策过程 MDP 实例示意图)

后续强化学习的算法设计，主要就是寻找合适的方法，来更稳定，更高效地寻找上述优化问题的最优解，从而得到更加强大的决策 AI。

Note：更详细的解释可以参考：

- 符号表：[传送门](#)
- OpenDILab 的强化学习基础概念入门文档：[传送门](#)

1.3 为什么选择 PPO

注：这一部分课程内容，读者可以结合 OpenDILab 制作的[微课短视频](#)帮助理解

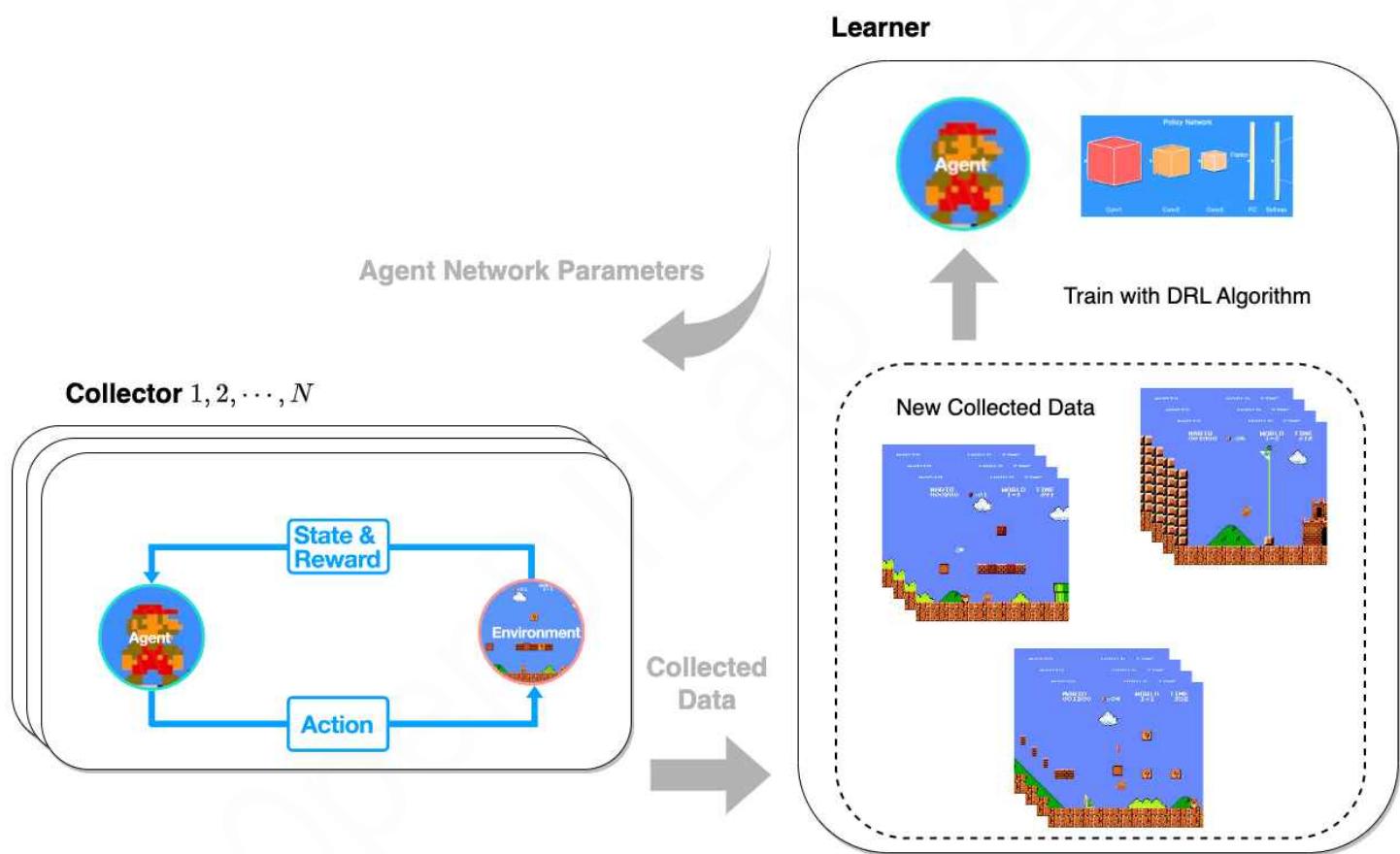
在本章节中，会详细讲述为什么选择 PPO（Proximal Policy Optimization）算法来构建最强大最通用的决策 AI 工具。首先，本文将从 PPO 算法的祖师爷——策略梯度（Policy Gradient）算法 [10] 开始，理解这类方法的算法原理和具体流程，然后从策略梯度的各种后续衍生工作展开，最终扩展到终极形态——PPO。

1.3.1 深入浅出策略梯度

在这一部分，将会类比“考取驾照”的过程来揭开策略梯度方法的神秘面纱。具体来说，想要训练得到强大的决策智能体，需要不断循环地执行下面三个核心步骤：**收集数据、设计目标、优化策略**。

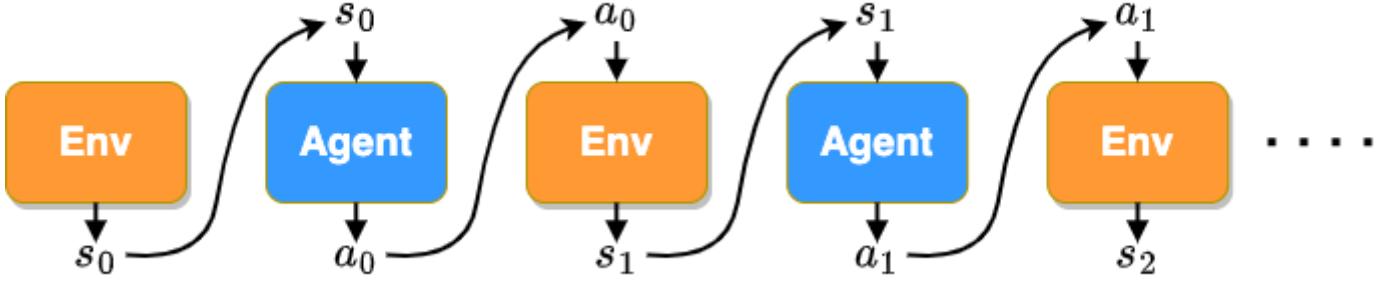
收集数据

首先，现今的深度强化学习算法（包括策略梯度）本质上都是数据驱动的学习方法，需要从数据中去挖掘信息、获取知识。就像考驾照的科目一环节，考生通常需要经历海量的刷题过程，正确理解并牢记题库中的各种交通规则和驾驶习惯。这个题库就是引导人学习驾驶技能的“数据”，只不过题库是被前人预先收集准备好的，对于强化学习智能体，也会有类似的环节，而且，面对不同的决策问题，智能体还需要和决策问题环境进行在线交互，去收集最新的数据。



(强化学习的训练pipeline示意图：数据收集器（Collector） + 学习器（Learner）)

如上图中的超级马里奥游戏，智能体就需要同时并行很多组数据收集器（Collector），每个收集器中智能体和环境交互产生轨迹数据，再把这些收集到的数据传到学习器（Learner），使用相应的深度强化学习算法进行训练，训练完成之后将最新更新的神经网络参数传递给数据收集器。那么具体的收集过程是什么样的呢？首先明确一下收集数据过程的标准定义：



(数据轨迹的形式化定义示意图)

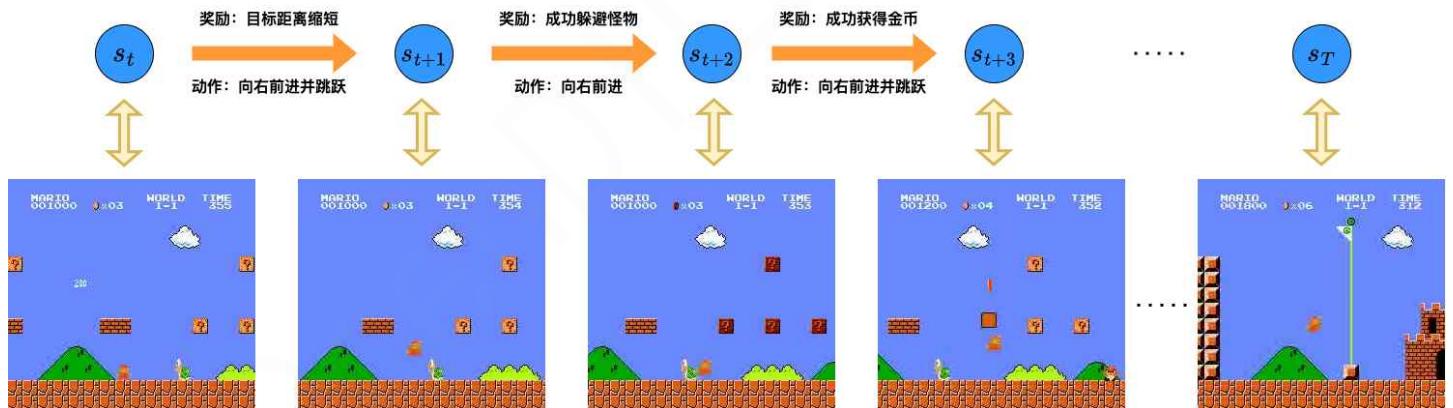
一般来说，收集器中会有一个智能体（Agent）和一个决策环境（Env），两者不断交互产生数据，把每一步产生的数据（包含状态 s_t ，动作 a_t ，奖励 r_t 等），按照交互的时间顺序串起来，组成一条训练用的数据轨迹（trajectory），即：

$$traj = (< s_0, a_0, r_0, \dots >, < s_1, a_1, r_1, \dots >, \dots, < s_{T-1}, a_{T-1}, r_{T-1}, \dots >)$$

在了解完抽象定义后，这里举一个具体的实例：经典红白机中的超级马里奥游戏（super mario）

- 注：该游戏的具体设定可以参考 [环境设定文档](#)

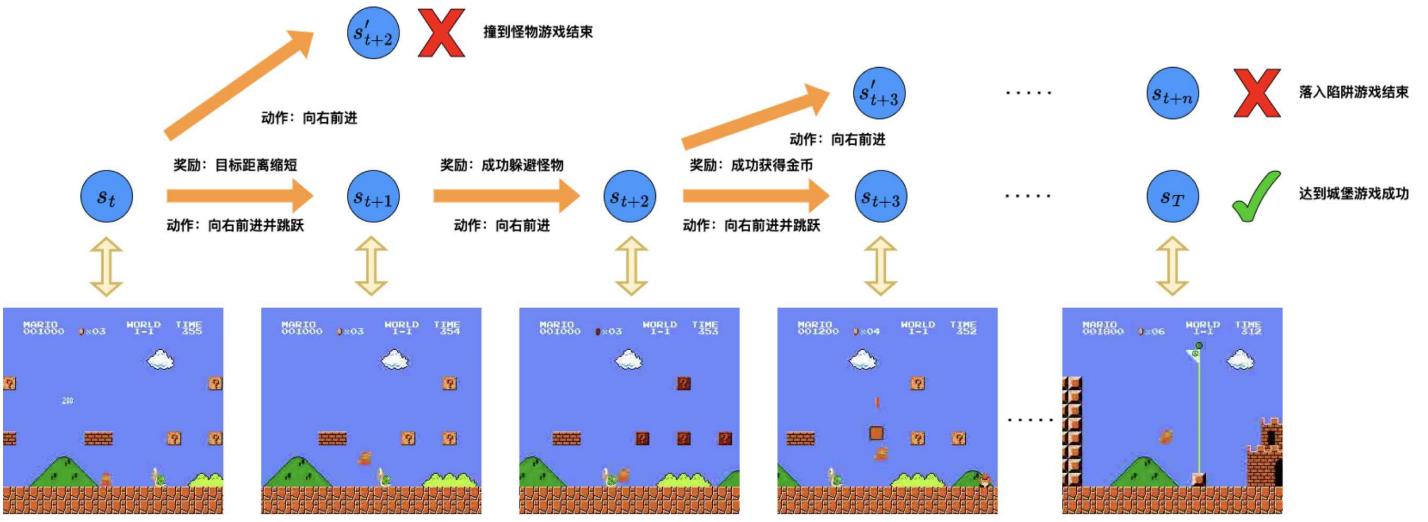
在这个游戏中，智能体像人类玩家一样，以图片画面中的信息为输入（观察空间），可以执行左右移动，跳跃等多种离散决策行为（动作空间），智能体需要闯过各种障碍，躲避怪物，最终抵达城堡。具体游戏中的一个片段示例如下所示：



(超级马里奥游戏成功通关的数据轨迹详细示意图)

注：智能体完整通关关卡 1-1 的视频可以参考 [Agent Demo List](#) 中的 mario 1-1

此外，收集数据时还有一个很关键的要素就是“探索”，因为强化学习正是一个不断提升、不断进化 的学习过程，需要在每次收集数据时适当尝试一些新的行为和策略，再根据这些新行为的收益高低，来决定是否要将让智能体学到这些知识。如果收集到的数据总是在相同的场景里重复已经学到的行为，那“强化”二字也就无从提起了。



(超级马里奥游戏中不同的轨迹示意图)

设计目标

策略梯度法的第二个关键步骤就是**设计智能体的决策目标**，即期望智能体运用上面最新收集到的数据能达到的效果。最朴素的想法是直接把任务目标交给智能体，比如自动驾驶就是让智能体把车从起点开到终点，围棋就是赢下这个对局，但这样的目标实在是太空洞了，信息量很少，智能体完全不知道该怎样去做。相应地，类比驾照考试的科目二和科目三，会设置一系列的考试科目（倒车入库，百米加减档），只有几乎通过所有考试才能被认为掌握了一定的驾驶技能。同理，把视角转换到策略梯度方法的设计中，智能体与环境交互就像是在练习各种科目，而训练数据中各种各样的轨迹就像在不同“科目”上的实践结果，其中既有表现好的轨迹，又有表现差的轨迹。**从轨迹生成的角度**，要想学会某项技能，可以通过增大高回报决策轨迹的概率来进行学习。具体来说，这里可以定量化计算每一个轨迹发生的概率，如下文中的公式所示：

$$P_\theta(\tau) = \rho(s_0)P_\theta(a_0|s_0)P(s_1|s_0, a_0)P_\theta(a_1|s_1)P(s_2|s_1, a_1) \dots$$

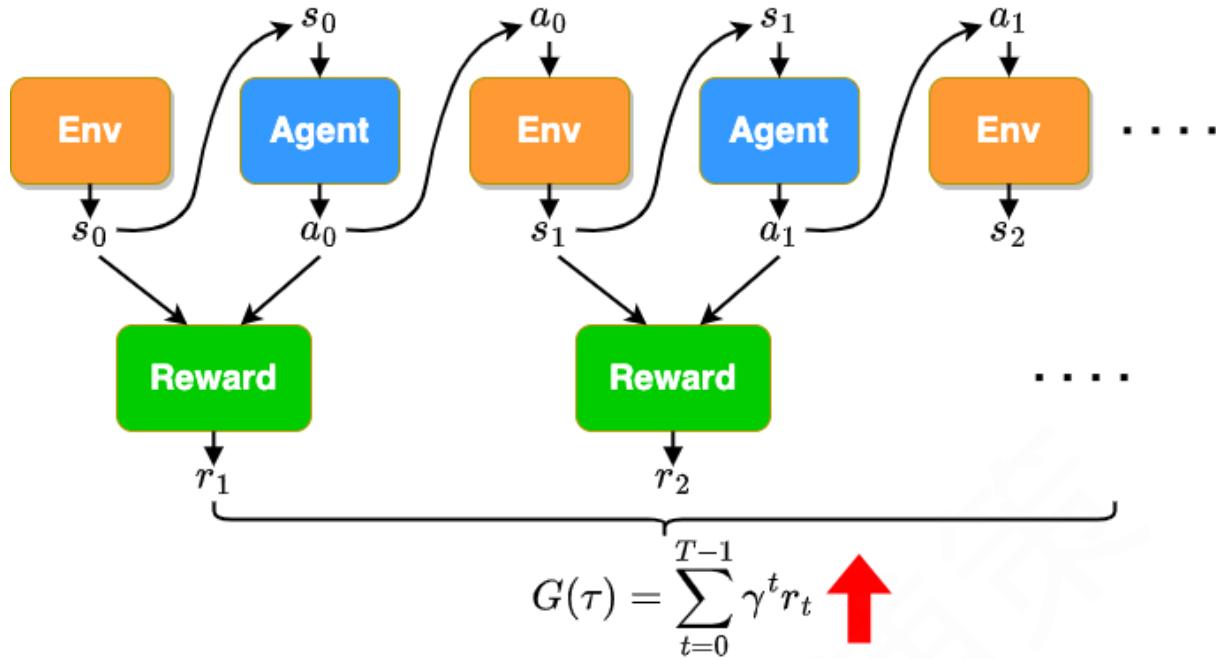
这个轨迹概率由三项组成

1. 环境的规则 $P(s_{t+1}|s_t, a_t)$ ：即环境内部状态转移的规则和逻辑（比如虚拟游戏规则，动力学模拟工具，自然物理定律等等），通常无法控制这部分，而是要通过学习过程，让智能体去适应和了解环境的规则。
2. 智能体的行为 $P_\theta(a_t|s_t)$ ，其中 θ 代表神经网络参数：这部分是算法可以控制的，不同的神经网络参数，决定了当给定一个状态 s_t 时对应输出的动作 a_t 。希望通过策略梯度方法来优化神经网络参数，使得智能体能在各种状态下都选择最优动作，获得最高的长期回报。
3. 初始状态分布 $\rho(s_0)$ ，即环境初始状态的分布。

总结一下，如下图所示，首先，将收集到的数据轨迹，定量地表示成相应的概率形式，然后，这些轨迹的期望回报又可以通过累加即时奖励值获得（注意：这里使用的是累积折扣奖励，通过折扣因子 γ 来平衡现在与未来的奖励），那么智能体的决策目标便呼之欲出：**最大化智能体产生高期望回报值的轨迹概率**。

注： $P_\theta(\tau)$ 和前文中的 $\tau \sim \pi_\theta$ 都是指，轨迹是根据参数化的策略生成的

$$P_\theta(\tau) = \rho(s_0)P_\theta(a_0|s_0)P(s_1|s_0, a_0)P_\theta(a_1|s_1)P(s_2|s_1, a_1) \cdots$$

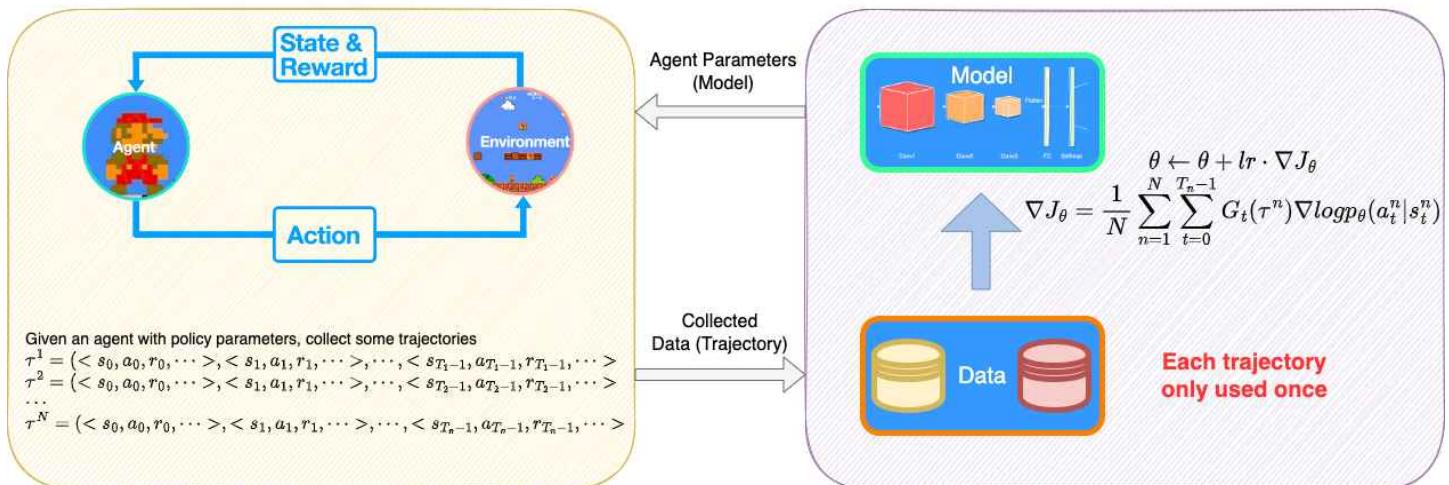


优化策略

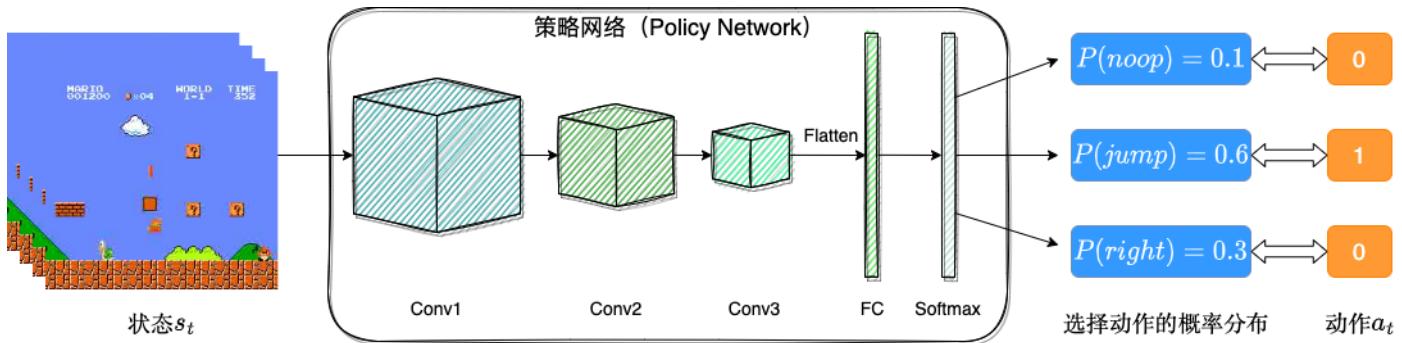
在完成收集数据，设计目标之后，最终就需要设计算法来优化智能体了。策略梯度方法的优化思路其实很简单：就是增大策略选择高回报值动作的概率，减小策略选择低回报值动作的概率。类比考驾照的过程，就是不断通过实际练车的经验，去引导手、脚、大脑之间的协调，在各种场景中都可以尽可能多做出正确的操作，从而尽可能避免做出一些危险驾驶行为。

对于上述的直观思想，强化学习中的经典结论——**策略梯度定理**将这个思路推广泛化到了多步的马尔科夫决策过程中，并给出了梯度相关计算公式。具体来说，如下图所示，策略梯度算法就是在上文所述的轨迹概率和期望回报的基础上，设计如下的梯度公式，并用随机梯度下降法（SGD）去执行优化更新神经网络模型参数。

注：有关策略梯度定理的详细数学推导可以参考附录1.4.1中的[揭秘策略梯度](#)



(策略梯度算法的数据收集过程和训练过程)

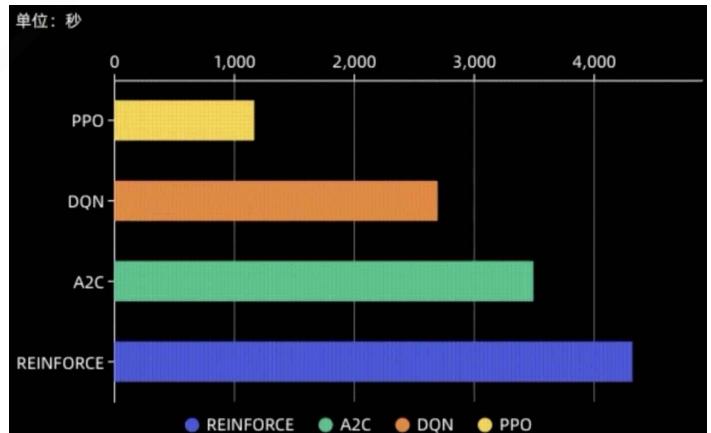
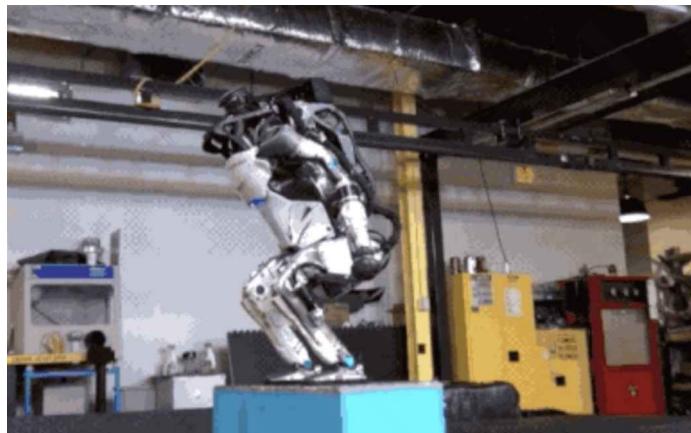


(结合神经网络的策略梯度算法的输入输出数据流图)

在代码实现中，其实就是将采样到的数据轨迹，传入神经网络并计算相应的目标函数，然后计算梯度更新神经网络。对于目标函数而言，其实就是将每一组 (s_t, a_t) 拿出来，计算它的对数概率 (log probability) 并执行反向传播计算梯度，同时梯度会乘以一个权重值，这个权重值就是这局游戏的累积回报。在计算完所有样本的梯度后，取梯度的平均值，执行一步 SGD 更新。而在深度学习方面，对于超级马里奥这样的游戏，需要结合卷积神经网络等神经网络经典架构知识来设计策略网络，输入图片状态，输出选择每个离散动作的概率，并根据策略梯度来优化神经网络。

Note：策略梯度算法是一种 on-policy 算法，每次采样的数据只会用一次，所以下一次循环需要使用最新的神经网络模型，执行收集数据过程，获取到最新的数据，再去进行下一次优化过程。

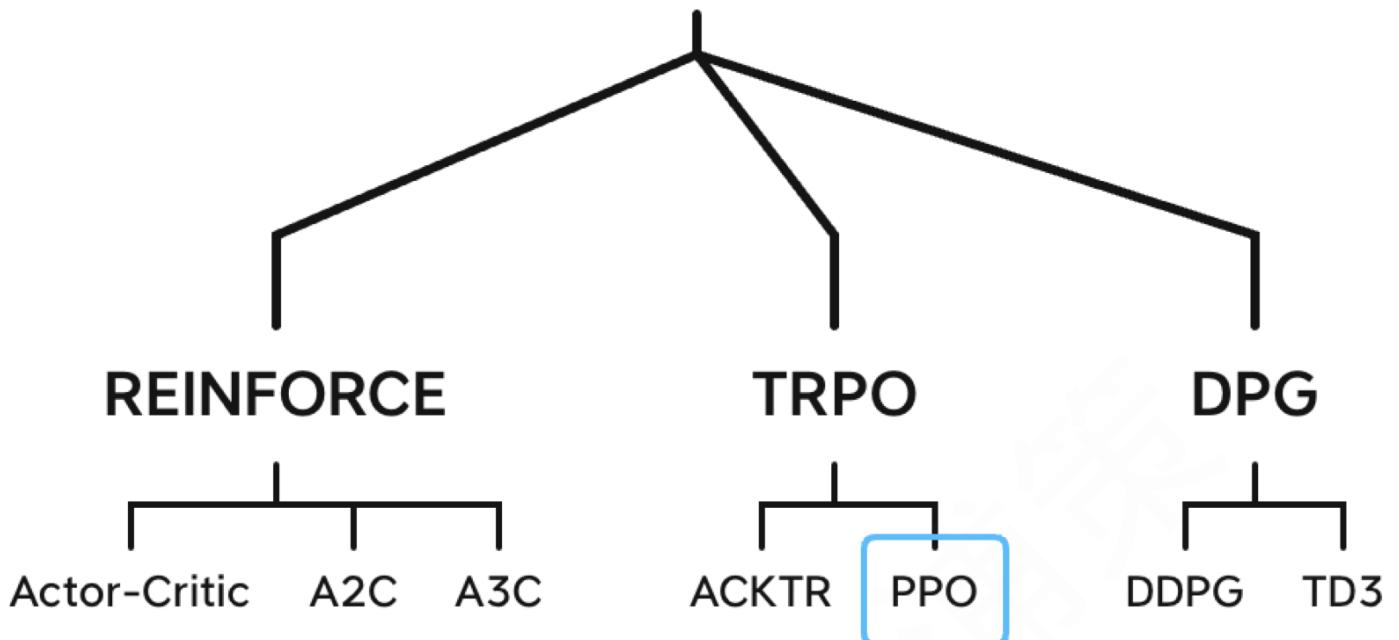
1.3.2 策略梯度各种改进的集大成之作：PPO



(算法的稳定性和效率)

在了解完策略梯度的核心思想之后，本章节展开讲解策略梯度的后续衍生工作，这些工作主要从**稳定性和效率**两个方面来进行优化。具体来说，因为策略梯度是采样近似估计回报函数，并且回报函数本身会受到环境奖励函数随机性的影响，所以实际的优化过程中，梯度的方差会很大。另一方面，策略梯度方法需要每次收集完整的 episode 轨迹，这对于一些规模较大的环境，数据收集和利用效率自然会变得非常低下。本章节中主要会结合跟 PPO 直接相关的改进方法展开讲解，其他衍生方法将在后续的课程中穿插进行对比和分析。

Policy Gradient



(策略梯度方法的后续衍生工作分类图)

Actor-Critic

- 问题：REINFORCE 策略梯度方法使用 episode return，梯度方差大，数据利用效率低
- 解决思路：结合 Value-Based RL 方法的思路，训练一个价值函数网络来估计这个值



(Actor-Critic 算法，即演员-评论家算法原理示意图)

价值函数，比如常使用的动作价值函数，简单来说可以定义为：

$$Q_\phi(s_t, a_t) = E_{s_{t+1}, a_{t+1}, \dots} \left[\sum_{l=0}^{\infty} \gamma^l r^{t+l} \right]$$

即在状态 s_t 下采用动作 a_t 后，后续动作服从策略 π 的情况下的累积期望回报。

通过将原本策略梯度公式中的回报函数替换为动作价值函数 $Q_\phi(s_t, a_t)$ ，进而得到 Actor-Critic 算法的更新公式，这样，就不用收集完整的 episode 数据来进行训练，只用收集定长的轨迹，结合价值函数来优化即可，从而提高数据收集和利用效率，减小梯度方差。

$$\nabla J_\theta = \frac{1}{N} \sum_{n=1}^N \sum_{t=0}^{T_n-1} Q_\phi(s_t, a_t) \nabla \log p_\theta(a_t | s_t)$$

不过，由于价值函数也是在训练过程中使用神经网络去不断近似学习的，实际产生的梯度会存在一定的偏差，这个问题将在之后的工作中被进一步优化。

注：关于价值函数及 Value-Based RL 方法的详细理解可以参考其他强化学习公开课，也可以通过这个[知乎博客](#)来直观了解一下。

A2C: Advantage Actor-Critic

- 问题：REINFORCE 方法梯度高方差，Actor-Critic 存在梯度偏差
- 解决思路：结合二者的优势，并减去一个基线函数来标准化，从而在保持无偏梯度估计的同时减小方差

$$\nabla J_\theta = \frac{1}{N} \sum_{n=1}^N \sum_{t=0}^{T_n-1} G_t(\tau) \nabla \log p_\theta(a_t | s_t) \quad + \text{no bias} \quad - \text{higher variance}$$

$$\nabla J_\theta = \frac{1}{N} \sum_{n=1}^N \sum_{t=0}^{T_n-1} Q_\phi(s_t, a_t) \nabla \log p_\theta(a_t | s_t) \quad - \text{not unbiased} \quad + \text{lower variance}$$

$$\nabla J_\theta = \frac{1}{N} \sum_{n=1}^N \sum_{t=0}^{T_n-1} (G_t(\tau) - V_\phi(s_t)) \nabla \log p_\theta(a_t | s_t) \quad + \text{no bias} \quad + \text{lower variance}$$

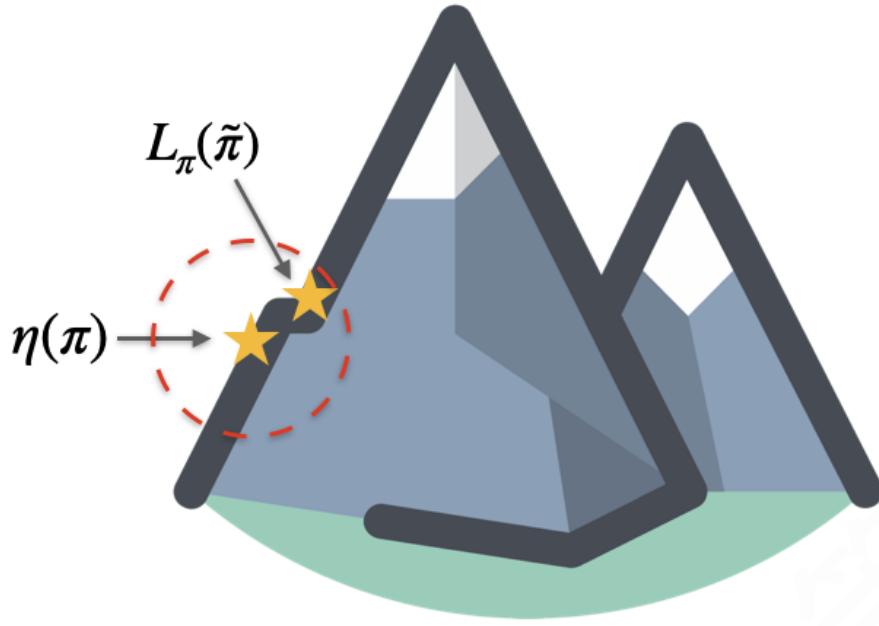
 可以用 Advantage 函数来表示

注：有关 A2C 中算法设计的详细数学原理推导可以参考附录1.4.1中的 [A2C 详解](#)

TRPO: Trust Region Policy Optimization

问题：如何让策略持久稳定提升

解决思路：可靠的方向，合适的步长



(信赖域示意图：替代函数和新策略之间的关系)

如何找到让策略持久稳定提升的方法呢，这里也可以使用类似轨迹生成的角度，定量化给出新旧策略之间的关系（这里的新策略指更新后的策略，旧策略指更新前的策略）：

$$\eta(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_{\tilde{\pi}}(s) \sum_a \tilde{\pi}(a | s) A_\pi(s, a)$$

只要让上式中加号右边的部分大于0，就可以获得稳定的策略提升，但是这一项实际过程中无法优化，因为其中含有新策略本身，所谓无法拿自己更新自己。

因此，一个合理的方式是使用替代函数近似，即：

$$\eta(\tilde{\pi}) \approx L_\pi(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_\pi(s) \sum_a \tilde{\pi}(a | s) A_\pi(s, a)$$

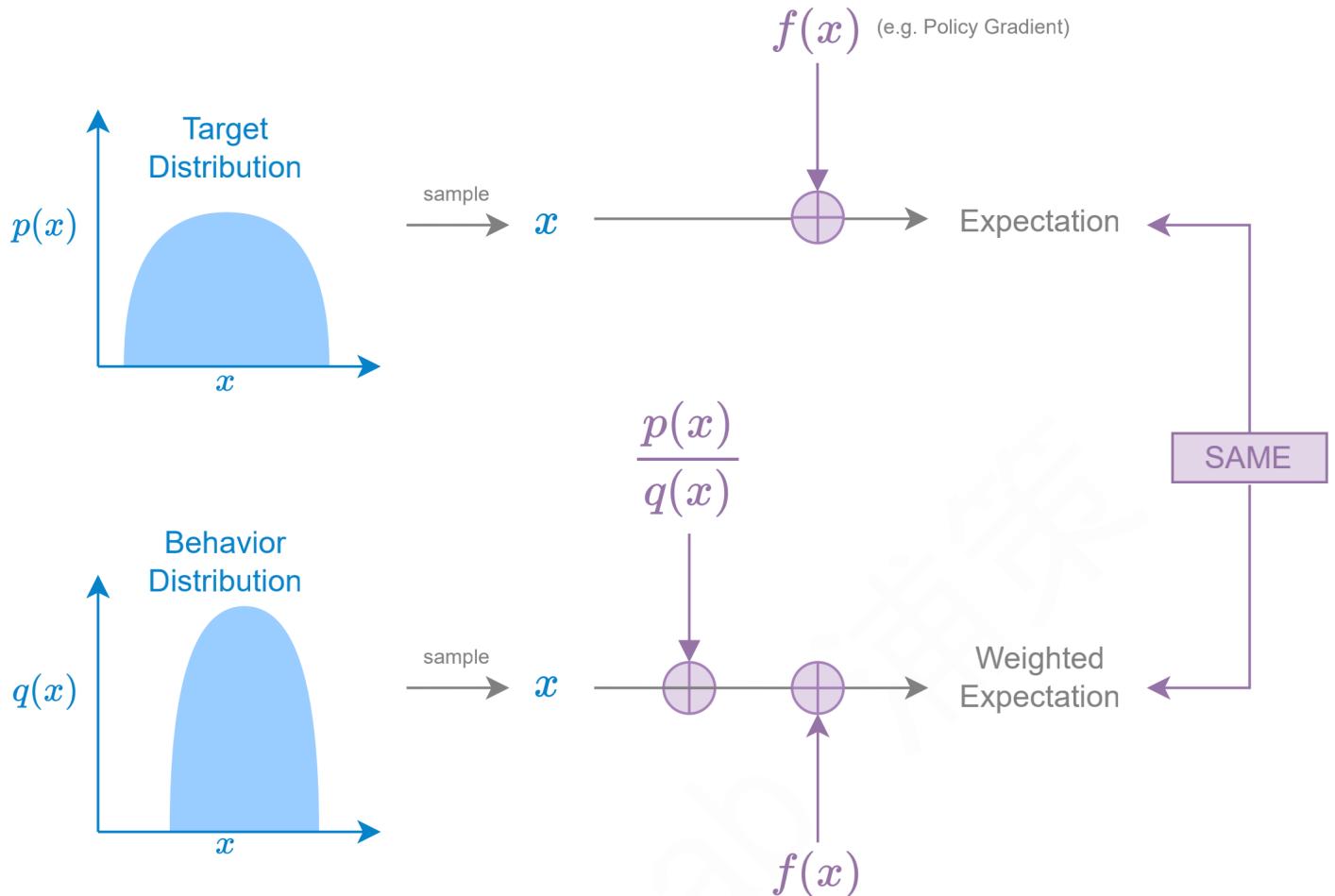
这个近似的效果怎么样呢，TRPO 这篇论文中就给出了新策略和替代函数之间的定量关系：

$$\eta(\tilde{\pi}) \geq L_\pi(\tilde{\pi}) - \frac{4\epsilon\gamma}{(1-\gamma)^2} \alpha$$

where $\alpha = \max_s D_{\text{KL}}(\pi(\cdot | s) \| \tilde{\pi}(\cdot | s))$, $\epsilon = \max_{s,a} |A_\pi(s, a)|$

基于上述不等式，就可以推导出相应的优化算法，保持策略的稳定更新。具体来说，优化方向由替代函数近似后的公式给出，而优化步长则通过不等式中的 KL 散度项来进行约束，即所谓的信赖域。

Importance Sampling



(重要性采样 (Importance Sampling, IS) 原理示意图)

另一方面，在上述的公式中还存在另一个对新策略的求和（期望）项，这里 TRPO 结合强化学习中的重要概念之一**重要性采样**来进行矫正。具体来说，原本的式子中是对旧策略的优势函数，求新策略对应动作分布的期望，但实际计算中往往使用旧策略采样获得的数据进行计算，因此将原有的求和式变成使用重要性采样的估计（实现中只是通过样本来估计重要性采样）矫正过后的形式。

$$\eta(\tilde{\pi}) \approx L_{\pi}(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_{\pi}(s) \sum_{a \sim \tilde{\pi}} \tilde{\pi}(a | s) A_{\pi}(s, a)$$

新策略的期望 旧策略的优势函数

$$\eta(\tilde{\pi}) \approx L_{\pi}(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_{\pi}(s) \sum_{a \sim \pi} IS \cdot \pi(a | s) A_{\pi}(s, a)$$

通过 IS 来进行矫正 $IS = \frac{\tilde{\pi}(a | s)}{\pi(a | s)}$

注：有关 TRPO 算法的详细数学原理推导可以参考附录1.4.1中的[步步深入 TRPO](#)

PPO: Proximal Policy Optimization

问题：TRPO 求解约束优化问题很麻烦，IS 计算的方差很大（由于通过采样估算 IS）

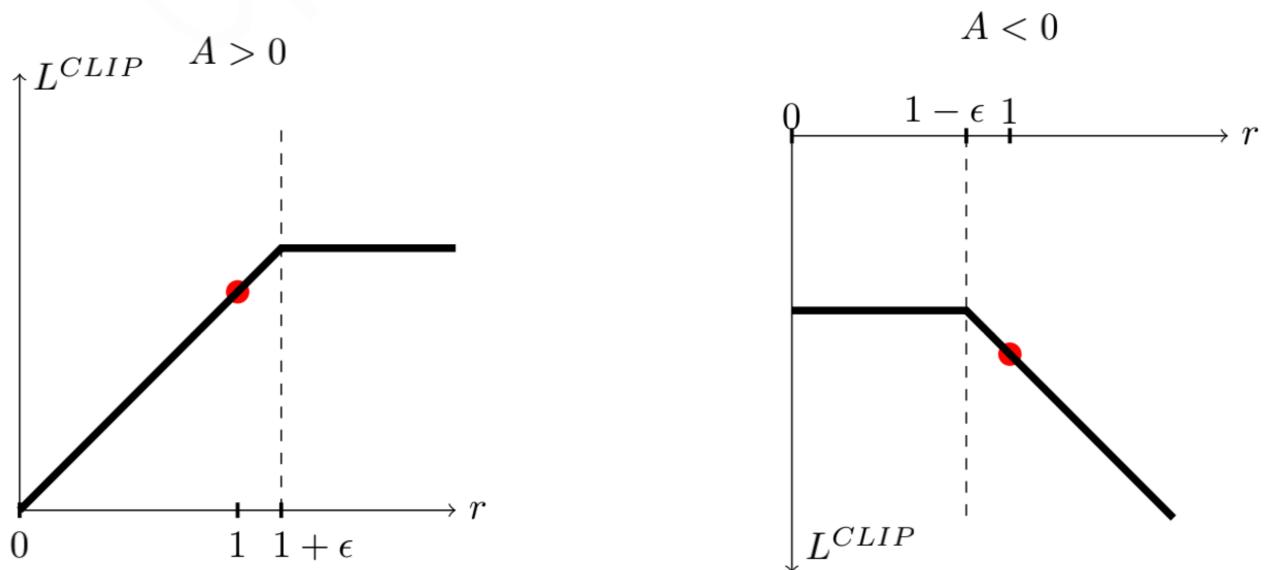
解决思路：Proximal Policy Optimization (PPO)

PPO 将 TRPO 中原始优化项和约束条件的组合，转化为一个没有约束条件的截断式优化目标：

$$\mathbb{E}_t [\min\left(\frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_k}(a_t | s_t)} A^{\theta_k}(s_t, a_t), \text{clip}\left(\frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_k}(a_t | s_t)}, 1 - \epsilon, 1 + \epsilon\right) A^{\theta_k}(s_t, a_t)\right)]$$

要注意的是，PPO 其实是一种悲观的约束 (pessimistic bound)，优化目标中包含一个对于原始优化项和截断式优化目标取最小值的操作。具体来讲，当预期收益较好时（即 $A > 0$ ），PPO 会保持谨慎，选择在 $r(\theta) > 1 + \epsilon$ 时使用截断值，避免激进的优化行为。而当预期收益较差时（即 $A < 0$ ），PPO 仍然会保持足够的惩罚力度，选择在 $r(\theta) < 1 - \epsilon$ 时使用截断值

注：下图中的 $r(\theta)$ or r 就是重要性采样系数 IS



Algorithm 1 Proximal Policy Optimization (PPO)

- 1: Input: initialize policy (actor) parameters θ and value (critic) parameters ϕ . Training epochs per collect E , trajectory length T , batch size B .
- 2: **for** $k = 0, 1, 2, \dots$ **do**
- 3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy π_{θ_k} interaction with environment.
- 4: Compute trajectory target return estimates \hat{R}_t . (e.g. n-step TD or other methods)
- 5: Compute advantage estimates \hat{A}^{θ_k} with value V_{ϕ_k} . (e.g. GAE or other methods)
- 6: **for** $e = 0, 1, \dots, E - 1$ **do**
- 7: **for** minibatch: $b \in \mathcal{D}_k$ **do**
- 8: Update the policy by maximizing the PPO-Clip objective with SGD:

$$L_\theta = \frac{1}{B \cdot T} \sum_{\tau \in b} \sum_{t=0}^{T-1} \min(r(\theta) \hat{A}^{\theta_k}(s_t, a_t), \text{clip}(r(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}^{\theta_k}(s_t, a_t))$$

$$r(\theta) = \frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_k}(a_t | s_t)}$$

- 9: Fit the value by regression on mean-squared error with SGD:

$$L_\phi = \frac{1}{B \cdot T} \sum_{\tau \in b} \sum_{t=0}^{T-1} (V_\phi(s_t) - \hat{R}_t)^2$$

- 10: **end for**
 - 11: **end for**
 - 12: **end for**
-

(PPO算法，PPO-Clip版本常用实现伪代码)

更具体地，结合上方的伪代码，可以得到关于 PPO 的一些特点：

- PPO 是一种 **on-policy 算法**（需要保持收集数据的策略和训练更新的策略都是最新的，不能使用过旧的策略收集数据）
- PPO 需要收集一定长度的序列轨迹数据，并预算像 Return 和 Advantage 这些估计值，用于训练端的更新
- 在收集到轨迹数据后，PPO 通过两重训练循环来提高数据利用效率，第一重循环是将数据重复训练 E 个 epoch，第二个循环是在每个 epoch 内，将数据划分成多个 minibatch 进行优化。
- 具体优化时，对于策略部分，PPO 结合截断式优化目标和悲观的约束 (pessimistic bound) 来指导策略进行更新，而对于价值函数部分，PPO 常利用经典的多步时序差分 (Temporal Difference) 方法来进行优化，比如 GAE。

1.4 附录 (可选阅读)

1.4.1 算法理论细节

本章节给出了1.3中部分深度强化学习算法的详细数学理论推导，有兴趣的读者可以详细梳理其中细节，明晰各个算法背后的数学原理。

- 揭秘策略梯度
- 为什么 Actor-Critic 添加 baseline 之后可以减小方差
- 步步深入 TRPO

1.4.2 延伸阅读材料

本章节列举了一些 PPO 算法具体实现中的技巧总结，有兴趣的读者可以深入了解，本课程也会整合其中最关键的内容，在第七讲详细进行讲解。

- <https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/>
- Engstrom, Logan, et al. "Implementation matters in deep policy gradients: A case study on PPO and TRPO." arXiv preprint arXiv:2005.12729 (2020).
- Marcin Andrychowicz, Anton Raichuk, Piotr Stanczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, Olivier Bachem: What Matters for On-Policy Deep Actor-Critic Methods? A Large-Scale Study. ICLR 2021

参考文献

- [1] Ramesh A, Dhariwal P, Nichol A, et al. Hierarchical Text-Conditional Image Generation with CLIP Latents. 2022[J]. URL: <https://doi.org/10.48550/arXiv.2204>.
- [2] Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning[J]. arXiv preprint arXiv:1312.5602, 2013.
- [3] Ho J, Ermon S. Generative adversarial imitation learning[J]. Advances in neural information processing systems, 2016, 29.
- [4] Torabi F, Warnell G, Stone P. Behavioral cloning from observation[J]. arXiv preprint arXiv:1805.01954, 2018.
- [5] Lillicrap T P, Hunt J J, Pritzel A, et al. Continuous control with deep reinforcement learning[J]. arXiv preprint arXiv:1509.02971, 2015.
- [6] Haarnoja T, Zhou A, Abbeel P, et al. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor[C]//International conference on machine learning. PMLR, 2018: 1861-1870.

- [7] Rashid T, Samvelyan M, Schroeder C, et al. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning[C]//International conference on machine learning. PMLR, 2018: 4295-4304.
- [8] Lanctot M, Zambaldi V, Gruslys A, et al. A unified game-theoretic approach to multiagent reinforcement learning[J]. Advances in neural information processing systems, 2017, 30.
- [9] Schulman J, Wolski F, Dhariwal P, et al. Proximal policy optimization algorithms[J]. arXiv preprint arXiv:1707.06347, 2017.
- [10] 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.
- [11] Mnih V, Badia A P, Mirza M, et al. Asynchronous methods for deep reinforcement learning[C]//International conference on machine learning. PMLR, 2016: 1928-1937.
- [12] Schulman J, Levine S, Abbeel P, et al. Trust region policy optimization[C]//International conference on machine learning. PMLR, 2015: 1889-1897.