摘　要

[TODO]

关键词： 强化学习，虚拟现实，机器控制算法

ABSTRACT

[TODO] **Keywords:** Reinforcement Learning, Virtual Reality, Robotic Control

目 录

第一章 绪 论

1.1 研究工作的背景与意义

机器学习是以知识的自动获取和产生为研究目标，是人工智能的核心问题之一。机器学习与统计学、心理学、机器人学等许多其他学科都有交叉。其中，学习心理学与机器学习的交叉综合直接促进了强化学习又称做增强学习或再励学习(Reinforcement Learning，RL) 理论与算法的产生和发展。所谓强化学习是一种以环境反馈作为输入的、特殊的、适应环境的机器学习方法，它的主要思想是与环境交互和试错，利用评价性的反馈信号实现决策的优化。这也是自然界中人类或动物学习的基本途径。

近年来，强化学习技术在人工智能、机器学习和自动控制等领域中得到了广泛的研究和应用，并被认为是设计智能系统的核心技术之一。随着强化学习算法和理论的深入，特别是强化学习的数学基础研究取得突破性进展之后，应用强化学习方法实现移动机器人行为对环境的自适应和控制器的优化成为机器人学领域研究和应用的热点之一。

与此同时，虚拟现实技术已被认为是人机接口技术的一场革命。它利用计算机和电子技术来产生逼真的视、听、触、力等三维感觉环境, 形成一种虚拟世界。它与人工智能、计算机图形学、人机接口技术、多媒体技术、传感技术以及高度并行的实时计算技术等领域联系十分紧密。虚拟现实技术可以最大限度地模拟真实环境，根据这个特点，人们可以在虚拟现实中开展各种训练任务，训练完成后将模型迁移到真实环境中。

1.2 强化学习算法的国内外研究历史与现状

强化学习作为机器学习领域另一个研究热点，已经广泛应用于工业制造。因此 RL 方法更加侧重于学习解决问题的策略。随着人类社会的飞速发展，在越来越多复杂的现实场景任务中，需要利用深度学习 (Deep Learning, DL) 来自动学习大规模输入数据的抽象表征，并以此表征为依据进行自我激励的 RL，优化解决问题的策略。RL 的基本思想是通过最大化智能体 (agent) 从环境中获得的累计奖赏值，以学习到完成目标的最优策略。由此，谷歌的人工智能研究团队 DeepMind 创新性地将具有感知能力的 DL 和具有决策能力的 RL 相结合，形成了人工智能领域新的研究热点，即深度强化学习 (Deep Reinforcement Learning，DRL)。此后，在很多挑战性领域中，DeepMind 团队构造并实现了人类专家级别的 agent。这些 agent 对

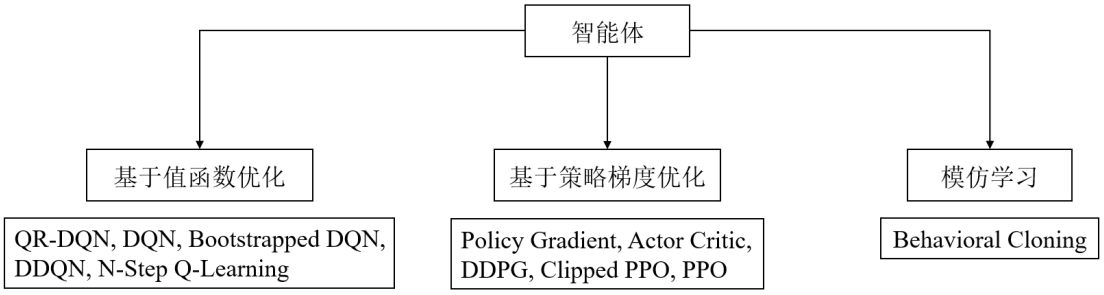

图 1-1 主流强化学习算法

自身知识的构建和学习都直接来自原始输入信号，无需任何的人工编码和领域知识。因此 DRL 是一种端对端 (end-to-end) 的感知与控制系统，具有很强的通用性。目前 DRL 技术在游戏等领域中得到了广泛的应用，并被认为是迈向通用人工智能 (Artificial General Intelligence，AGI) 的重要途径。

如图1-1所示，目前主流的强化学习算法主要分为如下几类：

1. 基于值函数优化 (Value Optimization) 的算法，代表算法为 Q-Learning，Mnih 等人将卷积神经网络与传统 RL 中的 Q-Learning 算法相结合，提出了深度 Q 网络 (Deep Q-Network, DQN) 模型，此后又涌现出双深度 Q 网络 (Double Deep Q-Network, DDQN)、分类 Q 网络 (Categorical Deep Q-Network，CDQN)、增强 Q 网络 (Bootstrapped Deep Q-Network) 等一系列值优化模型；
2. 基于策略梯度优化 (Policy Optimization) 的算法，此类算法可以能够直接在策略空间中搜索最优策略，从而比基于值函数优化的方法实用性更广，代表算法有区域信赖的策略最优化方法 (Trust Region Policy Optimization，TRPO)、近端梯度优化方法 (Proximal Policy Optimization，PPO)。
3. 行动者-评论者优化方法 (Actor-Critic)，Lillicrap 等人利用 DQN 扩展 Q-Learning 的思路对确定性梯度策略 (Deterministic Policy Gradient，DPG) 方法进行改造，提出了一种基于 AC 框架的深度确定性策略梯度 (Deep Deterministic Policy Gradient，DDPG) 算法，并在解决连续动作空间问题上取得了很好的效果。

1.3 本文的主要贡献与创新

本文尝试将虚拟现实技术中的 Unity 3D 环境与强化学习算法的训练结合起来，使得强化学习的训练时间大幅度减少。同时通过 Unity 强大的物理引擎尽可能地拟合真实环境，从而使得训练出的强化学习模型无需经过细致调参优化就可以很好地迁移应用到实体机械环境上。

1.4 本论文的结构安排

本文的主要章节安排如下：第二章主要介绍了强化学习领域的基础知识，并对主流的强化学习算法的发展路线进行梳理；第三章首先介绍了 Unity 相对于传统强化学习仿真框架的优势，然后对本文采用的强化学习算法进行介绍，之后分别介绍该算法在 2D 环境和 3D 环境中的性能表现，并对 Unity 与强化学习整合的可能性进行评估；第四章对全文进行总结和回顾。

第二章 强化学习理论

2.1 强化学习基础

RL 是一种从环境状态映射到动作的学习，目标是使 agent 在与环境的交互过程中获得最大的累积奖赏。马尔可夫决策过程 (Markov Decision Process，MDP) 可以用来对 RL 问题进行建模。通常将 MDP 定义为一个四元组 (S, A, ρ, f)，其中：

1. S 为所有环境状态的集合。$s_t \in S$ 表示 agent 在 t 时刻所处的状态；
2. A 为 agent 可执行动作的集合。$a_t \in A$ 表示 agent 在 t 时刻所采取的动作；
3. $\rho : S \times A \to R$ 为奖赏函数。$r_t \sim \rho(s_t, a_t)$ 表示 agent 在状态执行动作获得的立即奖赏值；
4. $f : S \times A \times S \to [0, 1]$ 为状态转移概率分布函数。$S_{t+1} \sim f(s_t, a_t)$ 表示 agent 在状态 s_t 执行动作 a_t 转移到下一状态 S_{t+1} 的概率。

在 RL 中，策略 $\pi : S \to A$ 是状态空间到动作空间的一个映射。表示为 agent 在状态 s_t 选择动作 a_t，执行该动作并以概率 $f(s_t, a_t)$ 转移到下一状态 s_{t+1}，同时接受来自环境反馈地奖赏 r_t。假设未来每个时间步所获得的立即奖赏都必须乘以一个折扣因子 γ，则从 t 时刻开始到 T 时刻结束时，奖赏之和定义为：

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_t \tag{2-1}$$

其中 $\gamma \in [0, 1]$，用来权衡未来奖赏对累计奖赏的影响。状态动作值函数 $Q^{\pi}(s, a)$ 指的是在当前状态 s 下执行该动作 a，并一直遵循策略 π 到情节结束，这一过程中 agent 所获得的累计回报表示为：

$$Q^{\pi}(s, a) = E[R_t | s_t = s, a_t = a, \pi] \tag{2-2}$$

对于所有的状态动作对，如果一个策略 π^* 的期望回报大于或等于其他所有策略的期望回报，那么称策略 π^* 为最优策略。最优策略可能不只一个，但它们共享一个状态动作值函数：

$$Q^{\pi}(s, a) = max_{\pi} E[R_t | s_t = s, a_t = a, \pi] \tag{2-3}$$

式2-3被称为最优状态值函数，且最优状态动作值函数遵循贝尔曼最优方程 (Bellman Optimality Equation)。即：

$$Q^*(s, a) = E_{s' \sim S}[r + \gamma max_{a'} Q(s', a') | s, a] \tag{2-4}$$

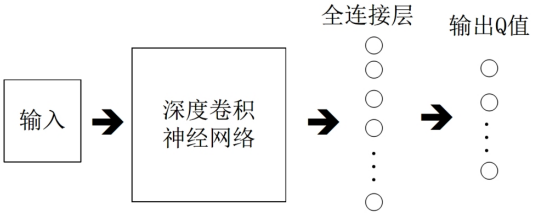

图 2-1 DQN 的模型结构

在传统的 RL 中，一般通过迭代贝尔曼方程求解 Q 值函数：

$$Q_{i+1}(s,a) = E_{s' \sim S}[r + \gamma max_{a'} Q_i(s',a') | s,a] \tag{2-5}$$

其中，当 $i \to \infty$ 时，$Q_i \to Q^*$。即通过不断地迭代会使状态动作值函数最终收敛，从而得到最优策略 $\pi^* = argmax_{a \in A} Q^*(s,a)$。然而对于实际问题来说，通过迭代式2-5求解最优策略显然是不可行的，因为在大状态空间下，用迭代贝尔曼方程求解 Q 值函数的方法计算代价太大．针对此问题，在 RL 算法中，通常使用线性函数逼近器来近似表示状态动作值函数，即 $Q(s,a,\theta) \approx Q^*(s,a)$。此外，也可以用深度神经网络等非线性函数逼近器去近似表示值函数或策略。然而将 RL 与深度神经网络相结合可能会出现算法不稳定等问题，这一直阻碍着 DRL 的发展与应用。

2.2 基于值函数的深度强化学习

2.2.1 深度 Q 网络

Mnih 等人将卷积神经网络与传统 RL 中的 Q 学习算法相结合，提出了深度 Q 网络（Deep Q-Network, DQN）模型．该模型用于处理基于视觉感知的控制任务，是 DRL 领域的开创性工作．

2.2.1.1 模型结构

DQN 模型的输入是距离当前时刻最近的 4 幅预处理后的图像。该输入经过 3 个卷积层和 2 个全连接层的非线性变换，最终在输出层产生每个动作的 Q 值。图2-1表示 DQN 的模型结构。

2.2.1.2 训练算法

图2-2描述了 DQN 的训练过程。为缓解非线性网络表示值函数时出现的不稳定等问题，DQN 主要对传统的 Q 学习算法做了 3 处改进。

1. DQN 在训练过程中使用经验回放机制 (experience replay)，在线处理得到的转移样本 $e_t = (s_t, a_t, r_t, s_{t+1})$。在每个时间步 t，将 agent 与环境交互得到的转

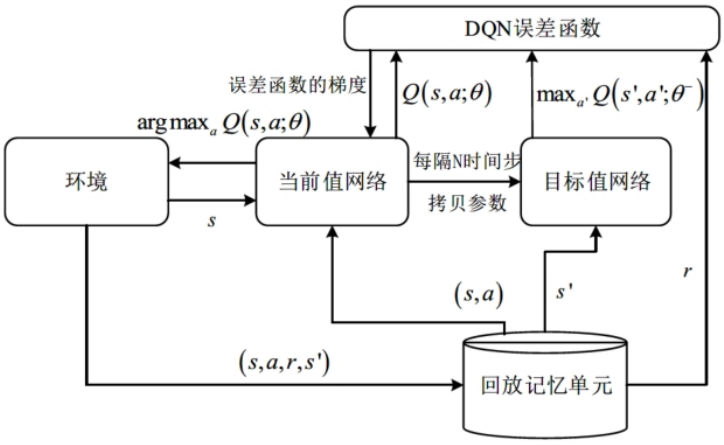

图 2-2 DQN 的训练过程

移样本存储到回放记忆单元 $D = e_1, ...e_t$ 中。训练时，每次从 D 中随机抽取小批量的转移样本，并使用随机梯度下降 (Stochastic Gradient Descent, SGD) 算法更新网络参数 θ。在训练深度网络时，通常要求样本之间是相互独立的。这种随机采样的方式，大大降低了样本之间的关联性，从而提升了算法的稳定性。

2. DQN 除了使用深度卷积网络近似表示当前的值函数之外，还单独使用了另一个网络产生目标 Q 值。具体地，$Q(s,a|\theta_i)$ 表示当前值网络的输出，用来评估当前状态动作对的值函数；$Q(s,a|\theta_i^-)$ 表示目标值网络的输出，一般采用近似表示值函数的优化目标，即目标 Q 值．当前值网络的参数 θ 是实时更新的，每经过 N 轮迭代，将当前值网络的参数复制给目标值网络。通过最小化当前 Q 值和目标 Q 值之间的均方误差来更新网络参数，误差函数为：

$$L(\theta_i) = E_{s,a,r,s'}[(Y_i - Q(s,a|\theta_i))^2] \tag{2-6}$$

对参数 θ 求偏导，得到一下梯度：

$$\nabla_{\theta_i} = E_{s,a,r,s'}[(Y_i - Q(s,a|\theta_i))\nabla_{\theta_i} Q(s,a|\theta_i)] \tag{2-7}$$

引入目标值网络后，在一段时间内目标 Q 值是保持不变的，一定程度上降低了当前 Q 值和目标 Q 值之间的相关性，提升了算法的稳定性。

3. DQN 将奖赏值和误差项缩小到有限的区间内，保证了 Q 值和梯度值都处于合理的范围内，提高了算法的稳定性。实验表明，DQN 在解决诸如 Atari 2600 游戏等类真实环境的复杂问题时，表现出与人类玩家相媲美的竞技水平，甚至在一些难度较低的非战略性游戏中，DQN 的表现超过了有经验的

人类玩家。在解决各类基于视觉感知的 DRL 任务时，DQN 使用了同一套网络模型、参数设置和训练算法，这充分说明 DQN 方法具有很强的适应性和通用性。

2.2.2 深度 Q 网络训练算法的改进

2.2.2.1 深度双 Q 网络

在 DQN 中使用 $Y_i = r + \gamma max_{a'} Q(s', a'|\theta_i^-)$ 近似表示值函数的优化目标时，每次都选取下一个状态中最大 Q 值所对应的动作。选择和评价动作都是基于目标值网络的参数 θ^-，这会引起在学习过程中出现过高估计 Q 值的问题。

Hasselt 等人基于双 Q 学习算法 (double Q-learning)，提出了深度双 Q 网络 (Deep DoubleQ-Network, DDQN) 算法。在双 Q 学习中有两套不同的参数：θ 和 θ^-。其中 θ 用来选择对应最大 Q 值的动作，θ 用来评估最优动作的 Q 值。两套参数将动作选择和策略评估分离开，降低了过高估计 Q 值的风险。因此 DDQN 使用当前值网络的参数 θ 来选择最优动作，使用目标值网络的参数 θ^- 来评估该最优动作。目标 Q 值的形式如下：

$$Y_i^{DDQN} = r + \gamma Q(x', argmax_a Q(s', a'|\theta_i^-)) \tag{2-8}$$

DDQN 在其他方面都与 DQN 保持一致。实验表明，DDQN 能够估计出更加准确的 Q 值，在一些 Atari 2600 游戏中可获得更稳定有效的策略。

2.2.2.2 基于优势学习的深度 Q 网络

根据2.2.2.1节可知，降低 Q 值的评估误差可以提升性能。Bellemare 等人在贝尔曼方程中定义新的操作符，来增大最优动作值和次优动作值之间的差异，以缓和每次都选取下一状态中最大 Q 值对应动作所带来的评估误差。具体的改进如下：基于采样得到的样本计算均方误差 $\Delta Q(s,a)^2$，其中误差项定义为：

$$\Delta Q(s,a) = r + \gamma V(s') - Q(s,a) \tag{2-9}$$

根据优势学习 (Advantage Learning，AL) 定义两种新的操作符，并将这两种操作符运用到上式中，分别得到 AL 误差项和一致性优势学习 (Persistent Advantage Learning，PAL) 误差项。其中 AL 误差项定义为：

$$\Delta_{AL} Q(s,a) = \Delta Q(s,a) - \alpha[V(s) - Q(s,a)] \tag{2-10}$$

为了定义 PAL 误差项，构造出如下式：

$$\Delta_{AL}Q'(s,a) = \Delta Q(s',a) - \alpha[V(s) - Q(s',a)] \tag{2-11}$$

得到 PAL 误差项的具体形式：

$$\Delta_{PAL}Q(s,a) = max\Delta_{AL}Q(s,a), \Delta_{AL}Q'(s,a) \tag{2-12}$$

实验表明，用 AL 和 PAL 误差项来替代贝尔曼方程中的误差项，可以有效地增加最优和次优动作对应值函数之间的差异，从而获得更加精确的 Q 值。即在 DQN 中加入 AL 和 PAL 误差项，可以有效地减小评估 Q 值时的偏差，促进学习效果的进一步提升，在许多 Atari 2600 游戏中取得了更好的表现。其中，采用 PAL 误差项时，最优和次优动作对应值函数之间的差异更大，Q 值的评估也更加精确。

2.2.2.3 基于优先级采样的深度 Q 网络

DQN 为了消除转移样本 $e_t = (s_t, a_t, r_t, s_{t+1})$ 之间的相关性，使用经验回放机制在线地存储和使用 agent 与环境交互得到的历史样本。在每个时刻，经验回放机制从样本池中等概率地抽取小批量的样本用于训练。然而等概率采样并不能区分不同样本的重要性，同时由于样本池 D 的存储量有限，某些样本还未被充分利用就已经被舍弃。针对该问题，Schaul 等人在 DDQN 的基础上提出了一种基于比例优先级采样的深度双 Q 网络 (double deep Q-Network with proportional prioritization)。该方法用基于优先级的采样方式来替代均匀采样，提高一些有价值样本的采样概率，从而加快最优策略的学习。具体的改进如下：

该抽样方法将每个样本的时间差分 (Temporal Difference,TD) 误差项作为评价优先级的标准。该误差为：$r + \gamma max_{a'}Q(s',a'|\theta^{-}) - Q(s,a|\theta)$，并且其绝对值越大，对应样本被采样的概率越高。在抽样过程中该方法使用随机比例化 (stochastic prioritization) 和重要性采样权重 (importance-sampling weights) 两种技巧。其中，随机比例化操作不仅能充分利用较大 TD 误差项对应的样本，而且保证了抽取样本的多样性。重要性采样权重的使用放缓了参数更新的速度，保证了学习的稳定性。实验表明，基于该抽样方式的深度双 Q 网络可以提升训练速度，并在很多 Atari 2600 游戏中获得了更高的分数。

另外，Lakshminarayanan 等人使用动态跳帧的方式来替代 DQN 中每个时刻重复 k 次的动作，提出了动态跳帧的 DQN(Dynamic Frame Skip DeepQ-Network，DFDQN) 算法。实验表明，DFDQN 在一些 Atari 2600 游戏中取得了更好的性能；Hasselt 等人使用一种称为 Pop-Art 的动态归一化操作来替代传统 DQN 中的区间裁

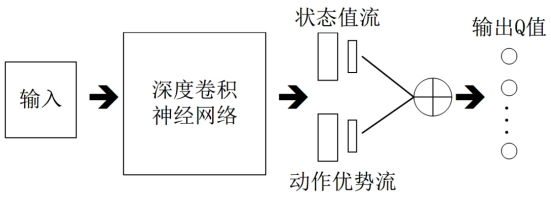

图 2-3 基于竞争架构的 DQN 模型结构

剪方法。在不流失重要状态信息的前提下，统一了不同任务中目标 Q 值的量级，提高了 agent 在很多 Atari2600 游戏中的表现；Vincent 等人在 DQN 中使用自适应的折扣因子和学习率，加速了深度网络收敛的速度。

2.2.3 DQN 模型结构的改进

对 DQN 模型的改进一般是通过向原有网络中添加新的功能模块来实现的。例如，可以向 DQN 模型中加入循环神经网络结构，使得模型拥有时间轴上的记忆能力。本节主要介绍两种重要的 DQN 模型的改进版本，分别是基于竞争架构的 DQN 和深度循环 Q 网络 (Deep Recurrent Q-Network，DRQN)。

2.2.3.1 基于竞争架构的 DQN

在很多基于视觉感知的 DRL 任务中，受不同动作的影响，状态动作对的值函数是不同的。然而在某些状态下，值函数的大小是与动作无关的。利用上述思想，Wang 等人设计了一种竞争网络结构 (dueling network)，并将其加入到 DQN 网络模型中。如图2-3，该网络结构与 DQN 模型的不同之处在于：DQN 将 CNN 提取的抽象特征经过全连接层后，直接在输出层输出对应动作的 Q 值，而引入竞争网络结构的模型则将 CNN 提取的抽象特征分流到两个支路中，其中一路代表状态值函数，另一路代表依赖状态的动作优势函数 (advantage function)。通过该种竞争网络结构，agent 可以在策略评估过程中更快地识别出正确的行为。 具体地，状态值函数表示为 $\hat{V}(s|\theta,\beta)$，动作优势函数表示为 $\hat{A}(s,a|\theta,\alpha)$。通过一种聚合操作将状态值流和动作优势流相结合：

$$Q(s,a|\theta,\alpha,\beta)=\hat{V}(s|\theta,\beta)+\hat{A}(s,a|\theta,\alpha) \tag{2-13}$$

其中，α、β 和 θ 分别代表状态值流、动作优势流和模型剩余部件的参数。然而在实际操作中，一般要将动作优势流设置为单独动作优势函数值减去某状态下所有动作优势函数的平均值。该技巧不仅可以保证该状态下各动作的优势函数相对排序不变，而且可以缩小 Q 值的范围，去除多余的自由度。实验表明，在 DQN 中加

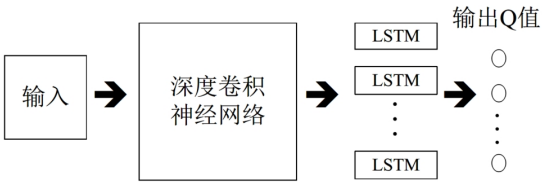

图 2-4 DRQN 模型结构

入竞争网络可以使得值函数的估计更加精确。在频繁出现 agent 采取不同动作但对应值函数相等的情形下，竞争架构的 DQN 模型性能提升最为明显。

2.2.3.2 深度循环 Q 网络

在传统的 RL 方法中，状态信息的部分可观察性一直是个亟待解决的难题。DQN 通过堆叠离当前时刻最近的 4 幅历史图像组成输入状态，有效缓解了状态信息的部分可观察问题，却增加了网络的计算和存储负担。针对此问题，Hausknecht 等人利用循环神经网络结构来记忆时间轴上连续的历史状态信息，提出了 DRQN 模型。 如图2-4所示，DRQN 将 DQN 中第 1 个全连接层的部件替换成了 256 个长短期记忆单元 (Long Short-TermMemory，LSTM)。此时模型的输入仅为当前时刻的一幅图像，减少了深度网络感知图像特征所耗费的计算资源。实验表明，在部分状态可观察的情况下，DRQN 表现出比 DQN 更好的性能。因此 DRQN 模型适用于普遍存在部分状态可观察问题的复杂任务。

随着 DL 领域中各种新颖网络模块的提出，未来 DRL 模型会朝着结构多样化、模块复杂化的方向发展。例如，可以利用深度残差网络所具备的强大感知能力来提高 agent 对复杂状态空间的表征效果；另外，可以在模型中加入视觉注意力机制 (Visual Attention Mechanism，VAM)，使得 agent 在不同状态下将注意力集中到有利于做出决策的区域，从而加速学习的进程。

2.3 基于策略梯度的深度强化学习

策略梯度是一种常用的策略优化方法，它通过不断计算策略期望总奖赏关于策略参数的梯度来更新策略参数，最终收敛于最优策略。因此在解决 DRL 问题时，可以采用参数为 □ 的深度神经网络来进行参数化表示策略，并利用策略梯度方法来优化策略。值得注意的是，在求解 DRL 问题时，往往第一选择是采取基于策略梯度的算法。原因是它能够直接优化策略的期望总奖赏，并以端对端的方式直接在策略空间中搜索最优策略，省去了繁琐的中间环节。因此与 DQN 及其改进模型相比，基于策略梯度的 DRL 方法适用范围更广，策略优化的效果也更好。

2.3.1 深度策略梯度的起源与发展

策略梯度方法是一种直接使用逼近器来近似表示和优化策略，最终得到最优策略的方法。该方法优化的是策略的期望总奖赏：

$$max_{\theta}E[R|\pi_{\theta}] \tag{2-14}$$

其中 $R=\sum_{t=0}^{T-1}$ 表示一个情节内所获得的奖赏总和。策略梯度最常见的思想是增加总奖赏较高情节出现的概率。策略梯度方法的具体过程如下：假设一个完整情节的状态、动作和奖赏的轨迹为：$\tau=(s_0,a_0,r_0,s_1,a_1,r_1,...,s_{T-1},a_{T-1},r_{T-1},s_T)$。则策略梯度表示为如下的形式：

$$g=R\nabla_{\theta}\sum_{t=0}^{T-1}log\pi(a_t|s_t;\theta) \tag{2-15}$$

利用该梯度调整策略参数：

$$\theta\leftarrow\theta+\alpha g \tag{2-16}$$

其中，α 是学习率，控制着策略参数更新的速率。式2-15中的 $\nabla_{\theta}\sum_{t=0}^{T-1}log\pi(a_t|s_t;\theta)$ 梯度项表示能够提高轨迹 τ 出现概率的方向，乘以得分函数 R 之后，可以使得单个情节内总奖赏越高的 τ 概率密度越大。即如果收集了很多总奖赏不同的轨迹，通过上述训练过程会使得概率密度向总奖赏更高的轨迹方向移动，最大化高奖赏轨迹 τ 出现的概率。然而在某些情形下，每个情节的总奖赏 R 都不为负，那么所有梯度 g 的值也都是大于等于 0 的。此时在训练过程中遇到每个轨迹 τ，都会使概率密度向正的方向“拉拢”，很大程度减缓了学习速度。这会使得梯度 g 的方差很大。因此可以对 R 使用某种标准化操作来降低梯度 g 的方差。该技巧使得算法能提高总奖赏 R 较大的轨迹 τ 的出现概率，同时降低总奖赏 R 较小的轨迹 τ 出现概率。根据上述思想，Williams 等人提出了 REINFORCE 算法，将策略梯度的形式统一为：

$$g=\nabla_{\theta}\sum_{t=0}^{T-1}log\pi(a_t|s_t;\theta)(R-b) \tag{2-17}$$

其中，b 是一个与当前轨迹 τ 相关的基线，通常设置为 R 的一个期望估计，目的是减小 R 的方差。可以看出，R 超过基准 b 越多，对应的轨迹 τ 被选中的概率越大。因此在大规模状态的 DRL 任务中，可以通过深度神经网络参数化表示策略，并采用传统的策略梯度方法来求解最优策略。

此外，优化策略的另一种思路是增加“好”的动作出现的概率。在 RL 中一般

是通过优势函数评价动作的好坏，因此可以利用优势函数项来构造策略梯度：

$$g = \nabla_\theta \sum_{t=0}^{T-1} \hat{A} log\pi(a_t|s_t;\theta) \tag{2-18}$$

其中 $\hat{A}_t$ 表示状态动作对 (s_t, a_t) 优势函数的一个估计，通常构造成以下形式：

$$\hat{A}_t^\gamma = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots - V(s_t) \tag{2-19}$$

其中，$\gamma \in [0,1]$ 表示折扣因子。此时带折扣的奖赏之和 $r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots$ 相当于式2-17中的 R，带折扣的状态值函数 $V(s_t)$ 相当于式2-17中的基准 b。当 $\hat{A}_t^\gamma > 0$ 时，会增加对应动作被选择的概率，而当 $\hat{A}_t^\gamma < 0$，会减少对应动作被选择的概率。

另外，Hafner 等人使用值函数来估计带折扣的奖赏和，进一步地缩小了梯度项地方差。此时进一步截断的 $\hat{A}_t^\gamma$ 表示为：

$$\hat{A}_t^\gamma = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{2-20}$$

类似地，两步截断的 $\hat{A}_t^\gamma$ 表示为：

$$\hat{A}_t^\gamma = r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) - V(s_t) \tag{2-21}$$

然而使用值函数估计带折扣的奖赏和，也会产生一定的估计偏差。为了缩小方差的同时还能保证偏差较小，Schulman 等人提出了广义优势函数 (generalized advantage function)：

$$\hat{A}_t^\gamma = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t-1}\delta_{T-1} \tag{2-22}$$

其中 $\delta_t = r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) - V(s_t)$。$\lambda$ 是一个调节因子，范围大小为 $0 < \lambda < 1$。当 λ 接近 0 时，$\hat{A}_t^\gamma$ 是低方差、高误差的；当 λ 接近于 1 时，$\hat{A}_t^\gamma$ 是高方差、低误差的。

基于广义优势函数的策略梯度方法的不足之处在于：在利用式2-16全局优化策略的过程中，很难确定一个合理的步长参数 α 来保证学习的稳定性。针对此问题，Schulman 等人提出了一种被称为区域信赖的策略最优化 (Trust Region Policy Optimization，) 方法。TRPO 的核心思想是：强制限制同一批次数据上新旧两种策略预测分布的 KL 差异，从而避免导致策略发生太大改变的参数更新步。为了将应用范围扩展到大规模状态空间的 DRL 任务中，TRPO 算法使用深度神经网络来参数化策略，在只接收原始输入图像的情况下实现了端对端的控制。实验表明，TRPO 在一系列 2D 场景下的机器人控制和 Atari 2600 游戏任务中都表现优异。此后，Schulman 等人又尝试将广义优势函数与 TRPO 方法相结合，在一系列 3D 场

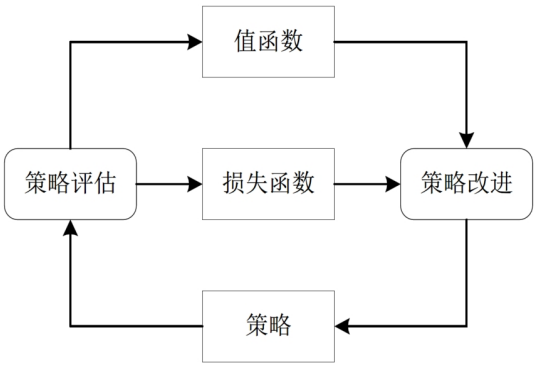

图 2-5 基于 AC 框架的深度策略梯度方法的学习结构

景下的机器人控制任务中取得了突破。

此外，深度策略梯度方法的另一个研究方向是通过增加额外的人工监督来促进策略搜索。例如著名的 AlphaGo 围棋机器人，先使用监督学习从人类专家的棋局中预测人类的走子行为，再用策略梯度方法针对赢得围棋比赛的真实目标进行精细的策略参数调整。然而在某些任务中是缺乏监督数据的，比如现实场景下的机器人控制，可以通过引导式策略搜索 (guided policy search) 方法来监督策略搜索的过程。在只接受原始输入信号的真实场景中，引导式策略搜索实现了对机器人的操控。

2.3.2 基于行动者评论家的深度策略梯度方法

2.3.1节中深度策略梯度方法的基本思想是通过各种策略梯度方法直接优化用深度神经网络参数化表示的策略。这类方法在每个迭代步，都需要采样批量大小为 N 的轨迹 $\tau_{i}^{N}{}_{i=1}$ 来更新策略梯度。然而在许多复杂的现实场景中，很难在线获得大量训练数据。例如在真实场景下机器人的操控任务中，在线收集并利用大量训练数据会产生十分昂贵的代价，并且动作连续的特性使得在线抽取批量轨迹的方式无法达到令人满意的覆盖面。以上问题会导致局部最优解的出现。针对此问题，可以将传统 RL 中的行动者评论家 (Actor-Critic，AC) 框架拓展到深度策略梯度方法中。图2-5展示了基于 AC 框架的深度策略梯度方法的学习结构。

下面阐述一种重要的基于 AC 框架的深度策略梯度算法。Lillicrap 等人利用 DQN 扩展 Q 学习算法的思路对确定性策略梯度 (Deterministic Policy Gradient，DPG) 方法进行改造，提出了一种基于 AC 框架的深度确定性策略梯度 (Deep Deterministic Policy Gradient，DDPG) 算法，该算法可用于解决连续动作空间上的 DRL 问题。DDPG 分别使用参数为 θ^{μ} 和 θ^{Q} 的深度神经网络来表示确定性策略 $a=\pi(s|\theta^{\mu})$ 和值函数 $Q(s,a|\theta^{Q})$。其中，策略网络用来更新策略，对应 AC 框架中

的行动者；值网络用来逼近状态动作对的值函数，并提供梯度信息，对应 AC 框架中的评论家。在 DDPG 中，目标函数被定义为带折扣的奖赏和：

$$J(\theta^{\mu}) = E_{\theta^{\mu}}[r_1 + \gamma r_2 + \gamma^2 r_3 + \dots] \tag{2-23}$$

然后，采用随机梯度下降方法来对目标函数进行端对端的优化。Silver 等人证明了目标函数关于 θ^{μ} 的梯度等价于 Q 值函数关于 θ^{μ} 的期望梯度：

$$\frac{\partial J(\theta^{\mu})}{\partial \theta^{\mu}} = E_s[\frac{\partial Q(s,a|\theta^{Q})}{\partial \theta^{\mu}}] \tag{2-24}$$

根据确定性策略 $a = \pi(s|\theta^{\mu})$ 可得：

$$\frac{\partial J(\theta^{\mu})}{\partial \theta^{\mu}} = E_s[\frac{\partial Q(s,a|\theta^{Q})}{\partial a}\frac{\partial \pi(s|\theta^{\mu})}{\partial \theta^{\mu}}] \tag{2-25}$$

通过 DQN 中更新值网络的方法来更新评论家网络，此时的梯度信息为：

$$\frac{\partial J(\theta^{Q})}{\partial \theta^{Q}} = E_{s,a,r,s' \sim D}[(y - Q(s,a|\theta^{Q}))\frac{\partial Q(s,a|\theta^{Q})}{\partial \theta^{Q}}] \tag{2-26}$$

其中，$y = r + \gamma Q(s', \pi(s'|\hat{\theta}^{\mu})|\hat{\theta}^{Q})$，$\hat{\theta}^{\mu}$ 和 $\hat{\theta}^{Q}$ 分别表示目标策略网络和目标值网络的参数。DDPG 使用经验回放机制从 D 中获得训练样本，并将由 Q 值函数关于动作的梯度信息从评论家网络传递给行动者网络。并根据式2-25沿着提升 Q 的方向更新策略网络的参数。

实验表明，DDPG 不仅在一系列连续动作空间的任务中表现稳定，而且求得最优解所需要的时间步也远远少于 DQN。与基于值函数的 DRL 方法相比，基于 AC 框架的深度策略梯度方法优化策略效率更高、求解速度更快。

然而在有噪声干扰的复杂环境下，策略一般都具有一定的随机性。DDPG 使用确定性的策略梯度方法。对于随机环境的场景，该方法并不适用。针对此问题，Heess 等人提出了一种适用于连续动作空间任务的通用框架，称为随机值梯度 (Stochastic Value Gradient，SVG) 方法。SVG 使用“再参数化”(re-parameterization) 的数学技巧来学习环境动态性的生成模型，将确定性策略梯度方法扩展为一种随机环境下的策略优化过程。Balduzzi 等人基于相容的值函数逼近器 (compatible function approximation) 理论，提出了值梯度反向更新 (Value-Gradient Backpropagation，GProp) 方法。Peng 等人融合多个策略网络和对应的值网络，提出了一种基于混合型行动者评论家指导 (Mixture of Actor Critic Experts，MACE) 的深度策略梯度方法。该方法在自适应机器人控制任务中取得了实质性的进展。MACE 相比于单个 AC 框架指导的深度策略梯度方法，有着更快的学习速度。随

后，Heess 等人使用循环神经网络，进一步扩展了 DPG 和 SVG 算法的适用范围，提出了循环确定性策略梯度 (Recurrent Deterministic Policy Gradient，RDPG) 和循环随机值梯度 (Recurrent Stochastic Value Gradient，RSVG) 方法。RDPG 和 RSVG 可以处理一系列部分可观察场景下连续动作的控制任务。Hausknecht 等人进一步将深度策略梯度方法扩展到了参数化的连续动作空间问题中。此后，Schulman 等人提出了一种形式化的随机计算图模型 (stochastic computation graphs)，开展了同时包含随机性和确定性操作的复杂深度策略梯度的研究。

2.3.3 异步优势行动者评论家算法

不同类型的深度神经网络为 DRL 中策略优化任务提供了高效运行的表征形式。为了缓解传统策略梯度方法与神经网络结合时出现的不稳定性，各类深度策略梯度方法 (如 DDPG、SVG 等) 都采用了经验回放机制来消除训练数据间的相关性。然而经验回放机制存在两个不足之处：

1. agent 与环境的每次实时交互都需要耗费很多的内存和计算力；
2. 经验回放机制要求 agent 采用离策略 (off-policy) 方法来进行学习，而离策略方法只能基于旧策略生成的数据进行更新。

针对这些问题，Mnih 等人根据异步强化学习 (Asynchronous Reinforcement Learning，ARL) 的思想，提出了一种轻量级的 DRL 框架，该框架可以使用异步的梯度下降法来优化网络控制器的参数，并可以结合多种 RL 算法。其中，异步的优势行动者评论家算法 (Asynchronous Advantage Actor-Critic，A3C) 在各类连续动作空间的控制任务上表现的最好。

具体地，A3C 算法利用 CPU 多线程的功能并行、异步地执行多个 agent。因此在任意时刻，并行的 agent 都将会经历许多不同的状态，去除了训练过程中产生的状态转移样本之间的关联性。因此这种低消耗的异步执行方式可以很好地替代经验回放机制。

A3C 算法在训练时降低了对硬件的要求。深度策略梯度算法十分依赖计算能力很强的图形处理器 (Graphics Processing Unit，GPU)，而 A3C 算法在实际的操作过程中只需要一个标准的多核 CPU。由表2-1可知，A3C 算法通过应用多线程技术，降低了模型对硬件的需求，在训练时间更少的情况下，A3C 算法在 Atari 2600 游戏任务上的平均性能有明显提升。而且 A3C 算法能够只根据原始的视觉输入学习到行走 3D 迷宫的有效策略。此外，A3C 算法还可以广泛应用于各种连续动作空间问题。综上所述，A3C 算法能够广泛应用于各种 2D、3D 离散和连续动作空间的任务，并且在这些任务中都取得了最佳的效果。这说明 A3C 是目前最通用和最

成功的一种 DRL 算法。当然，将 A3C 与近期的一些深度策略梯度算法相结合可能会进一步提升其性能。

表 2-1 不同的 DRL 模型在 57 个 Atari 游戏上的平均耗时以及游戏性能的提升

模型	训练条件	训练时间/天	平均性能提升
DQN	GPU	8	121.9%
DDQN	GPU	8	332.9%
Dueling DDQN	GPU	8	343.8%
Prioritized DQN	GPU	8	463.6%
A3C,FF	CPU	1	344.1%
A3C,FF	CPU	4	496.8%
A3C,LSTM	CPU	4	623.0%

除了基于值函数的 DRL 和基于策略梯度的 DRL 之外，还可以通过增加额外的人工监督来促进策略搜索的过程，即为基于搜索与监督的 DRL 的核心思想。蒙特卡洛树搜索 (Monte Carlo Tree Search，MCTS) 作为一种经典的启发式策略搜索方法，被广泛用于游戏博弈问题中的行动规划。因此在基于搜索与监督的 DRL 方法中，策略搜索一般是通过 MCTS 来完成的。例如 Google 公司的 AlphaGo 围棋算法就将深度神经网络和 MCTS 相结合，并取得了卓越的成就。

第三章 强化学习算法实践

3.1 强化学习常用仿真环境对比

3.1.1 OpenAI Gym 工具集

OpenAI 是一个非营利性质的人工智能公司，其致力于有益于人类的友好型 AI。Gym 是该公司推出的一套工具集，用于为强化学习算法提供一个通用的评价标准。该工具集不对用户的 agent 做出任何限制，这导致该工具集有着极强的兼容性，可以很好地兼容诸如 Tensorflow、Theano 或者 Keras 等主流机器学习库。

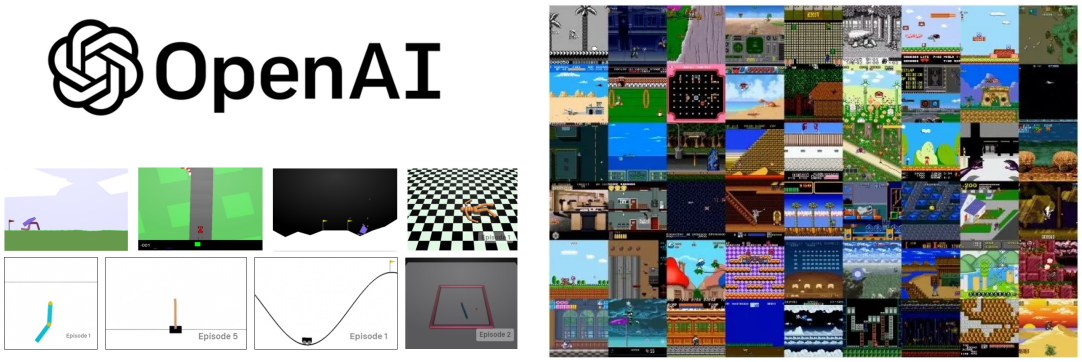

图 3-1 Gym 内置了多种多样的环境

Gym 工具集内置了多样化的测试环境，用户可以很方便地在这些环境中开发以及测试强化学习算法，这些环境共用同一套接口，这意味着用户可以用其编写通用性很强的算法。

如3-1所示，目前 Gym 已经包含了摆臂、自动驾驶汽车、简单机械臂、蜘蛛型机器人、人形机器人等环境，不仅如此，Gym 还包含了一个用于测试强化学习算法的游戏平台，涵盖了 70 余个雅达利和 30 多个世嘉平台游戏，并且 OpenAI 还发布了用于向 Gym 平台添加新游戏的工具。

Gym 极大地方便了学术界对强化学习算法及其泛化能力的研究，以往 RL 领域的研究主要集中在优化 agent 完成单个任务的能力上。

3.1.2 Robot Operating System(ROS) 平台

ROS(Robot Operating System, 机器人操作系统)，是专为机器人软件开发所设计出来的一套电脑操作系统架构。它是一个开源的元级操作系统 (后操作系统)，提供类似于操作系统的服务，包括硬件抽象描述、底层驱动程序管理、共用功能的

执行、程序间消息传递、程序发行包管理，它也提供一些工具和库用于获取、建立、编写和执行多机融合的程序。

ROS 的运行架构是一种使用 ROS 通信模块实现模块间 P2P(Peer to Peer, 点对点) 的松耦合的网络连接的处理架构，它执行若干种类型的通讯，包括：

- 基于服务的同步 RPC(Remote Progress Call, 远程过程调用) 通讯；
- 基于 Topic 的异步数据流通讯，还有参数服务器上的数据存储。

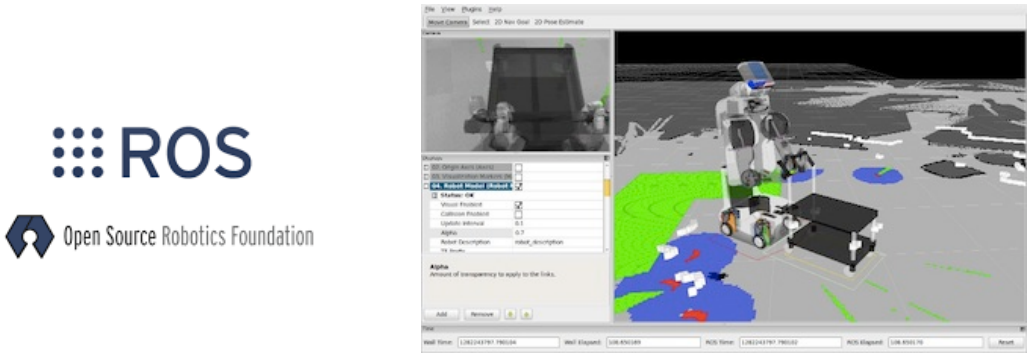

图 3-2 ROS 仿真环境

ROS 的首要设计目标是在机器人研发领域提高代码复用率。ROS 是一种分布式处理框架，这使可执行文件能被单独设计，并且在运行时松散耦合。这些过程可以封装到数据包 (Packages) 和堆栈 (Stacks) 中，以便于共享和分发。ROS 还支持代码库的联合系统。使得协作亦能被分发。这种从文件系统级别到社区一级的设计让独立地决定发展和实施工作成为可能。上述所有功能都能由 ROS 的基础工具实现。

为了实现“共享与协作”这一首要目标，人们制订了 ROS 架构中的其他支援性目标：

- 轻量级：ROS 是设计得尽可能方便简易，不必替换主框架与系统即可用于其他机器人软件框架中。
- 语言独立性：ROS 框架很容易在任何编程语言中执行。目前已经能在 Python 和 C++ 中顺利运行，同时添加有 Lisp、Octave 和 Java 语言库。
- 测试简单：ROS 有一个内建的单元/组合集测试框架，这使得集成调试和分解调试很容易。
- 扩展性：ROS 适合于大型实时系统与大型的系统开发项目。

3.1.3 Unity 游戏引擎

Unity 是一套跨平台的游戏引擎，可用于开发 Windows、MacOS、Linux 平台的单机游戏，或是 iOS、Android 移动设备的游戏。Unity 也可开发支持 WebGL 技术的网页游戏，或 PlayStation、XBox、Wii 主机上的游戏。目前 Unity 被广泛应用于虚拟现实应用的开发，是虚拟现实技术栈中不可或缺的一部分。

Unity 支持 PhysX 物理引擎、粒子系统，并且提供网络多人连接的功能，无需学习复杂的编程语言，匹配游戏制作上的各项需求。Unity 的推出降低游戏开发的门槛，即使是个人或小型团队制作游戏也不再是梦想。对于游戏公司而言，选择使用 Unity 引擎也可以缩短游戏的开发时间。

Unity 类似于 Director，Blender，Virtools 或 Torque Game Builder 等利用交互的图型化开发环境为首要方式的软件其编辑器运行在 Windows 和 Mac OS X 下，可发布游戏至 Windows、Wii、OSX 或 iOS 平台。也可以利用 Unity web player 插件发布网页游戏，支持 Mac 和 Windows 的网页浏览。它的网页播放器也被 Mac widgets 所支持。Unity3D 是一个用于创建诸如三维电子游戏、建筑可视化、实时三维动画等类型互动内容的综合型创作工具。

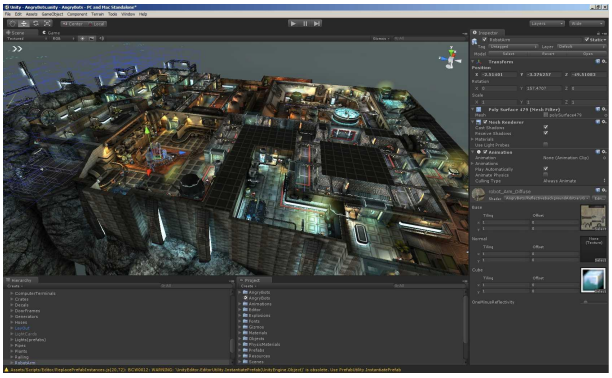

图 3-3 ROS 仿真环境

3.1.4 三者特性对比

严格来说，Unity 的主要用途是游戏开发，但是得益于其优异的物理仿真性能，将其用于强化学习算法的训练也并非不可能。表3-1将 Unity 与 Gym 和 ROS 的特性进行了对比，可以看到 Gym 由于专注于为学术界提供标准，其自定义性并不强，而 ROS 则为工业界而生，专业性和复杂度都相当高，其功能大而全，而 Unity 则很好地平衡了两者的优缺点，所以将 Unity 平台于强化学习算法结合相当值得一试。

表 3-1 Gym、ROS、Unity 三者特性对比

架构	复杂度	易用性	物理引擎	3D 模型导入	专业性
Gym	低	高	2D	不支持	高
ROS	高	低	3D	支持	高
Unity	低	高	2D/3D	支持	中

3.2 基于 A3C 架构的近端梯度优化算法

[TODO] PPO 算法原理介绍

3.3 二维机械臂环境中的 PPO 算法实践

[TODO] 基于 pyglet 游戏引擎仿真 PPO 算法

3.4 三维机械臂环境中的 PPO 算法实践

[TODO] 基于 unity 游戏引擎仿真 ppo 算法

第四章 全文总结与展望

4.1 全文总结

[TODO]

4.2 后续工作展望

[TODO]

致 谢

[TODO]

参考文献

[1] 王浩刚, 聂在平. 三维矢量散射积分方程中奇异性分析 [J]. 电子学报, 1999, 27(12): 68-71

[2] X. F. Liu, B. Z. Wang, W. Shao, et al. A marching-on-in-order scheme for exact attenuation constant extraction of lossy transmission lines[C]. China-Japan Joint Microwave Conference Proceedings, Chengdu, 2006, 527-529

[3] 竺可桢. 物理学 [M]. 北京: 科学出版社, 1973, 56-60

[4] 陈念永. 毫米波细胞生物效应及抗肿瘤研究 [D]. 成都: 电子科技大学, 2001, 50-60

[5] 顾春. 牢牢把握稳中求进的总基调 [N]. 人民日报, 2012 年 3 月 31 日

[6] 冯西桥. 核反应堆压力容器的 LBB 分析 [R]. 北京: 清华大学核能技术设计研究院, 1997 年 6 月 25 日

[7] 肖珍新. 一种新型排渣阀调节降温装置 [P]. 中国, 实用新型专利, ZL201120085830.0, 2012 年 4 月 25 日

[8] 中华人民共和国国家技术监督局. GB3100-3102. 中华人民共和国国家标准–量与单位 [S]. 北京: 中国标准出版社, 1994 年 11 月 1 日

[9] M. Clerc. Discrete particle swarm optimization: a fuzzy combinatorial box[EB/OL]. http://clere.maurice.free.fr/pso/Fuzzy_Discrere_PSO/Fuzzy_DPSO.htm, July 16, 2010

附　录

外文资料原文

Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov
OpenAI
{joschu, filip, prafulla, alec, oleg}@openai.com

Abstract

We propose a new family of policy gradient methods for reinforcement learning, which alternate between sampling data through interaction with the environment, and optimizing a "surrogate" objective function using stochastic gradient ascent. Whereas standard policy gradient methods perform one gradient update per data sample, we propose a novel objective function that enables multiple epochs of minibatch updates. The new methods, which we call proximal policy optimization (PPO), have some of the benefits of trust region policy optimization (TRPO), but they are much simpler to implement, more general, and have better sample complexity (empirically). Our experiments test PPO on a collection of benchmark tasks, including simulated robotic locomotion and Atari game playing, and we show that PPO outperforms other online policy gradient methods, and overall strikes a favorable balance between sample complexity, simplicity, and wall-time.

1 Introduction

In recent years, several different approaches have been proposed for reinforcement learning with neural network function approximators. The leading contenders are deep Q-learning [Mni+15], "vanilla" policy gradient methods [Mni+16], and trust region / natural policy gradient methods [Sch+15b]. However, there is room for improvement in developing a method that is scalable (to large models and parallel implementations), data efficient, and robust (i.e., successful on a variety of problems without hyperparameter tuning). Q-learning (with function approximation) fails on many simple problems[1] and is poorly understood, vanilla policy gradient methods have poor data effiency and robustness; and trust region policy optimization (TRPO) is relatively complicated, and is not compatible with architectures that include noise (such as dropout) or parameter sharing (between the policy and value function, or with auxiliary tasks).

This paper seeks to improve the current state of affairs by introducing an algorithm that attains the data efficiency and reliable performance of TRPO, while using only first-order optimization. We propose a novel objective with clipped probability ratios, which forms a pessimistic estimate (i.e., lower bound) of the performance of the policy. To optimize policies, we alternate between sampling data from the policy and performing several epochs of optimization on the sampled data.

Our experiments compare the performance of various different versions of the surrogate objective, and find that the version with the clipped probability ratios performs best. We also compare PPO to several previous algorithms from the literature. On continuous control tasks, it performs better than the algorithms we compare against. On Atari, it performs significantly better (in terms of sample complexity) than A2C and similarly to ACER though it is much simpler.

[1] While DQN works well on game environments like the Arcade Learning Environment [Bel+15] with discrete action spaces, it has not been demonstrated to perform well on continuous control benchmarks such as those in OpenAI Gym [Bro+16] and described by Duan et al. [Dua+16].

2 Background: Policy Optimization

2.1 Policy Gradient Methods

Policy gradient methods work by computing an estimator of the policy gradient and plugging it into a stochastic gradient ascent algorithm. The most commonly used gradient estimator has the form

$$\hat{g} = \hat{\mathbb{E}}_t\Big[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\hat{A}_t\Big] \tag{1}$$

where π_θ is a stochastic policy and $\hat{A}_t$ is an estimator of the advantage function at timestep t. Here, the expectation $\hat{\mathbb{E}}_t[\ldots]$ indicates the empirical average over a finite batch of samples, in an algorithm that alternates between sampling and optimization. Implementations that use automatic differentiation software work by constructing an objective function whose gradient is the policy gradient estimator; the estimator $\hat{g}$ is obtained by differentiating the objective

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t\Big[\log \pi_\theta(a_t \mid s_t)\hat{A}_t\Big]. \tag{2}$$

While it is appealing to perform multiple steps of optimization on this loss L^{PG} using the same trajectory, doing so is not well-justified, and empirically it often leads to destructively large policy updates (see Section 6.1; results are not shown but were similar or worse than the "no clipping or penalty" setting).

2.2 Trust Region Methods

In TRPO [Sch+15b], an objective function (the "surrogate" objective) is maximized subject to a constraint on the size of the policy update. Specifically,

$$\underset{\theta}{\text{maximize}} \quad \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right] \tag{3}$$

$$\text{subject to} \quad \hat{\mathbb{E}}_t[\mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]] \le \delta. \tag{4}$$

Here, θ_{old} is the vector of policy parameters before the update. This problem can efficiently be approximately solved using the conjugate gradient algorithm, after making a linear approximation to the objective and a quadratic approximation to the constraint.

The theory justifying TRPO actually suggests using a penalty instead of a constraint, i.e., solving the unconstrained optimization problem

$$\underset{\theta}{\text{maximize}}\, \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t - \beta\,\mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]\right] \tag{5}$$

for some coefficient β. This follows from the fact that a certain surrogate objective (which computes the max KL over states instead of the mean) forms a lower bound (i.e., a pessimistic bound) on the performance of the policy π. TRPO uses a hard constraint rather than a penalty because it is hard to choose a single value of β that performs well across different problems—or even within a single problem, where the the characteristics change over the course of learning. Hence, to achieve our goal of a first-order algorithm that emulates the monotonic improvement of TRPO, experiments show that it is not sufficient to simply choose a fixed penalty coefficient β and optimize the penalized objective Equation (5) with SGD; additional modifications are required.

外文资料译文

近端策略优化算法

1 摘要

本文提出一种新的基于策略梯度的强化学习算法，此算法交替地执行数据采样和策略优化过程，在数次与环境交互并取得批量采样数据之后，此算法会使用随机梯度下降法优化“代理”目标函数，如此循环往复。传统的策略梯度算法每采样一次数据就会执行一次梯度更新，我们提出了一种新的目标函数，可以采用多轮小批量数据进行梯度更新。我们提出的这种名为近端策略优化的算法，基于置信域策略优化算法发展而来，但与之相比更易于实现、泛化能力更强以及拥有更好的采样复杂度。我们在多个基准测试任务上测试了该算法，包括仿真机械运动和雅达利游戏，并且发现该算法取得了相比于其他在线策略梯度算法更好的性能，在采样复杂度、朴素性和时间复杂度上取得了很好的平衡。

2 介绍

近些年，强化学习领域出现了多种基于神经网络近似的算法，其中最具代表性的算法有：深度 Q 学习 [Mni+15]、普通策略梯度算法 [Mni+16] 和置信域策略梯度算法。然而，这些算法在可扩展性、效率和鲁棒性上还有很大的提升空间。Q 学习 (采用函数模拟的) 不仅在很多简单任务上表现很差，而且可解释性很弱，普通策略梯度算法效率和鲁棒性都无法令人满意；而置信域策略优化算法相对来说太过复杂，并且无法很好地兼容包含噪声 (如随机失活) 或者包含参数共享结构 (策略函数与值函数共享或者多任务参数共享) 的架构。

本文旨在提出一种比 TRPO 算法拥有更高的效率和更可靠的性能的仅使用了一阶优化函数的高效算法。我们提出的目标函数使用裁剪概率比率来评估策略性能的下界。为了优化策略，我们交替地批量采样数据并用这些数据优化目标函数。

我们在实验中对比了多种版本的代理目标函数，并且发现使用裁剪概率比率的版本表现最好。同时，我们也将该算法与其他文献中的算法进行了对比。在连续空间任务上，该算法比竞争算法表现都好。而在雅达利游戏上，该算法远远超出了 A2C 的表现，并在更低的复杂度上取得了与 ACER 类似的成绩。

3 背景知识：策略优化理论

3.1 策略梯度算法

策略梯度算法使用一个评价函数来评价策略梯度，并且使用随机梯度下降对该评价函数进行优化。最广泛使用的评价函数如式 (1) 所示。

$$\hat{g} = \hat{E}_t[\nabla_\theta log\pi_\theta(a_t|s_t)\hat{A}_t] \tag{1}$$

其中 π_θ 是一个随机变量，表示策略 (policy)，而 $\hat{A}_t$ 则是在时间步 t 时的优势评价指标。数学期望 $\hat{E}_t$ 表示有限数量样本的经验平均值，它通常在数据采样和优化时变化。为了能够使用自动求导来解决问题，通常需要构造一个目标函数，这个目标函数的梯度便是策略梯度评价指标；评价指标 $\hat{g}$ 可以通过对以下目标函数求导得到：

$$L^{PG}(\theta) = \hat{E}_t[log\pi_\theta(a_t|s_t)\hat{A}_t] \tag{2}$$

尽管我们希望损失函数 L^{PG} 在多步以后沿着同样的轨迹优化，但事实却并不如此，这种优化方式往往引起巨量的策略更新。

3.2 基于置信域的方法

在 TRPO 算法中，最大化目标函数时，使用策略更新的尺寸来约束，具体地，

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & \hat{E}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t] \\ \text{subject to} \quad & \hat{E}_t[KL[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]] \leq \delta \end{aligned}$$

其中，θ_{old} 是更新前的策略参数向量。在对目标函数进行线性近似并对约束条件进行二阶近似后，这个问题可以使用共轭梯度算法来有效地解决。

TRPO 算法在进行策略调整时，使用了一个惩罚项来代替约束，也就是说，求解包含系数 β 的非约束最优化问题

$$\underset{\theta}{\text{maximize}}\ \hat{E}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t] - \beta KL[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)] \tag{3}$$

该问题成立的前提是，存在这样一个事实：一个确定的代理目标函数 (通过计算状态的 KL 散度而不是其平均值) 形成了策略 π 的下界。TRPO 选用一个强约束条件而不是惩罚项来求解最优化问题，这是因为很难确定一个能够在多种问题上都取得很好效果的 β 值——甚至在单一问题上也很难确定合适的 β 值。因此，为了达到我们使用一阶函数改进 TRPO 算法的目标，实践证明，简单地选取一个固定的系数 β 然后去优化惩罚目标式 (3) 是远远不够的，我们需要进行更多的改动。