第二章　深度强化学习

在之前的章节中，我们主要关注于监督学习，而监督学习一般需要一定数量的带标签的数据。在很多的应用场景中，通过人工标注的方式来给数据打标签的方式往往行不通。比如我们通过监督学习来训练一个模型可以来自动下围棋，就需要将当前棋盘的状态作为输入数据，其对应的最佳落子位置（动作）作为标签。训练一个好的模型就需要收集大量的不同棋盘状态以及对应动作。这种做法实践起来比较困难，一是对于每一种棋盘状态，即使是专家也很难给出“正确”的动作，二是获取大量数据的成本往往比较高。对于下棋这类任务，虽然我们很难知道每一步的“正确”动作，但是其最后的结果（即赢输）却很容易判断。因此，我们希望可以通过大量的模拟数据，通过最后的结果（奖励）来倒推每一步棋的贡献，从而学习出“最佳”下棋策略，这就是强化学习。

强化学习（Reinforcement Learning，RL），也叫增强学习，是指一类从（与环境）交互中不断学习的问题以及解决这类问题的方法。强化学习问题可以描述为一个智能体从与环境的交互中不断学习以完成特定目标（比如取得最大奖励值）。强化学习就是智能体不断与环境进行交互，并根据经验调整其策略来最大化其长远的所有奖励的累积值。和深度学习类似，强化学习中的关键问题也是**贡献度分配问题** [Minsky, 1963]，每一个动作并不能直接得到监督信息，需要通过整个模型的最终监督信息（奖励）得到，并且有一定的延时性。

贡献度分配问题即一个系统中不同的组件（Components）对最终系统输出结果的贡献或影响。

强化学习也是机器学习中的一个重要分支。强化学习和有监督学习的不同在于，强化学习不需要给出“正确”的策略，只需要取得最大化的预期利益。现代强化学习可以追溯到两个来源：一个是心理学中的行为主义理论，即有机体如何在环境给予的奖励或惩罚的刺激下，逐步形成对刺激的预期，产生能获得最大利益的习惯性行为；另一个是控制论领域的最优控制问题，即在满足一定约束条件下，寻求最优控制策略，使得性能指标取极大值或极小值。

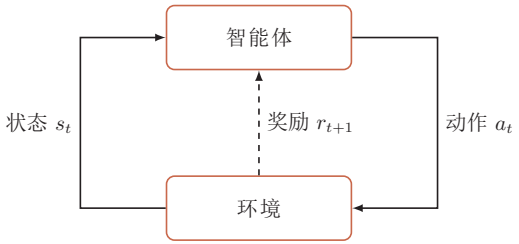

图 2.1: 智能体与环境的交互。

2.1 强化学习

我们先来描述下强化学习的任务定义。在强化学习中，有两个可以进行交互的对象：智能体和环境。

- **智能体**（Agent）可以感知外界环境的状态（State）和奖励反馈（Reward），并进行学习和决策。智能体的决策功能是指根据外界环境的状态来做出不同的动作（Action），而学习功能是指根据外界环境的奖励来调整策略。
- **环境**（Environment）是智能体外部的所有事物，并受智能体动作的影响而改变其状态，并反馈给智能体相应的奖励。

在强化学习中的基本要素包括：

- 环境的状态集合：$\mathcal{S}$；
- 智能体的动作集合：$\mathcal{A}$；
- 状态转移概率：$p(s'|s,a)$，即智能体根据当前状态 s 做出一个动作 a 之后，下一个时刻环境处于不同状态 s' 的概率；
- 即时奖励：$R:\mathcal{S}\times\mathcal{A}\times\mathcal{S}\to\mathbb{R}$，即智能体根据当前状态做出一个动作之后，环境会反馈给智能体一个奖励，这个奖励和动作之后下一个时刻的状态有关。

智能体与环境的交互如图2.1所示。为了简单起见，我们将智能体与环境的交互看作是离散的时间序列。智能体从感知到的初始环境 s_0 开始，然后决定做一个相应的动作 a_0，环境相应地发生改变到新的状态 s_1，并反馈给智能体一个

即时奖励r_1，然后智能体又根据状态s_1做一个动作a_1，环境相应改变为s_2，并反馈奖励r_2。这样的交互可以一直进行下去。

$$s_0, a_0, s_1, r_1, a_1, \cdots, s_{t-1}, r_{t-1}, a_{t-1}, s_t, r_t, \cdots, \tag{2.1}$$

其中r_t是t时刻的即时奖励，其依赖于$t-1$时刻的状态和动作，以及t时刻的状态，

$$r_t = R(s_{t-1}, a_{t-1}, s_t). \tag{2.2}$$

我们期望智能体执行一系列的动作来获得尽可能多的奖励。

2.1.1 典型例子

强化学习广泛应用在很多领域，比如电子游戏、棋类游戏、迷宫类游戏、控制系统、推荐等。这里我们介绍几个比较典型的强化学习例子。

多臂赌博机问题 给定K个赌博机，拉动每个赌博机的拉杆（arm），赌博机会按照一个事先设定的概率掉出一块钱或不掉钱。每个赌博机掉钱的概率不一样。**多臂赌博机问题**（Multi-armed bandit problem）是指，给定有限的机会次数T，如何玩这些老虎机才能使得期望累积收益最大化。多臂赌博机问题在广告推荐、投资组合等领域有着非常重要的应用。

也称为**K**臂赌博机问题（K-armed bandit problem）。

悬崖行走问题 在一个网格世界（Grid World）中，每个格子表示一个状态，其中有些格子为悬崖（Cliff）。如图2.2所示，有一个醉汉，从左下角的开始位置S，走到右下角的目标位置E。在每个状态$(i,j), 1 \le i \le 3, 1 \le j \le 6$，格子$(2,1)$到$(5,1)$是悬崖。如果走到悬崖，醉汉会跌落悬崖并死去。醉汉可以选择行走的路线，即在每个状态时，选择行走的方向：上下左右。动作空间$\mathcal{A} = \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$。但每走一步，都有一定的概率滑落到周围其他的格子。醉汉的目标是如何安全地到达目标位置。

2.1.2 总回报

对于一个策略π，在每一个时刻t，智能体感知到环境的状态$s_t \in \mathcal{S}$，然后根据策略选择一个动作$a_t = \pi(s_t)$。受动作的影响，环境变化到一个新的状态s_{t+1}，然后反馈与此改变(s_t, a_t, s_{t+1})相关联的即时奖励r_{t+1}。

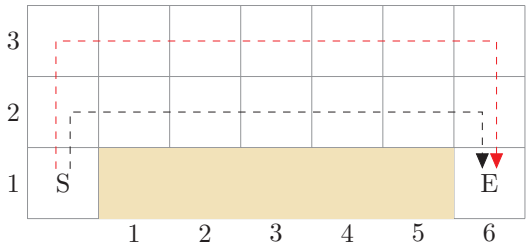

图 2.2: 醉汉悬崖问题

智能体的学习目标是找到一个策略，并根据一个可以得到尽可能多的累积奖励，即总回报（Return）。假设环境中有一个或多个特殊的**终止状态**（Terminal State），从起始状态到终止状态总共走了 T 步，其总回报为

$$G = \sum_{t=1}^{T} r_t. \tag{2.3}$$

如果环境中没有终止状态，即 $T = \infty$，其总回报也是无穷大。为了解决这个问题，我们可以引入一个折扣因子来降低远期回报的权重。因此，一个有折扣的回报（Discounted Return）定义为

$$G = \sum_{t=0}^{\infty} \gamma^t r_{t+1}, \tag{2.4}$$

其中，$\gamma \in [0, 1]$ 是折扣率。引入折扣因子也意味着智能体更在意短期回报而不是长期回报。

2.1.3 策略

智能体根据环境状态 s 来决定下一步的动作 a，就是智能体的**策略**（Policy）。策略可以分为确定性策略（Deterministic Policy）和随机性策略（Stochastic Policy）两组。

确定性策略是从状态空间到动作空间的映射函数 $\pi : \mathcal{S} \to \mathcal{A}$。随机性策略表示在给定状态时，动作空间的概率分布 $\pi(a|s) = p(a|s)$，$\sum_a \pi(a|s) = 1$。

为什么需要随机性策略呢？在很多强化学习的场合只有确定性策略是不可行的。比如在围棋中，面对一个空棋盘，确定性策略总是会在同一个位置上下

棋，会导致你的策略很容易被对手预测。此外，在“利用-探索”平衡中，也需要一定的随机性来“探索”环境。

2.2 马尔可夫决策过程

下篇我们正式开始学习强化学习，首先介绍一下马尔可夫决策过程。

马尔可夫过程（Markov Process）是指随时间变动的变量 $s_0, s_1, \cdots, s_t \in \mathcal{S}$，其过程具有马尔可夫性，即下一个时刻的状态只取决于当前状态。

$$p(s_{t+1}|s_t, \cdots, s_0) = p(s_{t+1}|s_t), \tag{2.5}$$

其中，$p(s_{t+1}|s_t)$ 为状态转移概率，$\sum_{s_{t+1}\in\mathcal{S}} p(s_{t+1}|s_t) = 1$。

对于离散的状态集合，状态转移概率 $p(s_{t+1}|s_t)$ 可以用一个状态转移矩阵来表示。

马尔可夫决策过程（Markov Decision Process，MDP）在马尔可夫过程的运行过程中加入一个额外的变量：动作 a，即下一个时刻的状态 s_{t+1} 和当前时刻的状态 s_t 以及动作 a_t 相关。

$$p(s_{t+1}|s_t, a_t, \cdots, s_0, a_0) = p(s_{t+1}|s_t, a_t), \tag{2.6}$$

其中，$p(s_{t+1}|s_t, a_t)$ 为状态转移概率，选择什么样的动作由策略 π 来决定，$a_t = \pi(s_t)$。

2.2.1 值函数

状态值函数 给定一个策略 π，从状态 s 开始，执行策略 π，我们可以得到一个动作和状态的序列 $(s_t, a_t, r_{t+1}), 0 \le t < \infty$。因为策略和状态转移都有一定的随机性，每次试验的序列都可以不一样，其收获的总回报也不一样。那么，从状态 s 开始，执行策略 π 得到的期望总回报为

$$V^\pi(s) = \sum_{(s_0=s, a_0, s_1, a_1, \cdots)} p(s_0 = s, a_0, s_1, a_1, \cdots) \sum_{t=0}^{\infty} \gamma^t r_{t+1} \tag{2.7}$$

$$= \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | \pi, s_0 = s], \tag{2.8}$$

其中，$a_t = \pi(s_t)$。

我们把 $V^\pi(s)$ 称为**状态值函数**（State Value Function）。根据马尔可夫性，我们将 $V^\pi(s)$ 展开可得到

$$\begin{aligned}
V^\pi(s) &= \mathbb{E}_\pi[r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots | \pi, s_0 = s] \\
&= \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}} p(s'|s,a) \mathbb{E}_\pi[R(s,a,s') + \gamma(r_2 + \gamma^1 r_3 + \cdots)|\pi, s_1 = s'] \\
&= \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}} p(s'|s,a) \Big(R(s,a,s') + \gamma \mathbb{E}_\pi[r_2 + \gamma^1 r_3 + \cdots | \pi, s_1 = s'] \Big) \\
&= \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}} p(s'|s,a) \Big(R(s,a,s') + \gamma V^\pi(s') \Big). \qquad (2.9)
\end{aligned}$$

公式 (2.9) 也称为**贝尔曼方程**（Bellman Equation），表示当前状态的值函数可以通过下个状态的值函数来计算。如果已知策略 $\pi(a|s)$，状态转移概率 $p(s'|s,a)$ 和奖励 $R(s,a,s')$，我们就可以通过迭代的方式来计算 $V^\pi(s)$。由于折扣的原因，迭代一定步数后，每个状态的值函数就会固定不变。

贝尔曼方程因提出者 Richard Bellman 而得名，也叫做“动态规划方程”。Richard Bellman（1920－1984），美国应用数学家，美国国家科学院院士，和动态规划的创始人。

状态-动作值函数 为了更方便优化策略，我们需要知道在状态 s 时，哪个动作会导致其总回报最大。因此，我们定义一个**状态-动作值函数**（State-Action Value Function）$Q^\pi(s,a)$，即初始状态为 s，并执行动作 a 后的所能得到的期望总回报。

$$Q^\pi(s,a) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | \pi, s_0 = s, a_0 = a] \qquad (2.10)$$

$$= \sum_{s'\in\mathcal{S}} p(s'|s,a) \Big(R(s,a,s') + \gamma V^\pi(s') \Big). \qquad (2.11)$$

状态值函数 $V^\pi(s)$ 是 $Q^\pi(s,a)$ 关于动作 a 的期望。

$$V^\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) Q^\pi(s,a). \qquad (2.12)$$

因此，根据公式 (2.11) 和 (2.12)，$Q^\pi(s,a)$ 也可以表示为

$$Q^\pi(s,a) = \sum_{s'\in\mathcal{S}} p(s'|s,a) \Big(R(s,a,s') + \gamma \sum_{a'\in\mathcal{A}} \pi(a'|s') Q^\pi(s',a') \Big). \qquad (2.13)$$

公式 (2.13) 也是一个贝尔曼方程。

2.2.2 最优策略

如果存在一个最优的策略π^*，其在所有状态上的期望回报最大，我们就成π^*为**最优策略**。

$$\forall s, \qquad \pi^* = \arg\max_{\pi} V^{\pi}(s). \tag{2.14}$$

最优策略π^*对应的值函数称为**最优值函数**。我们分别用$V^*(s)$和$Q^*(s,a)$来表示**最优状态值函数**和**最优状态-动作值函数**，

$$\forall s, \qquad V^*(s) = V^{\pi^*}(s), \tag{2.15}$$

$$\forall s, a, \qquad Q^*(s,a) = Q^{\pi^*}(s,a). \tag{2.16}$$

最优状态值函数$V^*(s)$和最优状态-动作值函数$Q^*(s,a)$的关系为：

$$V^*(s) = \max_a Q^*(s,a). \tag{2.17}$$

同样，最优状态值函数$V^*(s)$和最优状态-动作值函数$Q^*(s,a)$也可以进行迭代计算。

$$V^*(s) = \max_a \sum_{s' \in \mathcal{S}} p(s'|s,a)\Big(R(s,a,s') + \gamma V^*(s')\Big), \tag{2.18}$$

$$Q^*(s,a) = \sum_{s' \in \mathcal{S}} p(s'|s,a)\Big(R(s,a,s') + \gamma \max_{a'} Q^*(s',a')\Big), \tag{2.19}$$

这两个公式也都称为**贝尔曼最优方程**（Bellman Optimality Equation）。

参见习题(??)，第??页。

2.3 基于模型的强化学习算法

强化学习的目的求解马尔可夫决策过程(MDP)的最优策略。从贝尔曼最优方程可知，求解最优的策略需要知道马尔可夫决策过程的状态转移概率$p(s'|s,a)$和奖励$R(s,a,s')$。也就是说模型已知，我们直接可以通过贝尔曼最优方程来迭代计算最优策略。

这里，模型就是指马尔可夫决策过程。

我们把模型已知时的强化学习算法，都称为**基于模型的强化学习**（Model-based Reinforcement Learning）算法。常用的基于模型的强化学习算法主要有策略迭代算法和值迭代算法。

基于模型的强化学习，也叫作模型相关的强化学习，或有模型的强化学习。

2.3.1 策略迭代

策略迭代（Policy Iteration）算法是直接对策略进行迭代优化。先随机初始化一个策略，根据该策略，计算每个状态的值函数，然后根据这些状态的值函数和Bellman最优方程来计算新的策略。一直反复迭代直到收敛。

策略迭代如算法2.1所示。每次迭代可以分为两步：

1. 策略评估 (Policy Evaluation)：计算当前策略下，每个状态的值函数，即算法2.1中的3-6步。策略评估可以通过贝尔曼方程（公式(2.9)）进行迭代计算$V^\pi(s)$。如果状态数有限时，也可以通过直接求解Bellman方程来得到得到$V^\pi(s)$
2. 策略改进 (Policy Improvement)：根据状态价值得到新策略，即算法2.1中的7-8步。

算法 2.1: 策略迭代算法

输入: MDP 四元组：$\mathcal{S}, \mathcal{A}, P, R$;
折扣率：γ;
1 初始化：$\forall s, \forall a, \pi(a|s) = \frac{1}{|\mathcal{A}|}$;
2 **repeat**
// 策略评估
3 **repeat**
4 对于每一个状态s，根据贝尔曼方程（公式(2.9)），计算
5 $V^\pi(s) \leftarrow \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}} p(s'|s,a)\Big(R(s,a,s') + \gamma V^\pi(s')\Big)$;
6 **until** $\forall s$，$V^\pi(s)$ 收敛;
// 策略改进
7 根据公式(2.13)，计算$Q(s,a)$;
8 $\forall s, \pi(s) = \arg\max_a Q(s,a)$;
9 **until** $\forall s$，$\pi(s)$ 收敛;
输出: 策略π

在策略迭代中，每次迭代的时间复杂度最大为$O(|\mathcal{S}|^3|\mathcal{A}|^3)$，最大迭代次数为$|\mathcal{A}|^{|\mathcal{S}|}$。

2.3.2 值迭代

策略迭代中的策略评估和策略改进是交替轮流进行。其中，策略评估也是通过一个内部迭代来进行计算，其计算量比较大，而从降低了策略迭代算法的效率。事实上，我们不需要每次计算出每次策略对应的精确的值函数，也就是说内部迭代不需要执行到完全收敛。此外，也不需要每次显式地计算出策略π。更极端一些，将策略评估和策略改进同时进行，直接优化贝尔曼最优方程（公式(2.18)），即直接迭代计算最优值函数。这种方法称为**值迭代**（Value Iteration）算法。

值迭代如算法2.2所示。

算法 2.2: 值迭代算法

输入: MDP 四元组：$\mathcal{S}, \mathcal{A}, P, R$;
折扣率：γ;
收敛阈值：γ;

初始化：$\forall s \in \mathcal{S}, V(s) = 0$;
repeat
  $\forall s, V(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} p(s'|s,a)\Big(R(s,a,s') + \gamma V(s')\Big)$;
until $\forall s$，$V(s)$ 收敛;
根据公式 (2.13) 计算 $Q(s,a)$;
$\forall s, \pi(s) = \arg\max_a Q(s,a)$;

输出: 策略 π

值迭代中，每次迭代的时间复杂度最大为$O(|\mathcal{S}|^2|\mathcal{A}|)$，但迭代次数要比策略迭代算法更多。

策略迭代是根据贝尔曼方程来更新值函数，并根据当前的值函数来改进策略。而值迭代算法是直接使用贝尔曼最优方程来更新值函数，收敛时的值函数就是最优的值函数，其对应的策略也就是最优的策略。

值迭代和策略迭代都需要经过非常多的迭代次数才能完全收敛。在实际应用中，可以不必等到完全收敛。这样，当状态和动作数量有限时，经过有限次迭代就可以能收敛到近似最优策略。

基于模型的强化学习算法实际上是一种动态规划方法。在实际应用中有以下两点限制。

一是要求模型已知,即要事先给出马尔可夫决策过程的状态转移概率$p(s'|s,a)$和奖励函数$R(s,a,s')$，这个要求很难满足。如果是事先不知道模型，但仍然希望通过基于模型的学习算法，也可以通过与环境交互来学习出状态转移概率和奖励函数。一个简单的计算模型的方法为R-max [Brafman and Tennenholtz, 2002]，通过随机游走的方法来探索环境。每次随机一个策略并执行，然后收集状态转移和奖励的样本。在收集一定的样本后，就可以通过统计或监督学习来重构出马尔可夫决策过程。但是，这种基于采样的重构过程的复杂度也非常高，只能应用于状态数非常少的场合。

二是效率问题，当状态数量较大的时候，算法的效率比较低。但在实际应用中，很多问题的状态数量和动作数量非常多。比如，围棋有$19 \times 19 = 361$个位置，每个位置有黑子、白子或无子三种状态，整个棋局有$3^{361} \approx 10^{170}$种状态。动作（即落子位置）数量为361。不管是值迭代还是策略迭代，以当前计算机的计算能力，根本无法计算。一个有效的方法是通过一个函数（比如神经网络）来近似计算值函数，以减少复杂度，并提高泛化能力。

参见第2.5.2节，第32页。

2.4 模型无关的强化学习

在很多应用场景中，马尔可夫决策过程的状态转移概率$p(s'|s,a)$和奖励函数$R(s,a,s')$都是未知的。在这种情况下，我们一般需要智能体和环境进行交互，并收集一些样本。然后再根据这些样本来求解马尔可夫决策过程最优策略。这样模型未知，基于采样的学习算法都称为**模型无关的强化学习**（Model-free Reinforcement Learning）算法。常用的模型无关的强化学习算法主要有蒙特卡罗采样和时序差分学习两种方法。

模型无关的强化学习，也叫作无模型的强化学习。

2.4.1 蒙特卡罗采样方法

根据公式(2.10)，状态-动作值函数$Q^\pi(s,a)$为初始状态为s，并执行动作a后的所能得到的期望总回报。在已知模型时，可以通过动态规划的方法来计算。如果模型未知，一个直接的方法就是通过采样来计算。即固定策略π，从状态s，执行动作a开始，然后通过随机游走的方法来探索环境，并计算其得到的总回报。假设我们进行T次试验，得到T个总回报$G_t, 1 \leq t \leq M$。状态-动作值函

数 $Q^\pi(s,a)$ 可以通过这 T 次试验的总回报的平均 $\bar{G}$ 来近似。

$$\hat{Q}^\pi(s,a) = \bar{G} = \frac{1}{T}\sum_{t=1}^{M} G_t, \tag{2.20}$$

当 $T \to \infty$ 时，$\hat{Q}^\pi(s,a) \to Q^\pi(s,a)$。

在通过蒙特卡罗方法近似估计出状态-动作值函数 $Q^\pi(s,a)$ 之后，就可以通过类似策略迭代的方法来进行策略改进。然后在新的策略下重新通过蒙特卡罗方法来估计状态-动作值函数，并不断重复，直至收敛。

为了能更好地估计出值函数，就需要采样的轨迹尽可能覆盖所有的状态和动作。但是如果采用确定性的策略，每次试验得到的轨迹是一样的，因此也无法进一步改进策略。这样情况仅仅是对当前策略的*利用*（Exploitation），而缺失了对环境的*探索*（Exploration）以找到更好的策略。

这也可以看做是一个**多臂赌博机**问题。

为了平衡*利用*和*探索*，我们可以采用 **ϵ-贪心法**（ϵ-Greedy Method）。对于一个目标策略 π，其对应的 $\epsilon-$ 贪心法策略为

$$\pi^\epsilon(s) = \begin{cases} \pi(s), & \text{按概率} 1-\epsilon, \\ \text{随机选择}\mathcal{A}\text{中的动作}, & \text{按概率}\epsilon. \end{cases} \tag{2.21}$$

这样，ϵ-贪心法将一个仅利用的策略转为带探索的策略。每次选择动作 $\pi(s)$ 的概率为 $1-\epsilon+\frac{1}{|\mathcal{A}|}$，其它动作的概率为 $\frac{1}{|\mathcal{A}|}$。

将策略 $\pi^\epsilon(s)$ 代入到蒙特卡罗方法中进行评估并优化。由于采样的策略是 ϵ-贪心策略 $\pi^\epsilon(s)$ 而不是目标策略 $\pi(s)$，因此，我们不断改进策略也是 $\pi^\epsilon(s)$。我们把这种采样与改进的都是 ϵ-贪心策略的强化学习方法叫做**同策略**（On Policy）方法。但事实上，我们希望评估改进的策略是目标策略 π，而采样来自于 $\pi^\epsilon(s)$。这刚好符合重要性采样（Importance Sampling）的条件。因此，我们可以加权平均来对目标策略 π 进行评估。这种采样与改进分别使用不同策略的强化学习方法叫做**异策略**（Off Policy）方法。

总之，蒙特卡罗采样方法需要拿到完整的轨迹，才能对策略进行评估并更新模型，因此效率也比较低。

2.4.2 时序差分学习方法

时序差分学习（Temporal-Difference Learning）结合了动态规划和蒙特卡罗方法，比仅仅使用蒙特卡罗采样方法的效率要高很多 [Sutton and Barto, 2011]。

时序差分学习是模拟一段轨迹，每行动一步(或者几步)，就利用贝尔曼方程来评估行动前状态的价值。当时序差分学习中每次更新的动作数为最大步数时，就等价于蒙特卡罗方法。

首先，我们将蒙特卡罗方法中来评估值函数$\hat{Q}^{\pi}(s,a)$改为增量的计算方法，假设第T试验后值函数$\hat{Q}_T^{\pi}(s,a)$的平均为

$$\hat{Q}_T^{\pi}(s,a) = \frac{1}{T}\sum_{t=1}^{M} G_t \tag{2.22}$$

$$= \frac{1}{T}(G_T + \sum_{m=1}^{T-1} G_m) \tag{2.23}$$

$$= \frac{1}{T}\Big(G_T + (T-1)\hat{Q}_{T-1}^{\pi}(s,a)\Big) \tag{2.24}$$

$$= \hat{Q}_{T-1}^{\pi}(s,a) + \frac{1}{T}\Big(G_T - \hat{Q}_{T-1}^{\pi}(s,a)\Big). \tag{2.25}$$

值函数$\hat{Q}^{\pi}(s,a)$在第T试验后的平均等于第$T-1$试验后的平均加上一个增量。更一般性地，我们将权重系数$\frac{1}{T}$改为一个比较小的正数α。

$$\hat{Q}_T^{\pi}(s,a) = \hat{Q}_{T-1}^{\pi}(s,a) + \alpha\delta \tag{2.26}$$

$$= \hat{Q}_{T-1}^{\pi}(s,a) + \alpha\Big(G_T - \hat{Q}_{T-1}^{\pi}(s,a)\Big), \tag{2.27}$$

其中，增量δ称为**蒙特卡罗误差**，表示实际回报G_T与估计回报$\hat{Q}_{T-1}^{\pi}(s,a)$之间的差距。

在蒙特卡罗误差中，G_T为第T次试验所得到的总回报，需要采样整个轨迹来计算。为了提高效率，我们只需要采样下一步的状态和动作(s',a')，然后利用公式(2.13)中的贝尔曼方程来近似估计G_T，

$$G_T \approxeq r_T + \gamma\hat{Q}_{T-1}^{\pi}(s',a'), \tag{2.28}$$

参见习题(??)，第??页。

其中，r_T为第T次试验中，在状态s，执行动作a后，实际得到的即时奖励。因此，

$$\hat{Q}_T^{\pi}(s,a) = \hat{Q}_{T-1}^{\pi}(s,a) + \alpha\Big(r_T + \gamma\hat{Q}_{T-1}^{\pi}(s',a') - \hat{Q}_{T-1}^{\pi}(s,a)\Big) \tag{2.29}$$

$$= (1-\alpha)\hat{Q}_{T-1}^{\pi}(s,a) + \alpha\Big(r_T + \gamma\hat{Q}_{T-1}^{\pi}(s',a')\Big). \tag{2.30}$$

这种计算$\hat{Q}^{\pi}(s,a)$的方法需要知道当前状态s、当前动作a、奖励值r、下一步的状态s'和下一步的动作a'，因此成为称为**SARSA 算法**（State Action Reward State Action）[Rummery and Niranjan, 1994]。

算法 2.3: SARSA 算法

输入: 环境 E，动作空间 $\mathcal{A}$，折扣率 γ，学习率 α

初始化：$\forall s, \forall a, \pi(a|s) = \frac{1}{|\mathcal{A}|}$;

随机初始化 $Q(s,a)$;

repeat

  初始化起始状态 s;

  选择动作 $a = \pi^{\epsilon}(s)$;

  repeat

    执行动作 a，得到即时奖励 r 和新状态 s';

    在状态 s'，选择动作 $a' = \pi^{\epsilon}(s')$;

    $Q(s,a) \leftarrow Q(s,a) + \alpha\Big(r + \gamma Q(s',a') - Q(s,a)\Big)$;

    更新策略：$\pi(s) = \arg\max_{a \in |\mathcal{A}|} Q(s,a)$;

    $s \leftarrow s', a \leftarrow a'$;

  until s 为终止状态;

until $\forall s, a$，$Q(s,a)$ 收敛;

输出: 策略 $\pi(s)$

SARSA 算法2.3所示。SARSA 算法属于同策略算法。如果改为异策略，用非探索策略来计算时序差分误差，就是**Q 学习**（Q-Learning）算法 [Watkins and Dayan, 1992]。Q 学习算法2.4所示。与 SARSA 算法的不同是在第 8 步中，值函数的评估是贪婪的策略 $\pi(s') = \arg\max_{a \in |\mathcal{A}|} Q(s',a)$，而不是采样使用的 π^{ϵ} 策略。

事实上，Q 学习算法被提出的实际更早，SARSA 算法是 Q 学习算法的改进。

时间差分学习是强化学习的主要学习方法，其核心思想就是在每个步骤优化值函数来减少预期和现实的差距。这和动物学习的机制十分相像。在大脑神经元中，多巴胺的释放机制和时序差分学习十分吻合。Schultz [1998] 的一个实验中，通过监测猴子大脑释放的多巴胺浓度，发现如果猴子获得比预期更多的果汁，或者在没有预想到的时间喝到果汁, 多巴胺释放大增。如果本来预期的果汁没有喝到，多巴胺的释放就会大减。多巴胺的释放, 来自对于实际奖励和预期奖励的差异，而不是奖励本身。

多巴胺是一种神经传导物质，传递开心、兴奋有关的信息。

蒙特卡罗方法和时序差分的主要不同为：蒙特卡罗需要完整一个路径完成才能知道其总回报，也不依赖马尔可夫性质；而时序差分只需要一步，其总回报需要依赖马尔可夫性质来进行近似估计。

算法 2.4: Q 学习算法

输入: 环境 E，动作空间 $\mathcal{A}$，折扣率 γ，学习率 α

初始化：$\forall s, \forall a, \pi(a|s) = \frac{1}{|\mathcal{A}|}$;

随机初始化 $Q(s,a)$;

repeat

  初始化起始状态 s;

  repeat

    在状态 s，选择动作 $a = \pi^{\epsilon}(s)$;

    执行动作 a，得到即时奖励 r 和新状态 s';

    在状态 s'，选择动作 $a' = \pi(s')$;

    $Q(s,a) \leftarrow Q(s,a) + \alpha\Big(r + \gamma Q(s',a') - Q(s,a)\Big)$;

    更新策略：$\pi(s) = \arg\max_{a \in |\mathcal{A}|} Q(s,a)$;

    $s \leftarrow s'$;

  until s 为终止状态;

until $\forall s, a$，$Q(s,a)$ 收敛;

输出: 策略 $\pi(s)$

2.5 深度强化学习

深度强化学习（Deep Reinforcement Learning）是将强化学习和深度学习结合在一起，用强化学习来定义问题和优化目标，用深度学习来解决状态表示、策略表示等问题。深度强化学习在一定程度上具备解决复杂问题的通用智能，并在很多任务上都取得了很大的成功。

本章之前的强化学习算法中，我们都默认状态和动作好像都是离散的，因此值函数可以存储在表格中。在很多实际问题中，有些任务的状态和动作的数量非常多。比如围棋的棋局有 $3^{361} \approx 10^{170}$ 种状态，动作（即落子位置）数量为 361。还有些任务的状态和动作是连续的。比如在自动驾驶中，智能体感知到的环境状态是各种传统器数据，一般都是连续的。动作是操作方向盘的方向（$-90 \sim 90$ 度）和速度控制（$0 \sim 300$ 公里/小时），也是连续的。

为了有效地解决这些问题，可以一个复杂的函数（比如深度神经网络）来近似计算值函数以及策略，以减少强化学习算法的复杂度，并提高泛化能力。具体来讲，我们可以有三种不同的结合强化学习和深度学习的方式，分别用深度

神经网络来建模强化学习中的值函数、策略、模型，然后用误差反向传播算法来优化目标函数。

2.5.1 基于值函数的深度强化学习

为了在连续的状态和动作空间中计算值函数 $Q^\pi(s,a)$，我们可以用一个函数 $Q_\theta(\mathbf{s},\mathbf{a})$ 来表示近似计算，称为**值函数近似**（Value Function Approximation）。

$$Q_\theta(\mathbf{s},\mathbf{a}) \approx Q^\pi(s,a), \tag{2.31}$$

其中，$\mathbf{s}$ 和 $\mathbf{a}$ 分别是状态 s 和动作 a 的向量表示；函数 $Q_\theta(\mathbf{s},\mathbf{a})$ 为深度神经网络，参数为 θ，输出为一个实数，称为 **Q 网络**（Q-Network）。

如果动作为有限离散的 m 个动作 $a_1,\cdots,a_m$，我们可以让 Q 网络输出一个 m 维向量，其中每一维用 $Q_\theta(\mathbf{s},a_i)$ 来表示，对应值函数 $Q(s,a_i)$ 的近似值。

$$Q_\theta(\mathbf{s}) = \begin{bmatrix} Q_\theta(\mathbf{s},a_1) \\ \vdots \\ Q_\theta(\mathbf{s},a_m) \end{bmatrix} \approx \begin{bmatrix} Q^\pi(s,a_1) \\ \vdots \\ Q^\pi(s,a_m) \end{bmatrix}. \tag{2.32}$$

我们希望近似值函数 $Q_\theta(\mathbf{s},\mathbf{a})$ 可以逼近值函数 $Q^\pi(s,a)$。如果采样蒙特卡罗方法，就直接让 $Q_\theta(\mathbf{s},\mathbf{a})$ 去逼近总回报的平均 $\bar{G}$；如果采样时序差分方法，就让 $Q_\theta(\mathbf{s},\mathbf{a})$ 去逼近 $\mathbb{E}[r+\gamma Q_\theta(\mathbf{s}',\mathbf{a}')]$，$s',a'$ 是下一时刻的状态以及动作。

以 Q 学习为例，采用随机梯度下降，我们将目标函数设为

$$\mathcal{L}(s,a,s';\theta) = (r+\gamma \max_{a'} Q_\theta(\mathbf{s}',\mathbf{a}') - Q_\theta(\mathbf{s},\mathbf{a}))^2, \tag{2.33}$$

其中，s' 是下一时刻的状态。

然而，这个目标函数存在两个问题：一是目标不稳定，参数学习的目标依赖于参数本身；二是样本之间有很强的相关性。

为了解决这两个问题，Mnih et al. [2015] 提出了一种**深度 Q 网络**（Deep Q-Networks，DQN）。深度 Q 网络采取两个措施：一是目标网络冻结（Freezing Target Networks），即在一个时间段内固定目标中的参数，来稳定学习目标；二是经验回放 (Experience Replay)，构建一个经验池来去除数据相关性。经验池由智能体最近的经历组成的数据集。

经验回放可以形象地理解为在回忆中学习。

算法 2.5: 带经验回放的深度Q网络

输入: 环境 E，动作空间 $\mathcal{A}$，折扣率 γ，学习率 α

初始化经验池 $\mathcal{D}$，容量为 N;
随机初始化 Q 网络的参数 θ;
随机初始化目标 Q 网络的参数 $\theta^- = \theta$;
repeat
  初始化起始状态 s;
  repeat
    在状态 s，选择动作 $a = \pi^\epsilon$;
    执行动作 a，观测环境，得到即时奖励 r 和新的状态 s';
    将 s, a, r, s' 放入 $\mathcal{D}$ 中;
    从 $\mathcal{D}$ 中采样 ss, aa, rr, ss';
    $y = \begin{cases} rr, & ss' \text{为终止状态}, \\ r_j + \gamma \max_{a'} Q_{\theta^-}(\mathbf{ss}', \mathbf{a}'), & \text{否则} \end{cases}$;
    以 $\big(y - Q_\theta(\mathbf{s}, \mathbf{a})\big)^2$ 为损失函数来训练 Q 网络;
    $s \leftarrow s'$;
    每隔 C 步，$\theta^- \leftarrow \theta$;
  until s 为终止状态;
until $\forall s, a$，$Q_\theta(\mathbf{s}, \mathbf{a})$ 收敛;

输出: Q 网络

训练时，随机从经验池中抽取样本来来代替当前的样本用来进行训练。这样，也可以就打破了和相邻训练样本的相似性，避免模型陷入局部最优。经验回放在一定程度上类似于监督学习。先收集样本，然后在这些样本上进行训练。

深度Q网络的学习过程如算法2.5所示。

深度Q网络在Atari游戏中的应用

2.5.2 基于策略函数的深度强化学习

我们之前提到的策略优化都要依赖值函数,比如贪心策略 $\pi(s) = \arg\max_a Q(s, a)$。最优策略一般需要遍历当前状态s下的所有动作，并找出最优的 $Q(s, a)$。如果

动作空间离散但是很大时，那么遍历求最大需要很高的时间复杂度；如果动作空间是连续的并且 $Q(s,a)$ 非凸时，也很难求解出最佳的策略。

我们可以使用一种直接对策略进行参数化的方法，称为**策略搜索**（Policy Search）。和基于值函数的方法相比，策略搜索可以不需要值函数，直接优化策略。此外，参数化的策略能够处理连续状态和动作，可以直接学出随机性策略。

对于确定性策略，可以直接用深度神经网络来表示一个参数化的从状态空间到动作空间的映射函数：$a = \pi_\theta(\mathbf{s})$，其中 $\mathbf{s}$ 为状态 s 的向量表示，θ 为参数。对于随机性策略，可以让深度神经网络的输出为一个分布：$\pi_\theta(\mathbf{a}|\mathbf{s})$。

假设从状态 s 开始，执行策略 $\pi_\theta(\mathbf{a}|\mathbf{s})$，得到的轨迹为 $s_0 = s, a_0, r_1, s_1, a_2, \cdots$，记为 τ，其总回报记为 $G(\tau)$。因为最优的策略是使得在每个状态的总回报最大的策略，因此策略搜索的目标函数为

$$\mathcal{L}(s;\theta) = \mathbb{E}[G(\tau)|\pi_\theta] = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1}|\pi_\theta\right], \tag{2.34}$$

策略搜索是通过寻找参数 θ 使得目标函数 $\mathcal{L}(s;\theta)$ 最大。目标函数关于 θ 的导数为

$$\begin{aligned}
\frac{\partial \mathcal{L}(s;\theta)}{\partial \theta} &= \frac{\partial}{\partial \theta} \int p_\theta(\tau) G(\tau) d\tau && (2.35)\\
&= \int \nabla_\theta p_\theta(\tau) G(\tau) d\tau && (2.36)\\
&= \int p_\theta(\tau) \frac{1}{p_\theta(\tau)} \nabla_\theta p_\theta(\tau) G(\tau) d\tau && (2.37)\\
&= \mathbb{E}\left[\nabla_\theta \log p_\theta(\tau) G(\tau)\right], && (2.38)
\end{aligned}$$

公式 (2.38) 中，期望可以通过采样的方法来近似。对当前策略 π_θ，可以随机游走采集多个轨迹，对于每一个轨迹，计算总回报 $G(\tau)$，再乘以轨迹概率对参数 θ 的导数 $\nabla_\theta \log p_\theta(\tau)$，然后再计算平均。这种方法就叫做**策略梯度**（Policy Gradient）。

从公式 (2.38) 中可以看出，参数 θ 优化的方向是总回报 $G(\tau)$ 越大的轨迹 τ，其概率 $p_\theta(\tau)$ 也越大。

在优化算法上也可以分为基于模型的方法和模型无关的方法。

2.6 总结和深入阅读

强化学习是一种十分有趣的机器学习方法。和其他学习方法相比，强化学习更接近生物学习的本质，可以应对多种复杂的场景，而从更接近通用人工智能系统的目标。强化学习的主要参考文献为Sutton and Barto [2011] 的《Reinforcement learning: An introduction》。

强化学习和监督学习的区别在于：（1）强化学习的样本通过不同与环境进行交互产生，即试错学习，而监督学习的样本由人工收集并标注；（2）强化学习的反馈信息只有奖励，并且是延迟的；而监督学习需要明确的指导信息（每一个状态对应的动作）。

深度 Q 网络（DQN）在 Atari 游戏上取了很大的成功，但依然有一定的不足。一些 DQN 的改进，包括双 Q 网络 [Van Hasselt et al., 2016]、优先级经验回放 [Schaul et al., 2015]、决斗网络 [Wang et al., 2015] 等。

习题 2-1 证明公式 (2.18) 和 (2.19) 会收敛到最优解。

习题 2-2 比较证明公式 (2.13) 和 (2.28) 的不同之处。

习题 2-3 分析 SARSA 算法和 Q 学习算法的不同。

参考文献

Ronen I Brafman and Moshe Tennenholtz. R-max – a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3(Oct):213–231, 2002.

Marvin Minsky. Steps toward artificial intelligence. *Computers and thought*, 406:450, 1963.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineer-

ing, 1994.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Wolfram Schultz. Predictive reward signal of dopamine neurons. *Journal of neurophysiology*, 80(1):1–27, 1998.

Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. 2011.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, pages 2094–2100, 2016.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.