人工智能 - 强化学习

杜小勤

武汉纺织大学数学与计算机学院

2017/04/26

强化学习

- 引言；
- 基础方法；
- 高级方法；
- 应用；

引言

强化学习的基本思想：通过与环境的交互进行学习——从环境中获取信息，进行加工、处理，并驱动与影响环境，在这样一个交互中进行学习。

强化学习是一种机器学习方法。

引言

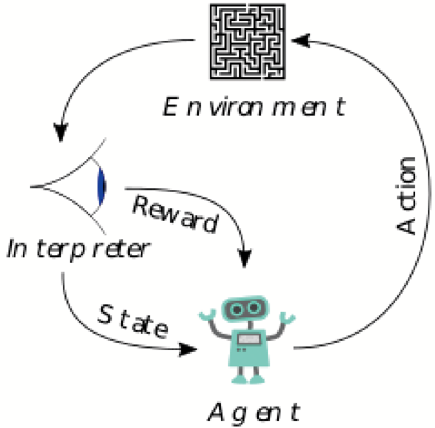

图 1-1: 强化学习的交互式框架

引言

实际上，从计算的观点看，强化学习要学习的是状态-动作的映射，以最大化从环境获得的（长期）总体奖励（回报）。

强化学习的两大特征：Trial-and-Error Search、Delayed Reward。

它的理论框架是 Markov Decision Process（MDP）。

引言

强化学习不同于监督式学习（Supervised Learning）。后者从实例（Examples）中进行学习，缺少交互式特征。

在很多情况下，获取这些 Examples 是不现实的。Agent 必须要能够从与环境的交互经验中进行学习。

引言

强化学习的一大挑战在于：Exploration V.S. Exploitation，学习算法要在它们之间取得某种折中——既要考虑已经试过的、能够获取到更多奖励的“好”的动作，又要考虑还没有试过的、可能潜在地获得更多奖励的动作。强化学习算法必须能够平衡它们。

此外，有些动作的直接奖励很高，但长期奖励却并不高，而另一些动作则刚好相反。

监督式学习并不具备交互式特征，因而也就不存在这个问题。

强化学习框架的基本要素

框架的基本要素：环境、Policy、Reward Function、Value Function，另外，也可以包括环境模型（可选）。

Policy：状态-动作的映射。一般而言，Policy 可以是 Stochastic。

Reward Function：它定义了学习的目标，指定了状态-奖励的映射，它表明了 Agent 对状态的直接渴望程度。该函数定义了直接回报。一般而言，Reward Function 可以是 Stochastic。

强化学习框架的基本要素

Value Function：它定义了学习的终极目标，指定了状态-值的映射，它表明了 Agent 对状态的长期渴望程度。与 Reward Function 不同的是，它定义了长期回报。

一个状态的直接奖励可能不高，但是长期回报可能很高，反之亦然。

强化学习算法的主要任务之一是估算 Value Function。

强化学习框架的基本要素

一些搜索方法，例如遗传算法、遗传规划、模拟退火以及其它一些函数优化方法，直接在策略空间进行搜索，而无需使用 Value Function。

这些方法被称为演化方法（Evolutionary Methods），这是因为它们与生物演化产生智能行为的方式很相似。另外，它们也无需与环境进行交互。

强化学习框架的基本要素

最后一个要素是：在某些强化学习系统中，可能会使用环境模型，它们用来模拟环境，例如，给定状态和动作，模型可能会预测下一个状态和奖励。

环境模型可以用于规划（Planning），这是强化学习领域中相对较新的进展。早期的强化学习完全通过 Trial-and-Error 来改进学习。

研究者们逐渐认识到，强化学习与动态规划方法关系密切，那么使用环境模型进行规划以及使用 Trial-and-Error 进行学习就是很自然的事情了。

Tic-Tac-Toe

强化学习算法解决 Tic-Tac-Toe 对弈问题。

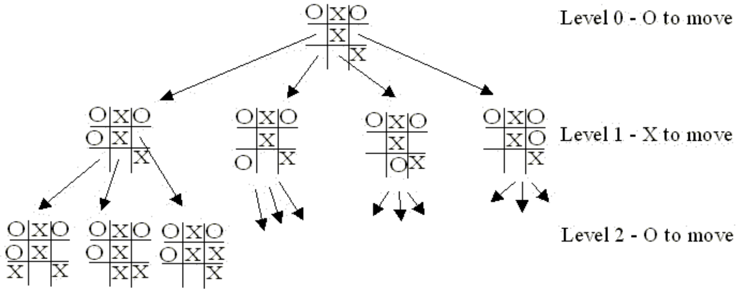

图 1-2: Tic-Tac-Toe 的状态树

Tic-Tac-Toe

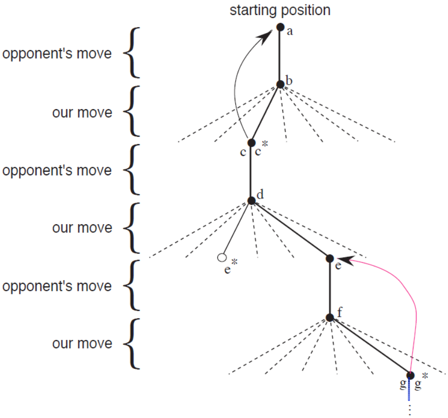

图 1-3: Tic-Tac-Toe 的移动序列及 Value Backup

Tic-Tac-Toe 的 Value Backup

$$V(s) \leftarrow V(s) + \alpha[V(s') - V(s)] \tag{1}$$

式中，s' 是 s 的后续状态，α 是学习率。

该公式是 Temporal-Difference Learning 更新规则中的一种。

程序演示

其它应用

实际上，强化学习算法已经成功地应用到了非常复杂的棋类游戏中，例如：Backgammon、国际象棋、围棋等。

Evaluative Feedback V.S. Instructive Feedback

Evaluative Feedback（EF）用来评估动作到底有多好，而 Instructive Feedback（IF）用来表明动作是好是坏。

IF 是监督式学习的基础，EF 是函数优化（包括演化方法）的基础。强化学习使用 EF，也可以将 EF 与 IF 结合起来。

N-armed Bandit 问题

问题描述：赌博者在一排老虎赌博机中选择一个老虎机，而每个老虎机提供了一个随机奖励（该奖励服从于老虎机特定的概率分布）。为了获得最多奖励，老虎机被选择的顺序及选择次数至关重要。在该问题中，赌博者的目标是最大化奖励。

该问题体现了 Exploration 和 Exploitation 的重要性，是研究强化学习算法的重要模型。

N-armed Bandit 问题

图 1-4: 拉斯维加斯的老虎赌博机

N-armed Bandit 问题

我们将对 N-armed Bandit 问题进行一定程度的简化。将赌博者选择老虎机看作是一个动作，获得的奖励对应着执行该动作后获得的回报。

我们还假定，Agent 事先并不知道每个动作执行后获得的回报，但是可以进行估算。

N-armed Bandit 问题

Agent 必须在 Exploration 和 Exploitation 之间取得某种平衡。

由于 Agent 可以维持对动作奖励的某种估算，那么 Agent 可能会倾向于选择目前奖励值最高的动作（Exploitation）。但是，这里存在一个潜在的问题，即当前最好的动作从长期来看未必是最佳的。

N-armed Bandit 问题

为了解决这个问题，Agent 必须要进行一定程度的 Exploration，即也要选择当前非最佳动作，这样就能够改善那些动作的估算值，以便能够确定从长期来看真正最佳的动作。

Action-Value 方法

$$Q_t(a) = \frac{r_1 + r_2 + ... + r_{k_a}}{k_a} \tag{2}$$

式中，$Q_t(a)$ 表示时刻 t 时动作 a 的奖励评估值，k_a 表示截止到时刻 t 时，该动作被选择的次数，r_i 表示第 i 次该动作获得的奖励。

$Q_0(a) = 0$，$\lim_{t \to \infty} Q_t(a) = Q^*(a)$，$Q^*(a)$ 表示实际值。这种估算方法被称为是 sample-average 方法。

Action-Value 方法

最简单的动作选择规则是 greedy 方法：

$$Q_t(a^*) = max_a Q_t(a) \tag{3}$$

这种方法总是 Exploitation——选择当前最好的动作。

Action-Value 方法

一个简单的改进

大部分时间仍然是 Exploitation，即选择当前最好的动作，但是以一个较小的概率 ϵ 随机选择一个其它的动作——Exploration。

这种方法被称为是 ϵ-greedy 方法。

实验结果

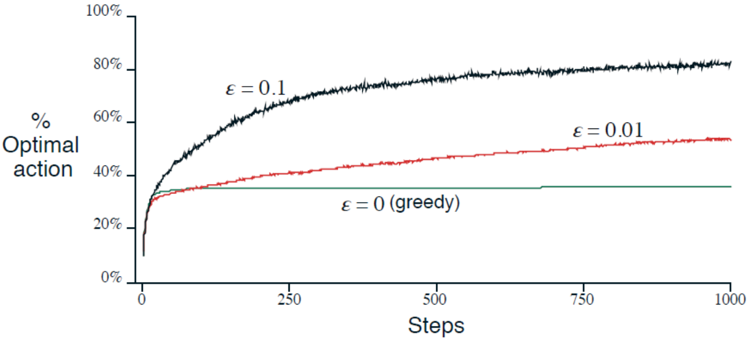

图 1-5: 10-armed 问题的实验结果

Softmax Action Selection

ϵ-greedy 方法的一个缺点是：在 Exploration 时，所有的动作有相同的选择概率。

解决办法：动作被选择的概率，将依据它们的值来确定，这就是 Softmax Action Selection 方法。

Softmax Action Selection

$$\frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(b)/\tau}} \tag{4}$$

上式是动作 a 被选择的概率，该式被称为 Gibbs 或 Boltzmann 分布，τ 是一个正数，被称为温度参数，高的数值将导致各个动作（近乎）等概率地被选中执行，低的数值将导致各个动作被选择的概率差异变大。$\tau \to 0$ 时该公式的效果与 greedy action selection 方法一样。

Incremental Implementation

Action-Value 计算公式 (2) 不利于实现——它要占用较多的空间及计算时间。改进如下：

$$
\begin{aligned}
Q_{k+1} &= \frac{1}{k+1}\sum_{i=1}^{k+1} r_i = \frac{1}{k+1}\left(r_{k+1} + \sum_{i=1}^{k} r_i\right) && (5)\\
&= \frac{1}{k+1}\left(r_{k+1} + kQ_k + Q_k - Q_k\right) && (6)\\
&= \frac{1}{k+1}\left(r_{k+1} + (k+1)Q_k - Q_k\right) && (7)\\
&= Q_k + \frac{1}{k+1}\left(r_{k+1} - Q_k\right) && (8)
\end{aligned}
$$

Incremental Implementation

实际上，公式 (5) 的更加通用的公式如下：

$$NewEstimate \leftarrow \quad (9)$$

$$OldEstimate + StepSize(Target - OldEstimate)(10)$$

注意，公式 (5) 中的 *StepSize* 是 $\frac{1}{k+1}$，它是随时间变化的，该公式很适合 stationary 环境——它均等地对待所有时刻获得的奖励，这被称为 sample-average 方法。

Incremental Implementation

为了处理 nonstationary 环境——Agent 更加重视最近获得的奖励，给予它更多的权值，我们可以引入常量 step-size 参数。这样，公式 (5) 改进如下：

$$Q_{k+1} = Q_k + \alpha(r_{k+1} - Q_k) \tag{11}$$

式中，α 是常量学习率。

Incremental Implementation

公式 (11) 更加重视最近的奖励，原因如下：

$$
\begin{aligned}
Q_k &= Q_{k-1} + \alpha(r_k - Q_{k-1}) && (12)\\
&= \alpha r_k + (1-\alpha)Q_{k-1} && (13)\\
&= \alpha r_k + (1-\alpha)\alpha r_{k-1} + (1-\alpha)^2 Q_{k-2} && (14)\\
&= \alpha r_k + (1-\alpha)\alpha r_{k-1} + (1-\alpha)^2 \alpha r_{k-2} + \ldots && (15)\\
&\quad + (1-\alpha)^{k-1}\alpha r_1 + (1-\alpha)^k Q_0 && (16)\\
&= (1-\alpha)^k Q_0 + \sum_{i=1}^{k} \alpha(1-\alpha)^{k-i} r_i && (17)
\end{aligned}
$$

Incremental Implementation

实际上，上式是 weighted-average 方法，因为：

$$(1-\alpha)^k + \sum_{i=1}^{k} \alpha(1-\alpha)^{k-i} = 1 \tag{18}$$

该方法与前面介绍的 sample-average 方法不同。有时候，我们也称这种方法为 exponential recency-weighted average 方法，因为它给予最近获得的奖励更多的权值。

Incremental Implementation

收敛条件：

$$\sum_{i=1}^{\infty} \alpha_k(a) = \infty \tag{19}$$

$$\sum_{k=1}^{\infty} \alpha_k^2(a) < \infty \tag{20}$$

式中，α_k^a 是 StepSize（学习率）。

第 1 个条件确保学习步数充分大，这样预测值就能够克服初始条件和随机扰动，并逐步逼近真实值；第 2 个条件确保 StepSize 逐渐变得充分小，这样就可以确保算法收敛。

Incremental Implementation

sample-average 方法满足这 2 个条件，它的 StepSize 为 $\alpha_k(a) = \frac{1}{k}$。

但是对于 weight-average 方法，它的 StepSize 为 $\alpha_k(a) = \alpha$，第 2 个条件并不满足：估算值从来不会完全收敛，而是持续地变化，以响应最近收到的奖励。实际上，这一点正是处于 nonstationary 环境中的 Agent 所希望的。

Incremental Implementation

另外，满足上述 2 个条件的学习算法通常收敛非常慢，需要做特别的调整以获得满意的收敛速度。

虽然这些算法经常用于理论中，但是却很少运用于实际应用和实验研究中。

Optimistic Initial Values

上述介绍的 Action-Value 更新规则在某种程度上依赖于动作初始估算值 $Q_0(a)$。

从统计学的角度来看，这被称为是初始估算偏置。

对于 sample-average 方法，从公式 (5) 中可以验证，只要所有的动作被选择至少一次，那么这些初始估算偏置就消失了。

Optimistic Initial Values

但是，对于 weighted-average 方法，这些初始估算偏置是永久存在的，虽然它们的作用随着学习步数的增加而减少。

在实际应用中，这种偏置通常都不是一个问题。相反，有时候是非常有用的。

偏置存在的不利方面是，学习算法要把它们当作学习参数。有利方面是，学习算法可以把偏置当作某种先验知识，以便更加利于学习。

Optimistic Initial Values

另外，初始估算偏置也可以用于鼓励学习算法进行 Exploration，即便动作选择策略一直是 greedy 的。

例如，在 10-armed testbed 实验中，可以把初始估算偏置都设置为 +5（最优的状态值）。在学习过程中，只要动作被选择执行，那么它的值都会小于初始估算值，在下一次进行动作选择时，其它的动作便会得到机会执行，这无形当中就鼓励了学习算法进行持续的 Exploration。

Optimistic Initial Values

这种方法被称为是 optimistic initial values，它适用于 stationary 环境，但是并不适用于 nonstationary 环境，因为后者对 Exploration 的需求是一直持续的。

Optimistic Initial Values

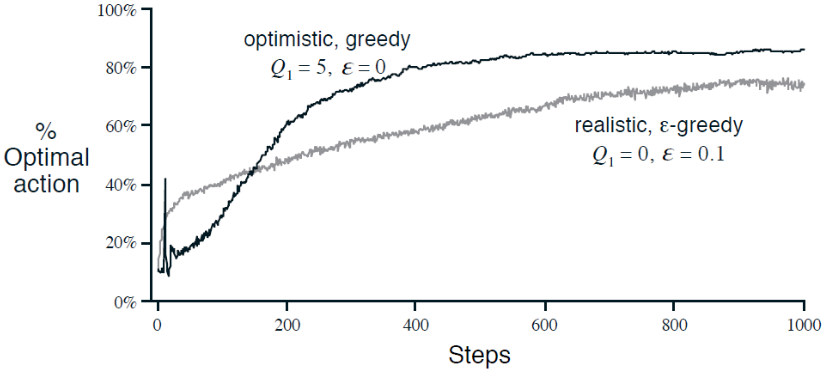

图 1-6: 10-armed - 使用 Optimistic Initial Values 技术

Reinforcement Comparison 方法

这是设计 Reinforcement Learning 算法的另一种思路。

它不是使用 Action-Value 进行动作选择。

该方法基于这一原理：获得高奖励的动作应该获得更多的执行机会，而获得奖励少的动作应该获得更少的执行机会。

Reinforcement Comparison 方法

Reference Rewards：基准奖励，一个自然的选择是使用平均奖励。

Large Reward：比平均奖励高的奖励。

Small Reward：比平均奖励小的奖励。

Reinforcement Comparison 方法

基于这一思想的学习算法被称为 Reinforcement Comparision 方法。

有些情况下，这种方法比 Action-Value 方法更为有效。

该方法也是 Actor-Critic 强化学习方法的前身。

Reinforcement Comparison 方法

该方法并不维护 Action-Values，而是维护一个总体平均奖励 $\overline{r_t}$ 以及每个动作的偏爱值 $p_t(a)$，然后使用偏爱值和 Softmax 公式来计算每个动作被选择的概率。相关公式如下：

$$\pi_t(a) = \frac{e^{p_t(a)}}{\sum_{b=1}^{n} e^{p_t(b)}} \tag{21}$$

$$p_{t+1}(a_t) = p_t(a_t) + \beta(r_t - \overline{r_t}) \tag{22}$$

$$\overline{r}_{t+1} = \overline{r_t} + \alpha(r_t - \overline{r_t}) \tag{23}$$

Reinforcement Comparison 方法

初始值 $\bar{r}_0$ 的设置可以采用 optimistic initial value 方法——鼓励 Exploration，或者使用先验知识。

初始的动作偏爱值 $p_0(a)$ 可以设置为 0。

StepSize（学习率）可以设置为 Constant。

该方法非常有效，有时候比 Action-Value 方法的效果要好。

Reinforcement Comparison 方法

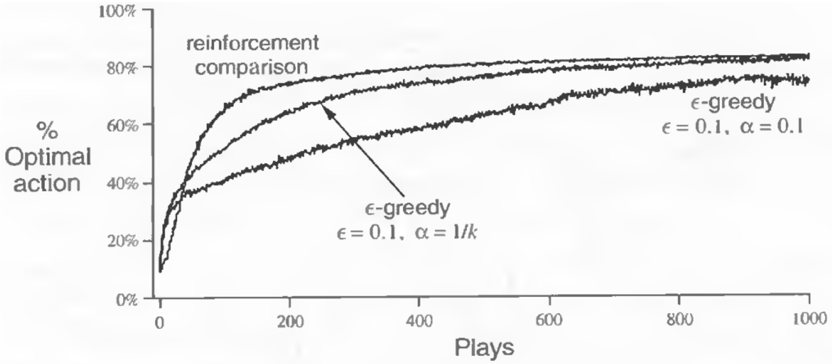

图 1-7: 10-armed - 使用 Reinforcement Comparison 方法

Pursuit 方法

Pursuit 方法既维护 Action-Value 估算，也维护 Action Preferences（动作偏爱值，就像 Reinforcement Comparision 方法那样）。

该方法让动作偏爱值持续地“追逐”最佳动作（具有当前 Action-Value 估算的最佳值）。

Pursuit 方法

在最简单的 Pursuit 方法中，动作偏爱值是每个动作的选择概率 $\pi_t(a)$。

在每次更新时，选择概率 $\pi_t(a)$ 将按如下规则更新——它将使得 greedy 动作的选择概率变得大一点，而其它非 greedy 动作的选择概率变得小一点。

Pursuit 方法

对于 greedy 动作（$a^*_{t+1} = argmax_a Q_{t+1}(a)$）的选择概率，其更新规则如下：

$$\pi_{t+1}(a^*_{t+1}) = \pi_t(a^*_{t+1}) + \beta(1 - \pi_t(a^*_{t+1})) \tag{24}$$

而对于所有其它 $a \neq a^*_{t+1}$ 动作的选择概率，其更新规则如下：

$$\pi_{t+1}(a) = \pi_t(a) + \beta(0 - \pi_t(a)) \tag{25}$$

Pursuit 方法

Action-Value——$Q_{t+1}(a)$ 的更新规则可以按 sample-average 或 weighted-average 方法进行。

Pursuit 方法

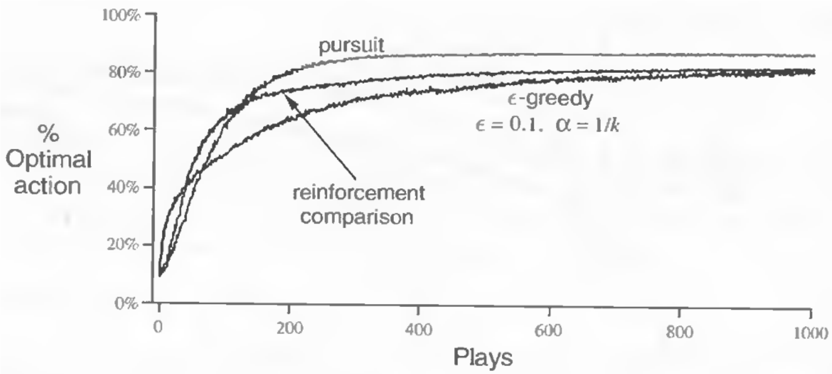

图 1-8: 10-armed - 使用 Pursuit 方法

Pursuit 方法

从图（1-8）可以看出，Pursuit 方法超越了其它 2 种方法。

但是，在其它类型的问题中，这 3 种方法的表现会与此不同，它们各有优缺点。

Associative Search

到目前为止，我们仅仅考虑了 nonassociative 任务，即只在单一的状态下进行动作的选择，且该动作的选择不会影响到下次动作的选择以及获得的奖励。

换一种说法就是，我们还没有将不同的动作与不同的状态进行关联，动作的执行也不会影响到下一状态的动作执行及其获得的奖励——因为我们还没有接触到这样的问题。

Associative Search

N-armed bandit 问题的扩展：假设我们有几个不同的 N-armed bandit 任务，在每次迭代时，你将面临它们中的一个（随机）。

当然，你也可以把这个任务看作是一个 nonstationary N-armed bandit 问题，并按照前述解决 nonstationary 问题的方法来解决它，例如 weighted-average 方法。

然而，除非真正的 Action-Value 变化较慢，否则，这种解决方法不会奏效。

Associative Search

实际上，这是一个 Associative Search 任务，它涉及到——Trial-and-error 学习（最佳动作），也涉及到状态-（最佳）动作映射（Policy）的学习。

Associative Search 介于 N-armed bandit 和完全的强化学习问题之间——与后者类似，它涉及到 Policy 的学习；与前者类似，每次动作执行后仅仅影响立即奖励，不会对后续状态下动作的执行产生什么影响。

如果动作的执行会影响到下一状态及奖励，那么它就是一个完全强化学习任务。

Agent-Environment Interface

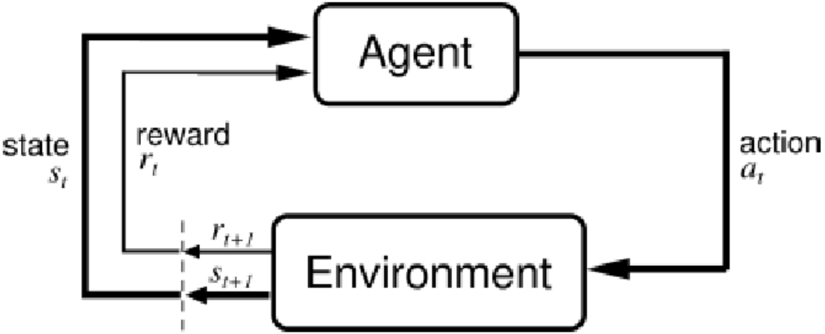

图 1-9: Agent-Environment Interface

Agent-Environment Interface

我们假定 Agent 与环境在每个离散时刻 $t = 0, 1, 2, \ldots$ 进行交互。

在某个时刻 t，Agent 处于状态 $s_t \in \mathcal{S}$，它选择执行动作 $a_t \in \mathcal{A}(s_t)$，在时刻 $t + 1$，Agent 收到一个奖励 $r_{t+1} \in \mathcal{R}$，并转移到状态 s_{t+1}。

Agent-Environment Interface

在每一时刻，Agent 将依据状态-动作映射来选择动作执行（该映射可以是确定的，也可以是非确定的，即以某个概率选择某个动作）。

Policy：状态-动作映射，用 π_t 来表示，例如 $\pi_t(s_t = s, a_t = a)$ 表示 t 时刻 Agent 处于状态 s 时选择动作 a 的概率。

Agent 的目标是最大化（长期总体累积）回报。

Returns

Returns：回报，或者期望回报（Expected Returns），用 R_t 来表示。

Agent 的目标是最大化 R_t。

Returns

强化学习任务可以分为：

- Episodic Tasks：Agent-Environment 的交互进程中有终止状态（Terminal State），例如，棋类游戏、迷宫搜索等;
- Continuing Tasks：Agent-Environment 的交互进程中没有终止状态，而是连续运行;

Returns - Episodic Tasks

最简单的回报形式：

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + ... + r_T \tag{26}$$

式中，T 是一个 episode 的终止时刻。这样的任务称为是 episodic tasks。

Returns - Continuing Tasks

折扣回报（Discounted Return）形式：

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \quad (27)$$

式中，γ 是折扣率，$0 \leq \gamma \leq 1$。

Returns - Continuing Tasks

折扣率 γ 确定了将来奖励的当前价值：第 k 步时的奖励是 $\gamma^{k-1} r_{t+k}$。

$\gamma = 0$，Agent 从来不会考虑将来的奖励，而只考虑直接奖励，该 Agent 被认为是“短视的”。

$\gamma < 1$，公式（27）的值是有限的，只要奖励序列 r_k 有界。γ 越接近 1，Agent 越有远见：对将来的奖励更具期望。

Pole-Balancing

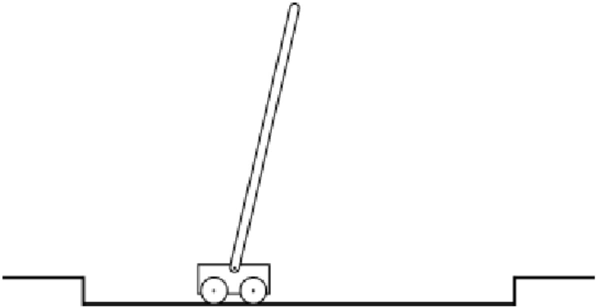

图 1-10: The pole-balancing task

Pole-Balancing

Episodic: 每一步的直接奖励值可以是 $+1$，一直到失败为止。每个 episodic 收到的回报将是 *steps*（在一个 episodic 内保持平衡的步数）。

Continuing: 使用折扣率 γ，每一步的奖励值可以是 0，一直到失败为止，失败时将收到奖励值 -1。每次失败时收到的回报将是 $-\gamma^{steps}$。

不论哪种方式，Agent 要最大化回报。

Episodic 与 Continuing

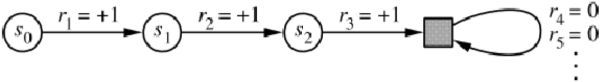

图 1-11: Episodic tasks as continuing tasks

Episodic 与 Continuing

将 Episodic 与 Continuing 任务统一起来：

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1} \tag{28}$$

式中，$T = \infty$ 或者 $\gamma = 1$，两者取其一。

参考文献

[1] Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction, February 1998. The MIT Press.
[2] Reinforcement Learning Wikipedia.
[3]Multi-armed bandit