Reinforcement Learning 读书笔记

黄帅

January 12, 2019

Contents

Preface

About Me

- **GitHub** https://github.com/ShuaiHuang/
- **Homepage** http://shuaihuang.github.io/
- **Email** shuaihuang.sjtu@gmail.com

Chapter 1 Introductions

1.1 笔记摘要

本章主要是对强化学习进行简单的介绍, 大部分内容和周志华所写的机器学习 [1] 第 16 章介绍部分相同. 第一节主要介绍了强化学习的概念; 第二节举例说明了强化学习智能体; 第三节从马尔可夫决策过程说明了强化学习四元组的内容; 第四节强调了本书主要关注决策过程. 另外还将强化学习与进化计算进行了比较, 强化学习主要侧重于通过与环境交互来达到最大化长期累积奖励的目的; 而进化计算则主要侧重于在搜索空间中进行启发式搜索, 与环境的交互并不是主要关注点. 第五节通过一个简单的下棋的例子对强化学习进行了详细阐述. 在介绍完下棋规则后, 对比了博弈论, 启发式搜索, 进化计算与强化学习的区别. 从这些方法对比中得出强化学习最重要特点是以结果为导向, 通过优化策略来指导每一步行棋动作. 每一个行棋动作都可以视为与环境的一次交互, 可以通过状态函数的优化规则

$$V(s) \leftarrow V(s) + \alpha\Big(V(s') - V(s)\Big) \tag{1.1}$$

其中 α 是调整权重;s' 是当前的状态值;s 是上一步的权重值. 那么这里还有一个问题, 状态 $V(\cdot)$ 应该如何定义, 应该符合什么要求?

1.2 我的思考

之所以选择强化学习作为接下来的学习目的, 主要是因为想在个人的知识体系以及技术发展上执行卡位策略. 在如今深度学习已经这么火爆的条件下, 如果仍然死守着机器学习的领域不放, 势必会错失先机. 接下来计划是:

1. 三个月内阅读完成这一本书;

2. 记录下相应的读书笔记和自己的思考;

3. 调研强化学习在学术界和工业界的应用现状和前景.

同时也应该注意到, 在学习机器学习和深度学习的过程中的不足需要改进:

- 对于实践应用不够重视;
- 没有挑战复杂项目下的实践应用;
- 没有成体系的输出.

Bibliography

[1] 机器学习. 清华大学出版社, 2016.

Part I

Tabular Solution Methods

Chapter 2 Multi-armed Bandits

本章通多摇臂赌博机 (Multi-armed Bandits) 这一简单的例子, 对探索和利用进行了说明, 更进一步地, 阐述了在探索和利用之间进行折中的几种方法.

2.1 A k-armed Bandit Problem

多摇臂赌博机是贯穿本书多个章节的重要例子, 因此在本章开头的第一节首先介绍了多摇臂赌博机问题的描述, 然后介绍了该问题的数学模型.

强化学习的一个典型特征是以目标为导向, 达到长期累计奖赏最大化的目的. 多摇臂赌博机问题也具有这个特征, 它的目的是长期的累计奖赏的期望[1]. 由此引入如下概念:

- **动作价值** 动作价值本质上是某一动作对应的奖赏期望

$$q_*(a) \doteq \mathbb{E}[R_t \mid A_t = a] \tag{2.1}$$

- **估计动作价值** 实际中通常使用估计动作价值 $Q_t(a)$, 使其尽可能贴近 $q_*(a)$.

贪心动作是指动作价值最大的动作, 探索和利用的区别在于是否是否采用了贪心动作. 实际情况下, 需要对探索和利用进行折中. 折中需要结合先验知识来确定, 适用场景很受限制.

2.2 Action-value Methods

定义了动作价值函数之后, 紧接着的问题就是:

[1]如何理解这里的期望? 可以把每一次动作对应的奖赏视为一个随机分布, 我们的目的是要将这些分布的期望累加最大化.

1. 如何评估动作的价值?

2. 如何依据这些估计量去选择动作?

本节定义了最直观的动作价值估计方法,*sample-average* 方法, 即

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}} \tag{2.2}$$

还定义了相应的动作选择方法,ϵ-贪心法, 该方法是以 $1-\epsilon$ 的概率选择贪心动作

$$A_t \doteq \arg\max_a Q_t(a) \tag{2.3}$$

进行利用, 以 ϵ 的概率进行探索.

2.3 The 10-armed Testbed

基于上一节的 *sample-average* 方法, 本节构造了一个 10-摇臂赌博机的例子, 然后评估了探索-利用不同探索利用折中所带来的不同结果. 本节所举的例子很直观, 也很容易理解, 同时借此也可以简单地了解一下强化学习的学习过程. 由此产生一个疑问: 这个过程中, 训练和测试数据来自同一个分布, 所以不存在过拟合问题. 不知道是否可以这么理解?[2]

2.4 Incremental Implementation

前面两节只是从定义上介绍了动作价值估计方法及其应用, 但是在实际应用情形下, 如何高效计算动作价值函数? 本节将给出动作价值函数的增量式计算方式.

由 *sample-average* 方法定义有

$$Q_n \doteq \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1} \tag{2.4}$$

其中, R_i 是某个动作第 i 次后的反馈; Q_n 是某个动作被选择 $n-1$ 次后评估的奖赏.

[2]后面阅读的过程中关注一下过拟合.

改成增量式, 有

$$
\begin{aligned}
Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} R_i \\
&= \frac{1}{n}\Big(R_n + \sum_{i=1}^{n-1} R_i\Big) \\
&= \frac{1}{n}\Big(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\Big) \\
&= \frac{1}{n}\Big(R_n + (n-1)Q_n\Big) \\
&= \frac{1}{n}\Big(R_n + nQ_n - Q_n\Big) \\
&= Q_n + \frac{1}{n}\Big[R_n - Q_n\Big]
\end{aligned}
\tag{2.5}
$$

总结上述计算形式, 可以得到如下增量式迭代过程

Algorithm 1 A simple bandit algorithm

For $a = 1$ to k: $Q(a) \leftarrow 0, N(a) \leftarrow 0$

loop

$$
A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\epsilon \\ \text{a random action} & \text{with probability } \epsilon \end{cases}
$$

$R \leftarrow bandit(A)$

$N(A) \leftarrow N(A) + 1$

$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}[R - Q(A)]$

end loop

更广义地来看, 式2.5中的更新公式可以抽象为

$$
\text{NewEstimate} \leftarrow \text{OldEstimate} + \text{StepSize}\Big[\text{Target} - \text{OldEstimate}\Big] \tag{2.6}
$$

2.5 Tracking a Nonstationary Problem

紧接着前两节的思路, 在摇臂赌博机的问题中, 如果每一个动作对应的奖赏是会随着时间动态变化的, 那我们该如何计算 (估计) 奖赏函数呢?

不妨假设, 在奖赏动态分布的情况下我们更倾向于关注近期发生的动作和相应的奖赏, 根据式2.5有

$$Q_{n+1} \doteq Q_n + \alpha\Big[R_n - Q_n\Big] \tag{2.7}$$

其中步长参数 $\alpha \in (0, 1]$ 是一个常量. 对上式进行展开, 有

$$\begin{aligned} Q_{n+1} &= Q_n + \alpha\Big[R_n - Q_n\Big] \\ &= \alpha R_n + (1-\alpha)Q_n \\ &= \alpha R_n + (1-\alpha)\Big[\alpha R_{n-1} + (1-\alpha)Q_{n-1}\Big] \\ &= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2 Q_{n-1} \\ &= (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i \end{aligned} \tag{2.8}$$

这种评估奖赏的方式被称为 *exponential recency-weighted average*.

步长参数通常会有多种选择, 但是为了保证动作价值函数能够收敛, 步长参数需要满足

$$\sum_{i=1}^{\infty} \alpha_n(a) = \infty \quad \text{and} \quad \sum_{i=1}^{\infty} \alpha_n^2(a) < \infty \tag{2.9}$$

第一个条件是为了让步长足够大, 价值函数可以避免初始值和迭代过程随机波动的影响; 第二个条件使得后期步长足够小, 确保价值函数可以收敛.

2.6 Optimistic Initial Values

既然动作价值函数的估计是一个迭代过程, 那么就不可避免地涉及到初始值选取的问题.

从统计学上看, 初始值是一种偏差 (bias). 初始动作价值可以视为一种促进探索的手段, 但这种手段并不具有普适性, 需要根据具体问题进行具体分析.

2.7 Upper-Confidence-Bound Action Selection

估计出动作价值函数后, 就需要根据动作价值函数选择出相应的动作. 本节提供了一种动作选择方法.

为了使得非贪心动作的奖赏接近于或者大于贪心动作的奖赏, 可以使用如下的动作选择方式

$$A_t \doteq \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right] \tag{2.10}$$

其中 $N_t(a)$ 定义了动作 a 在前 t 次动作中被选择的次数;$c > 0$ 控制着探索的程度. 如果有 $N_t(a) = 0$, 那么 a 就是使得 A_t 最大化的动作.

需要注意的是, 式2.10的定义脱离了 ϵ-贪心算法的框架, 直接把探索和利用整合到一起. 具体来看, 式2.10中平方根项的功能是用来估计动作 a 的不确定性或者方差.

2.8 Gradient Bandit Algorithms

上一节 (第2.7节) 介绍的 *UCB 动作选择*方法是一种比较好的动作选择方法, 但除了这种方法之外, 还有其他表现优异的动作选择方法. 本节介绍了动作*偏好* (preference), 以及基于动作偏好的随机梯度下降法动作选择.

针对每一个动作 a, 定义数字量化动作偏好 $H_t(a)$. 需要注意的是,$H_t(a)$ 与奖励并无直接关联, 单个动作偏好的绝对数量值没有意义, 多个动作偏好数值的相对比值才有意义. 基于此, 有

$$\Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \doteq \pi_t(a) \tag{2.11}$$

其中,$\pi_t(a)$ 表示在 t 时刻, 采取动作 a 的概率.

因此, 有动作偏好更新策略

$$\begin{aligned} H_{t+1}(A_t) &\doteq H_t(A_t) + \alpha(R_t - \bar{R}_t)(1 - \pi_t(A_t)), \quad \text{and} \\ H_{t+1}(a) &\doteq H_t(a) - \alpha(R_t - \bar{R}_t)\pi_t(a), \quad \text{for all } a \neq A_t \end{aligned} \tag{2.12}$$

其中,α 是步长参数;$\bar{R}_t \in \mathbb{R}$ 是包含时刻 t 在内的历史奖赏均值. 如果动作 A_t 对应的奖赏 R 大于历史均值, 那么动作 A_t 的偏好就会上升, 否则就会下降.

看书的时候没有深刻理解书中所列的算法, 这里把算法细节详细摘抄一遍加深记忆.

Gradient Bandit 算法可以从随机梯度上升估计的角度进行理解. 在精确梯度上升算法中, 每一个动作偏好 $H_t(a)$ 的增量正比于期望奖赏的增量

$$H_{t+1}(a) \doteq H_t(a) + \alpha \frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} \tag{2.13}$$

其中

$$\mathbb{E}[R_t] = \sum_b \pi_t(b) q_*(b) \tag{2.14}$$

但是在实际情况中, 精确计算梯度上升是不可行的, 因为 $q_*(b)$ 是未知的. 但是我们注意到, 我们要计算的是奖赏对动作偏好的偏导, 因此可以用随机梯度上升法进行估算

$$\begin{aligned} \frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} &= \frac{\partial}{\partial H_t(a)} \Big[\sum_b \pi_t(b) q_*(b) \Big] \\ &= \sum_b q_*(b) \frac{\partial \pi_t(b)}{\partial H_t(a)} \\ &= \sum_b (q_*(b) - X_t) \frac{\partial \pi_t(b)}{\partial H_t(a)} \end{aligned} \tag{2.15}$$

其中 X_t 是不依赖于 b 的常量. 之所以可以在此引入 X_t 等号仍然成立是因为 $\frac{\partial \pi_t(b)}{\partial H_t(a)}$ 在各个方向上的和等于 0,$\sum_b \frac{\partial \pi_t(b)}{\partial H_t(a)} = 0$. 当 $H_t(a)$ 变化时, 有些动作的概率上升, 有些动作的概率下降, 但是动作概率对动作偏好的偏导总和为 0, 因为所有动作概率的总和等于 1. 进一步将式2.15写成期望形式 (将所有可能动作 b 替换成动作变量 A_t), 有

$$\begin{aligned} \frac{\partial \pi_t(b)}{\partial H_t(a)} &= \sum_b \pi_t(b) \Big(q_*(b) - X_t \Big) \frac{\partial \pi_t(b)}{\partial H_t(a)} / \pi_t(b) \\ &= \mathbb{E}\Big[(q_*(A_t) - X_t) \frac{\partial \pi_t(A_t)}{\partial H_t(a)} / \pi_t(A_t) \Big] \\ &= \mathbb{E}\Big[(R_t - \bar{R}_t) \frac{\partial \pi_t(A_t)}{\partial H_t(a)} / \pi_t(A_t) \Big] \end{aligned} \tag{2.16}$$

其中, 式2.15中的 $X_t = \bar{R}_t$;$q_*(A_t)$ 被替换为 R_t, 因为有 $\mathbb{E}[R_t \mid A_t] = q_*(A_t)$, 并且还由于 R_t 在 A_t 给定的条件下与其他变量不存在耦合关系. 稍后将证

明 $\frac{\partial \pi_t(b)}{\partial H_t(a)} = \pi_t(b)(\mathbb{1}_{a=b} - \pi_t(a))$, 基于此继续推导式2.16

$$\begin{aligned} &= \mathbb{E}[(R_t - \bar{R}_t)\pi_t(A_t)(\mathbb{1}_{a=A_t} - \pi_t(a))/\pi_t(A_t)] \\ &= \mathbb{E}[(R_t - \bar{R}_t)(\mathbb{1}_{a=A_t} - \pi_t(a))] \end{aligned} \tag{2.17}$$

到这里, 可以证明式2.5

$$H_{t+1}(a) = H_t(a) + \alpha(R_t - \bar{R}_t)(\mathbb{1}_{a=A_t} - \pi_t(a)), \quad \text{for all } a \tag{2.18}$$

至此, 还剩下 $\frac{\partial \pi_t(b)}{\partial H_t(a)} = \pi_t(b)(\mathbb{1}_{a=b} - \pi_t(a))$ 的证明:

$$\begin{aligned} \frac{\partial \pi_t(b)}{\partial H_t(a)} &= \frac{\partial}{\partial H_t(a)}\pi_t(b) \\ &= \frac{\partial}{\partial H_t(a)}\left[\frac{e^{H_t(b)}}{\sum_{c=1}^{k} e^{H_t(c)}}\right] \\ &= \frac{\frac{\partial e^{H_t(b)}}{\partial H_t(a)} \sum_{c=1}^{k} e^{H_t(c)} - e^{H_t(b)} \frac{\partial \sum_{c=1}^{k} e^{H_t(c)}}{\partial H_t(a)}}{\left(\sum_{c=1}^{k} e^{H_t(c)}\right)^2} \\ &= \frac{\mathbb{1}_{a=b} e^{H_t(b)} \sum_{c=1}^{k} e^{H_t(c)} - e^{H_t(b)} e^{H_t(a)}}{\left(\sum_{c=1}^{k} e^{H_t(c)}\right)^2} \\ &= \frac{\mathbb{1}_{a=b} e^{H_t(b)}}{\sum_{c=1}^{k} e^{H_t(c)}} - \frac{e^{H_t(b)} e^{H_t(a)}}{\left(\sum_{c=1}^{k} e^{H_t(c)}\right)^2} \\ &= \mathbb{1}_{a=b}\pi_t(b) - \pi_t(b)\pi_t(a) \\ &= \pi_t(b)(\mathbb{1}_{a=b} - \pi_t(a)) \end{aligned} \tag{2.19}$$

上述算法是梯度上升算法的特例, 它可以保证算法具有较强的收敛鲁棒性[3]. 另外值得注意的是,X_t 的选取并不会影响算法的更新结果, 但是它会影响更新的方差以及收敛速率.

2.9 Associative Search (Contextual Bandits)

本章的前面几节介绍的都是非关联任务 (nonassociative tasks), 也就是说动作与状态 (situations) 之间没有关联. 在实际中更多的情况是, 动作与状态之间有关联关系, 策略必须基于状态映射到相应的动作上. 但是本节并没有将关联任务做展开.

[3]也有可能收敛至局部极小值.

2.10 Summary

本节除了总结之外, 还提到了如何对不同的算法进行评估. 这里用到的是学习曲线 (learning curve). 此外, 还有一些常见的探索-利用折中方法, 如 *Gittins indices*, *Baysian* 方法, *posterior* 采样, *Thompson* 采样.

Chapter 3 Finite Markov Decision Processes

本章主要展开介绍了第2.9节所提到的关联任务 (associative tasks) 及其对应的 MDP 模型.

3.1 The Agent-Environment Interface

本节主要介绍了什么是 *Markov Decision Processes* (MDP), 以及为什么要在强化学习中引入 MDP.

强化学习之所以使用 MDP 作为问题框架, 是因为 MDP 在本质上是一种从交互中进行学习并最终达到一定目的的问题框架, 这与强化学习的思路高度重合. 本章重点介绍的是有限 *MDP* (finite MDP), 有限指的是状态集 $\mathcal{S}$, 动作集 $\mathcal{A}$, 奖赏集 $\mathcal{R}$ 中所包含的元素个数均是有限的. 对于任意 $s, s' \in \mathcal{S}$, $r \in \mathcal{R}$, 以及 $a \in \mathcal{A}(s)$, 有

$$p(s', r \mid s, a) \doteq \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\} \tag{3.1}$$

基于这个定义可以推导出许多信息, 如

- **状态转移概率 (state-transition probabilities)**

$$p(s' \mid s, a) \doteq \Pr\{S_t = s' \mid S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in \mathcal{R}} p(s', r \mid s, a) \tag{3.2}$$

- **状态动作对期望奖赏 (expected rewards for state-action pairs)**

$$r(s, a) \doteq \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r \mid s, a) \tag{3.3}$$

- **状态动作状态三元组期望奖赏 (expected rewards for state-action-next-state triple**

$$r(s,a,s') \doteq \mathbb{E}[R_t \mid S_{t-1}=s, A_{t-1}=a, S_t=s'] = \sum_{r\in\mathcal{R}} r\frac{p(s',r\mid s,a)}{p(s'\mid s,a)} \tag{3.4}$$

除了对于 MDP 的介绍以外, 本章还介绍了智能体 (agent) 与环境的划分并不是通常的物理意义上的划分, 这种界限划分还具有灵活性.MDP 框架并不一定是最好的框架, 但它是一种具有普适性的框架.

3.2 Goals and Rewards

本节介绍的内容是目标和奖赏.

强化学习的最终目标是长期累积奖赏, 由短期奖赏累计得到最终的目标在实际应用中具有灵活性和普适性. 值得注意的是, 奖赏是你想让智能体达到什么目的, 而不是怎样让智能体达到目的.

> The all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward).

3.3 Returns and Episodes

在本节中, 主要介绍了 *return* 这一概念和一种任务种类的划分. 那么随之而来的一个问题就是 *return* 和 *reward* 之间有什么联系? 二者有什么区别?

假设经过 t 步之后, 后续的奖赏序列 $R_{t+1}, R_{t+2}, \cdots, R_T$. 但是我们最终想要最大化的是 *expected return* G_t, 也就是说奖赏序列的特定函数. 最简单的例子是

$$G_t \doteq R_{t+1} + R_{t+2} + \cdots + R_T \tag{3.5}$$

其中 T 是到达最后一步时的步数.

当智能体和环境之间的交互序列可以很自然地划分为子序列时, 我们把这种情况称为 *episode*, 相应的任务称为 *episodic task*. 相应的, 如果智能

体和环境之间的交互是连续的并且没有时间限制情况下, 相应的任务被称为 *continuing task*. 在 *continuing task* 中, 如果还使用式3.5形式的 *expected return*, 就会有一些问题: 由于 $T = \infty$, 等式很容易趋向于无穷大. 因此在本书中使用 *discounting expected return*

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3.6}$$

其中 $0 \leq \gamma \leq 1$, 被称为打折比例 (discount rate).

3.4 Unified Notation for Episodic and Continuing Tasks

为了后续讨论便利性, 最好将第3.3节中的两种类型任务进行统一定义.

我们定义 $S_{t,i}$ 为在第 i 个 episode, 第 t 步时的状态. 在不引起歧义的情况下也会使用 S_t 指代 $S_{t,i}$.

对于 *expected return*, 我们可以将式3.5视为3.6的一种特殊形式. 因此,*expected return* 可以统一定义为

$$G_t \doteq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \tag{3.7}$$

其中, 可以包括 $T = \infty$ 或者 $\gamma = 1$ 这两种特殊情况 (不能同时成立).

3.5 Policies and Value Functions

有了 *return* 这一定义并且将不同种类任务的任务进行归一化后, 接下来所要做的就是在最大化 *return* 这一目标下确定策略 (policy), 对于每一步的动作选择, 则依赖于价值函数或者动作价值函数.

强化学习在本质上学习的其实是策略. 价值函数或者动作价值函数在不同的策略下才有具体意义. 基于此, 状态 s 在策略 π 下的价值定义为:

$$v_\pi(s) \doteq \mathbb{E}[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right] \quad \text{for all } s \in \mathcal{S} \tag{3.8}$$

同样的, 动作价值函数定义为

$$q_\pi(s,a) \doteq \mathbb{E}_\pi\Big[G_t \mid S_t = s, A_t = a\Big] = \mathbb{E}_\pi\Big[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\Big] \tag{3.9}$$

状态价值函数的意义为, 基于 t 时刻的状态 s 能获得的对未来 *return* 的期望; 动作价值函数的意义为, 基于 t 时刻的状态 s, 所能获得的对未来 *return* 的期望.

强化学习和动态规划中所使用的价值函数满足递归性质, 即

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)\Big[r + \gamma \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s']\Big] \\ &= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_\pi(s')\Big] \end{aligned} \tag{3.10}$$

其中,$s, s' \in \mathcal{S}$, 分别代表当前状态以及采取动作 a 后的下一个状态;$a \in \mathcal{A}(s)$ 代表根据状态 s 所采取的动作. 式3.10又被称为 *Bellamn equation for* v_π.

3.6 Optimal Policies and Optimal Value Functions

强化学习的训练过程是一个不断迭代优化策略的过程. 如何比较两个策略的好坏, 训练结束后得到的策略是否是最优的策略 *optimal policy*, 是本节主要的阐述内容.

如果一个策略 π 在所有状态下的期望 return 均大于或等于策略 π' 的期望 return, 那么策略 π 比 π' 好或者两者一样好[1]. 基于最优策略, 那么就可以定义

- **最优状态值函数**

$$v_*(s) \doteq \max_\pi v_\pi(s) \tag{3.11}$$

对于所有的 $s \in \mathcal{S}$ 均成立.

[1]这里仅仅从定义上定义了策略的好坏, 但是在实际的训练过程中有什么评价指标?

- **最优动作值函数**

$$q_*(s,a) \doteq \max_\pi q_\pi(s,a) \tag{3.12}$$

对于所有的 $s \in \mathcal{S}, a \in \mathcal{A}(s)$ 均成立.

由于 v_* 是基于策略定义的最优价值函数, 那么它必定与式3.10满足逻辑自洽[2], 因此有最优 *Bellman* 等式

$$\begin{aligned} v_*(s) &\doteq \max_{a\in\mathcal{A}(s)} q_{\pi_*}(s,a) \\ &= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_*(s')] \end{aligned} \tag{3.13}$$

$$\begin{aligned} q_*(s,a) &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r \mid s, a)[r + \gamma \max_{a'} q_*(s', a')] \end{aligned} \tag{3.14}$$

在有限 MDP 的情况下, 式3.13有唯一解. 因为其本质上是一个方程组, 包含有 n 个未知量[3],n 个等式.

一旦确定了 v_*, 那么最优策略的确定也就相对容易 (*Once one has* v_**, it is relatively easy to determine an optimal policy).*[4]使用最优价值函数来确定最优策略的好处是, 根据当前的状态, 选择使得最优价值函数值最大的那个动作, 虽然这样看来是短期贪婪的选择, 但是在本质上长期累计奖赏也是最大的 (因为 v_* 已经考虑到了长期累计奖赏). 更进一步,q_* 会使得动作选择更加容易. 由于使用了状态—动作对来计算价值函数, 那么就不用关心后继状态以及它们的价值, 也就是说不用关注环境的动态.

直接计算式3.13得到最优策略仅仅是理论上可行, 但是在强化学习实践中由于状态空间过大, 实际上并不可行. 通常使用近似估计方法来求取最优策略. 另外还有启发式算法, 将 Bellman 等式视为树.

[2]这里的逻辑自洽是指最优状态值函数必定符合一般状态值函数所遵从的 Bellman 等式

[3]序列 $v_*(s_i), i = 1, 2, \cdots, n$

[4]那么在实际训练过程中, 到底是由最优策略确定最优价值函数, 还是由最优价值函数确定最优策略? 从本节的描述来看, 应该是由最优价值函数来确定最优策略.

3.7 Optimality and Approximation

本节简略介绍了为什么要使用近似方法而不是精确方法来计算最优策略.

虽然理论上可以通过最优价值函数求取最优策略, 但是在实际情况下, 可以提供给智能体的计算能力很有限, 尤其是在但不情况下计算量受到很大限制. 内存是另外一项限制因素.MDP 框架也迫使我们使用近似方法来解决强化学习问题. 强化学习的在线特性可以对常见的状态进行学习, 对不常见的状态进行忽略.

Chapter 4 Dynamic Programming

动态规划 (dynamic programming) 指的是在 MDP 模型下可以计算出最优策略的一系列算法. 传统 DP 有两大限制, 第一是它要求对环境进行精确的构建; 第二是它的计算量很大. 但其仍然有重要的理论地位.DP 的核心思想是使用价值函数对搜索过程进行组织和构建, 以得到最优策略.

4.1 Policy Evaluation (Prediction)

本节主要阐述的问题是在给定策略的情况下如何计算价值函数.

根据式3.10, 可以从理论上计算出策略 π 下的价值函数. 计算方法虽然直接, 但是计算量很大. 在实际情况中, 如果对环境有精确的建模, 那么可以使用迭代的方法逼近策略 π 下的价值函数

$$\begin{aligned} v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_k(s')\Big] \end{aligned} \tag{4.1}$$

其中 $\pi(a \mid s)$ 是策略的体现;$p(s', r \mid s, a)$ 是对环境进行精确建模的体现; 注意此处的 k 对应的并不是不同的单独状态 s', 而是状态 s 所有可能的后继状态.

更进一步, 从节省内的角度来看, 可以使用 in-place 算法. 并且 in-place 算法还具有收敛速度快的优势[1].

4.2 Policy Improvement

上一节中引入特定策略下的价值函数的目的是为了找到更好的策略. 本节主要介绍了策略提升理论.

[1]因为迭代出的新值可以立即使用, 而不用等到下一轮迭代才能使用.

Algorithm 2 Iterative policy evaluation (in-place version)

Input: π, the policy to be evaluated

Initializ an array $V(s) = 0$, for all $s \in \mathcal{S}^+$

repeat

$\Delta \leftarrow 0$

for each $s \in \mathcal{S}$ **do**

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

end for

until $\Delta < \theta$ (a small positive number)

Output: $V \approx v_\pi$

设 π 和 π' 是一对确定性策略, 对于所有的 $s \in \mathcal{S}$, 有

$$q_\pi(s, \pi'(s)) \geq v_\pi(s) \tag{4.2}$$

那么我们称策略 π' 和 π 一样好或者策略 π' 好于 π. 也就是说, 对于所有的 $s \in \mathcal{S}$:

$$v_{\pi'}(s) \geq v_\pi(s) \tag{4.3}$$

将上述的过程拓展至所有的状态 s 和所有的动作 a, 即从初始策略 π 作为起始点获取到贪心策略 π' 的过程被称为策略提升过程.

在实际情况下, 策略给出的是动作的概率分布, 那么我们就只要对所有的最优动作进行按权重相加即可. 权重的具体数值没有硬性规定, 只要非最优动作权重被赋值为 0 即可.

4.3 Policy Iteration

上一节介绍了策略提升, 本节就在策略提升的基础上介绍了寻找最优策略的算法框架.

寻找最优策略的迭代过程为

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_* \tag{4.4}$$

其中 $\xrightarrow{\text{E}}$ 是策略评估过程;$\xrightarrow{\text{I}}$ 是策略迭代过程.

通常情况下, 策略迭代只要使用较少的迭代轮数就可以收敛.

4.4 Value Iteration

策略评估过程由于需要对所有状态进行多次遍历, 所以其本身就是一个拖沓的计算过程. 引入值迭代 *(value iteration)* 后, 在原来的策略迭代环节只需要进行一轮状态迭代.

Algorithm 3 Value Iteration

Input: Initialize array V arbitrarily.

repeat

$\Delta \leftarrow 0$

for each $s \in \mathcal{S}$ **do**

$v \leftarrow V(s)$

$V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

end for

until $\Delta < \theta$ (a small positive number)

Output: $\pi(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')]$

相比于式4.1, 价值迭代

$$\begin{aligned} v_{k+1}(s) &\doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(s_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_k(s)] \end{aligned} \tag{4.5}$$

绕开了对策略的依赖.

4.5 Asymchronous Dynamic Programming

上一节提到的价值迭代需要对整个状态集合进行迭代, 在状态集合特别大的情况下, 计算量也会急剧增大. 本节针对这一问题引入了异步动态规划算法.

异步算法不用遍历整个状态集合, 只要对某些特定的状态子集进行遍历. 同时需要注意的是, 节省计算资源不是最终目的, 策略的快速收敛才是引入异步算法的最终目的. 在 MDP 构建的过程中可以使用异步动态规划算法进行并行的计算迭代.

4.6 Generalized Policy Iteration

本节对动态规划的迭代过程进行了一般性拓展, 引入了广义策略迭代 *(Generalized Policy Iteration, GPI)* 的概念.

GPI 是指不对策略评估和策略提升这两个过程的间隔尺寸和粒度做硬性规定的策略迭代过程.

4.7 Efficiency of Dynamic Programming

动态规划算法作为解决 *MDP* 问题的一种经典算法, 其算法复杂度也是值得关注的一个方面.

动态规划算法是多项式时间复杂度. 此外还有线性规划算法, 但是不是特别适合解决大型问题.

4.8 Summary

本章展现了动态规划算法的一个重要特性 *bootstrapping*[2]. 接下来的一章将会介绍一种不需要对环境进行建模, 并且不需要 bootstrapping 的方法. 之后还会介绍不需要进行环境建模但是需要 bootstrapping 的方法.

[2]That is, they update estimates on the basis of other estimates.

Chapter 5 Monte Carlo Methods

相比于动态规划方法, *Monte Carlo* 方法只依赖于经验, 不需要对环境进行精确建模, 只要有部分状态和动作的采样即可. *Monte Carlo* 基于对 *return* 进行采样取平均的方法来解决强化学习问题.

5.1 Monte Carlo Prediction

本节解决的是 *MC* 方法的 *prediction* 问题.

Algorithm 4 First-visit MC prediction, for estimating $V \approx v_\pi$

Input:

$\pi \leftarrow$ policy to be evaluated

$V \leftarrow$ an arbitary state-value function

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

loop

Generate an epsiode using π {MC 方法是 epsiode-by-epsiode 方式的.}

for each state s appearing in the epsiode **do**

$G \leftarrow$ the return that follows the *first occurence* of s

Append G to $Return(s)$

$V(s) \leftarrow$ average($Return(s)$) {可以用之前提到的增量式方式计算均值}

end for

end loop

prediction 问题也就是给定策略计算出价值函数的问题. 所有的 MC 方法都暗含着一个思想, 即当越来越多的 return 被观察到时, 他们的均值应该收敛到期望价值. 假设在状态 s 下, 有策略 π, 我们想要估计价值函数 $v_\pi(s)$.

如果我们只估计整个序列中首个状态 s 后的价值函数, 那么这种情况被称为 *first-visit MC 方法*. First-visit MC 算法见 Algorithm 4. 如果我们估计的是序列中每个状态 s 后的价值函数, 那么这种情况被称为 *every-visit MC 方法*.

MC 方法一个值得注意的点是, 每个状态的估计值都是相互独立的.

5.2 Monte Carlo Estimation of Action Values

前一节中的 *prediction* 问题估计的是某一个策略下的价值函数. 但是如果在模型未知的情况下, 很难从 *MDP* 四元组 *(式3.1)* 中获取到不同状态之间的状态迁移情况, 所以通常情况下估计的是动作价值函数.

由于动作-状态对的数量很多, 如果直接套用 MC 方法可能会有许多动作-状态对未被访问到. 这个就是探索保持问题 (maintaining exploration). 对于这个问题, 通常有两种解决方法:

- 采样时候依次指定每一个状态-动作对作为起点;
- 每一个状态-动作对初始概率赋值为非 0 值.

5.3 Monte Carlo Control

在定义了 *prediction* 问题之后, 紧接着需要解决 *control* 问题.

类比于 DP 方法的策略迭代式4.4, 可以定义 MC 方法的策略迭代

$$\pi_0 \xrightarrow{\mathrm{E}} q_{\pi_0} \xrightarrow{\mathrm{I}} \pi_1 \xrightarrow{\mathrm{E}} q_{\pi_1} \xrightarrow{\mathrm{I}} \pi_2 \xrightarrow{\mathrm{E}} \cdots \xrightarrow{\mathrm{I}} \pi_* \xrightarrow{\mathrm{E}} q_* \tag{5.1}$$

其中, 策略提升可以通过贪心选择动作, 使

$$\pi(s) \doteq \arg\max_a q(s,a) \tag{5.2}$$

最大化实现.

Algorithm 5 First-visit MC prediction, for estimating $V \approx v_\pi$

Input: for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$

$Q(s,a) \leftarrow$ arbitary

$\pi(s) \leftarrow$ arbitary

$Returns(s,a) \leftarrow$ empty list

loop

Choose $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}$ s.t. all pairs have probability > 0

Generate an epsiode starting from S_0, A_0 following π

{policy evaluation begin}

for each pair s, a appearing in the epsiode **do**

$G \leftarrow$ the return that follows the *first occurence* of s, a

Append G to $Return(s,a)$

$Q(s,a) \leftarrow$ average($Return(s,a)$)

end for

{policy evaluation end}

{policy improvement begin}

for each s in the episode **do**

$\pi(s) \leftarrow \arg\max_a Q(s,a)$

end for

{policy improvement end}

end loop

MC 方法为什么可以在只有样本, 没有环境建模的情况下找出最优策略? 因为对于所有的 $s \in \mathcal{S}$, 有

$$\begin{aligned} q_{\pi_k}(s, \pi_{k+1}(s)) &= q_{\pi_k}(s, \arg\max_a q_{\pi_k}(s,a)) \\ &= \max_a q_{\pi_k}(s,a) \\ &\geq q_{\pi_k}(s, \pi_k(s)) \\ &\geq v_{\pi_k}(s) \end{aligned} \tag{5.3}$$

最后一个不等号成立是因为对于 $q_{\pi_k}(s, \pi_k(s))$, 相当于状态 s 的最大奖赏与 $v_{\pi_k}(s')$ 的加和, s' 是 s 采取动作 $\arg\max_a q_{\pi_k}(s,a)$ 后的下一个状态.

至此, 我们做了两点假设确保 MC 方法可以收敛

1. 每一个 episode 都有探索起始点;
2. 策略评估基于无限多个 episode 完成.

本章后续内容将会重点阐述去除第 1 条假设. 本节接下来阐述如何去除第 2 条假设.

不管是 DP 方法还是 MC 方法, 去除第 2 条假设有两种方法:

- 在每一次策略评估中对 q_{π_k} 进行近似估计;
- 基于 GPI, 避免每一次策略提升之前都进行策略评估.

当策略和动作价值函数都达到最优时, 算法也达到稳定状态.

5.4 Monte Carlo Control without Exploring Starts

本节紧接着上一节的内容, 介绍了去除 exploring starts 假设的方法之一, on-policy 方法.

On-policy 方法与 off-policy 方法的区别在于是否对判别用的策略和产生数据用的策略作区分.

既然 on-policy 方法在 policy iteration 中产生数据和判别使用的是同一个策略, 那么就在数据生成环节做一些文章.

Algorithm 6 On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Input: for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$

$Q(s,a) \leftarrow$ arbitary

$Returns(s,a) \leftarrow$ empty list

$\pi(a \mid s) \leftarrow$ an arbitary ε-soft policy

loop

Generate an episode using π

{policy evaluation begin}

for each pair s, a appearing in the episode **do**

$G \leftarrow$ the return that follows the first occurrence of s, a

Append G to $Return(s,a)$

$Q(s,a) \leftarrow$ average$(Return(s,a))$

end for

{policy evaluation end}

{policy improvement begin}

for each s in the episode **do**

$A^* \leftarrow \arg\max_a Q(s,a)$ {with ties broken arbitary}

for all $a \in \mathcal{A}(s)$ **do**

$$\pi(a \mid s) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq A^* \end{cases}$$

end for

end for

{policy improvement end}

end loop

需要注意的是, GPI 并不要求策略始终保持贪心选择, 只要策略可以趋近于贪心策略即可. 因此可以使用 ε-贪心方法, 如 Algorithm 6所示. 任意一个 ε-贪心方法对应的 q_π 可以由 ε-soft 策略 π 确保提升, 是因为有 *policy improvement theorem*. 假设 π' 是 ε-贪心策略, 对于任意 $s \in \mathcal{S}$ 有

$$\begin{aligned} q_\pi(s, \pi'(s)) &= \sum_a \pi'(a \mid s) q_\pi(s, a) \\ &= \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) + (1-\varepsilon) \max_a q_\pi(s, a) \\ &\geq \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) + (1-\varepsilon) \sum_a \frac{\pi(a \mid s) - \frac{\varepsilon}{|\mathcal{A}(s)|}}{1-\varepsilon} q_\pi(s, a) \\ &= \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) - \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) + \sum_a \pi(a \mid s) q_\pi(s, a) \\ &= v_\pi(s) \end{aligned} \tag{5.4}$$

[1]因此基于动作价值函数, 也可以提升策略.

在判定环节, 从价值函数的角度来看, ε-soft 策略与原始策略是一样好的. 设 $\tilde{v}_*$ 和 $\tilde{q}_*$ 定义的是 ε-soft 条件下的最优价值函数. 那么当且仅当 $v_\pi = \tilde{v}_*$ 时, 策略 π 才是最优的.

从定义上, 可以知道 $\tilde{v}_*$ 是下式的唯一解.

$$\begin{aligned} \tilde{v_\pi}(s) &= (1-\varepsilon) \max_a \tilde{q}_*(s, a) + \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a \tilde{q}_*(s, a) \\ &= (1-\varepsilon) \max_a \sum_{s', r} p(s', r \mid s, a)[r + \gamma \tilde{v}_*(s')] \\ &\quad + \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a \sum_{s', r} p(s', r \mid s, a)[r + \gamma \tilde{v}_*(s')] \end{aligned} \tag{5.5}$$

根据式5.4, 有

$$\begin{aligned} v_\pi(s) &= (1-\varepsilon) \max_a q_*(s, a) + \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_*(s, a) \\ &= (1-\varepsilon) \max_a \sum_{s', r} p(s', r \mid s, a)[r + \gamma v_*(s')] \\ &\quad + \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a \sum_{s', r} p(s', r \mid s, a)[r + \gamma v_*(s')] \end{aligned} \tag{5.6}$$

[1]The sum is a weighted average with nonnegative weights summing to 1, and as such it must be less than or equal to the largest number averaged.

由于 $\tilde{v}_*$ 是唯一解, 因此有 $v_\pi = \tilde{v}_*$.

由此有最终结论, GPI 也可以用于 ε-soft 方法.

5.5 Off-policy Prediction via Importance Sampling

本节介绍了去除 *exploring starts* 前提假设的另一种方法—*off-policy* 方法.

On-policy 方法在判别环节和数据生成环节使用的是同一种策略, 而 off-policy 方法使用的是两种不同的策略. 学习并且最终得到的策略是目标策略 (target policy), 用于探索生成数据的策略是行为策略 (behavior policy).

虽然 off-policy 方法具有方差大和收敛速度慢的缺点, 但是它能力更强, 更具有一般性.

首先需要解决的是 off-policy 方法的 prediction 问题, 即给定行为策略 b 的情况下, 如何评估目标策略 π 的价值 v_π 或者动作价值 q_π. 为了保证问题有解, 要求 π 下所采取的动作, 同样可以被策略 b 采取, 这就是 *coverage* 假设.

几乎所有的 off-policy 方法都可以使用 *importance sampling* 方法解决. 假设在策略 π 下, 由起始状态 S_t 开始有状态动作轨迹 $A_t, S_{t+1}, A_{t+1}, \cdots, S_T$, 那么有

$$\begin{aligned}
&\Pr\{A_t, S_{t+1}, A_{t+1}, \cdots, S_T \mid S_t, A_{t:T-1} \sim \pi\} \\
&\quad = \pi(A_t \mid S_t) p(S_{t+1} \mid S_t, A_t) \pi(A_{t+1} \mid S_{t+1}) \cdots p(S_T \mid S_{T-1}, A_{T-1}) \\
&\quad = \prod_{k=t}^{T-1} \pi(A_k \mid S_k) p(S_{k+1} \mid S_k, A_k)
\end{aligned} \tag{5.7}$$

其中 p 是式3.2所指代的状态转移概率. 因此 importance-smpling ratio 为

$$\rho_{t:T-1} \doteq \frac{\prod_{k=t}^{T-1} \pi(A_k \mid S_k) p(S_{k+1} \mid S_k, A_k)}{\prod_{k=t}^{T-1} b(A_k \mid S_k) p(S_{k+1} \mid S_k, A_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k \mid S_k)}{b(A_k \mid S_k)} \tag{5.8}$$

那么, 基于 MC 方法有两种不同的 importance sampling 方法

- **ordinary importance sampling**

$$V(s) \doteq \frac{\sum_{t\in\mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{|\mathcal{T}(s)|} \tag{5.9}$$

- **weighted importance sampling**

$$V(s) \doteq \frac{\sum_{t\in\mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\sum_{t\in\mathcal{T}(s)} \rho_{t:T(t)-1}} \tag{5.10}$$

上述两种方法的差异在于偏差和方差[2].

5.6 Incremental Implementation

前一节主要从理论层面介绍了什么是 *off-policy* 方法, 以及 *off-policy* 方法背后的 *importance sampling*. 本节主要从实现的角度介绍了增量式实现 *off-policy* 的 *prediction* 问题. 具体实现方法见 Algorithm 7.

5.7 Off-policy Monte Carlo Control

本节主要介绍如何实现增量式目标策略估计. 具体算法见 Algorithm 8.

将目标策略和行为策略区分开的优势在于, 在目标策略固定的情况下, 行为策略可以持续地对所有的动作进行采样.

如果一个 episode 中非贪心动作比重过多, 就会减缓学习速率. 因此, 引入了时序差分学习方法, 这也是下一章的主要内容.

[2]仔细考虑在偏差和方差上有什么差异. 见原书 P86

Algorithm 7 Off-policy MC prediction. for estimating $Q \approx q_\pi$

Input: an arbitary target policy π

Initialize, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$

$Q(s,a) \leftarrow$ arbitary

$C(s,a) \leftarrow 0$

loop

$b \leftarrow$ any policy with coverage of π

Generate an episode using b: $S_0, A_0, R_1, \cdots, S_{T-1}, A_{T-1}, R_T, S_T$

$G \leftarrow 0$

$W \leftarrow 1$

for $t = T-1, T-2, \cdots$ down to 0 **do**

$G \leftarrow \gamma G + R_{t+1}$

$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$

$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)}[G - Q(S_t, A_t)]$

$W \leftarrow W \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$

if $W = 0$ **then**

exit For loop

end if

end for

end loop

Algorithm 8 Off-policy MC control. for estimating $\pi \approx \pi_*$

Input:

Initialize, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$

$Q(s,a) \leftarrow$ arbitary

$C(s,a) \leftarrow 0$

$\pi(s) \leftarrow \arg\max_a Q(S_t, a)$

loop

$b \leftarrow$ any soft policy

Generate an episode using b: $S_0, A_0, R_1, \cdots, S_{T-1}, A_{T-1}, R_T, S_T$

$G \leftarrow 0$

$W \leftarrow 1$

for $t = T-1, T-2, \cdots$ down to 0 **do**

$G \leftarrow \gamma G + R_{t+1}$

$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$

$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)}[G - Q(S_t, A_t)]$

$\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$

if $A_t \neq \pi(S_t)$ **then**

exit For loop

end if

$W \leftarrow W \frac{1}{b(A_t|S_t)}$

end for

end loop

Chapter 6 Temporal-Difference Learning

本章主要介绍了时序差分方法, TD 方法将 DP 方法和 MC 方法结合在一起, 与这两种方法有关联之处, 但是在 prediction 问题上有明显区别.

6.1 TD Prediction

与前两章一样, 首先介绍的是 TD 方法的 prediction 问题.

对于 MC 方法, 状态价值函数更新的公式为

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big] \tag{6.1}$$

但是对于 TD 方法, 则有

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big] \tag{6.2}$$

即 TD 方法的状态价值函数更新不用等到整个 episode 进行完才能进行, 只要得到下一个状态 S_{t+1} 以及对应的状态价值函数就可以进行更新. 于是, 算法流程如 Algorithm 9所示. 之所以可以这么做是因为有

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \end{aligned} \tag{6.3}$$

同时可以得到, MC 方法和 TD 方法都是 *sample update*, 即通过样例的状态转换进行价值函数的更新. 与此相对应的, DP 方法是 *expected update*, 它是通过整个分布上的状态转换情况进行价值函数的更新.

此外, 一般将 S_t 的价值和估计值 $R_{t+1}+\gamma V(S_{t+1})$ 之间的差值称为 *TD error*.

$$\delta_t \doteq R_{t+1}+\gamma V(S_{t+1})-V(S_t) \tag{6.4}$$

Algorithm 9 Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated

Initialize $V(s)$ arbitarily (e.g., $V(s)=0$, for all $s\in\mathcal{S}^+$)

repeat

{for each step of episode}

$A \leftarrow$ action given by π for S

Take action A, observe R, S'

$V(S) \leftarrow V(S)+\alpha\Big[R+\gamma V(S')-V(S)\Big]$

until S is the terminal

6.2 Advantages of TD Prediction Methods

在对 TD 的 prediction 问题进行了介绍后, 就需要思考 TD 方法有什么优势. 前文也提到过, TD 是 bootstrap 的, 这种形式是好是坏?

相比于 DP 方法, TD 方法的优势在于, 不需要对环境进行建模, 不需要对奖赏进行建模, 不需要对下一个状态迁移的分布进行建模. 相比于 MC 方法, TD 方法的优势在于在线式和增量式实现.

整体上来说, TD 方法的收敛性是可以得到保证的, 这个将在第 9 章[1]进行详细阐述. 相比于 MC 方法, TD 的收敛速率并无定论, 但从复杂任务的观测结果来看, TD 方法的手链速率比 constant-α MC 更快一些.

6.3 Optimality of TD(0)

TD(0) 方法的最优性体现在收敛速率上. 本节更好的翻译可以参考这里[2].

[1] TODO: 替换成引用形式

[2] https://blog.csdn.net/coffee_cream/article/details/70194456

假如训练集中的经验数量是有限的, 增量式学习方法不断重复利用这些经验, 直到该方法收敛到某个值. 即给定一个价值函数估计 V, 在每一个时刻访问到一个非终止状态并计算出一个增量. 但是价值函数只被更新一次, 更新所有增量的加和. 这种更新方式被称为 *batch update*. 在 batch update 模式下, 只要 α 足够小, 那么 TD(0) 始终会收敛到同一个值. 同样条件下, constant-α MC 方法也会收敛, 但是每次收敛到的数值有可能会不同.

在批模式下, TD(0) 比 MC 方法收敛速率快的原因在于, TD(0) 计算的是等确定性估计[3](certainty-equivalence estimate). 等确定性估计在某些情况下是一种最优方案, 但是它的计算量比较大. TD 方法是估计等确定性的唯一可行方法.

6.4 Sarsa: On-policy TD Control

Sarsa 算法主要用来解决同策略 TD control 问题.

前面几节主要关注的是基于状态价值函数的 TD(0) 方法, 这一节我们主要介绍的是基于动作价值函数的 TD(0) 方法, 其更新公式为

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big] \tag{6.5}$$

如果 S_{t+1} 是终止状态, 那么 $Q(S_{t+1}, A_{t+1})$ 的值为 0. 算法流程见 Algorithm 10.

Sarsa 算法的收敛性质基于策略对 Q 的依赖本质. 这句话应该如何理解?

6.5 Q-learning: Off-policy TD Control

Q-Learning 算法主要用来解决异策略 TD 方法的 control 问题.

[3]Given this model, we can compute the estimate of the value function that would be exactly correct if the model were correct. This is called the *certainty-equivalence estimate*, because it is equivalent to assuming that the estimate of the underlying process was known with certainty rather than being approximate. (On Page 104)

Algorithm 10 Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Input: Initialize $Q(s,a)$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}$, arbitarily, and $Q(terminal\text{-}state,\cdot) = 0$

repeat

{for each episode}

Initialize S

Choose A from S using policy derived from Q (e.g. ϵ-greedy)

repeat

{for each step of episode}

Take action A, observe R, S' using policy derived from Q (e.g. ϵ-greedy)

$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$

$S \leftarrow S'$, $A \leftarrow A'$

until S is terminal

until

Q-Learning 算法的更新公式为

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t)\Big] \tag{6.6}$$

具体算法流程见 Algorithm 11.

6.6 Expected Sarsa

本节介绍的 Expected Sarsa 算法是对 Sarsa 算法的进一步改进.

虽然 Expected Sarsa 算法相比于 Sarsa 有着更高的计算复杂度, 但是它消除了由于随机选择 A_{t+1} 而引入的方差. 其更新公式如下

$$\begin{aligned} Q(S_t,A_t) &\leftarrow Q(S_t,A_t) + \alpha\Big[R_{t+1} + \gamma \mathbb{E}[Q(S_{t+1},A_{t+1}) \mid S_{t+1}] - Q(S_t,A_t)\Big] \\ &\leftarrow Q(S_t,A_t) + \alpha\Big[R_{t+1} + \gamma \sum_a \pi(a \mid S_{t+1}) Q(S_{t+1},a) - Q(S_t,A_t)\Big] \end{aligned} \tag{6.7}$$

Algorithm 11 Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Input: Initialize $Q(s,a)$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}$, arbitarily, and $Q(terminal\text{-}state, \cdot) = 0$

repeat

{for each episode}

Initialize S

repeat

{for each step of episode}

Choose A from S using policy derived from Q (e.g. ϵ-greedy)

Take action A, observe R, S'

$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$

$S \leftarrow S'$

until S is terminal

until

6.7 Maximization Bias and Double Learning

到目前为止介绍的 *control* 问题都涉及到最大化价值操作, 这就会导致正向偏差. 本节主要介绍一种方法, 用以改进这个问题.

之所以出现正向偏差, 是因为使用同样的采样方法去决定使得价值函数最大化的动作, 同时去最大化价值. 那么 Double Learning 算法 (Algorithm 12) 就有效地解决了这一问题.

如果 Q_1 和 Q_2 的功能是固定的, 那么最终的结果仍然有可能是有偏估计, 所以要对 Q_1 和 Q_2 的功能进行轮换.

6.8 Games, Afterstates, and Other Special Cases

除了状态价值和动作价值外, 还有其他种类的价值选择方式.

本节主要介绍的后状态价值 (afterstates) 可以对状态进行合理的简化.

Algorithm 12 Double Q-learning

Input:

Initialize $Q_1(s,a)$ and $Q_2(s,a)$, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$, arbitarily

Initialize $Q_1(\textit{terminal-state},\cdot) = Q(\textit{terminal-state},\cdot) = 0$

repeat

{for each episode}

Initialize S

repeat

{for each step of episode}

Choose A from S using policy derived from Q_1 and Q_2 (e.g., ϵ-greedy in $Q_1 + Q_2$)

Take action A, observe R, S'

With 0.5 probability:

$$Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big(R + \gamma Q_2(S', \arg\max_a Q_1(S',a)) - Q_1(S,A)\Big)$$

else:

$$Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\Big(R + \gamma Q_1(S', \arg\max_a Q_2(S',a)) - Q_2(S,A)\Big)$$

$S \leftarrow S'$

until S is terminal

until

Chapter 7 n-step Bootstrapping

本章介绍的 *n-step TD* 方法是对位于两个极端的 *MC* 方法和 *TD(0)* 方法的折中, 更适用于一般性的情形.

7.1 n-step TD Prediction

本节主要介绍的是 *n-step TD* 方法的 *prediction* 问题.

与 MC 方法和 TD(0) 方法的 prediction 问题一样, 首先对 n-step 方法的 return 进行定义

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n}) \tag{7.1}$$

对于所有的 n, t 有 $n \geq 1$ 以及 $0 \leq t \leq T - n$. 如果 $t + n \geq T$, 那么所有的缺失项都被定义成 0. 相应的, 状态价值函数学习算法更新公式为

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha \Big[G_{t:t+n} - V_{t+n-1}(S_t) \Big] \tag{7.2}$$

其中 $0 \leq t \leq T$. 更新时其他所有状态的价值都保持不变, 即对于所有的 $s \neq S_t$, 有 $V_{t+n}(s) = V_{t+n-1}(s)$.

n-step 算法使用价值函数 V_{t+n-1} 来校正 R_{t+n} 之后的 return 缺失部分, 期望 n-step return 的最差误差不大于 V_{t+n-1} 下的误差的 γ^n 倍. 对于所有的 $n \geq 1$, 有

$$\max_s \Big| \mathbb{E}[G_{t:t+n} \mid S_t = s] - v_\pi(s) \Big| \leq \gamma^n \max_s \Big| V_{t+n-1}(s) - v_\pi(s) \Big| \tag{7.3}$$

具体算法流程见 Algorithm 13.

Algorithm 13 n-step TD for estimating $V \approx v_\pi$

Input:

Initialize $V(s)$ arbitrarily, $s \in \mathcal{S}$

Parameters: step size $\alpha \in (0, 1]$, a positive integer n

All store and access operations (for S_t and R_t) can take their index mod n

loop {for each episode}

Initialize and store $S_0 \neq$ terminal

$T \leftarrow \infty$

repeat $\{t = 0, 1, 2, \cdots\}$

if $t < T$ **then**

Take an action according to $\pi(\cdot \mid S_t)$

Observe and store the next reward as R_{t+1} and the next state as S_{t+1}

if S_{t+1} is terminal **then**

$T \leftarrow t + 1$

end if

end if

$\tau \leftarrow t - n + 1$ {τ is the time whose estimate is being updated}

if $\tau \geq 0$ **then**

$G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$

if $\tau + n < T$ **then**

$G \leftarrow G + \gamma^n V(S_{\tau+n})$

end if

$V(S_\tau) \leftarrow V(S_\tau) + \alpha[G - V(S_\tau)]$

end if

until $\tau = T - 1$

end loop

7.2 n-step Sarsa

本节主要展示了将 *n-step* 方法与 *Sarsa* 方法直接结合，形成一种 *on-policy TD control* 方法.

n-step 方法的核心思想是将式7.2中的状态价值函数替换成动作状态价

值函数,

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha\Big[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)\Big], \quad 0 \le t \le T, \tag{7.4}$$

对于所有的 $s \neq S_t$ 和 $a \neq A_t$, 所有的动作价值函数保持不变 $Q_{t+n}(s,a) = Q_{t+n-1}(s,a)$. 然后使用 ϵ-greedy 方法估计策略.

另外, 对于 expected n-step Sarasa, 有

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \sum_a \pi(a \mid S_{t+n}) Q_{t+n-1}(S_{t+n}, a) \tag{7.5}$$

对于所有的 n 和 t 均有 $n \ge 1$ 和 $0 \le t \le T - n$.

具体算法流程见 Algorithm 14.

Algorithm 14 n-step Sarsa for estimating $Q \approx q_*$, or $Q \approx q_\pi$ for a given π

Input:

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}$, $a \in \mathcal{A}$

Initialize π to be ϵ-greedy with respect to Q, or to a fixed given policy

Parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$, a positive integer n

All store and access operations (for S_t, A_t and R_t) can take their index mod n

loop {for each episode}

Initialize and store $S_0 \neq$ terminal

Select and store an action $A_0 \sim \pi(\cdot \mid S_0)$

$T \leftarrow \infty$

repeat $\{t = 0, 1, 2, \cdots\}$

if $t < T$ **then**

Take an action A_t

Observe and store the next reward as R_{t+1} and the next state as S_{t+1}

if S_{t+1} is terminal **then**

$T \leftarrow t + 1$

else

Select and store an action $A_t \sim \pi(\cdot \mid S_{t+1})$

end if
end if
$\tau \leftarrow t - n + 1$ {τ is the time whose estimate is being updated}
if $\tau \geq 0$ **then**
$G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$

if $\tau + n < T$ **then**
$G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$
end if
$Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha[G - Q(S_\tau, A_\tau)]$
If π is being learned, then ensure that $\pi(\cdot \mid S_\tau)$ is ϵ-greedy wrt Q
end if
until $\tau = T - 1$
end loop

7.3 n-step Off-policy Learning by Importance Sampling

本节主要介绍的是使用 *off-policy* 方法解决 *n-step TD* 方法的 *control* 问题.

off-policy 的更新公式为

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha \rho_{t:t+n-1}[G_{t:t+n} - V_{t+n-1}(S_t)], \quad 0 \leq t \leq T \tag{7.6}$$

其中 $\rho_{t:t+n-1}$ 被称为 *importance sampling ratio*, 具体形式为

$$\rho_{t:h} \doteq \prod_{k=t}^{\min(h,T-1)} \frac{\pi(A_k \mid S_k)}{b(A_k \mid S_k)} \tag{7.7}$$

类似的, n-step Sarsa 在 off-policy 下的更新公式为

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha \rho_{t+1:t+n-1}[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)] \tag{7.8}$$

需要注意的是, 更新从 $t+1$ 步开始, 因为需要获取到完整的动作状态对.

详细算法流程参见 Algorithm 15

Algorithm 15 Off-policy n-step Sarsa for estimating $Q \approx q_*$, or $Q \approx q_\pi$ for a given π

Input:

an arbitrary behavior policy b such that $b(a \mid s) > 0$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}$

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}$, $a \in \mathcal{A}$

Initialize π to be ϵ-greedy with respect to Q, or to a fixed given policy

Parameters: step size $\alpha \in (0,1]$, small $\epsilon > 0$, a positive integer n

All store and access operations (for S_t, A_t and R_t) can take their index mod n

loop {for each episode}

Initialize and store $S_0 \neq$ terminal

Select and store an action $A_0 \sim b(\cdot \mid S_0)$

$T \leftarrow \infty$

repeat $\{t = 0, 1, 2, \cdots\}$

if $t < T$ **then**

Take an action A_t

Observe and store the next reward as R_{t+1} and the next state as S_{t+1}

if S_{t+1} is terminal **then**

$T \leftarrow t + 1$

else

Select and store an action $A_{t+1} \sim b(\cdot \mid S_{t+1})$

end if

end if

$\tau \leftarrow t - n + 1$ {τ is the time whose estimate is being updated}

if $\tau \geq 0$ **then**

$\rho \leftarrow \prod_{i=\tau+1}^{\min(\tau+n-1,T-1)} \frac{\pi(A_i|S_i)}{b(A_i|S_i)}$

$G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$

if $\tau + n < T$ **then**

$G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$

end if

$Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\rho[G - Q(S_\tau, A_\tau)]$

If π is being learned, then ensure that $\pi(\cdot \mid S_\tau)$ is ϵ-greedy wrt Q

end if

until $\tau = T - 1$

end loop

7.4 Per-reward Off-policy Methods

本节介绍的方法对 off-policy 方法的效率进行提升.

普通的 n-step 方法的 return 主要遵从下列定义

$$G_{t:h} = R_{t+1} + \gamma G_{t+1:h} \tag{7.9}$$

per-reward 定义的 return 格式如

$$G_{t:h} \doteq \rho_t(R_{t+1} + \gamma G_{t+1:h}) + (1 - \rho_t)V_{h-1}(S_t), \quad t \le h \le T \tag{7.10}$$

其中 $G_{t:t} \doteq V_{t-1}(S_t)$. 上式等号右端第一项表示第 $t+1$ 步的 return 进行 importance sampling, 第二项表示 t 步之后的价值估计.

本节介绍的这种方法增大了方差, 导致学习速率较慢, 但是有解决方法. 具体参考文献可以参见本书第 123 页下方.

7.5 Off-policy Learning Without Importance Sampling: The n-step Tree Backup Algorithm

Tree-backup 是另外一种 off-policy 算法.

在之前介绍的算法中, 由于我们没有对未使用的动作进行采样, 所以在 tree-backup 算法中, 我们使用未使用动作的估计采样值对目标值进行估计.

由此, 相应的 return 定义为

$$\begin{aligned}
G_{t:t+n} &\doteq R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a \mid S_{t+1}) Q(S_{t+1}, a) + \gamma \pi(A_{t+1} \mid S_{t+1}) G_{t+1:t+n} \\
&= \delta'_t + Q_{t-1}(S_t, A_t) - \gamma \pi(A_{t+1} \mid S_{t+1}) Q_t(S_{t+1} \mid A_{t+1}) \\
&\quad + \gamma \pi(A_{t+1} \mid S_{t+1}) G_{t+1:t+n} \\
&= Q_{t-1}(S_t, A_t) + \delta'_t + \gamma \pi(A_{t+1} \mid S_{t+1})(G_{t+1:t+n} - Q_t(S_{t+1}, A_{t+1})) \\
&= Q_{t-1}(S_t, A_t) + \delta'_t + \gamma \pi(A_{t+1} \mid S_{t+1}) \delta'_{t+1} \\
&\quad + \gamma^2 \pi(A_{t+1} \mid S_{t+1}) \pi(A_{t+1} \mid S_{t+1}) \pi(A_{t+2} \mid S_{t+2}) \delta'_{t+2} + \cdots \\
&= Q_{t-1}(S_t, A_t) + \sum_{k=t}^{\min(t+n-1, T-1)} \delta'_k \prod_{i=t+1}^{k} \gamma \pi(A_i \mid S_i)
\end{aligned} \tag{7.11}$$

详细算法流程参见 Algorithm 16.

Algorithm 16 n-step Tree Backup for estimating $Q \approx q_*$, or $Q \approx q_\pi$ for a given π

Input:

Initialize $Q(s, a)$ arbitrarily, for all $s \in \mathcal{S}$, $a \in \mathcal{A}$

Initialize π to be ϵ-greedy with respect to Q, or to a fixed given policy

Parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$, a positive integer n

All store and access operations (for S_t, A_t and R_t) can take their index mod n

loop {for each episode}

Initialize and store $S_0 \neq$ terminal

Select and store an action $A_0 \sim \pi(\cdot \mid S_0)$

Store $Q(S_0, A_0)$ as Q_0

$T \leftarrow \infty$

repeat $\{t = 0, 1, 2, \cdots\}$

if $t < T$ **then**

Take an action A_t

Observe and store the next reward as R_{t+1} and the next state as S_{t+1}

if S_{t+1} is terminal **then**
$T \leftarrow t+1$
Store $R - Q_t$ as δ_t
else
Store $R + \gamma \sum_a \pi(a \mid S_{t+1}) Q(S_{t+1}, a) - Q_t$ as δ_t
Select arbitrarily and store an action as A_{t+1}
Store $Q(S_{t+1}, A_{t+1})$ as Q_{t+1}
Store $\pi(A_{t+1} \mid S_{t+1})$ as π_{t+1}
end if
end if
$\tau \leftarrow t - n + 1$ {τ is the time whose estimate is being updated}
if $\tau \geq 0$ **then**
$Z \leftarrow 1$
$G \leftarrow Q_\tau$
for $k = \tau, \cdots, \min(\tau + n + 1, T - 1)$ **do**
$G \leftarrow G + Z\delta_k$
$Z \leftarrow \gamma Z_{\pi_{k+1}}$
end for
$Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha[G - Q(S_\tau, A_\tau)]$
If π is being learned, then ensure that $\pi(a \mid S_\tau)$ is ϵ-greedy wrt $Q(S_\tau, \cdot)$
end if
until $\tau = T - 1$
end loop

7.6 A Unify Algorithm: n-step $Q(\sigma)$

详细算法流程参见 Algorithm 17.

Algorithm 17 Off-policy n-step $Q(\sigma)$ for estimating $Q \approx q_*$, or $Q \approx q_\pi$ for a given π

Input:

an arbitrary behavior policy b such that $b(a \mid s) > 0$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}$
Initialize $Q(s, a)$ arbitrarily, for all $s \in \mathcal{S}$, $a \in \mathcal{A}$
Initialize π to be ϵ-greedy with respect to Q, or to a fixed given policy
Parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$, a positive integer n
All store and access operations can take their index mod n
loop {for each episode}
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim \pi(\cdot \mid S_0)$
  Store $Q(S_0, A_0)$ as Q_0
  $T \leftarrow \infty$
  repeat $\{t = 0, 1, 2, \cdots\}$
    if $t < T$ **then**
      Take an action A_t
      Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
      if S_{t+1} is terminal **then**
        $T \leftarrow t + 1$
        Store $R - Q_t$ as δ_t
      else
        Select and store an action $A_{t+1} \sim b(\cdot \mid S_{t+1})$
        Select and store σ_{t+1}
        Store $Q(S_{t+1}, A_{t+1})$ as Q_{t+1}
        Store $R + \gamma\sigma_{t+1}Q_{t+1} + \gamma(1 - \sigma_{t+1}) \sum_a \pi(a \mid S_{t+1})Q(S_{t+1}, a) - Q_t$ as δ_t
        Sotre $\pi(A_{t+1} \mid S_{t+1})$ as π_{t+1}
        Store $\frac{\pi(A_{t+1}|S_{t+1})}{b(A_{t+1}|S_{t+1})}$ as ρ_{t+1}
      end if
    end if
    $\tau \leftarrow t - n + 1$ {τ is the time whose estimate is being updated}
    if $\tau \geq 0$ **then**
      $\rho \leftarrow 1$

$Z \leftarrow 1$

$G \leftarrow Q_\tau$

for $k = \tau, \cdots, \min(\tau + n + 1, T - 1)$ **do**

$G \leftarrow G + Z\delta_k$

$Z \leftarrow \gamma Z[(1 - \sigma_{k+1})\pi_{k+1} + \delta_{k+1}]$

$\rho \leftarrow \rho(1 - \sigma_k + \sigma_k \rho_k)$

end for

$Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\rho[G - Q(S_\tau, A_\tau)]$

If π is being learned, then ensure that $\pi(a \mid S_\tau)$ is ϵ-greedy wrt $Q(S_\tau, \cdot)$

end if

until $\tau = T - 1$

end loop

Chapter 8 Planning and Learning with Tabular Methods

本章从本质上对 *model-free* 方法和 *model-based* 方法进行总结, 找到二者的共通之处并对二者进行理论上的统一.

8.1 Model and Planning

本节主要引入一些概念, 为本章之后的展开做理论铺垫.

强化学习可以分为两种类型:

- Model-based Methods: 指的是需要对环境进行精确构建的方法, 该方法主要依赖 *planning* 部分.
- Model-free Methods: 指的是不需要对环境进行精确构建的方法, 该方法主要依赖 *learning* 部分.

上面提到的 planning 过程指的是以 model 作为输入, 计算得到或者更新 policy 的过程. planning 分为两种:

- Plan-space planning: 直接由 model 计算得出 policy 的方法;

$$\text{model} \xrightarrow{\text{planning}} \textit{policy}$$

- State-space planning: 经过中间步骤计算出 policy 的方法.

$$\text{model} \rightarrow \text{simulated experience} \xrightarrow{\text{backups}} \text{values} \rightarrow \textit{policy}$$

本书主要介绍的是 state-space planning.

虽然 Planning 和 learning 都基于回溯动作来计算价值函数, 但是 planning 使用的是 model 模拟出的经验, 而 learning 使用的是环境产生的真实经验.

一种简单的 planning 算法如 Algorithm 18所示.

Algorithm 18 Random-sample one-step tabular Q-planning

loop {forever}

Select a state, $S \in \mathcal{S}$, and an action, $A \in \mathcal{A}(s)$, at random

Send S, A to a sample model, and obtain a sample next reward, R, and a sample next state, S'

Apply one-step tabular Q-learning to S, A, R, S':

$$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$$

end loop

8.2 Dyna: Intergrating Planning, Acting, and Learning

本节介绍的 *Dyna-Q* 方法对于经验的两个用途做了简单的整合, 后续还会对其做进一步优化.

强化学习智能体通过与环境交互而获得的新信息可以用来更新模型, 进而提升 planning 过程的结果[1]. Dyna-Q 提供了一种对这两者进行整合的方法. 算法流程如 Algorithm 19所示.

Algorithm 19 Tabular Dyna-Q

Input: Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

loop {forever}

$S \leftarrow$ current(nonterminal) state

$A \leftarrow \epsilon - \text{greedy}(S,Q)$

Execute action A: observe resultant reward, R, and state, S'

[1]一般性的图例可以参考原书 P133 最下方, 具体到 Dyna-Q 的整合方法图例可以参考 P134 最下方.

$Q(S,A) \leftarrow Q(S,A)+\alpha[R+\gamma \max_a Q(S',a)-Q(S,A)]$ {Using experience to update the value function.}
$Model(S,A) \leftarrow R, S'$ (assuming deterministic environment) {Using experience to update the model.}
loop {n times}
$S \leftarrow$ random previously observed state
$A \leftarrow$ random action previous taken in S
$R, S' \leftarrow Model(S,A)$
$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
end loop
end loop

8.3 When the Model Is Wrong

本节主要介绍当环境与模型之间存在差异时, 会对最终结果产生什么影响.

由于环境在不断变化过程中, 所以会产生错误的模型. 而错误的模型会导出次优的策略, 在某些情况下次优的策略会带来探索和模型错误修正. 但是如果环境变好, 那么策略就有可能反应不出来这种好的变化. 从本质上来看, 这也是一种探索-利用窘境. 在 planning 中, 探索是指尝试会对模型提升有帮助的动作, 而利用是指基于现有的模型寻找最优动作序列. 一般解决探索-利用窘境的方法是启发式搜索. 除此之外, 还可以在奖赏函数中加入对探索的奖励[2], 用来缓解探索-利用窘境. 例如, 将奖赏函数定义为 $r + \kappa\sqrt{\tau}$, 其中 κ 是一个很小的常量, τ 是该转移没有被访问过的次数.

8.4 Prioritized Sweeping

前一节提到的 *Dyna* 算法选择模拟状态迁移起点时做的是均匀选择, 这样的策略是低效的. 本节介绍了一种提升更新状态迁移分布的方法.

[2]bonus reward

直观上看, 只更新到达目标状态的动作-状态对可以使得价值函数加速收敛, 但这种方式并不具有一般性. 更一般地, 我们考虑对有更新的动作-状态对进行反推更新, 这种方法被称为 *backward focusing*. 即便如此, 反推得到的动作-状态对规模会很大, 就需要按照更新量的大小进行优先级排列. 这种方法被称为 *prioritized sweeping*, 具体算法可以参见 Algorithm 20. 这就是算法从定性到定量的完善过程.

在随机环境中, 使用期望更新方式的计算量会很大, 可以考虑使用采样更新的方式.

需要注意的是, 本节介绍的只是一种可能的策略, 还有其他的诸如 *forward focusing* 的策略.

Algorithm 20 Prioritized sweeping for a deterministic environment

---

Input: Initialize $Q(s,a)$, $Model(s,a)$, for all s,a and $PQueue$ to empty

loop {forever}

$S \leftarrow$ current(nonterminal) state

$A \leftarrow policy(S,Q)$

Execute action A: observe resultant reward, R, and state, S'

$Model(S,A) \leftarrow R, S'$

$P \leftarrow |R + \gamma \max_a Q(s',a) - Q(S,A)|$.

if $P > \theta$ **then**

Insert S,A into $PQueue$ with priority P

end if

loop {n times, while $PQueue$ is not empty}

$S,A \leftarrow first(PQueue)$

$R,S' \leftarrow Model(S,A)$

$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$

loop {for all $\overline{S},\overline{A}$ predicted to lead to S}

$\overline{R} \leftarrow$ predicted reward for $\overline{S},\overline{A},S$

$P \leftarrow |\overline{R} + \gamma \max_a Q(S,a) - Q(\overline{S},\overline{A})|$

if $P > \theta$ **then**

Insert $\overline{S},\overline{A}$ into $PQueue$ with priority P

end if

end loop
end loop
end loop

8.5 Expected vs. Sample Updates

价值函数的更新有三个维度:

- 更新的价值函数是状态价值还是动作价值;
- 同策略还是异策略;
- 期望更新 (expected updates) 还是采样更新 (sample updates).

本节重点强调的是最后一个维度中的两种不同的更新方式.

通常来说, 期望更新对样本的容错性更强, 但是计算量更大. 但是从另外一个角度来看, 在计算能力一定的情况下, 是使用较少的迭代轮数的期望更新, 还是使用采样更新去迭代更多的轮数? 原书 P144 上部的图给出了一个理想条件下的对比. 从理论上来看, 分支数 b 越大, 错误下降地越快. 但是在实际中, 样本更新对后继状态的更新速率很快, 这也是样本更新的第二大优势.

8.6 Trajectory Sampling

本节主要比较了两种分布式更新方法:

- 穷举式状态/状态-动作扫描;
- 按照同策略分布 (on-policy distribution) 对状态/状态-动作空间进行采样[3].

其中前一种方式在大规模数据场景下应用受限; 第二种方式指的是基于模拟轨迹对遇到的状态/状态-动作对进行更新, 也被称为轨迹采样 (Trajactory

[3]It means that according to the distribution observed when following the current policy

Sampling). 当策略发生变化时, 分布也会发生变化. 相比于策略评估, 分布的计算也需要一定的计算量.

同策略分布在 *function approximation* 方法中有明显的优势, 这一点将在下一章进行详细阐述.

从短期来看, 同策略分布的收敛速率更快, 但是从长期来看这种方法是有害的, 因为它会重复采样已经出现的状态. 在实际中, 对于状态空间规模特别大的问题, 同策略分布方法还是适用的.

8.7 Real-time Dynamic Programming

本节介绍的是轨迹采样的实例 *RTDP* 算法.

在轨迹采样中需要的是最优部分策略[4](optimal partial policy), 这也就意味着对于当前所使用的策略, 存在一部分始终未被访问到的状态.

RTDP 的收敛条件为:

- 目标状态的初始价值为 0;
- 至少存在一个策略保证可以从任意起始状态以概率 1 到达目标状态;
- 非目标状态之间的状态转移奖赏是严格的负值;
- 所有的初始价值必须等于或者大于它们的最优值.

对比 RTDP 算法, 传统的 DP 算法的策略和价值函数可能会收敛于局部最优.

8.8 Planning at Decision Time

Planning 有两种使用方式:

- 提升策略和价值函数;
- 在得到状态后计算相应的动作.

[4]A policy that is optimal for the relevant states but can specify arbitrary actions.

对于后一种方式, 对于状态可以看得更加深入, 产生许多不同的状态和奖赏轨迹.

同时需要注意的是, decision-time planning 在实时性方面的表现比较差.

8.9 Heuristic Search

传统算法中的启发式算法与强化学习算法有许多共同之处. 本节主要介绍了这两者之间的异同.

传统的启发式搜索算法的价值函数是固定的, 如果将价值函数引入更新机制, 则可以得到强化学习算法. 另外一方面, 启发式算法可以视为贪心策略对步数的拓展. 搜索更多的步数有助于获取更好的动作选择. 此外, 启发式搜索的一个重要出发点是当前的状态, 所有的价值函数估计以及更新都是立足于当前状态这一出发点的.

8.10 Rollout Algorithms

Rollout 算法是对于当前状态，从每一个可能的动作开始，之后根据给定的策略进行路径采样，根据多次采样的奖励和来对当前状态的行动值进行估计。当当前估计基本收敛时，会根据行动值最大的原则选择动作进入下一个状态再重复上述过程。在蒙特卡洛控制中，采样的目的是估计一个完整的，最优价值函数，但是 rollout 中的采样目的只是为了计算当前状态的行动值以便进入下一个状态，而且这些估计的行动值并不会被保留。在 rollout 中采用的策略往往比较简单被称作 rollout 策略 (rollout policy)。[5]

Rollout 算法是将 MC control 应用于模拟轨迹上的决策时规划算法. 相比于 MC 算法, 它仅仅只针对当前状态在给定策略的情况下评估价值函数.

Rollout 算法的目标是提升默认的策略, 而不是寻找一个最优的策略. 这句话应该如何理解?

[5] https://zhuanlan.zhihu.com/p/33637351

8.11 Monte Carlo Tree Search

MCTS 算法在 rollout 算法基础上进行了拓展, 其核心在于关注从当前状态作为起始点的模拟轨迹, 根据历史模拟轨迹的奖赏, 对具有较高奖赏的轨迹进行拓展. MCTS 算法的具体步骤如下:

1. *Selection* 从根节点出发, 使用 tree policy 选择一个叶节点;
2. *Expansion* 以一定的概率对当前结点使用未探索过的动作, 进行叶节点的拓展;
3. *Simulation* 对于选定的节点, 使用 rollout 算法进行模拟, 直到终止状态;
4. *Backup* 根据模拟轨迹更新树内节点的状态.

MCTS 首先被应用于二人竞技博弈.

Part II

Approximate Solution Methods

Chapter 9 On-policy Prediction with Approximation

本书第二部分主要讲第一部分构建模型的状态空间拓展至任意大的规模, 这样就需要对算法所需要消耗的计算资源和存储资源作出限制. 此外, 在目任务中遇到的状态很有可能在训练过程中未遇到过, 因此对算法的泛化能力也有一定的要求. 本章主要使用参数化函数 $\hat{v}(\boldsymbol{s};\boldsymbol{w})$ 去估计给定策略下的价值 $v_\pi(\boldsymbol{s})$, 其中 $\boldsymbol{w}$ 所包含分量的个数 d 远小于状态个数, 即 $d \ll |\mathcal{S}|$. 因此估计出的函数具有一定的泛化能力, 相应的算法也可用于部分可观测问题.

9.1 Value-function Approximation

为了叙述简单, 定义符号 $s \mapsto u$, 其中 s 是被更新的状态, u 是被更新的状态 s 对应的价值. 有了这个定义, 就可以将前面章节的内容和本章之后的内容做一个统一.

虽然理论上所有的监督学习模型都可以用于函数估计方法, 但是强化学习增量式在线获取数据, 目标函数是动态变化的这两个特点决定了并不是所有的监督学习都适用于强化学习.

9.2 The Prediciton Objective ($\overline{VE}$)

为了应对状态空间连续的情形, 首先引入状态权重的概念. 即有状态权重 $\mu(\boldsymbol{s}) \geq 0, \sum_{\boldsymbol{s}} \mu(\boldsymbol{s}) = 1$[1]. 并且在此基础上定义价值均方误差

$$\overline{VE}(\mathbf{w}) \doteq \sum_{s \in \mathcal{S}} \mu(\boldsymbol{s})[v_\pi(\boldsymbol{s}) - \hat{v}(\boldsymbol{s}, \mathbf{w})]^2 \tag{9.1}$$

9.3 Stochastic-gradient and Semi-gradient Methods

从最小化 $\overline{VE}$ 的角度来看, 权重的更新公式为

$$\begin{aligned} \mathbf{w}_{t+1} &\doteq \mathbf{w}_t - \frac{1}{2}\alpha \nabla [v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)]^2 \\ &= \mathbf{w}_t + \alpha [v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)] \nabla \hat{v}(S_t, \mathbf{w}_t) \end{aligned} \tag{9.2}$$

其中 α 是步长参数.

SGD 算法如 Algorithm 21所示, 其中 $S_t \mapsto U_t$, $U_t \doteq G_t$ 是 $v_\pi(S_t)$ 的无偏估计.

Algorithm 21 Gradient Monte Carlo Algorithm for Estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated

Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$

Initialize value-function weights $\mathbf{w}$ as appropriate (e.g. $\mathbf{w} = \mathbf{0}$)

loop {forever}

Generate an episode $S_0, A_0, R_1, S_1, A_1, \cdots, R_T, S_T$ using π

for $t = 0, 1, \cdots, T-1$ **do**

$\mathbf{w} \leftarrow \mathbf{w} + \alpha [G_t - \hat{v}(S_t, \mathbf{w})] \nabla \hat{v}(S_t, \mathbf{w})$

end for

end loop

与此相对应的有 Semi-gradient 方法, 适用于 bootstrapping 的情形. Semi-gradient 方法具有收敛速率更快, 更适合在线学习的优势. 算法流程如 Algorithm 22所示.

[1] $\mu(\boldsymbol{s})$ 具体计算过程可以参考原书 P163

Algorithm 22 Semi-gradient TD(0) for Estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated

Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(terminal, \cdot) = 0$

Initialize value-function weights $\mathbf{w}$ as appropriate (e.g. $\mathbf{w} = \mathbf{0}$)

loop {for each episode}

Initialize S

repeat {for each step of episode}

Choose $A \sim \pi(\cdot \mid S)$

Take action A, observe R, S'

$\mathbf{w} \leftarrow \mathbf{w} + \alpha[R + \gamma\hat{v}(S', \mathbf{w}) - \hat{v}(S_t, \mathbf{w})]\nabla\hat{v}(S_t, \mathbf{w})$

$S \leftarrow S'$

until S' is terminal

end loop

9.4 Linear Methods

最具有代表性的情形是如

$$\hat{v}(s, \mathbf{w}) \doteq \mathbf{w}^T\mathbf{x}(s) \doteq \sum_{i=1}^{d} w_i x_i(s) \tag{9.3}$$

所示的线性情形, 其中 $\mathbf{x}(s)$ 被称作特征向量 *(feature vector)*.

对于 semi-gradient 方法, 在时刻 t 权重的更新公式为

$$\begin{aligned} \mathbf{w}_{t+1} &\doteq \mathbf{w}_t + \alpha(R_{t+1} + \gamma\mathbf{w}^T\mathbf{x}_{t+1} - \mathbf{w}_t^T x_t)\mathbf{x}_t \\ &= \mathbf{w}_t + \alpha(R_{t+1}\mathbf{x}_t + \mathbf{x}_t(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})^T\mathbf{w}_t) \end{aligned} \tag{9.4}$$

其中 $\mathbf{x}_t = \mathbf{x}(S_t)$. 当达到稳定状态后, 对于任意给定的 $\mathbf{w}_t$, 下一时刻的权重期望可写为

$$\mathbb{E}[\mathbf{w}_{t+1} \mid \mathbf{w}_t] = \mathbf{w}_t + \alpha(\mathbf{b} - \mathbf{A}\mathbf{w}_t) \tag{9.5}$$

其中

$$\begin{aligned} \mathbf{b} &\doteq \mathbb{E}[R_{t+1}\mathbf{x}_t] \in \mathbb{R}^d \\ \mathbf{A} &\doteq \mathbb{E}[\mathbf{x}_t(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})^T] \in \mathbb{R}^d \times \mathbb{R}^d \end{aligned} \tag{9.6}$$

此时有

$$\begin{aligned} \mathbf{b} - \mathbf{A}\mathbf{w}_{TD} &= \mathbf{0} \\ \Rightarrow \mathbf{b} &= \mathbf{A}\mathbf{w}_{TD} \\ \Rightarrow \mathbf{w}_{TD} &\doteq \mathbf{A}^{-1}\mathbf{b} \end{aligned} \tag{9.7}$$

其中 $\mathbf{w}_{TD}$ 被称为 *TD fixed point*.

9.5 Feature Construction for Linear Methods

本节主要介绍一些适合于线性方法的特征构造方式.

线性方法 (Linear Methods) 有着收敛性有保证, 计算效率高的优势. 但是线性方法很难表征出特征分量间的交互关系. 因此, 需要对特征进行精心设计.

9.5.1 Polynomials

> Suppose each state s corresponds to k numbers, $s_1, s_2, \cdots, s_k$, with each $s_i \in \mathbb{R}$. For this k-dimensional state space, each order-n polynomial-basis feature x_i can be written as
>
> $$x_i(s) = \prod_{j=1}^{k} s_j^{c_{i,j}}$$
>
> where each $c_{i,j}$ is an integer in the set $\{0, 1, \cdots, n\}$ for an integer $n \geq 0$. These features make up the order-n polynomial basis for dimension k, which contains $(n+1)^k$ different features.

但是多项式方法构造的特征容易导致维度灾难, 需要引入特征选择.

9.5.2 Fourier Basis

傅里叶基特征构造方法将原始空间的状态映射至频域进行分析, 适用范围广. 但是在表达不连续点时有困难, 并且维度增加时, 特征维度以指数趋势上升, 同样需要进行特征选择.

9.5.3 Coarse Coding

Coarse Coding 方法用重叠的圆形对连续二维状态空间进行划分, 并以圆形是否包含目标状态点做编码. 圆形的大小, 形变会影响泛化性能.

9.5.4 Tile Coding

Tile Coding 方法是 Coarse Coding 方法的拓展形式, 适用于多维连续状态空间. 只有一个 tile 的时候退化成 state aggregation 情形, 有多个 tile 的时候则是 coarse tile 情形. 除了标准矩形以外, 还可以使用异形作为基本元素对状态空间进行划分. 此外, 从 tile 角度上来看, hashing 方法本质上是对 tile 的压缩.

9.5.5 Radial Basis Functions

Coarse Coding 向连续情形进行拓展, 就得到了 RBF. RBF 的一大优势是它编码产生的估计函数是平滑且可导的.

9.6 Nonlinear Function Approximation: Artifical Neural Networks

本节主要讨论了人工神经网络是如何应用于价值函数估计的. 与深度学习一书有很大的重合性, 在此不做展开.

9.7 Least-Squares TD

本章之前的章节使用的是迭代方法计算线性估计方程. 本章提出了一种非迭代方式计算线性估计方程.

Least-Squares TD algorithm 通过对 $\mathbf{A}$ 和 $\mathbf{b}$ 进行估计

$$\hat{\mathbf{A}}_t \doteq \sum_{k=0}^{t-1} \mathbf{x}_k (\mathbf{x}_k - \gamma \mathbf{x}_{k+1})^T + \epsilon \mathbf{I} \tag{9.8}$$

$$\hat{\mathbf{b}}_t \doteq \sum_{k=0}^{t-1} R_{t+1}\mathbf{x}_k \tag{9.9}$$

直接计算出 TD fix point

$$\mathbf{w}_t \doteq \hat{\mathbf{A}}_t^{-1}\hat{\mathbf{b}}_t \tag{9.10}$$

直接计算 $\hat{\mathbf{A}}_t$ 的逆复杂度为 $O(d^3)$, 可以由一下迭代公式计算

$$\begin{aligned}\hat{\mathbf{A}}_t^{-1} &= (\hat{\mathbf{A}}_{t-1} + \mathbf{x}_t(\mathbf{x}_t + \gamma\mathbf{x}_{t+1})^T)^{-1} \\ &= \hat{\mathbf{A}}_{t-1}^{-1} - \frac{\hat{\mathbf{A}}_{t-1}^{-1}\mathbf{x}_t(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})^T\hat{\mathbf{A}}_{t-1}^{-1}}{1 + (\mathbf{x}_t - \gamma\mathbf{x}_{t+1})^T\hat{\mathbf{A}}_{t-1}^{-1}\mathbf{x}_t}\end{aligned} \tag{9.11}$$

具体算法流程如 Algorithm 23所示.

Algorithm 23 LSTD for Estimating $\hat{v} \approx v_\pi$ ($O(d^2)$ version)

Input: feature representation $\mathbf{x}(s) \in \mathbb{R}^d$, for all $s \in \mathcal{S}, \mathbf{x}(\text{terminal}) \doteq \mathbf{0}$

$\widehat{\mathbf{A}^{-1}} \leftarrow \epsilon^{-1}\mathbf{I}$

$\hat{b} \leftarrow \mathbf{0}$

loop {for each episode}

Initialize S; obtain corresponding $\boldsymbol{x}$

repeat {for each step of episode}

Choose $A \sim \pi(\cdot \mid S)$

Take action A, observe R, S'; obtain corresponding $\mathbf{x}$'

$\mathbf{v} \leftarrow \widehat{A^{-1}}^T(\mathbf{x} - \gamma\mathbf{x}')$

$\widehat{\mathbf{A}^{-1}} \leftarrow \widehat{\mathbf{A}^{-1}} - (\widehat{\mathbf{A}^{-1}}\mathbf{x})\mathbf{v}^T/(1 + \mathbf{v}^T\mathbf{x})$

$\hat{\mathbf{b}} \leftarrow \hat{\mathbf{b}} + R\mathbf{x}$

$\theta \leftarrow \widehat{\mathbf{A}^{-1}}\hat{\mathbf{b}}$

$S \leftarrow S'$; $\mathbf{x} \leftarrow \mathbf{x}'$

until S' is terminal

end loop

9.8 Memory-based Function Approximation

本节介绍的方法与之前的方法不同之处在于, 先存储样本, 再进行被动学习.

基于内存的方法相比于参数方法的优势在于, 不用受限于预先设定好的函数形式 (分布); 函数估计只受到相邻样本的影响; 智能体的经验只对其相邻状态空间的状态价值函数产生影响, 而不是对全局的状态价值函数产生影响.

基于内存的方法主要计算量在于距离计算.

9.9 Kernel-based Function Approximation

Kernel-based 方法基于不同的角度看待距离. 可以参考机器学习中的核方法.

9.10 Looking Deeper at On-Policy Learning: Interest and Emphasis

本节考虑不同状态权重不一样的情形.

首先, 定义一个随机变量 I_t 表征 *interest*, 即在时刻 t 的状态的兴趣值. 接着, 定义一个非负整型随机变量 M_t 表征 *emphasis*. 那么则有

$$\mathbf{w}_{t+n} \doteq \mathbf{w}_{t+n-1} + \alpha M_t [G_{t:t+n} - \hat{v}(S_t, \mathbf{w}_{t+n-1})] \nabla \hat{v}(S_t, \mathbf{w}_{t+n-1}) \quad 0 \leq t < T \tag{9.12}$$

其中,

$$M_t = I_t + \gamma^n M_{t-n} \quad 0 \leq t < T$$

对于 $t < 0$ 有 $M_t \doteq 0$.

Chapter 10 On-policy Control with Approximation

本章主要介绍的是同策略的 *control* 问题. 先后对前一章内容做了以下拓展:

- 从状态价值拓展至动作价值;
- 拓展至同策略 *GPI* 一般形式;
- 从 *episodic* 形式拓展至 *continuing* 形式.

10.1 Episodic Semi-gradient Control

本章将 *approximation* 从状态价值拓展至动作价值.

对于动作价值, 相应的权重更新公式为

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\Big[U_t - \hat{q}(S_t, A_t, \mathbf{w}_t)\Big]\nabla\hat{q}(S_t, A_t, \mathbf{w}_t) \tag{10.1}$$

除了权重更新公式, control 方法还需要选择合适的策略提升方法和动作选择方法. 具体的算法可以参考 Algorithm 24.

Algorithm 24 Episodic Semi-gradient Sara for Estimating $\hat{q} \approx q_*$

Input: a differentiable function $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$

Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitarily (e.g. $\mathbf{w} = \mathbf{0}$)

loop {for each episode}

$S, A \leftarrow$ initial state and action of episode (e.g. ϵ-greedy)

loop {for each step of episode}

Take action A, observe R, S'

if S' is terminal **then**

$\mathbf{w} \leftarrow \mathbf{w} + \alpha\Big[R - \hat{q}(S, A, \mathbf{w})\Big]\nabla\hat{q}(S, A, \mathbf{w})$

Go to next episode

end if

Choose A' as a function of $\hat{q}(S', \mathbf{w})$ (e.g. ϵ-greedy)

$\mathbf{w} \leftarrow \mathbf{w} + \alpha[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})]\nabla\hat{q}(S, A, \mathbf{w})$

$S \leftarrow S'$

$A \leftarrow A'$

end loop

end loop

10.2 n-step Semi-gradient Sarsa

本节主要介绍如何将 *Sarsa* 从 *tabular* 形式拓展至 *function approximation* 形式.

首先将 n-step return 拓展至 function-approximation 形式:

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{q}(S_{t+n}, A_{t+n}, \mathbf{w}_{t+n-1}) \quad n \geq 1, 0 \leq t \leq T-n \tag{10.2}$$

与前文一样, 如果 $t + n \geq T$, 有 $G_{t:t+n} \doteq G_t$. 相应的更新公式为

$$\mathbf{w}_{t+n} \doteq \mathbf{w}_{t+n-1} + \alpha[G_{t:t+n} - \hat{q}(S_t, A_t, \mathbf{w}_{t+n-1})]\nabla\hat{q}(S_t, A_t, \mathbf{w}_{t+n-1}) \quad 0 \leq t < T \tag{10.3}$$

具体算法流程如 Algorithm 25所示.

Algorithm 25 Episodic Semi-gradient n-step Sara for Estimating $\hat{q} \approx q_*$, or $\hat{q} \approx q_\pi$

Input: a differentiable function $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$, possibly π

Initialize value-function weights $\mathbf{w}$ arbitarily (e.g. $\mathbf{w} = \mathbf{0}$)

Parameters: step size $\alpha > 0$, small $\epsilon > 0$, a positive integer n

All store and access operations (S_t, A_t and R_t) can take their index mod n

loop {for each episode}

Initialize and store $S_0 \neq$ terminal

Select and store an action $A_0 \approx \pi(\cdot \mid S_0)$ or ϵ-greedy wrt $\hat{q}(S_0, \cdot, \mathbf{w})$
$T \leftarrow \infty$
for $t = 0, 1, , 2, \cdots$ **do**
  if $t < T$ **then**
    Take action A_t
    Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
    if S_{t+1} is terminal **then**
      $T \leftarrow t + 1$
    else
      Select and store $A_{t+1} \approx \pi(\cdot \mid S_{t+1})$ or ϵ-greedy wrt $\hat{q}(S_{t+1}, \cdot, \mathbf{w})$
    end if
  end if
  $\tau \leftarrow t - n + 1$ {τ is the time whose estimate is being updated}
  if $\tau \geq 0$ **then**
    $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau+1} R_i$
    if $\tau + n < T$ **then**
      $G \leftarrow G + \gamma^n \hat{q}(S_{\tau+n}, A_{\tau+n}, \mathbf{w})$
    end if
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha[G - \hat{q}(S_\tau, A_\tau, \mathbf{w})]$
  end if
end for
end loop

10.3 Average Reward: A New Problem Setting for Continuing Tasks

本节引入了 *average reward* 的概念, 引入这一概念的原因在下一节进行展开.

策略 π 的量级 (quality) 被定义为

$$\begin{aligned} r(\pi) &\doteq \lim_{h\to\infty} \frac{1}{h}\sum_{t=1}^{h} \mathbb{E}[R_t \mid A_{0:t-1} \approx \pi] \\ &= \lim_{t\to\infty} \mathbb{E}[R_t \mid A_{0:t-1} \approx \pi] \\ &= \sum_s \mu_\pi(s) \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) r \end{aligned} \tag{10.4}$$

相应的, 在 average reward 中, return 被定义为

$$G_t \doteq R_{t+1} - r(\pi) + R_{t+2} - r(\pi) + R_{t+3} - r(\pi) + \cdots \tag{10.5}$$

也被称为 *differential return.*

10.4 Deperecating the Discounted Setting

本节主要介绍了为什么要引入 *average reward.*

在连续情形下, discounted return 正比于 π 的 quality, 也就是说 discounting 是实效的. 因此需要引入 average return.

10.5 n-step Differential Semi-gradient Sarsa

本节介绍了引入 *average return* 之后的 *Sarsa* 算法.

由上节, 定义

$$G_{t:t+n} \doteq R_{t+1} - \overline{R}_{t+1} + R_{t+2} - \overline{R}_{t+2} + \cdots + R_{t+n} - \overline{R}_{t+n} + \hat{q}(S_{t+n}, A_{t+n}, \mathbf{w}_{t+n-1}) \tag{10.6}$$

其中 $\overline{R}$ 是 $\tau(\pi)$ 的估计, $n \geq 1$, 并且 $t + n < T$. 如果 $t + n \geq T$, 则规定 $G_{t:t+n} \doteq G_t$.

另外, 定义 TD 误差 (TD error)

$$\delta_t \doteq G_{t:t+n} - \hat{q}(S_t, A_t, \mathbf{w}) \tag{10.7}$$

Chapter 11 Off-policy Methods with Approximation

本章主要讨论异策略函数估计情形下的收敛性问题. 主要面临着两种挑战: 第一, 更新目标的挑战 (注意与目标策略的概念进行区分); 第二, 更新分布的挑战. 为了应对这两种挑战, 引入两种方法: 第一, 使用 *importance sampling*, 但是这一次是将更新分布映射回同策略分布, 这样 *semi-gradient* 方法就能确保收敛; 第二, 使用真实的不依赖于任何特定分布的梯度方法, 确保收敛的稳定性.

11.1 Semi-gradient Methods

本节将 *semi-gradient* 方法拓展至异策略函数估计方法. 主要基于改变更新分布的方法来应对挑战.

首先定义单步情形下的 importance sampling ratio:

$$\rho_t \doteq \rho_{t:t} = \frac{\pi(A_t \mid S_t)}{b(A_t \mid S_t)} \tag{11.1}$$

对于单步状态价值算法, 有

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\rho_t\delta_t\nabla\hat{v}(S_t, \mathbf{w}_t) \tag{11.2}$$

其中, δ_t 在 episodic and discounted 以及 continuing and average reward 情形下的定义分别为:

$$\delta_t \doteq R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t) \tag{11.3}$$

$$\delta_t \doteq R_{t+1} - \overline{R}_t + \hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t) \tag{11.4}$$

对于单步动作价值算法, 有

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\delta_t\nabla\hat{q}(S_t, A_t, \mathbf{w}_t) \tag{11.5}$$

$$\delta_t \doteq R_{t+1} + \gamma\sum_a \pi(a \mid S_{t+1})\hat{q}(S_{t+1}, a, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t) \tag{11.6}$$

$$\delta_t \doteq R_{t+1} - \overline{R}_t + \sum_a \pi(a \mid S_{t+1})\hat{q}(S_{t+1}, a, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t) \tag{11.7}$$

注意, 在这种情形下并没有使用 *importance sampling.* 因为在 *function approximation* 情形下, *importance sampling* 的作用并不明确. 因为在所有的状态动作对对于全局 *function approximation* 均生效的情况下, 我们可能希望对状态动作对赋以不同的权重. 这个问题留待后文进行解释.

在多步的情况下, 状态价值和动作价值算法都使用了 importance sampling:

$$\mathbf{w}_{t+n} \doteq \mathbf{w}_{t+n-1} + \alpha\rho_{t+1}\cdots\rho_{t+n-1}[G_{t:t+n} - \hat{q}(S_t, A_t, \mathbf{w}_{t+n-1})]\nabla\hat{q}(S_t, A_t, \mathbf{w}_{t+n-1}) \tag{11.8}$$

$$G_{t:t+n} \doteq R_{t+1} + \cdots + \gamma^{n-1}R_{t+n} + \gamma^n\hat{q}(S_{t+n}, A_{t+n}, \mathbf{w}_{t+n-1}) \tag{11.9}$$

$$G_{t:t+n} \doteq R_{t+1} - \overline{R}_t + \cdots + R_{t+n} - \overline{R}_{t+n-1} + \hat{q}(S_{t+n}, A_{t+n}, \mathbf{w}_{t+n-1}) \tag{11.10}$$

11.2 Examples of Off-policy Divergence

本节列举了异策略发散的例子.

除了本节中提到的不稳定例子以及原因分析, 还有两种防止不稳定情况的方式:

- Q-learning 对于所有的 control 方法都有收敛性保证;
- 针对函数估计使用特殊的方法.

11.3 The Deadly Triad

紧接着上一节的逻辑, 提出了三个因素同时出现会产生发散情况.
当以下三种情况同时聚集时, 会出现发散现象:

- Function approximation
- Boostrapping
- Off-policy training

但是如果只有任何两个现象同时出现, 则不会发散. 同时, 还分析了这三个因素哪些是可以避免的:

- function approximation 在问题规模比较大的情况下是不能避免的;
- boostrapping 可以避免, 但是会导致计算量上升和数据利用效率下降;
- off-policy 是否可以避免要视情况而定.

11.4 Linear Value-function Geometry

为了从更加抽象的层面分析函数估计, 并进一步理解稳定性的挑战, 本节首先提出问题优化目标.

通常情况下, 需要用估计的价值函数去逼近真实价值函数, 需要度量二者之间的差异, 首先定义范数

$$\|v\|_\mu^2 \doteq \sum_{s\in\mathcal{S}} \mu(s)v(s)^2 \tag{11.11}$$

其中, 权重 $\mu : \mathcal{S} \to \mathbb{R}$ 度量的是不同的状态被准确估计价值的重要程度. 接着, $\overline{VE}$ 被定义为

$$\overline{VE}(\mathbf{w}) = \|v_\mathbf{w} - v_\pi\|_\mu^2 \tag{11.12}$$

最后定义投影操作 Π 表示在上述范数空间中与任意价值函数最接近的价值函数

$$\Pi_v \doteq v_\mathbf{w} \quad \text{where} \quad \mathbf{w} = \arg\min_\mathbf{w} \|v - v_\mathbf{w}\|_\mu^2 \tag{11.13}$$

为了量化真实价值函数 v_π 与 function approximation 的估计值 $v_{\mathbf{w}}$ 之间的差异, 定义状态 s 下的 *Bellman error* 为

$$\begin{aligned}\overline{\delta}_{\mathbf{w}}(s) &\doteq \Big(\sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_{\mathbf{w}}(s')]\Big) - v_{\mathbf{w}}(s) \\ &= \mathbb{E}[R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1}) - v_{\mathbf{w}}(S_t) \mid S_t = s, A_t \sim \pi]\end{aligned} \tag{11.14}$$

基于 *Bellman error*, 有 *Mean Squared Bellman Error*:

$$\overline{BE}(\mathbf{w}) = \|\overline{\delta}_{\mathbf{w}}\|^2_\mu \tag{11.15}$$

相当于把 *Bellman operator*$B_\pi : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$ 应用于估计价值函数. 对于所有的 $s \in \mathcal{S}$ 以及 $v : \mathcal{S} \to \mathbb{R}$, 有如下 *Bellman operator* 定义 $\overline{BE}$:

$$(B_\pi v)(s) \doteq \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma v(s')] \tag{11.16}$$

如果将 *Bellman error* 映射回原始表达空间, 有 *Mean Square Projected Bellman Error*, 定义为 $\overline{PBE}$:

$$\overline{PBE}(\mathbf{w}) = \|\Pi\overline{\delta}_{\mathbf{w}}\|^2_\mu \tag{11.17}$$

11.5 Stochastic Gradient Descent in the Bellman Error

本节主要提出了解 *objective* 的方法以及两种不可行的 *objective* 定义方式.

首先从最简单的情形入手, 定义 *TD error*

$$\delta_t = R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t) \tag{11.18}$$

那么有 *Mean Squared TD Error*

$$\begin{aligned}\overline{TDE}(\mathbf{w}) &= \sum_{s \in \mathcal{S}} \mu(s) \mathbb{E}\left[\delta_t^2 \mid S_t = s, A_t \sim \pi\right] \\ &= \sum_{s \in \mathcal{S}} \mu(s) \mathbb{E}\left[\rho \delta_t^2 \mid S_t = s, A_t \sim b\right] \\ &= \mathbb{E}_b\left[\rho_t \delta_t^2\right]\end{aligned} \tag{11.19}$$

那么权重更新公式为:

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t - \frac{1}{2}\alpha\nabla(\rho_t\delta_t^2) \\ &= \mathbf{w}_t - \alpha\rho_t\delta_t\nabla\delta_t \\ &= \mathbf{w}_t + \alpha\rho_t\delta_t(\nabla\hat{v}(S_t,\mathbf{w}_t) - \gamma\nabla\hat{v}(S_{t+1},\mathbf{w}_t))\end{aligned} \tag{11.20}$$

使用上述定义的更新算法被称为 *naive residual-gradient* 算法. 虽然这一算法收敛性比较鲁棒, 但是并不能保证收敛到一个理想的位置.

另外一种更加完善的思想是最小化 *Bellman error*. 那么估计函数的参数更新公式为

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t - \frac{1}{2}\alpha\nabla(\mathbb{E}_\pi[\delta_t]^2) \\ &= \mathbf{w}_t - \frac{1}{2}\alpha\nabla(\mathbb{E}_b[\rho_t\delta_t]^2) \\ &= \mathbf{w}_t - \alpha\mathbb{E}_b[\rho_t\delta_t]\nabla\mathbb{E}_b[\rho_t\delta_t] \\ &= \mathbf{w}_t - \alpha\mathbb{E}_b[\rho_t(R_{t+1} + \gamma\hat{v}(S_{t+1},\mathbf{w}) - \hat{v}(S_t,\mathbf{w}))]\nabla\mathbb{E}_b[\rho_t\nabla\delta_t] \\ &= \mathbf{w}_t - \alpha\Big[\mathbb{E}_b[\rho_t(R_{t+1} + \gamma\hat{v}(S_{t+1},\mathbf{w})))] - \hat{v}(S_t,\mathbf{w})\Big]\Big[\nabla\hat{v}(S_t,\mathbf{w}) - \gamma\mathbb{E}_b[\rho_t\nabla\hat{v}(S_{t+1},\mathbf{w})]\Big]\end{aligned} \tag{11.21}$$

这种更新算法被称为 *residual gradient algorithm*. 为了让 *residual gradient algorithm* 生效, 需要保证以下两点:

- 需要保证环境是确定的;
- 在两个期望中, 从 S_t 到 S_{t+1} 的采样是相互独立的.

同时还要注意 *residual gradient algorithm* 不满足收敛性的三种表现:

- 收敛速度慢;
- 收敛到错误的值;
- 第三种表现将在下一节进行详细阐述.

11.6 The Bellman Error is Not Learnable

在本节中, 主要介绍了 Bellman error 不可学习的原因.
强化学习与机器学习中可学习性的区别在于

- RL 的可学习性重点指学习的结果, 而不关注算法效率;
- 在给定的环境内部结构中进行学习, 而不是从经验数据中进行学习.

上一节学习了两个不可学习的目标, 那么什么才是可学习的的目标? 这里引入了一个定义 *Mean Square Return Error*, 缩写为 $\overline{RE}$:

$$\begin{aligned}\overline{RE}(\mathbf{w}) &= [(G_t - \hat{v}(S_t, \mathbf{w}))^2] \\ &= \overline{VE} + \mathbb{E}[(G_t - v_\pi(S_t))^2]\end{aligned} \tag{11.22}$$

11.7 Gradient-TD Methods

本节考虑使用 SGD 方法最小化 $\overline{PBE}$.
直接对 $\overline{PBE}$ 求导, 有

$$\nabla\overline{PBE}(\mathbf{w}) = 2\mathbb{E}\left[\rho_t(\gamma\mathbf{x}_{t+1} - \mathbf{x}_t)\mathbf{x}_t^T\right]\mathbb{E}\left[\mathbf{x}_t\mathbf{x}_t^T\right]^{-1}\mathbb{E}\left[\rho_t\delta_t\mathbf{w}_t\right] \tag{11.23}$$

除了直接的求导方法, 另外一种方法是分别估计上述三个期望值, 然后将它们结合, 进而求出一个无偏的梯度估计. 如果三个期望中的两个被估计和储存, 那么第三个期望就可以在这两个期望的基础上求取得到. 首先定义

$$\mathbf{v}_{t+1} \approx \mathbb{E}\left[\mathbf{x}_t\mathbf{x}_t^T\right]^{-1}\mathbb{E}\left[\rho_t\delta_t\mathbf{w}_t\right] \tag{11.24}$$

那么有 *GTD2* 算法更新公式:

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t - \frac{1}{2}\alpha\nabla\overline{PBE}(\mathbf{w}_t) \\ &= \mathbf{w}_t - \frac{1}{2}\alpha 2\mathbb{E}\left[\rho_t(\gamma\mathbf{x}_{t+1} - \mathbf{x}_t)\mathbf{x}_t^T\right]\mathbb{E}\left[\mathbf{x}_t\mathbf{x}_t^T\right]^{-1}\mathbb{E}\left[\rho_t\delta_t\mathbf{w}_t\right] \\ &= \mathbf{w}_t + \alpha\mathbb{E}\left[\rho_t(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})\mathbf{x}_t^T\right]\mathbb{E}\left[\mathbf{x}_t\mathbf{x}_t^T\right]^{-1}\mathbb{E}\left[\rho_t\delta_t\mathbf{w}_t\right] \\ &\approx \mathbf{w}_t + \alpha\mathbb{E}\left[\rho_t(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})\mathbf{x}_t^T\right]\mathbf{v}_t \\ &\approx \mathbf{w}_t + \alpha\rho_t(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})\mathbf{x}_t^T\mathbf{v}_t\end{aligned} \tag{11.25}$$

稍加变换, 有 *TDC* 算法更新公式:

$$\begin{aligned}
\mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha\mathbb{E}\left[\rho_t(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})\mathbf{x}_t^T\right]\mathbb{E}\left[\mathbf{x}_t\mathbf{x}_t^T\right]^{-1}\mathbb{E}\left[\rho_t\delta_t\mathbf{w}_t\right] \\
&= \mathbf{w}_t + \alpha(\mathbb{E}\left[\rho_t\mathbf{x}_t\mathbf{x}_t^T\right] - \gamma\mathbb{E}\left[\rho_t\mathbf{x}_{t+1}\mathbf{x}_t^T\right])\mathbb{E}\left[\mathbf{x}_t\mathbf{x}_t^T\right]^{-1}\mathbb{E}\left[\rho_t\delta_t\mathbf{w}_t\right] \\
&= \mathbf{w}_t + \alpha(\mathbb{E}\left[\mathbf{x}_t\rho_t\delta_t\right] - \gamma\mathbb{E}\left[\rho_t\mathbf{x}_{t+1}\mathbf{x}_t\right]\mathbb{E}\left[\mathbf{x}_t\mathbf{x}_t^T\right]^{-1}\mathbb{E}\left[\rho_t\delta_t\mathbf{w}_t\right]) \qquad (11.26) \\
&\approx \mathbf{w}_t + \alpha(\mathbb{E}\left[\mathbf{x}_t\rho_t\delta_t\right] - \gamma\mathbb{E}\left[\rho_t\mathbf{x}_{t+1}\mathbf{x}_t\right]\mathbf{v}_t) \\
&\approx \mathbf{w}_t + \alpha\rho_t(\delta_t\mathbf{x}_t - \gamma\mathbf{x}_{t+1}\mathbf{x}_t\mathbf{v}_t)
\end{aligned}$$

11.8 Emphatic-TD Methods

另外一种异策略的函数估计方法是对状态而不是状态转移分配权重. 这种方法被称为 *Emphatic-TD 方法*. 权重更新方式如下:

$$\begin{aligned}
\delta_t &= R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t) \\
\mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha M_t\rho_t\delta_t\nabla\hat{v}(S_t, \mathbf{w}_t) \qquad (11.27) \\
M_t &= \gamma\rho_{t-1}M_{t-1} + I_t
\end{aligned}$$

11.9 Reducing Variance

控制方差对于基于 importance sampling 的异策略方法尤为重要. 本节重点介绍了几种降低方差的方法, 可以参考相关文献进行进一步阅读.

Chapter 12 Eligibility Traces

Eligibility Traces 是强化学习的一项重要机制, 通过引入 *eligibilty trace* $\mathbf{z}_t \in \mathbb{R}_d$, 使得强化学习算法具有了短期记忆.

12.1 The λ-return

本节通过 *compound update* 引入 *λ-return*, 统一了 *MC* 算法与 *TD* 算法.

通过对简单的更新成分求平均而得到的更新被称为 *compound update*. TD(λ) 算法可以被视为 n-step update 的特例, 相应的有 λ-return:

$$G_t^\lambda \doteq (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_{t:t+n} \tag{12.1}$$

更进一步, 我们可以将终点状态的 return 剥离出来, 得到:

$$G_t^\lambda \doteq (1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}G_{t:t+n} + \lambda^{T-t-1}G_t \tag{12.2}$$

由此, 我们得到离线 λ-return 算法的更新公式

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\Big[G_t^\lambda - \hat{v}(S_t,\mathbf{w}_t)\Big]\nabla(S_t,\mathbf{w}_t), \quad t=0,\cdots,T-1 \tag{12.3}$$

12.2 TD(λ)

TD(λ) 算法相比于离线 λ-return 算法有以下改进:

- 在每一步更新权重而不仅仅只是在 episode 结束时更新权重;
- 计算量在整个学习过程中是均匀分布的, 而不是集中在 episode 结束时集中计算;

- 除了离散问题, 也可以被用于解决连续问题.

Eligibility trace 向量的增量化计算公式为

$$\begin{aligned}\mathbf{z}_{-1} &\doteq \mathbf{0} \\ \mathbf{z}_t &\doteq \gamma\lambda\mathbf{z}_{t-1} + \nabla(S_t, \mathbf{w}_t), \quad 0 \le t \le T\end{aligned} \tag{12.4}$$

相应的算法流程如 Algorithm 26所示.

Algorithm 26 Semi-gradient TD(λ) for Estimating $\hat{v} \approx v_\pi$

Input: The policy π to be evaluated

Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal}, \cdot) = 0$

Initialize value-function weights $\mathbf{w}$ arbitarily (e.g. $\mathbf{w} = \mathbf{0}$)

Parameters: step size $\alpha > 0$, small $\epsilon > 0$, a positive integer n

All store and access operations (S_t, A_t and R_t) can take their index mod n

loop {for each episode}

Initialize and store S

$\mathbf{z} \leftarrow \mathbf{0}$

$T \leftarrow \infty$

loop {for each step of episode}

Choose $A \sim \pi(\cdot \mid S)$

Take action A, observe R, S'

$\mathbf{w} \leftarrow \gamma\lambda\mathbf{z} + \nabla\hat{v}(S, \mathbf{w})$

$\delta \leftarrow R + \gamma\hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$

$\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\mathbf{z}$

$S \leftarrow S'$

end loop

end loop

当 $\lambda = 0$ 时, $TD(\lambda)$ 等价于 one-step semi-gradient TD 更新算法; 当 $\lambda < 1$ 时, 早期的状态对 TD error 的贡献更小; 当 $\lambda = 1$ 时, 等价于 eligibility trace 算法; 当 $\lambda = 1$ 并且 $\gamma = 1$ 时, 等价于 MC 算法 (undiscounted, episodic task). 此外当 $\lambda = 1$ 时的算法统称为 TD(1) 算法.

12.3 n-step Truncated and λ-return Methods

λ-return 只有到 *episode* 结束时才知道结果. 本节针对这一问题进行修正.

首先定义 *truncated λ-return*. 在 t 时刻, 如果仅仅知道后续 h 步的数据, 那么有

$$G_{t:h}^{\lambda} \doteq (1-\lambda)\sum_{n=1}^{h-t-1}\lambda^{n-1}G_{t:t+n} + \lambda^{h-t-1}G_{t:h} \quad 0 \le t < h \le T \tag{12.5}$$

相应的价值估计函数更新公式为

$$\mathbf{w}_{t+n} \doteq \mathbf{w}_{t+n-1} + \alpha\Big[G_{t:t+n}^{\lambda} - \hat{v}(S_t, \mathbf{w}_{t+n-1})\Big]\nabla\hat{v}(S_t, \mathbf{w}_{t+n-1}) \tag{12.6}$$

12.4 Redoing Updates: The Online λ-return Algorithm

本节主要介绍在线 *λ-return* 算法.

更新的一般公式为

$$\mathbf{w}_{t+1}^{h} \doteq \mathbf{w}_{t}^{h} + \alpha\Big[G_{t:h}^{\lambda} - \hat{v}(S_t, \mathbf{w}_{t}^{h})\Big]\nabla\hat{v}(S_t, \mathbf{w}_{t}^{h}) \quad 0 \le t < h \le T \tag{12.7}$$

在线 λ-return 算法的缺点是每一步的计算复杂度特别大.

12.5 True Online TD(λ)

本节提出了一种可以实际应用的在线算法, 将 *forward-view* 算法应用到 *back-view* 中提升效率.

在上一节中提到的权重公式, 其实只用到了 $\mathbf{w}_t^t$ 项, 因此需要找到一个更加高效的计算方式:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\delta_t\mathbf{z}_t + \alpha(\mathbf{w}_t^T\mathbf{x}_t - \mathbf{w}_{t-1}^T\mathbf{x}_t)(\mathbf{z}_t - \mathbf{x}_t) \tag{12.8}$$

其中, $\mathbf{x}_t \doteq \mathbf{x}(S_t)$, δ_t 同 TD(λ) 中的定义相同, $\mathbf{z}_t$ 定义为

$$\mathbf{z}_t \doteq \gamma\lambda\mathbf{z}_{t-1} + (1 - \alpha\gamma\lambda\mathbf{z}_{t-1}^T\mathbf{x}_t)\mathbf{x}_t \tag{12.9}$$

具体算法流程如 Algorithm 27所示.

Algorithm 27 True Online TD(λ) for Estimating $\mathbf{w}^T\mathbf{x} \approx v_\pi$

Input: The policy π to be evaluated

Initialize value-function weights $\mathbf{w}$ arbitarily (e.g. $\mathbf{w} = \mathbf{0}$)

loop {for each episode}

Initialize state and obtain initial feature vector $\mathbf{x}$

$\mathbf{z} \leftarrow \mathbf{0}$ {an d-dimensional vector}

$V_{old} \leftarrow 0$ {a saclar temporary variable}

repeat {for each step of episode}

Choose $A \sim \pi$

Take action A, observe $R, \mathbf{x}'$ {feature vector of the next state}

$V \leftarrow \mathbf{w}^T\mathbf{x}$

$V' \leftarrow \mathbf{w}^T\mathbf{x}'$

$\delta \leftarrow R + \gamma V' - V$

$\mathbf{z} \leftarrow \gamma\lambda\mathbf{z} + (1 - \alpha\gamma\lambda\mathbf{z}^T\mathbf{x})\mathbf{x}$

$\mathbf{w} \leftarrow \mathbf{w} + \alpha(\delta + V - V_{old})\mathbf{z} - \alpha(V - V_{old})\mathbf{x}$

$V \leftarrow V_{old}$

$\mathbf{x} \leftarrow \mathbf{x}'$

until $\mathbf{x}' = \mathbf{0}$ {signaling arrival at terminal state}

end loop

12.6 Dutch Traces in Monte Carlo Learning

本节提出的算法将 *eligibility trace* 与 *MC* 进行结合.

权重更新公式为

$$\begin{aligned}\mathbf{w}_T &= \mathbf{w}_{T-1} + \alpha(G - \mathbf{w}_{T-1}^T\mathbf{x}_{T-1})\mathbf{x}_{T-1} \\ &= \mathbf{w}_{T-1} + \alpha\mathbf{x}_{T-1}(-\mathbf{x}_{T-1}^T\mathbf{w}_{T-1}) + \alpha G\mathbf{x}_{T-1} \\ &= (\mathbf{I} - \alpha\mathbf{x}_{T-1}\mathbf{x}_{T-1}^T)\mathbf{w}_{T-1} + \alpha G\mathbf{x}_{T-1} \\ &= \mathbf{F}_{T-1}\mathbf{w}_{T-1} + \alpha G\mathbf{x}_{T-1}\end{aligned} \tag{12.10}$$

其中, $\mathbf{F}_t \doteq I - \alpha\mathbf{x}_t\mathbf{x}_t^T$, 是 *forgetting matrix / fading matrix*.

12.7 Sarsa(λ)

本节将 *eligibility traces* 扩展至动作价值函数.

权重更新公式为

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\delta_t\mathbf{z}_t \tag{12.11}$$

由动作价值函数计算出的 TD error 为

$$\delta_t \doteq R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t) \tag{12.12}$$

动作价值函数形势下的 eligibility trace 为

$$\begin{aligned} \mathbf{z}_{-1} &\doteq \mathbf{0} \\ \mathbf{z}_t &\doteq \gamma\lambda\mathbf{z}_{t-1} + \nabla\hat{q}(S_t, A_t, \mathbf{w}_t) \quad 0 \le t \le T \end{aligned} \tag{12.13}$$

具体算法流程如 Algorithm 28所示.

Algorithm 28 Sarsa(λ) with binary features and linear function approximation for Estimating $\hat{q} \approx q_\pi$ or $\hat{q} \approx q_*$

Input: a function $\mathcal{F}(s, a)$ returning the set of (indices of) active features for s, a

Input: a policy π to be evaluated, if any

Initialize value-function weights $\mathbf{w} = (w_1, \cdots, w_n)$ arbitarily (e.g. $\mathbf{w} = \mathbf{0}$)

loop {for each episode}

Initialize S

Choose $A \sim \pi(\cdot \mid S)$ or ϵ-greedy according to $\hat{q}(S, \cdot, \mathbf{w})$

$\mathbf{z} \leftarrow \mathbf{0}$

loop {for each step of episode}

Take action A, observeR, S'

$\delta \leftarrow R$

loop {for i in $\mathcal{F}(S, A)$}

$\delta \leftarrow \delta - w_i$

$z_i \leftarrow z_i + 1$ or $z_i \leftarrow 1$

end loop

if S' is the terminal **then**
$\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\mathbf{z}$
Go to the next episode
end if
Choose $A' \sim \pi(\cdot \mid S')$ or near greedily $\sim \hat{q}(S', \cdot, \mathbf{w})$
loop {for i in $\mathcal{F}(S', A')$}
$\delta \leftarrow \delta + \gamma w_i$
end loop
$\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\mathbf{z}$
$\mathbf{z} \leftarrow \gamma\lambda\mathbf{z}$
$S \leftarrow S'$; $A \leftarrow A'$
end loop
end loop

同样的, 动作价值函数也有 online 版本和 true online 版本. true online 版本算法流程如 Algorithm 29所示.

Algorithm 29 True Online Sarsa(λ) for Estimating $\hat{q} \approx q_\pi$ or $\hat{q} \approx q_*$

Input: a feature function $\mathbf{x} : \mathcal{S}^+ \times \mathcal{A} \to \mathbb{R}^d$ s.t. $\mathbf{x}(terminal, \cdot) = \mathbf{0}$
Input: the policy π to be evaluated, if any
Initialize parameter $\mathbf{w}$ arbitarily (e.g. $\mathbf{w} = \mathbf{0}$)
loop {for each episode}
Initialize S
Choose $A \sim \pi(\cdot \mid S)$ or near greedily from S using $\mathbf{w}$
$\mathbf{x} \leftarrow \mathbf{x}(S, A)$
$\mathbf{z} \leftarrow \mathbf{0}$
$\mathbf{Q}_{old} \leftarrow 0$
repeat {for each step of episode}
Take action A, observeR, S'
Choose $A' \sim \pi(\cdot \mid S')$ or near greedily from S' using $\mathbf{w}$
$\mathbf{x}' \leftarrow \mathbf{x}(S', A')$
$Q \leftarrow \mathbf{w}^T\mathbf{x}$
$Q' \leftarrow \mathbf{w}^T\mathbf{x}'$

$\delta \leftarrow R + \gamma Q' - Q$
$\mathbf{z} \leftarrow \gamma\lambda\mathbf{z} + (1 - \alpha\gamma\lambda\mathbf{z}^T\mathbf{x})\mathbf{x}$
$Q_{old} \leftarrow Q'$
$\mathbf{x} \leftarrow \mathbf{x}'$
$A \leftarrow A'$
until S' is terminal
end loop

12.8 Variable λ and γ

本节主要讨论自适应参数 λ 和 γ 的选定策略.

对于自适应 λ 有 $\lambda_t \doteq \lambda(S_t, A_t)$. 对于 γ 有 $\gamma_t \doteq \gamma(S_t)$. 其中后一种情况被称为 *state-dependent discounting*. 相应的有 return 定义

$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma_{t+1} G_{t+1} \\ &= \sum_{k=t}^{\infty} R_{k+1} \prod_{i=t+1}^{k} \gamma_i \end{aligned} \tag{12.14}$$

为了保证上式的和是有限值, 我们要求 $\prod_{k=t}^{k} \gamma_k = 0$.

12.9 Off-policy Eligibility Traces

在本节引入 *importance sampling*.

引入 importance sampling 后的计算公式为:

$$G_t^{\lambda s} \doteq \rho_t\Big(R_{t+1} + \gamma_{t+1}((1-\lambda_{t+1})\hat{v}(S_{t+1}, \mathbf{w}_t) + \lambda_{t+1} G_{t+1}^{\lambda s})\Big) + (1 - \rho_t)\hat{v}(S_t, \mathbf{w}_t) \tag{12.15}$$

Off-policy eligibility traces 可以处理本章开始提到的第一种挑战, 但是对第二种挑战即分布式更新没有帮助.

12.10 Warkins's $Q(\lambda)$ to Tree-Backup(λ)

本节介绍将 *Q-learning* 拓展至 *eligibility traces* 的方法.

原始 *Warkins's* $Q(\lambda)$ 算法, 只要采取贪心动作便在 eligibility traces 上进行削减, 对于第一个非贪心动作, 将 traces 设为 0.

对于$TD(\lambda)$, 有

$$\begin{aligned} G_t^{\lambda a} &\doteq \gamma_{t+1}\Big((1-\lambda_{t+1})\overline{Q}_{t+1} + \lambda_{t+1}\Big(\\ &\quad \sum_{a\neq A_{t+1}} \pi(a \mid S_{t+1})\hat{q}(S_{t+1}, a, \mathbf{w}_t) + \pi(A_{t+1} \mid S_{t+1})G_{t+1}^{\lambda a}\Big)\Big) \\ &= R_{t+1} + \gamma_{t+1}\Big(\overline{Q}_{t+1} + \lambda_{t+1}\pi(A_t+1 \mid S_{t+1})\Big(G_{t+1}^{\lambda a} - \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t)\Big)\Big) \end{aligned} \tag{12.16}$$

12.11 Stable Off-policy Methods with Traces

使用 *eligibility traces* 提升 *off-policy* 情形下稳定性的 *4* 种方法.

$GTD(\lambda)$ 的更新公式为

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\delta_t^s\mathbf{z}_t - \alpha\gamma_{t+1}(1-\lambda_{t+1})(\mathbf{z}_t^T\mathbf{v}_t)\mathbf{x}_{t+1} \tag{12.17}$$

$GQ(\lambda)$ 的更新公式为

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\delta_t^a\mathbf{z}_t - \alpha\gamma_{t+1}(1-\lambda_{t+1})(\mathbf{z}_t\mathbf{v}_t)\overline{\mathbf{x}}_{t+1} \tag{12.18}$$

$HTD(\lambda)$ 的更新公式为

$$\begin{aligned} \mathbf{w}_{t+1} &\doteq \mathbf{w}_t + \alpha\delta_t^s\mathbf{z}_t + \alpha\Big((\mathbf{z}_t - \mathbf{z}_t^b)\mathbf{v}_t\Big)\Big(\mathbf{x}_t - \gamma_{t+1}\mathbf{x}_{t+1}\Big) \\ \mathbf{v}_{t+1} &\doteq \mathbf{v}_t + \beta\delta_t^s\mathbf{z}_t - \beta(\mathbf{z}_t^{bT}\mathbf{v}_t)(\mathbf{x}_t - \gamma_{t+1}\mathbf{x}_{t+1}) \quad \mathbf{v}_0 \doteq \mathbf{0} \\ \mathbf{z}_t &\doteq \rho_t(\gamma_t\lambda_t\mathbf{z}_{t-1}\mathbf{x}_t) \quad \mathbf{z}_{-1} \doteq \mathbf{0} \\ \mathbf{z}_t^b &\doteq \gamma_t\lambda_t\mathbf{z}_{t-1}^b + \mathbf{x}_t \quad \mathbf{z}_{-1}^b \doteq \mathbf{0} \end{aligned} \tag{12.19}$$

Emphatic TD(λ*)* 的更新公式为

$$\begin{aligned} \boldsymbol{\theta}_{t+1} &\doteq \boldsymbol{\theta}_t + \alpha\delta_t\mathbf{z}_t \\ \delta_t &\doteq R_{t+1} + \gamma_{t+1}\boldsymbol{\theta}_t^T\mathbf{x}_{t+1} - \boldsymbol{\theta}_t^T\mathbf{x}_t \\ \mathbf{z}_t &\doteq \rho_t(\gamma_t\lambda_t\mathbf{z}_{t-1} + M_t\mathbf{x}_t) \quad \mathbf{z}_{-1} \doteq \mathbf{0} \\ M_t &\doteq \lambda_t I_t + (1-\lambda_t)F_t \\ F_t &\doteq \rho_{t-1}\gamma_t F_{t-1} + I_t \quad F_0 \doteq i(S_0) \end{aligned} \tag{12.20}$$

12.12 Implementations Issues

在实际过程中, eligibility traces 中的 λ 和 γ 两个参数都很接近于 0, 只有最近访问过的状态对应的参数才会显著大于 0. 在实践过程中, 者正是这些不多的状态需要被更新, 以使得算法的估计值能够贴近真实值.

Chapter 13 Policy Gradient Methods

Policy Gradient 方法的核心思想是学习一个参数化的策略, 可以不借助价值函数而进行动作选择. 此外, 对价值函数和策略同时进行学习的方法被称为 *actor-critic methods.*

13.1 Policy Approximation and its Advantages

本节主要介绍 *policy approximation* 的定义形式以及要求. 另外还介绍其优势.

策略也可以用参数化的形式进行表示, 定义为 $\pi(a \mid s, \boldsymbol{\theta})$, 并且其对参数可微分, 即 $\nabla_\theta \pi(a \mid s, \boldsymbol{\theta})$ 存在且为有限值. 在实际中, 我们通常认为策略应该是一个不确定值.

直接使用策略进行强化学习的优势也很明显:

- 策略更容易被估计;
- 策略更容易与先验知识进行结合.

13.2 The Policy Gradient Theorem

本节主要介绍了策略梯度算法的理论支撑.

除了前一节介绍的应用层面的优势, 从理论层面来看, 策略梯度还具有连续性好的优势.

首先定义性能度量标准

$$J(\boldsymbol{\theta}) \doteq v_{\pi_{\boldsymbol{\theta}}}(s_0) \tag{13.1}$$

接下来就有一个问题: 在策略变化对状态分布影响未知的情况下, 如何确定策略梯度方法的性能度量标准的准确性? 这里就有 *policy gradient theorem*,

即

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla_{\boldsymbol{\theta}} \pi(a \mid s, \boldsymbol{\theta}) \tag{13.2}$$

具体的证明过程可见原书 P269.

13.3 REINFORCE: Monte Carlo Policy Gradient

本节在上一节提出的策略梯度理论上提出了一个可行的算法方案.

从采样定理的角度看式13.2, 通过采样将分布 $\mu(s)$ 隐式化, 可以得到

$$\begin{aligned}
\nabla J(\boldsymbol{\theta}) &\propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla_{\boldsymbol{\theta}} \pi(a \mid s, \boldsymbol{\theta}) \\
&= \mathbb{E}_\pi \Big[\sum_a q_\pi(S_t, a) \nabla_{\boldsymbol{\theta}} \pi(a \mid S_t, \boldsymbol{\theta}) \Big] \\
&= \mathbb{E}_\pi \Big[\sum_a \pi(a \mid S_t, \boldsymbol{\theta}) q_\pi(S_t, a) \nabla_{\boldsymbol{\theta}} \frac{\pi(a \mid S_t, \boldsymbol{\theta})}{\pi(a \mid S_t, \boldsymbol{\theta})} \Big] \\
&= \mathbb{E}_\pi \Big[q_\pi(S_t, A_t) \frac{\pi(a \mid S_t, \boldsymbol{\theta})}{\pi(a \mid S_t, \boldsymbol{\theta})} \Big] \\
&= \mathbb{E}_\pi \Big[G_t \frac{\pi(a \mid S_t, \boldsymbol{\theta})}{\pi(a \mid S_t, \boldsymbol{\theta})} \Big]
\end{aligned} \tag{13.3}$$

策略参数更新公式为

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha G_t \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t \mid S_t, \boldsymbol{\theta}_t)}{\pi(A_t \mid S_t, \boldsymbol{\theta}_t)} \tag{13.4}$$

具体算法流程见 Algorithm 30.

Algorithm 30 REINFORCE A Monte-Carlo Policy-Gradient Methods(episodic)

Input: a differentiable policy parameterization $\pi(a \mid s, \boldsymbol{\theta})$

Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$

loop {forever}

Generate an episode $S_0, A_0, R_1, \cdots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot \mid \cdot, \boldsymbol{\theta})$

for each step of the episode $t = 0, \cdots, T-1$ **do**

$G \leftarrow$ return from step t

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G \nabla_{\boldsymbol{\theta}} \ln \pi(A_t \mid S_t, \boldsymbol{\theta})$

end for

end loop

同时不难得到, 作为一种 Monte Carlo 方法, *REINFORCE* 的方差比较大, 导致学习过程缓慢.

13.4 REINFORCE with Baseline

本节介绍了一种降低 REINFORCE 方差的方法.

将 policy gradient theorem 引入 baseline $b(s)$, 有

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a \Big(q_\pi(s,a) - b(s)\Big) \nabla_{\boldsymbol{\theta}} \pi(a \mid s, \boldsymbol{\theta}) \tag{13.5}$$

其中 $b(s)$ 可以是任何函数甚至是随机变量, 只要求其满足不随着 a 变化. 上式成立的原因是

$$\sum_a b(s) \nabla_{\boldsymbol{\theta}} \pi(a \mid s, \boldsymbol{\theta}) = b(s) \nabla_{\boldsymbol{\theta}} \sum_a \pi(a \mid s, \boldsymbol{\theta}) = b(s) \nabla_{\boldsymbol{\theta}} 1 = 0 \tag{13.6}$$

相应的权重更新公式为

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha\Big(G_t - b(S_t)\Big) \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t \mid S_t, \boldsymbol{\theta}_t)}{\pi(A_t \mid S_t, \boldsymbol{\theta}_t)} \tag{13.7}$$

具体算法流程见 Algorithm 31.

Algorithm 31 REINFORCE with Baseline(episodic)

Input: a differentiable policy parameterization $\pi(a \mid s, \boldsymbol{\theta})$

Input: a differentiable state-value parameterization $\hat{v}(s, \mathbf{w})$

Parameters: step size $\alpha^{\boldsymbol{\theta}} > 0, \alpha^{\mathbf{w}} > 0$

Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$

loop {forever}

Generate an episode $S_0, A_0, R_1, \cdots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot \mid \cdot, \boldsymbol{\theta})$

for each step of the episode $t = 0, \cdots, T-1$ **do**
$G_t \leftarrow$ return from step t
$\delta \leftarrow G_t - \hat{v}(S_t, \mathbf{w})$
$\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \gamma^t \delta \nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla_{\boldsymbol{\theta}} \ln \pi(A_t \mid S_t, \boldsymbol{\theta})$
end for
end loop

13.5 Actor-Critic Methods

本节介绍在参数化策略的基础上, 通过引入参数化状态价值函数, 形成 *actor-critic* 的算法.

REINFORCE-with-baseline 中虽然也使用了状态价值函数, 但是它的价值函数仅仅被用作 baseline, 并没有被用于 boostrapping, 因此并不能称作 actor-critic 算法.

One-step actor-critic 算法将 REINFORCE 中的 full return 替换为 one-step return:

$$\begin{aligned}
\boldsymbol{\theta}_{t+1} &\doteq \boldsymbol{\theta}_t + \alpha\Big(G_{t:t+1} - \hat{v}(S_t, \mathbf{w})\Big) \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t \mid S_t, \boldsymbol{\theta}_t)}{\pi(A_t \mid S_t, \boldsymbol{\theta}_t)} \\
&= \boldsymbol{\theta}_t + \alpha\Big(R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})\Big) \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t \mid S_t, \boldsymbol{\theta}_t)}{\pi(A_t \mid S_t, \boldsymbol{\theta}_t)} \\
&= \boldsymbol{\theta}_t + \alpha \delta_t \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t \mid S_t, \boldsymbol{\theta}_t)}{\pi(A_t \mid S_t, \boldsymbol{\theta}_t)}
\end{aligned} \tag{13.8}$$

具体算法流程见 Algorithm 32.

Algorithm 32 One step Actor Critic(episodic)

Input: a differentiable policy parameterization $\pi(a \mid s, \boldsymbol{\theta})$
Input: a differentiable state-value parameterization $\hat{v}(s, \mathbf{w})$
Parameters: step size $\alpha^{\boldsymbol{\theta}} > 0, \alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$
loop {forever}
Initialize S (first state of episode)

$I \leftarrow 1$

while S is not terminal **do**

$A \sim \pi(\cdot \mid S, \boldsymbol{\theta})$

Take action A, observe S', R

$\delta \leftarrow R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$ {if S' is terminal, then $\hat{v}(S', \mathbf{w}) \doteq 0$}

$\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} I \delta \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})$

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla_{\boldsymbol{\theta}} \ln \pi(A \mid S, \boldsymbol{\theta})$

$I \leftarrow \gamma I$

$S \leftarrow S'$

end while

end loop

将上述 one-step 方法拓展至 multi-step 方法, 即引入 eligibility traces, 有 Algorithm 33.

Algorithm 33 Actor-Critic with Eligibility Traces(episodic)

Input: a differentiable policy parameterization $\pi(a \mid s, \boldsymbol{\theta})$

Input: a differentiable state-value parameterization $\hat{v}(s, \mathbf{w})$

Parameters: trace-decay rates $\lambda^{\boldsymbol{\theta}} \in [0, 1], \lambda^{\mathbf{w}} \in [0, 1]$; step size $\alpha^{\boldsymbol{\theta}} > 0, \alpha^{\mathbf{w}} > 0$

Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$

loop {forever}

Initialize S (first state of episode)

$\mathbf{z}^{\boldsymbol{\theta}} \leftarrow \mathbf{0}$

$\mathbf{z}^{\mathbf{w}} \leftarrow \mathbf{0}$

$I \leftarrow 1$

while S is not terminal **do**

$A \sim \pi(\cdot \mid S, \boldsymbol{\theta})$

Take action A, observe S', R

$\delta \leftarrow R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$ {if S' is terminal, then $\hat{v}(S', \mathbf{w}) \doteq 0$}

$\mathbf{z}^{\mathbf{w}} \leftarrow \gamma \lambda^{\mathbf{w}} \mathbf{z}^{\mathbf{w}} + I \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})$

$\mathbf{z}^{\boldsymbol{\theta}} \leftarrow \gamma \lambda^{\boldsymbol{\theta}} \mathbf{z}^{\boldsymbol{\theta}} + I \nabla_{\boldsymbol{\theta}} \ln \pi(A \mid S, \boldsymbol{\theta})$

$\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \mathbf{z}^{\mathbf{w}}$

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \delta \mathbf{z}^{\boldsymbol{\theta}}$
$I \leftarrow \gamma I$
$S \leftarrow S'$
end while
end loop

13.6 Policy Gradient for Continuing Problems

本节将 *policy gradient* 拓展至状态空间连续的情形.

首先, 需要定义如下的效果评估指标:

$$\begin{aligned} J(\boldsymbol{\theta}) \doteq r(\pi) &\doteq \lim_{h\to\infty} \frac{1}{h} \sum_{t=1}^{h} \mathbb{E}[R_t \mid A_{-:t-1} \sim \pi] \\ &= \lim_{t\to\infty} \mathbb{E}[R_t \mid A_{0:t-1} \sim \pi] \\ &= \sum_s \mu(s) \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) r \end{aligned} \tag{13.9}$$

具体算法流程见 Algorithm 34.

Algorithm 34 Actor-Critic with Eligibility Traces(continuing)

Input: a differentiable policy parameterization $\pi(a \mid s, \boldsymbol{\theta})$
Input: a differentiable state-value parameterization $\hat{v}(s, \mathbf{w})$
Parameters: trace-decay rates $\lambda^{\boldsymbol{\theta}} \in [0,1], \lambda^{\mathbf{w}} \in [0,1]$; step size $\alpha^{\boldsymbol{\theta}} > 0, \alpha^{\mathbf{w}} > 0, \eta > 0$
$\mathbf{z}^{\boldsymbol{\theta}} \leftarrow \mathbf{0}\{d'\text{-component eligibility trace vector}\}$
$\mathbf{z}^{\mathbf{w}} \leftarrow \mathbf{0}\{d'\text{-component eligibility trace vector}\}$
Initialize $\overline{R} \in \mathbb{R}$(e.g. to 0)
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$(e.g. to $\mathbf{0}$)
Initialize $S \in \mathcal{S}$(e.g. to s_0)
loop {forever}
$A \sim \pi(\cdot \mid S, \boldsymbol{\theta})$
Take action A, observe S', R
$\delta \leftarrow R - \overline{R} + \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$ {if S' is terminal, then $\hat{v}(S', \mathbf{w}) \doteq 0$}

$\overline{R} \leftarrow \overline{R} + \eta\delta$
$\mathbf{z}^{\mathbf{w}} \leftarrow \lambda^{\mathbf{w}}\mathbf{z}^{\mathbf{w}} + \nabla_{\mathbf{w}}\hat{v}(S, \mathbf{w})$
$\mathbf{z}^{\boldsymbol{\theta}} \leftarrow \lambda^{\boldsymbol{\theta}}\mathbf{z}^{\boldsymbol{\theta}} + \nabla_{\boldsymbol{\theta}} \ln \pi(A \mid S, \boldsymbol{\theta})$
$\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}}\delta\mathbf{z}^{\mathbf{w}}$
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}}\delta\mathbf{z}^{\boldsymbol{\theta}}$
$S \leftarrow S'$
end loop

13.7 Policy Parameterization for Continuous Actions

紧跟着上一节的思路, 将连续状态价值拓展至连续动作价值函数.

在这里, 对动作的分布进行建模, 而不是分别求取众多动作中的每一个动作的概率. 相应的, 参数化策略可以定义为在动作上的一个正态分布:

$$\pi(a \mid s, \boldsymbol{\theta}) \doteq \frac{1}{\sigma(s, \boldsymbol{\theta})\sqrt{2\pi}} \exp\Big(- \frac{(a - \mu(s, \boldsymbol{\theta}))^2}{2\sigma(s, \boldsymbol{\theta})^2}\Big) \tag{13.10}$$

其中, $\mu : \mathcal{S} \times \mathbb{R}^{d'} \to \mathbb{R}$, $\sigma : \mathcal{S} \times \mathbb{R}^{d'} \to R^+$ 是两个参数化估计函数. $\boldsymbol{\theta}$ 可以分为两个部分, $\boldsymbol{\theta} = [\boldsymbol{\theta}_\mu, \boldsymbol{\theta}_\sigma]^T$, 相应的估计为

$$\begin{aligned} \mu(s, \boldsymbol{\theta}) &\doteq \boldsymbol{\theta}_\mu^T \mathbf{x}(s) \\ \sigma(s, \boldsymbol{\theta}) &\doteq \exp\Big(\boldsymbol{\theta}_\sigma^T \mathbf{x}(s)\Big) \end{aligned} \tag{13.11}$$

Part III

Looking Deeper

Chapter 14 Psychology

本章从心理学的角度介绍了心理学理论与强化学习的关联. 主要目的有两个: 第一, 动物学习行为与强化学习的共通之处; 第二, 强化学习如何反作用于动物学习行为的研究.

14.1 Prediction and Control

本节引入了两个概念, *classical conditioning* 和 *instrumental conditioning*, 分别对应于强化学习中的 *prediction* 和 *control*. 但是这两个概念只是对应关系, 不是严格意义上的对等.

14.2 Classical Conditioning

将新的刺激与先天反射建立起关联的过程被称为经典条件反射 *(classical conditioning)* 或者 *Pavlovian conditioning*. 与此相对应的还有操作条件反射 *(instrumental / operant / conditioning)* 除此之外, 这里还定义了如下几个概念:

- **UR** unconditioned response
- **US** unconditioned stimuli
- **CR** conditioned response
- **CS** conditioned stimuli

14.2.1 Blocking and High-order Conditioning

Blocking occurs when an animal fails to learn a CR when a potential CS in presented along with another CS that had been used previously to condition the animal to produce that CR.

当一个 CS 在另一个 CS 之前出现, 并且阻止后一个 CS 产生 CR, 我们就称这种情况为 *blocking*.

Higher-order conditioning occurs when a previously-conditioned CS acts as a US in conditioning another initially neutral stimulus.

当一个预先定义好的 CS 作为一个 US 反射的基础, 我们称这种情况为 *higher-order conditioning*.

14.2.2 The Rescorla-Wagner Model

本节主要介绍 *Rescorla-Wanger* 模型是什么以及为什么由这个模型可以发现 *blocking*.

首先介绍了模型本身, 然后将其迁移到 TD 模型上.

14.2.3 The TD Model

本小节将 *Rescorla-Wanger* 模型拓展至实时模型.

14.2.4 TD Model Simulations

本小节介绍了 *TD* 模型又是如何促进心理学研究的.

14.3 Instrumental Conditioning

本节主要阐述 *instrumental conditioning* 的概念.

In instrumental conditioning experiments learning depends on the consequences of behavior: the delivery of a reinforcing stimulus is contingent on what the animal does.

操作条件反射实验学习到的结论视动物的行为的结果而定: 依照动物的行为释放不同的强化刺激源.

The law of effect 揭示了强化学习算法与动物学习行为之间的关联. 从强化学习的角度来看,

- **selectional** 强化学习是一种选择性的算法;
- **associative** 强化学习是一种具有关联性的算法.

与之对应的, the law of effect 则强调了 *search* 与 *memory*.

14.4 Delayed Reinforcement

The law of effect 需要经历一段时间才能确定某一行为对于最终结果的影响. 我们称这个问题为 *delayed reinforcement*. 为解决这个问题, 我们有两种方法:

- 使用 eligibility traces;
- 使用 TD 方法获得快速反馈.

14.5 Cognitive Maps

本节主要介绍心理学理论与 *model-based* 算法间的关联.

主要介绍了动物是否使用环境模型. 如果使用的话, 使用的是什么样的模型.

14.6 Habitual and Goal-directed Behavior

本节主要介绍心理学理论与 *model-free* 算法间的关联.

Chapter 15 Neuroscience

本章阐述了强化学习与神经科学之间的联系. 本章的主要目的有两个:

1. 多巴胺 *(dopamine)* 神经活动的奖赏预测误差假设;

2. 脑科学对于强化学习的促进.

15.1 Neuroscience Basics

本节主要阐述了神经科学的基本概念. 包括

- 轴突 (dendrites)
- 树突 (axon)
- 突触 (synapse)
- 递质 (neurotransmitter)
- 神经调节物质 (neuromodulator)
- 可塑性 (synaptic plasticity)

15.2 Reward Signals, Reinforcement Signals, Values, and Prediction Errors

本节将计算强化学习与神经科学中的概念建立起对应关系.

15.3 The Reward Prediction Error Hypothesis

本节主要介绍了奖赏预测误差理论.

突变活动中, 多巴胺产生的神经元传递的是误差, 这样神经科学便与 TD 算法建立起了关联. 对照这两个概念, 需要建立以下两条假设:

1. TD 误差可以是负值, 但是神经元不能有负的触发率;
2. 关于经典条件反射实验中访问到的状态是如何在学习算法中作为输入进行表示的.

15.4 Dopamine

本节主要介绍了多巴胺的概念.

15.5 Experimental Support for the Reward Prediction Error Hypothesis

本节主要介绍了奖赏预测误差理论的实验依据.

15.6 TD Error / Dopamine Correspondence

本节将上一节实验中的多巴胺神经元 *phasic* 响应与 *TD* 误差建立起对应关系.

为了建立对应关系, 需要做出如下假设:

1. 智能体已经学到了能够获取到奖赏的动作;
2. 像之前一样, 我们需要做出一个假设, 学习算法所需要的状态如何进行表示, 将直接影响到 TD 误差与多巴胺神经元活动的接近程度;

3. 智能体使用 TD(0) 算法学习到了价值函数 V, 这个价值函数初始状态下存储在查找表中, 所有的状态都被置为 0.

TD 误差和多巴胺神经元还有一些差异不能通过上述的方法进行解释.
从另一方面看, TD 误差和多巴胺同时捕捉到了大脑的奖励过程.

15.7 Neural Actor-Critic

本节主要阐述了 *actor-critic* 与脑功能的联系.
Actor-critic 算法的两个特性可以说明大脑有类似的机制:

1. 大脑的两个区域 dorsal 和 ventral 的功能类似于 actor 与 critic;

2. 大脑中的 TD 误差同时有 critic 和 actor 的功能.

15.8 Actor and Critic Learning Rules

本节主要介绍 *actor / critic* 分别如何对待 *TD* 误差.

15.9 Hedonistic Neurons

本节主要介绍享乐主义神经元理论.

15.10 Collective Reinforcement Learning

本节主要从集体的角度去研究强化学习.

15.11 Model-based Methods in the Brain

本节主要介绍 *model-based* 和 *model-free* 算法有助于探索脑部机制.

15.12 Addiction

大脑成瘾与 *TD* 算法的关联.

Chapter 16 Applications and Case Studies

本章重点是强化学习的实例, 主要展示实际问题中的折中与问题.

16.1 TD-Gammon

本节将 *TD(λ)* 算法应用与西洋双陆棋进行说明.
值得注意的是原书 P350 下半部分介绍的状态空间的编码方式.

16.2 Samuel's Checkers Player

本节主要介绍 *Samuel* 的方法与现代强化学习.

16.3 Watson's Daily-Double Wagering

IBM WASTON 实例介绍.

16.4 Optimizing Memory Control

强化学习中的内存优化问题.

16.5 Human-level Video Game Play

本节通过介绍电视游戏以引入价值函数的表现以及存储问题.

在之前介绍的 function approximation 方法依赖于手工提取特征, 但是通过人工神经网络则可以将这一过程自动化. 但是使用人工神经网络则没有引入领域知识的途径. 基于此, 就产生了 DQN 的方法.

DQN 使用 Q-learning 的原因:

- 耗时问题;
- experience replay 要求使用 off-policy 方法.

同样, 针对 Q-learning 存在的问题, 有以下改进途径:

- 使用 experience replay 方法;
- 使用 bootstrap, 基于当前的动作价值函数估计出目标奖赏进行更新;
- 对误差项进行截断, 以增加算法稳定性.

16.6 Mastering the Game of Go

本节主要介绍如何在围棋领域开发出强化学习程序, 并展示了 *AlphaGo* 和 *AlphaZero* 的例子.

传统的暴力搜索主要困难在于定义完备的位置评估函数, 因此引入了 MCTS, 但是还不足以完全解决问题.

16.6.1 AlphaGo

开发 AlphaGo 的主要动因有: 全新版本的 MCTS; 价值函数; 初始化的神经网络.

16.6.2 AlphaGo Zero

AlphaGo Zero 在 AlphaGo 的基础上不使用人工标注的数据或者人工引导. 它使用人工神经网络指导进行博弈.

16.7 Personalized Web Services

本节主要介绍了个性化网络服务引入强化学习的方法以及优势.
在个性化网络服务领域, 强化学习与传统方法的区别在于:

- 贪婪算法的目标在于最大化点击过程中间状态的概率;
- 强化学习算法的目的在于提升用户多次访问同一个网站时的点击次数.

16.8 Thermal Soaring

本节主要介绍用强化学习解决热气流流向预测问题.
首先需要论证是否是 MDP, 并定义了问题的模型.

Chapter 17 Frontiers

17.1 General Value Functions and Auxiliary Tasks

通用价值函数可以用于辅助任务学习, 并进一步提升主任务的效果.

通用价值函数 (general value function), 定义如下:

$$v_{\pi,\gamma,C(s)} = \mathbb{E}\Big[\sum_{k=t}^{\infty} C_{t+1} \prod_{i=t+1}^{k} \gamma(S_i) \mid S_t = s, A_{t:\infty} \sim \pi\Big] \tag{17.1}$$

辅助任务可以对环境进行多维度的构建, 并和主任务共享特征表示.

学习辅助任务与心理学中的条件反射存在共同之处. 结合 built-in 算法提升算法表现.

辅助任务可以打破状态是固定的这一假设.

17.2 Temporal Abstraction via Options

本节主要介绍了如何在算法中进行时间尺度的缩放.

对于 MDP, 可以考虑构造与规则在不同的时间尺度. 设计 option 的目的在于它们是可交换的动作. 为了 plan, 必须获取到 options model, 学习 options model 的方法.

17.3 Observations and State

感知到的环境仅仅是部分准确的, 为此有以下 4 个步骤:

1. 改变问题定义, 假设奖赏是观察值的一个已知函数;

2. 尝试从观察值和动作序列中恢复出状态值;

3. 妥善处理确定性计算考虑;

4. 重新引入估计方法.

17.4 Designing Reward Signals

如何实际奖赏信号是一个难点, 具体困难在于奖赏信号的稀疏性与不可预测性. 正是因为如此, 在实际项目中, 设计奖赏信号是一个关键环节.

为了解决这一问题, 有以下 3 个切入点:

1. 对价值函数设定一个假想值;

2. 如果没有假想值, 则借助于专家知识进行确定;

3. 自动化方式进行搜索.

17.5 Remaining Issues

强化学习仍然遗留有以下问题:

1. 表征能力强的参数化估计函数;

2. 泛化性能好的特征;

3. 拓展性能好的方法用于模型推导与学习;

4. 子问题的自动化选择.

17.6 Reinforcement Learning and the Future Artifical Intelligence

目前 RL 智能体还是以从仿真经验中学习为主, 但是最终还是要从真实环境中进行学习. 在真实环境中学习难免会遇到风险. 因此风险管控是一个问题.