《Artificial Intelligence——A Modern Approach》第 21 章强化学习读书报告

尹达恒
（东南大学，江苏　南京）

1　引言

1.1　何谓之“强化学习”?

1、“状态”：Agent 所处的环境的状况；

2、“动作”：Agent 对环境进行某种操作，进而使得“状态”发生改变；

3、“强化”，或称“回报”：Agent 从环境中获得什么是“好情况”和什么是“坏情况”的反馈；

4、“强化学习”：Agent 利用观察到的“回报”来学习针对某个“状态”的最优“动作”。

1.2　强化学习的应用领域

在许多复杂领域中，人很难对大量的数据给出精确一致的评价，例如在棋类运动中，人很难评价每种棋局的好坏。替代地，人可以告诉机器什么时候赢了或输了，使机器能运用类似的信息来学习评价函数，进而对从任何给定的棋局出发对获胜的概率进行精确的估计。

1.3　强化学习的 Agent 简单分类

1、 基于“效用”的 Agent：学习关于状态的效用函数，选择到达效用最大的状态的动作；

2、 基于“Q-learning”的 Agent：学习在给定状态下采取特定动作所产生的效用的函数（Q 函数），选择效用最大的动作；

3、 基于“反射”的 Agent：学习一组将状态直接映射到动作的策略。

1.4　强化学习的目标

对于系统的每个状态 $S_t(t \in \mathbb{N}_+)$，都对应一个回报值 $R(S_t) \in \mathbb{R}$；当采取某个策略 π 时，与非终止状态相关联的效用期望值可以定义为：

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right] \quad (S_0 = s)$$

强化学习的目标就是要学习这个效用函数的输出与状态每个 s 之间的对应关系

2 被动强化学习

2.1 直接效用估计

直接效用估计的核心思想是：一个状态的回报是从该状态往后的总回报期望。即贝尔曼方程：

$$U^*(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s))U^*(s')$$

在学习过程中，每次试验对于每个访问到的状态提供了一个样本每个样本都以状态为输入，以观察到的未来回报为输出，进而将强化学习问题简化为标准的归纳学习问题。

然而，直接效用估计只有在序列的最后才能计算经过的每个状态的未来回报，因为它没有考虑到不同状态效用之间的联系，而将效用视为孤立的值。实际上，从强化学习的目标可以明显看出，一个状态的效用值与其后所有状态的效用值都有关。从这个层面上讲，直接效用估计错失了很多学习的机会，直到序列的最后才能开始学习，因而收敛很慢。

2.2 自适应动态规划

自适应动态规划 Agent 通过学习连接状态的转移模型，并使用动态规划方法求解马尔可夫决策过程，以利用效用之间的约束。相当于把学到的转移模型 $P(s'|s, \pi(s)$ 以及观察到的回报 $R(s)$ 带入到贝尔曼方程中，以计算状态的效用。

由于环境的完全可观察性，学习模型本身的过程是容易的。这意味着我们面临一个有监督的学习任务，其输入是一个状态行动对，输出是结果状态，具体地，是对在状态 s 执行动作 a 后能够到达状态 s' 的转移概率 $P(s'|s, a)$ 的估计值。

对转移概率 $P(s'|s, a)$ 的估计有两种方法，分别对应于基于贝叶斯方法的强化学习和基于鲁棒控制理论的强化学习。

2.2.1 基于贝叶斯理论的强化学习

假设 u_h^π 是通过在模型 h 中执行策略 π 而获得的期望效用，$P(h|\boldsymbol{e})$ 是环境中出现 h 的概率，通常通过给定目前为止的观察用贝叶斯规则获得，那么基于贝叶斯理论的强化学习可以表示为：

$$\pi^* = \underset{\pi}{argmax} \sum_h P(h|\boldsymbol{e})u_h^\pi$$

2.2.2 基于鲁棒控制理论的强化学习

基于鲁棒控制理论的强化学习所输出的策略为在最坏情况下输出的最好策略：

$$\pi^* = \underset{\pi}{argmax}\ \underset{h}{min}\ u_h^\pi$$

2.3 时序差分学习

与求解贝尔曼方程的求解方法不同，时序差分学习使用观察到的转移来调整观察到的状态的效用，使得它们满足约束方程：

$$U^*(s) \leftarrow U^*(s) + \alpha(R(s) + \gamma(s') - U^\pi(s))$$

其中 α 是学习速度参数。

时序差分学习的基本思想是将效用估计朝着理想均衡方向调整，当效用估计正确时，理想均衡是局部成立的，其均衡就是贝尔曼方程。

自适应动态规划和时序差分实际上是紧密相关的，它们都试图对效用估计进行局部调整，以使得每一状态都与其后继状态相一致。区别在于，时序差分调整状态使其与已观察到的状态相一致，而自适应动态规划则调整状态使其使其与所有可能出现的后继状态相一致，并根据概率进行加权。此外，时序差分学习对每个观察到的转移都只进行单一的调整，而自适应动态规划为了重建效用估计和环境模型之间的一致性会进行尽可能多的调整。所以，时序差分学习可以看作是对自适应动态规划的一个粗略而有效的一阶近似。

3 主动强化学习

3.1 主动自适应动态规划学习 Agent

被动学习 Agent 有固定的策略决定其行为，而主动学习 Agent 必须决定自己应该采取什么行动。

一个显然的方案是，让 Agent 运用策略迭代，简单地执行最优策略所建议的行动，这类 Agent 称为贪婪 Agent。但这种贪婪 Agent 很容易陷入局部最优而无法找到真正的最优解，因为学校到的模型和真实环境并一定不相同。行动不仅仅根据当前学习到的模型提供回报，它们也通过影响所接收的感知信息对真实模型的学习做出贡献，通过改进模型，Agent 将在未来得到更高的回报。因此，一个 Agent 必须要在充分利用信息以最大化回报和探索以最大化长期利益之间进行折中。技术上，任何这样的方案在无穷探索的极限下都必然是贪婪的 (GLIE, greedy in the limit of infinite exploration)。一个 GLIE 方案必须对每个状态下的每个行动进行无限次数的尝试，以避免由于一系列不常见的糟糕结果而错过最优行动的有限概

率。一个 GLIE 方案最终还必须变得贪婪，以使得 Agent 的行动对于学习到的模型而言是最优的。

存在几种不同的 GLIE 方案：

1、 随机选择行动：收敛速度缓慢；

2、 给 Agent 尝试较少的行动加权，同时避免已确信具有低效用的行动：

$$U^+(s) \leftarrow R(s) + \gamma \max_a f\left(\sum_{s'} P(s'|s,a)U^+(s'), N(s,a)\right)$$

其中：

$$f(u,n) = \begin{cases} R^+ n < N_e \\ uothers \end{cases}$$

3.2 主动时序差分学习 Agent

与被动学习的情况相比，主动时序差分学习 Agent 最明显的差别是 Agent 不再具有固定的策略。和主动自适应动态规划学习 Agent 不同的是，由于主动时序差分学习 Agent 使用观察到的转移来调整观察到的状态的效用，在从被动 Agent 成为主动 Agent 时，其更新规则不需要做出任何修改。

3.3 Q-Learning

Q-Learning 是时序差分学习的一种，它学习一种行动-效用表示而不是学习效用。Q-Learning 的核心——Q 函数 $Q(s,a)$ 代表在状态 s 进行行动 a 的价值，其与效用值直接相关：

$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a')$$

可以看出，Q-Learning 不需要状态转移模型，而只需要 Q 值。Q-Learning 的更新公式为：

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

4 策略搜索

一个策略 π 就是一个将状态映射到行动的函数，它具有比状态空间中的状态少得多的参数。例如，可以用一个参数化的 Q 函数集合表示 π，每个行动用一个函数，并选取具有最高预测值的行动：

$$\pi(s) = \max_a \hat{Q}_\theta(s,a)$$

每个 Q 函数都可以是诸如神经网络那样的一个非线性函数，然后策略搜索会调整参数 θ 来改进策略，直到找到一个 θ，使 $\hat{Q}_\theta$ 接近 Q。

离散的行动使得策略价值的变化不连续，令基于梯度的搜索变得困难，因此，策略搜索方法经常使用 $\pi_\theta(s,a)$ 的一种随机搜索表示方法，指定在状态 s 中选择行动 a 的概率。一个常用的表示是 softmax 函数：

$$\pi_\theta(s,a) = e^{\hat{Q}_\theta(s,a)} / \sum_{a'} \hat{Q}_\theta(s,a)$$

5 强化学习的应用

1、 Arthur Samuel 的下棋程序：加权线性函数用于形势评估，使用时序差分更新权值；

2、 Gerry Tesauro 的 TD-GAMMON 双陆棋程序：使用具有 80 个隐单元的神经网络，使用时序差分学习；

3、 小车连杆平衡：控制小车的位置以使连杆保持平衡，已有数千篇关于强化学习及相关控制理论的论文；

4、 直升机自动控制：通常使用策略搜索以及基于一个已经学习好的转移模型的进行仿真的算法。