Reinforcement Learning: An Introduction
notebook

黎雷蕾

2017 年 12 月 7 日

目录

Chapter 8

Planning and Learning with Tabular Methods

8.1 Models and Planning

- distribution models: 所有可能性;
  - 优点: 可以生成所有可能的过程, 以及各个过程的概率分布.
  - 缺点: 难以构建, 需要对环境足够的先验知识, 并且工程量浩大, 一旦模型出现错误无法更改.
- sample models: 所有可能性中的抽样（蒙特卡洛）;
  - 优点: 可以通过 agent 与环境交互的经验构建模型, 构建模型的方式更为简单, 并且可以更改模型中的错误.
  - 缺点: 只能生成一种可能的过程, 并且在概率估计上可能出现偏差.
- State-space planning: 通过状态空间搜索一个最优的政策或一个目标的最佳路径;
- plan-space planning: 动作导致状态到状态之间的转化，并且 V 值通过状态来进行计算;

Random-sample one-step tabular Q-planning

无限重复如下步骤：

1. 随机选择一个状态 S 和一个动作 A;
2. 将 S, A 发送给一个抽样模型，获得下一个状态 S' 和下一个 Reward(R);
3. $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',A) - Q(S,A)]$;

8.2 Dyna: Integrating Planning, Acting, and Learning

真实经验 (real experience) 的作用:

- 提升模型的性能 (model-learning)；
- 提升价值函数或策略 (direct reinforcement learning)；

它们的关系如下：

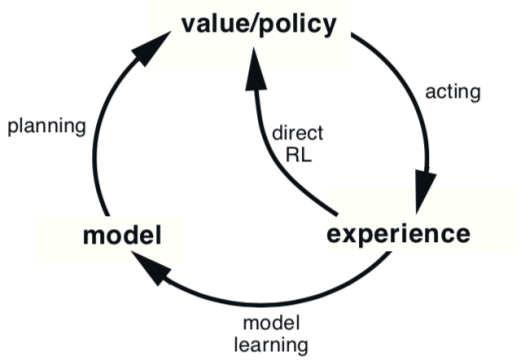


Figure 8.1: Relationships among learning, planning, and acting.

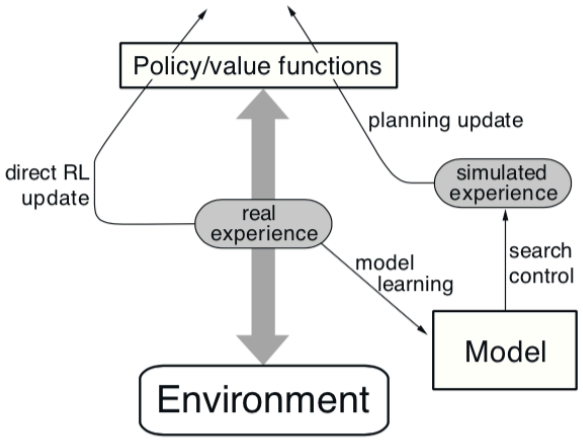


Figure 8.2: The general Dyna Architecture. Real experience, passing back and forth between the environment and the policy, affects policy and value functions in much the same way as does simulated experience generated by the model of the environment.

Tabular Dyna-Q

1. 初始化 $Q(s,a)$ 和 $Model(s,a)$;
2. 一直循环:
   (a) $S \leftarrow$ 当前非中断状态;
   (b) $A \leftarrow \epsilon - greedy(S,Q)$;
   (c) 执行动作 A，观察由此产生的 R, S';
   (d) $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',A) - Q(S,A)]$;
   (e) $Model(S,A) \leftarrow R, S'$;
   (f) 以下步骤重复 n 次:
      i. $S \leftarrow$ 先前随机观测到的状态;
      ii. $A \leftarrow$ 在 S 状态下采用的 A;
      iii. $R, S' \leftarrow Model(S,A)$;
      iv. $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',A) - Q(S,A)]$;

8.3 When the Model Is Wrong

如果模型存在误差，那么计算出来的可能就不是最优策略了。

一般来说，Dyna-Q+ 相比于 Dyna-Q 来说具有一定的启发搜索能力，对于时间 τ 内未被尝试的搜索状态转移，其回报是 $r + \kappa\sqrt{\tau}$.

8.4 Prioritized Sweeping

如果我们采用均匀采样的话，会有许多价值函数的取值是 0，那么过多的 0 会导致 backup 操作无效而被浪费。我们从目标状态向前 (backward) 搜索 (回溯) 可能会是一个有效的策略. 我们希望该方法能够用于生成奖励函数，对于目标状态，仅仅是其中的一个特例而已。

prioritized sweeping 的思想是：在随机的环境中，变量状态转移的概率也对变化量的大小产生影响，可以根据 backup 的优先级进行排序，首先进行优先级别高的进行 backup：

Prioritized sweeping for a deterministic environment

1. 初始化 $Q(s,a)$,$Model(s,a)$，一个空队列 $PQueue$；
2. 一直重复:
   (a) $S \leftarrow$ 当前的非中断状态;
   (b) $A \leftarrow policy(S,Q)$;
   (c) 执行动作 A，观察由此产生的 R, S';
   (d) $Model(S,A) \leftarrow R, S'$;
   (e) $P \leftarrow |R + \gamma \max_a Q(S',a) - Q(S,A)|$;
   (f) 如果 $P > \theta$，以优先级 P 将 S, A 插入 $PQueue$;
   (g) 如果 $PQueue$ 不为空，那么重复下面步骤 n 次：
      i. $S, A \leftarrow first(PQueue)$;
      ii. $R, S' \leftarrow Model(S,A)$;
      iii. $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',A) - Q(S,A)]$;

iv. 对于所有的 $\overline{S}, \overline{A}$ 预测值为 S 的，重复如下步骤:

A. $\overline{R} \leftarrow \overline{S}, \overline{A}, S$ 的预测回报;

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

C. 如果 $P > \theta$，以优先级 P 将 $\overline{S}, \overline{A}$ 插入 $PQueue$;

8.5 Full vs. Sample Backups

Full backup 没有抽样误差，对全局的估计将会更好，但是开销很大；在实际中用处不大，一般都是采用 sampling backups。

full backup 的动作价值函数可以写为:

$$Q(s, a) \leftarrow \sum_{s', r} \hat{p}(s', r|s, a) \left[r + \gamma \max_{a'} Q(s', a') \right] \tag{8.1}$$

其中 $\hat{p}(s', r|s, a)$ 是一个动态的估计模型，示意图如下:

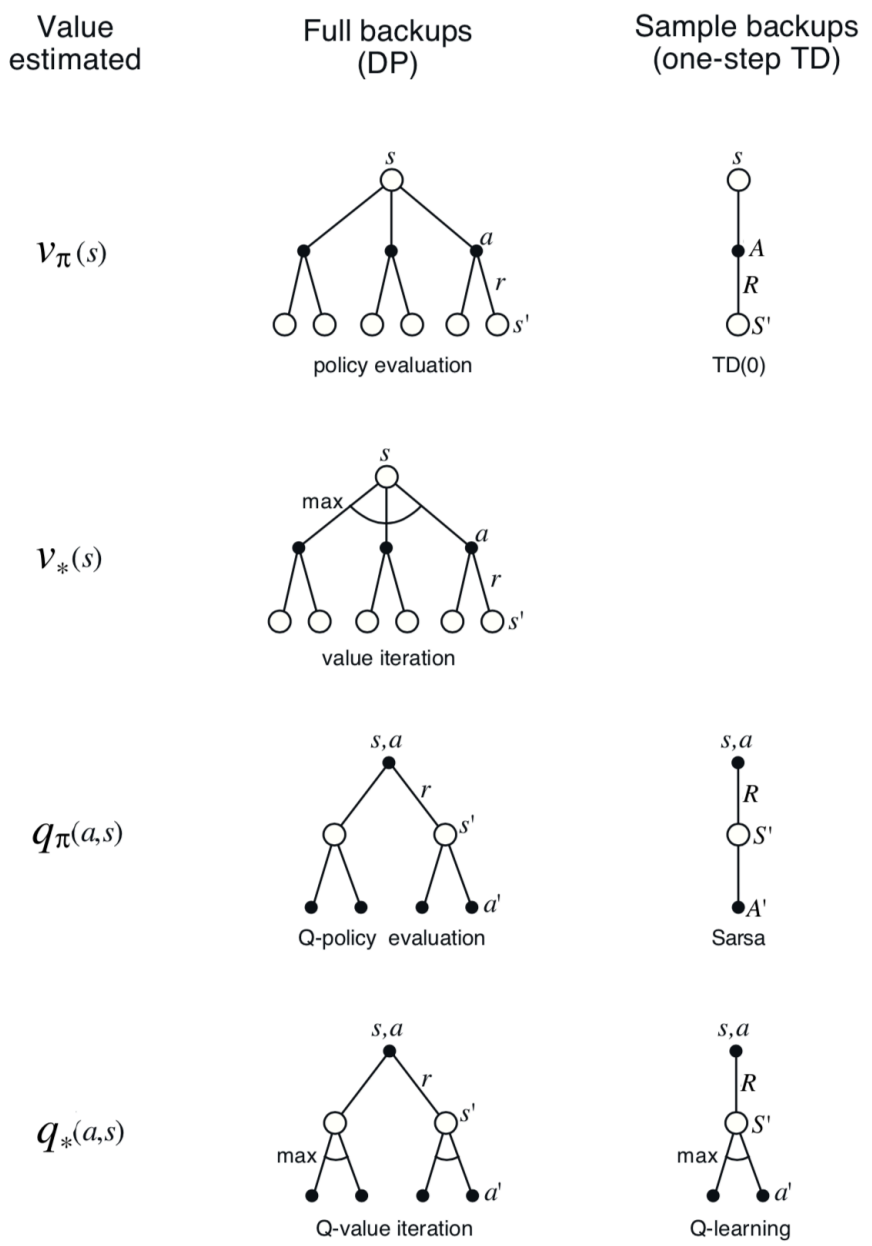


Figure 8.9: The one-step backups.

在这种条件下 Q-learning 的更新公式如下:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R + \gamma \max_{a'} Q(S', a') - Q(s,a) \right] \tag{8.2}$$

这两种情况的不同的是环境的随机性，特别是给定一个状态和动作，许多可能情况都有许多可能性。如果有足够多的时间来进行一个 full backup，由于没有抽样误差，那么它的效果肯定会优于 sampling backups。

8.6 Trajectory Sampling

- The classical approach from dynamic programming: 通过对整个状态空间进行扫描，备份每一个状态-动作对。
- The second approach is to sample from the state or state–action space according to some distribution: 相对于第一种方式，第二种采用的是抽样代替全部扫描，这样的话可以加快扫描速度，避免一些极低概率的情况。

8.7 Real-time Dynamic Programming, RTDP

RTDP 是一个异步 (asynchronous) 的 DP 算法，在 RTDP 中，备份的顺序访问真实或者模拟的轨迹的状态顺序决定的。相比于普通 DP，它可以跳过一些不需要的状态，从而加快处理的速度。

RTDP 只处理与目标状态相关的状态，而忽略其余不相关的状态。

8.8 Planning at Decision Time

- background planning: 在当前状态 S_t，选择一个动作前。planning 需要对许多状态都选择动作。在这种情况下，planning 不集中在当前的状态了。
- decision-time planning: 每次遇到新的状态 S_t 就用 planning 来完成它, 不同于前面的 planning, 这在里，它将会确定一个特别方向。

8.9 Heuristic Search

启发式搜索 (Heuristic Search) 中，每个遇到的状态，可以将其认为是一个存在着许多可能性的一棵树。近似值函数 (the approximate value function) 可以认为是叶结点，然后以此回溯到根结点的当前状态。回溯完了选择最优结点即可。

不止一步 (one-step) 的深入搜索可以获得更好的动作选择，如果是一个好的模型和一个不完善的动作-价值函数，深入搜索可以获得更加好的策略。若进化到蒙特卡洛方法，那么这个不完善的动作-价值函数的误差将会被消除。如果搜索的深度为 k 而使得 γ^k 非常小，那么得到的策略也是接近最优的。但是越深的搜索会带来更大的计算开销。

一种做法是找出 backup 时最明显的部分，启发式搜索的有效性往往时集中在它的搜索树能够很快地得到当前的状态。如下图所示：

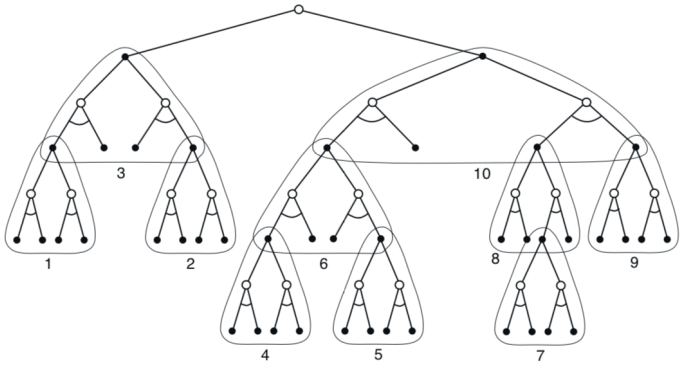


Figure 8.12: The deep backups of heuristic search can be implemented as a sequence of one-step backups (shown here outlined). The ordering shown is for a selective depth-first search.

8.10 Rollout Algorithms

Rollout 算法是一个基于蒙特卡洛的时间规划算法，适用于所有从当前环境状态开始的轨迹。是通过平均值来估计动作-价值函数。最优也是一个选择最大化价值函数的过程。

不同于蒙特卡洛方法，rollout 仅仅是针对当前的状态和给定的策略 (称

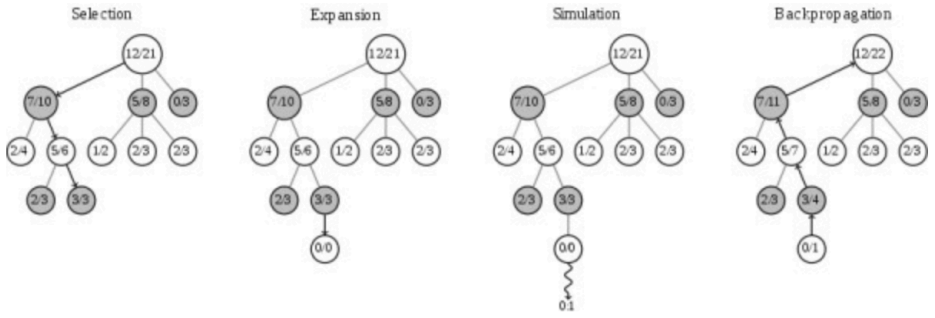
为起始策略)，往后估计蒙特卡洛值，而不是从头开始。这样可以使得计算变得比较简单，不需要对每个动作状态对产生采样，不需要在状态空间上产生近似函数。

8.11 Monte Carlo Tree Search

蒙特卡洛树搜索与蒙特卡洛方法相比：

- 蒙特卡洛方法得到的 reward 永远是后续步骤的 reward 均值，不会进步。在围棋游戏中存在偏差 (Bias) 和方差 (Variance).
- 蒙特卡洛树搜索可以向最优解进行收敛，在围棋游戏中不存在偏差，只有方差；但是对于复杂的局面，它仍有可能长期陷入陷阱，直到很久之后才能收敛得到正确答案。

以 AlphaGo 为例：



图中的数字代表：

$$\text{黑棋胜利次数/这个结点被访问次数} \tag{8.3}$$

图中的根结点 (12/21) 表示共模拟了 21 次，其中黑棋胜利了 12 次。下面介绍蒙特卡洛树搜索算法：

蒙特卡洛树搜索算法 (循环很多次)

1. 选择 (Selection): 从根结点往下走，在下层结点中按照如下公式选择一个结点：

$$S_{SelectBlack} = \arg\max \left(x_{child} + C \cdot \sqrt{\frac{\log(N_{parent})}{N_{child}}} \right) \tag{8.4}$$

其中 x_{child} 是子结点的胜率估计，N_{parent}, N_{child} 分别代表父结点和子结点的访问次数，而 C 是一个常数，C 越大就越偏向于广度搜索，C 越小就越偏向于深度搜索。由这个公式我们可以选择黑棋的最优结点。当到白棋走时，选择黑棋最不利的走法，即黑棋胜率最低的结点。

$$S_{SelectWlack} = \arg\min \left(x_{child} + C \cdot \sqrt{\frac{\log(N_{parent})}{N_{child}}} \right) \tag{8.5}$$

不断往下走，直到来到一个未拓展的结点 (叶结点)，如图上的 (3/3) 结点。

2. 扩展 (Expansion): 给 (3/3) 结点加入一个 (0/0) 的叶结点，即一种没有试过的走法。

3. 模拟 (Simluation): 通过这个新加入的结点，采用快速走子策略 (Rollout Policy) 走到底，得到一个胜负结果。

4. 回溯 (Backpropagation): 把胜负结果加到该叶结点，并加到该叶结点的所有父结点，如上图中得到的模拟结果是 (0/1), 则该结点的所有父结点都要加上 (0/1)。