Reinforcement Learning: An Introduction notebook

黎雷蕾

2017 年 11 月 23 日

目录

Chapter 5

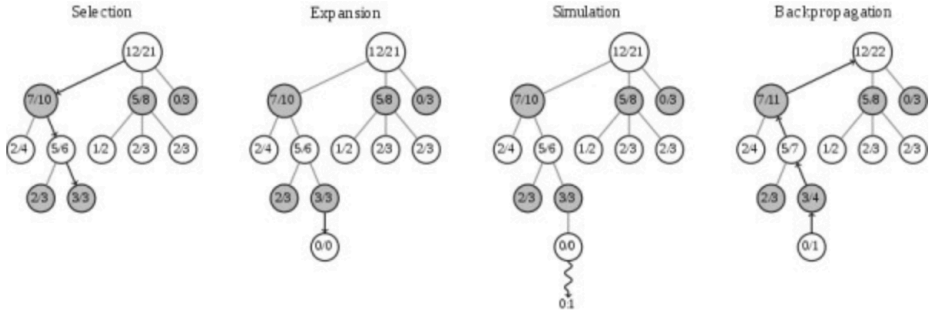
Monte Carlo Methods

5.11 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：

$$\text{黑棋胜利次数/这个结点被访问次数} \tag{5.1}$$

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：

$$S_{SelectBlack} = \arg\max \left(x_{child} + C \cdot \sqrt{\frac{\log(N_{parent})}{N_{child}}} \right) \tag{5.2}$$

   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。

$$S_{SelectWlack} = \arg\min \left(x_{child} + C \cdot \sqrt{\frac{\log(N_{parent})}{N_{child}}} \right) \tag{5.3}$$

   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)。

Chapter 6

Temporal-Difference Learning

6.1 TD Prediction

时序差分算法 (TD) 和蒙特卡洛算法 (MC) 都是根据策略 π 进行模拟，根据当前状态 S_t 和 v_π 来更新相应的 V 值.

α 步长-MC 算法可以表示为:

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

其中 G_t 表示在时间 t 时整个模拟的回报，α 是一个恒定步长参数 (constant step-size parameter)。上述公式表示 MC 算法必须在一次完整的模拟之后 (必须得到 G_t) 才能更新 $V(S_t)$。

TD 算法只需要等待下一步，在 $t+1$ 时立即用观察到的 R_{t+1} 和估计的 $V(S_{t+1})$ 进行更新:

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

上式可以被称为 TD(0)，或者 one-step TD。算法可以表示为:

TD(0) for estimating v_π

1. IN: 需要估计的策略 π;
2. 初始化 $V(s) = 0$;
3. 重复多次轨迹采样直到达到结束条件:
   (a) 初始化 S;
   (b) 通过当前的 S 和所给的策略 π 来更新动作 A;
   (c) 通过动作 A, 观察得到 R_{t+1} 和 S_{t+1}，并根据公式 6.2更新 $V(S)$;
   (d) $S \leftarrow S_{t+1}$;

TD(0) 是 bootstrapping 的算法，即:

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s \right] \tag{6.3}$$

$$= \mathbb{E}_\pi \left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s \right] \tag{6.4}$$

其中 6.3是 MC 的目标而 6.4是 TD 的目标。

注意到当前的估计状态值 $V(S_t)$ 与更好的估计 $R_{t+1} + \gamma V(S_{t+1})$ 会存在一定的差异，我们将其称为 $TD\ error$，用 δ_t 表示:

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \tag{6.5}$$

注意到 δ_t 的产生只与当前时间 t 和下一时间 $t+1$ 有关，如果 V 在某次估计中没有进行改变，那么蒙特卡洛 error，可以表示为:

$$\begin{aligned} G_t - V(S_t) &= R_{t+1} + \gamma G_{t+1} - V(S_t) + \gamma V(S_{t+1}) - \gamma V(S_{t+1}) \\ &= \sum_{k=t}^{T-1} \gamma^{k-t} \delta_k \end{aligned} \tag{6.6}$$

6.2 Advantages of TD Prediction Methods

TD 算法是通过 bootstrap 进行迭代的。与 MC 算法比较如下:

- MC 具有高方差 (variance)，零偏差 (bias)；
  - 具有很好的收敛 (convergence) 性质 (可以采用函数逼近)；
  - 对于初始化的值不敏感；
  - 非常简单易用；
- TD 具有低方差和偏差；
  - 一般来说比 MC 效率高；
  - TD(0) 收敛于 $v_\pi(s)$(但不是总能采用函数逼近)；
  - 对初值比较敏感；

6.3 Optimality of TD(0)

批量更新 (batch updating): 我们可以理解为当处理完一批数据后再进行更新，而不是每处理一个数据就进行更新了。

假设第 k 次模拟的蒙特卡洛轨迹为：

$$\langle s_1^k, a_1^k, r_2^k, \cdots, s_{T_K}^k \rangle \tag{6.7}$$

- MC 收敛于最小均方误差, 契合观察到的结果：

$$\sum_{k=1}^{K}\sum_{t=1}^{T_K}(G_t^k - V(s_t^k))^2 \tag{6.8}$$

- TD 收敛于最大似然马尔可夫模型，得到的 MDP$\langle \mathcal{S}, \mathcal{A}, \hat{\mathcal{P}}, \hat{\mathcal{R}}, \gamma \rangle$ 切合数据：

$$\begin{aligned} \hat{\mathcal{P}}_{s,s'}^a &= \frac{1}{N(s,a)}\sum_{k=1}^{K}\sum_{t=1}^{T_K}\ell(s_t^k, a_t^k, s_{t+1}^k = s, a, s') \\ \hat{\mathcal{R}}_s^a &= \frac{1}{N(s,a)}\sum_{k=1}^{K}\sum_{t=1}^{T_K}\ell(s_t^k, a_t^k = s, a) r_t^k \end{aligned} \tag{6.9}$$

通过上面的分析，TD 更适合马尔可夫环境，MC 更加适合非马尔可夫环境。

6.4 Sarsa: On-policy TD Control

TD(0) 的 Q 值可由如下动作价值公式进行更新:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \quad (6.10)$$

如果到 S_{t+1} 时发生中断，那么 $Q(S_{t+1}, A_{t+1}) = 0$;

sarsa 算法采用五元组 $\langle S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1} \rangle$ 进行迭代, 对应下图:

图 6.1: sarsa 算法示意图

对应的算法可以表示为:

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

1. 初始化 $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}$ 随机或者为 0，$Q(terminal-state, \cdot) = 0$;
2. 重复多次轨迹采样直到 S 到达中断状态:
   (a) 初始化 S;
   (b) 采用 ϵ-贪心算法从 Q 中获取 A, S;
   (c) 对于每条轨迹的每一步 (each step in one episode), 进行如下循环:
      i. 通过选取的 A，观察 R S‘;
      ii. 根据 S' 从 Q 中通过 ϵ-贪心算法选取 A';
      iii. 更新 $Q(S,A)$:

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

      iv. 更新 $S \leftarrow S'$;
      v. 更新 $A \leftarrow A'$;

6.5 Q-learning: Off-policy TD Control

$Q-learning$ 通过下面的公式来更新 $Q(S_t, A_t)$:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t)] \quad (6.11)$$

对应的算法可以表示为:

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

1. 初始化 $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}$ 随机或者为 0，$Q(terminal-state, \cdot) = 0$;
2. 重复多次轨迹采样直到 S 到达中断状态：
   (a) 初始化 S;
   (b) 采用 ϵ-贪心算法从 Q 中获取 A, S;
   (c) 对于每条轨迹的每一步 (each step in one episode), 进行如下循环:
      i. 通过选取的 A，观察 R S‘;
      ii. 根据 S' 从 Q 中通过 ϵ-贪心算法选取 A';
      iii. 更新 $Q(S,A)$:

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

      iv. 更新 $S \leftarrow S'$;

6.6 Expected Sarsa

对于类似 $Q-learning$ 之类的算法，如果不是采用最大值，而是采用期望值，即照着下面的公式进行更新：

$$\begin{aligned} Q(S_t,A_t) &\leftarrow Q(S_t,A_t) + \alpha\left[R_{t+1} + \gamma \mathbb{E}[Q(S_{t+1},A_{t+1})|S_{t+1}] - Q(S_t,A_t)\right] \\ &\leftarrow Q(S_t,A_t) + \alpha\left[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1},a) - Q(S_t,A_t)\right] \end{aligned} \tag{6.12}$$

这个算法被称为 Expected Sarsa, 相比于 Sarsa，Expected Sarsa 计算将会更加复杂，但是由于 A_{t+1} 采用的是随机选择而不是最大化，能够有效地降低方差。

6.7 Maximization Bias and Double Learning

前面六节所提到的所有算法在优化的过程中都需要最大化目标策略，但是这样做很容易导致一个最大化偏差 (Maximization Bias)。

想要避免最大化偏差，若我们一直使用估计的最大值作为真实值的估计 $(Q-learning)$，那么会产生一个正的偏差，造成这个的原因是由于在确定最大化 Q 值的动作和估计状态值时采用的是相同的样本。

为了解决这个问题，我们把样本分成两个子集 (set)，分别独立地学习两个估计，分别称为 $Q_1(a)$ 和 $Q_2(a)$, 我们可以用 $Q_1(a)$ 来决定最大化动作：$A^* = \arg\max_a Q_1(a)$，采用 $Q_2(a)$ 对其价值函数进行估计：$Q_2(A^*) = Q_2(\arg\max_a Q_1(a))$. 若 $\mathbb{E}[Q_2(A^*)] = q(A^*)$。那么将会是无偏估计。我们还可以产生第二个无偏估计 $Q_1(\arg\max_a Q_2(a))$,这个算法被称为 $doubled\,learning$。这个算法需要两倍的存储空间，但是每一步的计算开销没有增加。下面给出的是 $Q-learning$ 的 double 版：

Double Q-learning

1. 随机地初始化 $Q_1(s,a), Q_2(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$;
2. 初始化中断状态: $Q_1(terminal-state,\cdot) = Q_2(terminal-state,\cdot) = 0$;
3. 对于每次估计 (episode):
   (a) 初始化 S;
   (b) 对于当前估计的每一步:
      i. 在 Q_1+Q_2 中采用 ϵ-贪心算法从 S 中获取 A;
      ii. 通过选择的 A，观察 R, S':
      iii. 以 0.5 的概率随机选择下面两个公式其中的一个:

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

      iv. 更新 $S \leftarrow S'$

6.8 Games, Afterstates, and Other Special Cases

在游戏模拟中，可能从不同状态选取不同的动作，最后得到的状态是相同的，这被称为 Afterstates。Afterstates 在我们知道部分初始环境的变化的情况后比较有用，我们不需要知道全部的动态，在 Afterstates 中，Q 值按照如下方式更新:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1},A_{t+1}) - Q(S_t,A_t)] \quad (6.13)$$