第三课 - 利用动态规划来寻找最优策略
Planning by Dynamic Programming

David Silver[1], Zhenyue Qin[2*]

[1] Google Deep Mind, London

[2] 不正常 AI 研究中心, 深度人工智障

[*] 请联系这个电子邮箱地址: zhenyue.qin@anu.edu.au.

2018, Feb 3

本节主要如何利用动态规划来做强化学习中的规划. 也就是说, 在已知模型的基础上判断一个策略的价值函数. 除此之外, 我们还希望能够一步一步地, 也即动态地, 找到最优的策略和最优的价值函数. 我们将介绍三种基本的动态规划的范式. 这些范式包括策略评估, 也就是定量地去评价一个策略的好坏; 还包括策略迭代, 也即一步一步地找到更好的策略, 直至找到最优的策略; 类似地, 我们还有价值迭代, 也就是一步一步地找到更好的价值函数, 直到每一个状态的价值达到最优. 我们最终会给出一些为什么这些方法能够成功的直观解释.

1 导论 Introduction

1.1 何为动态规划 What is Dynamic Programming?

Dynamic Programming (动态规划) 中的 programming 不是编程的意思, 而是规划的意思. 换句话说, 是怎么样做一件事情做得更好 (优化).

动态 (dynamic) 的意思是一个问题中的顺序性的或者随时间的部分. 也就是说, 这个东西的后面像链一样依次取决于前面. 对于动态问题, 我们需要一步一步地解决这个问题. 后一步基于前一步的结果. 所以对于这一个系列, 我们希望把每一步都走得最好. 这样, 最终的结果也会最好.

规划 (programming) 就像我们之前说的, 是一个数学过程. 是说怎么样优化事情.

动态规划是一种解决复杂问题的方法. 我们把复杂问题分解为子问题. 然后我们通过一步一步地解决这一个一个" 小目标", 最后解决我们的大问题. 也就是说, 我们可以重复利用我们之前已经解决的小问题, 而不需要每一次重复地去解决已经解决的问题.

1.2 动态规划的使用条件 Requirements for Dynamic Programming

动态规划有两个属性:

- 第一, 一个复杂的问题的最优解通常由若干个小问题的最优解组成 (1).
- 第二, 小问题通常会反复出现. 也就是说, 解决了一个小问题, 储存下来这个小问题的答案. 在将来, 在这个小问题再度出现时, 我们可以直接给出答案, 而不用再次进行计算.

马尔科夫决定过程 (MDP) 满足上述两个属性. 我们在上一节提到了贝尔曼最优方程. 贝尔曼方程将 MDP 分解为递归的小问题. 具体来说, 贝尔曼方程将最优价值函数分解成为两部分, 即当前的及时奖励和下一步的奖励. 这是一个一步一步的过程, 我们把每一步做到最好, 就能把整体做到最好. 状态价值函数把它之后的未来的价值总结并储存

了起来. 使得它之前的状态可以直接使用它的价值, 而不需要重复计算它之后的未来的价值.

同时, 价值函数将在当前时刻的价值存储起来, 以便在将来重复使用.

1.3 使用动态规划规划 Planning by Dynamic Programming

如果我们使用动态规划来解决 MDP 的话, 我们是开了上帝视角的. 也就是说, 我们对于环境非常清楚. 我们的问题是怎么样在这个环境里面表现的最好. 也就是说, 我们想要解决**规划**问题. 规划问题包括**预测**和**控制**.

- 对于预测, 我们是给定一个策略, 找这个策略的价值函数.

  输入: MDR $< S, A, P, R, \gamma >$ 和一个策略 π

  或者: MRP $< S, P^{\pi}, R^{\pi}, \gamma >$

  我们想要得到价值函数 v_{π}

- 对于控制, 我们是找到最好的策略.

  输入: MDR $< S, A, P, R, \gamma >$ 和一个策略 π

  输出: 最优价值函数 v_{*}

  和: 最优策略 π_{*}

控制更重要, 因为我们的目的就是找到做事情的最好的方式.

2 策略评估 Policy Evaluation

2.1 迭代型策略评估 Iterative Policy Evaluation

我们的问题是评估一个给定的策略 π 怎么样. 我们利用贝尔曼期望方程来做这件事情. 具体来说, 我们不断反向迭代贝尔曼方程来做优化. 贝尔曼期望对于不同的策略会

给出不同的结果. 所以我们要不断地更新策略, 从而使贝尔曼期望不断增长, 最终增长到最优情形.

我们使用**同步反向迭代**. 也就是说, 我们先计算后面所要用到的价值. 我们可以这样做是因为最后一步是在最终状态, 而我们在最终状态可以直接得到最优价值. 然后一步一步地找到当前步的最优价值. 具体来说,

- 在第 $k+1$ 次迭代,
- 对于所有的状态 $s \in S$,
- 用 $v_k(s)$ 来更新 $v_k(s)$, s' 是 s 的后继状态.

利用这种方法, 最终我们所有状态的价值达到收敛. 形式化地,

$$v_{k+1}(s) = \sum_{a \in A} \pi(a|s)(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s'))$$

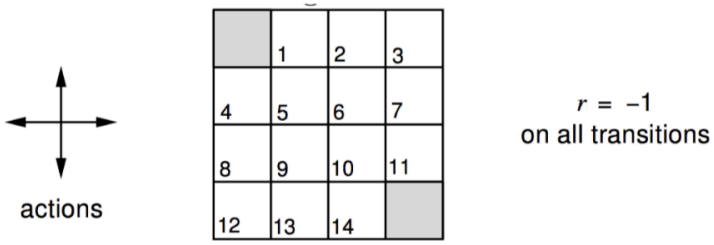
这个公式是说, 状态 s 的价值等同于从该状态所能做的所有的行为的奖励的期望, 加上从该状态所能达到的接下来的状态的价值的期望. 也就是说, 我们需要利用后继的状态来计算当前的状态, 这就是反向迭代.

写成向量的形式. (向量形式一次就能更新所有的状态): $v_{k+1} = R^\pi + \gamma P^\pi v^k$.

2.2 例子 - 格子世界 Grid World

如下图所示, 我们有一个 4×4 的大方块. 这个方块由 16 个小格子组成. 其中, 左上角和右下角所对应的格子为终止格子. 其余的 14 个格子, 每一个格子对应值为-1 的奖励. 每一个非终止的格子的策略, 选择走上下左右的概率是相同的 (都是 0.25). 如果走到边界, 那么不能越出边界. 越出边界的那一步相当于停留.

我们有两种走法. 一种是完全随机的走法, 另外一种是下一步朝着价值最高的格子移动. 对于我们这个例子, 后者比前者的收敛速度要快很多. 后者又叫策略迭代.



- Undiscounted episodic MDP ($\gamma = 1$)
- Nonterminal states $1, ..., 14$
- One terminal state (shown twice as shaded squares)
- Actions leading out of the grid leave state unchanged
- Reward is -1 until the terminal state is reached
- Agent follows uniform random policy

$$\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25$$

图 1: Grid World Example (2)

作者瞎写: 马尔科夫过程很神奇啊. 一旦确定了起始状态, 每一步就变成确定性的过程了. 也就是说, 最终的分布在一开始就确定了.

3 策略迭代 Policy Iteration

3.1 怎样改进策略 How to Improve a Policy

具体来说, 给定一个策略 π,

- 我们去计算这个策略 π, 使用 $v_\pi(s) = E[R_{t+1} + \gamma R_{t+2} + ...|S_t = s]$
- 贪心地改进 π', 通过 v_π. 具体来说, $\pi' = greedy(v_\pi)$

很有意思的是, 这种贪心策略总是收敛的.

通俗地说, 在当前策略上迭代计算 v 值，再根据 v 值贪婪地更新策略，如此反复多次，最终得到最优策略 π^* 和最优状态价值函数 V^* (1).

窃以为一旦初始化之后, 每一个步骤都是确定的.

如下图所示, 我们起始的状态价值 V 并不重要. 我们根据当前的 V 贪心地更新策略 π, 如此循环往复. 每一次计算当前的状态价值 v 叫做策略评估 (policy evaluation). 每一次更新策略 π 叫做策略改进 (policy improvement).

见招拆招, 直到达到我们的目的.

3.2 **策略改进** Policy Improvement

我们现在考虑一个确定性的策略, 形式化地, $a = \pi(s)$. 换句话说, 现在我们的策略不在是一个分布, 而是一个确定的行为. 所以现在, 一旦我们确定了初始状态. 每一步都将是确定的, 而不仅仅是期望是确定的.

我们采用贪心策略来选择我们的行为. 形式化地, $\pi'(s) = argmax_{a \in A} q_\pi(s, a)$. 我们一开始随机给一些策略, 通过这些随机给出的策略计算出所有状态的价值. 然后我们贪心地更新策略 (根据第一次的随机结果, 更新下策略, 使得每个状态价值更好一些). 总之, 状态价值只会增加, 不会减少. 这就是序列呀. 根据过去, 改进未来.

增加到最后, 如果 q 值不再增加. 那么此时, 我们将满足贝尔曼最优方程. 也就是说, 我们找到了使得状态的价值最大的行为. 也就是说, 我们找到了最好的行为.

思考：很多时候，策略的更新较早就收敛至最优策略，而状态价值的收敛要慢很多，是否有必要一定要迭代计算直到状态价值得到收敛呢？(1)

3.3 **修饰过的策略迭代** Modified Policy Iteration

有时候不需要持续迭代至最有价值函数, 可以设置一些条件提前终止迭代. 比如设定一个 ϵ; 比较两次迭代的价值函数平方差; 直接设置迭代次数; 以及每迭代一次更新一次策略等 (1).

4 **价值迭代** Value Iteration

4.1 **优化原理** Principle of Optimality

一个最优策略可以被分解为两个部分: * 首先采取了最优状态 A_*. * 在后继状态 s' 的时候也采取了最优策略.

定理 (优化原理): 我们说一个策略 $\pi(a|s)$ **从**状态 s 达到最优状态, $v_\pi(s) = v(s)$, 当且仅当 * 对于任意的可以从 s 达到的状态 s' * π 在状态 s' 也能够达到最优价值, $v_\pi(s') = v(s')$.

也就是说, 在一个状态表现的很好就在后续的状态都表现的很好. 在一开始是你大爷就永远是你大爷.

References and Notes

1. 叶强, 《强化学习》第三讲动态规划寻找最优策略 (2017).

2. D. Silver, Lecture 3: Planning by dynamic programming (2015).