Reinforcement Learning and Control

强化学习和控制

在监督学习中，我们见过的算法在给定训练集的情况下，尝试输出标签y。这种情况下，对于每个输入x，都有一个明确的标签Y与之对应。

但是在一些连续决策和控制问题中，很难给算法提供清晰准确的监督。

比如让四条腿的机器人走路，我们并不知道应该采取什么样的动作。又比如设计下象棋的AI，每走一步都是一个决策的过程，我们并没有特定的标签提供给算法。

比如让四条腿的机器人走路，我们并不知道应该采取什么样的动作。又比如设计下象棋的AI，每走一步都是一个决策的过程，我们并没有特定的标签提供给算法。

强化学习中，我们提供**奖励函数(reward function)**。这个函数会在机器人正确行走时做出奖励，在走错或者跌倒时给出惩罚。随时间推移，学习算法就会找到一系列动作使得奖励函数取得最大值。

马尔可夫决策(Markov decision processes, MDP)

对强化学习的探索，要先从马尔可夫决策开始，这个概念给出了强化学习问题的常见形式。

MDP是一个五元组 $(S, A, \{P_{sa}\}, \gamma, R)$

1. 状态集S（在直升机自动飞行例子中，S可能是直升机的所有位置和方向的集合）
2. 动作集A（还以直升机为例，A可以是遥控器上面能够操作的所有方向的集合）
3. 状态转移概率 P_{sa} 在状态 s 下进行一个动作 a 而要转移到的状态的分布。
4. 折现因子(discount factor) $\gamma \in [0, 1)$
5. 奖励函数 $R : S \times A \mapsto \mathbb{R}$（有时候只写成状态S的函数，即 $R : S \mapsto \mathbb{R}$）

MDP过程

以某个起始状态 s_0 开始

选择动作 $a_0 \in A$ 使状态随机的转移到某个后继状态 s_1，并且 $s_1 \sim P_{s_0 a_0}$

此时状态为 s_1，然后再选择另一动作 a_1，再转移到某个后继状态 s_2

一直这样下去.....

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \cdots$$

通过序列中的所有状态和对应的动作，就能得到总收益(total payoff)

$$R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \cdots .$$

如果之写成状态的函数，为 $R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots .$

强化学习的目标是找到一组动作，使得总收益的均值最大 $\mathrm{E}\left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots\right]$

注意：在时间步长t上的奖励函数随着参数 γ^t 而进行了缩减，所以为了使期望最大，我们应尽可能早积累正面奖励，而推迟负面奖励的出现。

定义一种策略(policy)是一个函数 π：$S \mapsto A$，从状态到特征的映射。

如果在状态s，我们执行动作 $a = \pi(s)$，就可以说在执行某种策略 π

我们为 π 定义值函数（value function）

$$V^{\pi}(s) = \mathrm{E}\left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots \,\middle|\, s_0 = s, \pi\right]$$

$V^{\pi}(s)$ 是从状态s开始，执行动作 π 来积累的discounted reward的期望之和。

这个值函数由两部分组成：第一部分是立即获得的奖励函数值（immediate reward）$R(s_0)$；第二部分是未来的discounted reward函数的期望之和，其实也就是 $V^{\pi}(S_1)$，可以写成 $\mathrm{E}_{s' \sim P_{s\pi(s)}}[V^{\pi}(s')]$

注：discounted reward 不知道啥意思。。

给定一个固定的策略函数 π，它的值函数满足贝尔曼等式(Bellman equations)

$$V^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^{\pi}(s')$$

我们定义最优化值函数(optimal value function)：

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s')$$

最优化值函数的贝尔曼等式

第一项R（s）是定值

第二项就是在状态s执行了动作a之后，可能到达的所有状态的 $V^*(s)$ 的未来期望之和的最大值

再定义一个策略 $\pi^* : S \mapsto A$

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s')$$

注：这里的a是能使上个式子max取得最大值的a

则对于每个状态s和每个策略 π：都有 $V^*(s) = V^{\pi^*}(s) \geq V^{\pi}(s)$

第一个等式说明，对每个状态s来说，π^* 的值函数 V^{π^*}，与 V^* 的最优化值函数是相等的

第二个不等式说明，π^* 是能使 $V^{\pi}(s)$ 取得最大值的最优策略

注：这里的 π^* 是全局最优策略，也就是不管是从哪个状态s开始MDP过程，最优策略都是 π^*

值迭代(value iteration)和策略迭代(policy iteration)

值迭代和策略迭代都可以解决有限状态下的MDP过程，目前为止，我们只考虑有限状态和有限的动作空间 $|\mathrm{S}| < \infty, |\mathrm{A}| < \infty$。

值迭代算法

1. For each state s, initialize $V(s) := 0$.
2. Repeat until convergence {

   For every state, update $V(s) := R(s) + \max_{a \in A} \gamma \sum_{s'} P_{sa}(s') V(s')$

   }

利用贝尔曼等式重复的更新预计的值函数

在循环体中有两种更新算法

同步更新(synchronous)：对每个状态s计算V(s)的新值，并且以此覆盖掉原来的旧值。在这个例子中，这个算法可以看做贝尔曼备份运算符。这个运算法接收值函数的当前估计，并且把它映射到新的估计。

异步更新(asynchromous)：使用某种次序遍历状态集，每次更新其中一个值

不管用哪种更新算法，最终值迭代都能使V收敛到 V^*，然后再使用 $\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s')$ 来找出最优策略

策略迭代算法

1. Initialize π randomly.
2. Repeat until convergence {

   (a) Let $V := V^{\pi}$.

   (b) For each state s, let $\pi(s) := \arg\max_{a \in A} \sum_{s'} P_{sa}(s') V(s')$

   }

内部循环计算当前策略的值函数，然后使用当前值函数来更新策略（b步骤的策略 π 也可以叫做V的贪心策略）

迭代了某个最大迭代次数之后，V 将会收敛到 V^*，而 π 会收敛到 π^*

值迭代和策略迭代都是解决MDPs的标准算法， 而且目前对于这两个算法哪个更好，还没有一个统一的一致意见。对小规模的 MDPs 来说，策略迭代通常非常快，迭代很少的次数就能收敛。然而，对有大规模状态空间的 MDPs，确切求解 V π就要涉及到求解一个非常大的线性方程组系统，可能非常困难。对于这种问题，就可以更倾向于选择值迭代。因此，在实际使用中，值迭代通常比策略迭代更加常用。

学习一个马尔可夫决策模型

我们上面的所有推理，都基于一个假设，那就是状态转移概率和奖励函数都是已知的。但是在许多现实中，这种假设并不成立，我们只能从数据中来估算它们（usually, S, A, γ是已知的）

例如，在倒立摆问题中(习题4),在MDP中进行了一系列实验，过程如下

$$s_0^{(1)} \xrightarrow{a_0^{(1)}} s_1^{(1)} \xrightarrow{a_1^{(1)}} s_2^{(1)} \xrightarrow{a_2^{(1)}} s_3^{(1)} \xrightarrow{a_3^{(1)}} \cdots$$
$$s_0^{(2)} \xrightarrow{a_0^{(2)}} s_1^{(2)} \xrightarrow{a_1^{(2)}} s_2^{(2)} \xrightarrow{a_2^{(2)}} s_3^{(2)} \xrightarrow{a_3^{(2)}} \cdots$$
$$\cdots$$

$s_i^{(j)}$ 表示第j次实验的第i次状态；$a_i^{(j)}$ 表示该状态下的对应动作。

有了在 MDP 中一系列试验得到的“经验”，就可以对状态转移概率（state transition probabilities）推导出最大似然估计

$$P_{sa}(s') = \frac{\#\text{times took we action } a \text{ in state } s \text{ and got to } s'}{\#\text{times we took action a in state } s}$$

如果上面出现了0/0的情况，那么就是在状态 s 之前没进行过任何动作 a，这样就可以简单估计 $P_{sa}(s') = 1/|S|$，也就是说把 Psa(s') 估计为在所有状态上的均匀分布（uniform distribution）

如果在 MDP 过程中我们能观察更多次数，就能利用新经验来更新估计的状态转移概率。具体来说就是保存下来等式中的分子和分母的计数，那么观察到更多的试验的时候，就可以很简单地累积这些计数数值。

类似地，，如果奖励函数（reward） R 未知，我们也可以选择在状态 s 下的期望即时奖励函数（expected immediate reward） R(s) 的估计来作为在状态 s 观测到的平均奖励函数（average reward）。

学习了MDP模型后，我们可以用值迭代或者策略迭代，并且使用估计的状态转移概率和奖励函数，来求解MDP问题。

下面是一种可行的算法

1. 随机初始化 π
2. 循环{
   (a)在 MDP 中执行 π 作为若干次试验
   (b)利用上面在 MDP 积累的经验（accumulated experience），更新对 P_{sa} 的估计（如果可以的话也对奖励函数 R 进行更新）。
   (c)把估计的状态转移概率和奖励函数应用到值迭代上，来得到新的估计的值函数V
   (d)更新 π 作为V的贪婪策略
   }