黑色的仙人掌

中度可信度描述已自动生成

**计算机与信息学院**

**《人工智能》课程作业**

**2024年秋季学期**

|  |  |
| --- | --- |
| 课程类型： | 专业核心课 |
| 学 号： | 202210120518 |
| 姓 名： | 胡国昌 |
| 专 业： | 计算机科学与技术 |
| 授课教师： | 臧兆祥 |

完成日期：2024年 11月 5日

**强化学习-机器人导航**

# 案例内容与要求

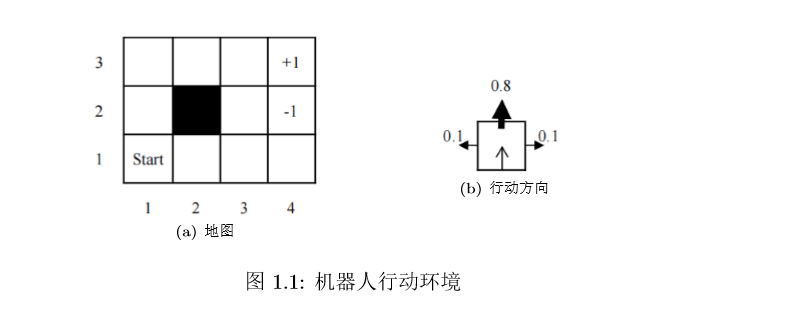
**1.1 实验内容与任务**

图1.1(a)是一个机器人导航问题的地图。机器人从起点Start出发，每一 个时间点，它必须选择一个行动(上下左右)。在马尔可夫决策中实验中，机 器人是根据环境模型中的转移矩阵 P(x,a,x0) 来进行价值函数和最优策略 的计算，但是本次实验中，机器人并不知道这个转移矩阵。已知机器人行 动之后，环境会告知机器人两件事情–新的实际位置以及到达新位置所得到 的报酬。因此，如果机器人有一个策略π(x)，那么在与环境的交换中，机 器人会具有这样一个数据序列：(x0,a0,r0,x1,a1,r1,..., an−1,rn−1,xn)，其 中xi,ri 是环境告知的状态和该状态的报酬，ai = π(xi)是机器人自己的决 策，x0是Start, xn 是一个终止状态。这个数据序列也称为一个样本路径。 强化学习的任务是让机器人在环境中运行多次，得到多条样本路径，通过 这些样本路径，来求解最优策略。

样本路径是与环境的交互中产生的，你先要实现一个环境模型。假设 实际位置由环境按图1.1(b) 的方式决定：机器人每次移动的实际结果是机 器人以0.8 的概率移向所选择方向，也可能是以0.1的概率移向垂直于所选 方向。如果实际移动的方向上有障碍物，则机器人会停在原地。机器人移 动到图中每个格子，会获得一个报酬, 图1.1(a) 中标有+1和-1 的格子中标 记的就是该格子的报酬，其他格子的报酬是-0.04. 报酬会随着时间打折，假 设折扣是1。

**1.2 实验过程及要求**

1. 实验环境要求：Windows/Linux操作系统，Python编译环境，numpy、 scipy等程序库。



2. 编写一个环境，它能跟机器人交互，主要提供行动的结果–下一步的 状态及获得的报酬。

3. 已知机器人的策略π(x), 通过与环境交互学习在该策略下的价值函 数U(x)（或者叫效用函数）。

4. 机器的行动价值函数是Q(x,a)，且π(x) = argmax a

Q(x,a)是最优决策,

通过与环境交互学习这个行动价值函数Q(x,a)。

5. 已知机器人的策略π(x), 用一个线性函数来逼近价值函数，通过与环 境交互学习这个线性函数。

6. 撰写实验报告。

**1.3 教学目标**

1. 理解和掌握强化学习的原理。

2. 能够应用时序差分方法，计算状态的价值函数。

3. 能够应用时序差分方法，计算状态的行为价值函数，从而得到最优决 策。

4. 能够提出一个新函数来逼近价值函数，并用梯度下降法来计算该新函数的参数。

5. 能够分析不同方案的优缺点，提高对复杂工程问题建模和分析的能力。

**1.4 相关知识及背景**

马尔可夫决策过程的实验中，通过状态转移矩阵和状态的报酬，用价 值迭代法和策略迭代法来求解状态的效用以及最优决策。在没有状态转移 矩阵和报酬定义的情况下，机器人可以通过反复与环境进行交互，获得状 态变化数据以及报酬数据，并通过这些数据来学习状态效用和最优决策， 这个过程称为强化学习。强化学习、监督学习、无监督学习构成了机器学 习的几大类别。 强化学习特别适用于类似游戏智能，通过在游戏中获得奖励，引导Agent采 取正确行动。2016年在围棋中击败顶级人类选手的人工智能AlphaGo，主要 应用的是搜索技术、有监督学习技术和强化学习技术，通过大量的战例来 训练模型。 本次实验通过奖励机制，应用时序差分方法来学习机器人的价值函数 以及行动价值函数，并获得最优策略。

**1.5 实验教学与指导**

**1.5.1 环境模型**

需要模拟一个环境，由这个环境告知机器人的行动的结果。这个模型 可以自定义一个转移矩阵P，并用它生成机器人的新状态。在这个模型中， 除了P不能被学习算法使用外，其他的属性和方法可以被学习算法使用。

class Env:

def \_\_init\_\_(self, name):

self.Name = name

self.N = 11 *# 状态数*

self.A = np.arange(4) *# 动作数*

self.X = np.arange(self.N) *# 状态空间*

self.Gamma = 0.9 *# 折扣因子*

self.StartState = 0 *# 起始状态*

self.EndStates = [6, 10] *# 终止状态*

self.makeP() *# 定义转移概率矩阵*

self.makeR() *# 定义报酬向量*

def action(self, x, a):

x\_ = np.random.choice(self.N, p=self.P[x, a, :]) *# 根据转移概率选择新状态*

return x\_

**1.5.2 被动学习-时序差分方法TD Learning**

给定一个策略π(x)，让机器人学习状态的价值函数，称为被动学习。通 过与环境交互，机器人获得一个样本路径sample = (x0,r0,a0,x1,r1,a1,..., an−1,xn,rn)。价值函数Uπ(x)的定义是机器人从状态x出发，按照策略π连 续移动所获得的报酬总和的期望值。在通过样本计算时，可以近似认为

U(xi) ≈

≈ ri + γU(xi+1) (1.1)

在进行迭代时，可以使用时序差分迭代公式：

U(xi) := (1−α)U(xi) + α(ri + γU(xi+1) := U(xi) + α(ri + γU(xi+1)−U(xi)) (1.2) 其中α是一个权重，称为学习率。

class TD:

def \_\_init\_\_(self, E):

self.E = E

self.Alpha = 0.5 *# 学习率*

self.Pi = [3, 2, 2, 2, 3, 3, 0, 0, 0, 0, 0] *# 策略*

self.U = np.zeros(self.E.N) *# 初始化价值函数*

def train(self):

x = np.random.choice([0, 1, 2, 3, 4, 5, 7, 8, 9]) *# 随机选择起始状态*

while x not in self.E.EndStates:

a = self.Pi[x] *# 按策略选择动作*

\_x = self.E.action(x, a) *# 执行动作，获得新状态*

r = self.E.R[x] *# 获得报酬*

*# 更新状态价值*

self.U[x] = self.U[x] + self.Alpha \* (r + self.E.Gamma \* self.U[\_x] - self.U[x])

x = \_x

**1.5.3 主动学习-Q Learning**

假设机器人由一个行为价值函数Q(x,a)，定义在状态x下采取行动a的 价值，那么根据Q函数机器人可以按概率选取一个行动，并与环境进行交 互，通过奖励来更新Q函数。如果学到了正确的Q 函数，则贪心策略就是最 优策略，

π(x) = argmax a Q(x,a) 而且状态x的价值满足

U(x) = max a

Q(x,a)

。

Q 函数的时序差分迭代公式为

Q(xi,ai) := Q(xi,ai) + α(ri + γ max a0

Q(xi+1,a0)−Q(si,ai)) (1.3)

class Q\_Learning:

def \_\_init\_\_(self, E):

self.E = E

self.Alpha = 0.5 *# 学习率*

self.Q = np.ones((self.E.N, len(self.E.A))) / 4 *# 初始化Q值*

def train(self):

x = np.random.choice([0, 1, 2, 3, 4, 5, 7, 8, 9]) *# 随机选择起始状态*

while x not in self.E.EndStates:

a = np.argmax(self.Q[x]) if np.random.rand() > 0.1 else np.random.choice(self.E.A)

\_x = self.E.action(x, a)

r = self.E.R[x]

*# Q值更新*

self.Q[x, a] = self.Q[x, a] + self.Alpha \* (r + self.E.Gamma \* np.max(self.Q[\_x]) - self.Q[x, a])

x = \_x

1.5.4 价值函数的线性逼近

当状态数过多而价值函数不宜用列表方式表达时，可以用函数逼近的 方式。有些价值函数可以用线性函数做很好的逼近。在本实验中，状态x的 特征为行列坐标(r,c)，如果令U(x) = w1 +w2r +w3c, 则可以定义平方损失 函数J = 1 2(U(x)−S)2, 其中，S是从状态x开始的带折扣报酬序列总和。可 以应用梯度下降法来求U(x)的参数w = (w1,w2,w3)

class FTD:

def \_\_init\_\_(self, E):

self.w = np.array([0.5, 0.5, 0.5]) *# 权重向量初始化*

self.E = E

self.Alpha = 0.001 *# 学习率*

self.Pi = [3, 2, 2, 2, 3, 3, 0, 0, 0, 0, 0] *# 策略*

def U(self, x):

if x in self.E.EndStates:

return self.E.R[x]

(row, col) = divmod(x, int(np.sqrt(self.E.N)))

return np.dot(np.array([1, row, col]), self.w) *# 线性逼近的状态价值*

def dU(self, x):

(row, col) = divmod(x, int(np.sqrt(self.E.N)))

return np.array([1, row, col])

def train(self):

x0 = np.random.choice([0, 1, 2, 3, 4, 5, 7, 8, 9])

Rsum = self.E.R[x0]

x = x0

gamma = self.E.Gamma

while x not in self.E.EndStates:

a = self.Pi[x]

x = self.E.action(x, a)

Rsum += gamma \* self.E.R[x]

gamma \*= self.E.Gamma

*# 更新权重*

self.w = self.w + self.Alpha \* (Rsum - self.U(x0)) \* self.dU(x0)