作业 4：强化学习

清华大学软件学院
机器学习, 2024 年秋季学期

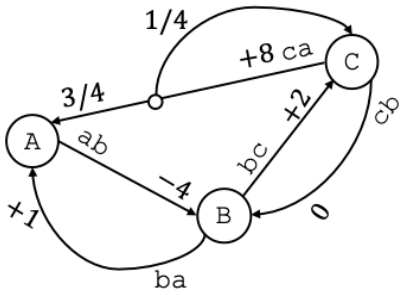
1 介绍

本次作业需要提交说明文档（PDF 形式）和 Python 的源代码。注意事项如下：

- 作业按点给分，因此请在说明文档中按点回答，方便助教批改。
- REINFORCE & AC 题目中使用的 Pytorch 主要用于计算梯度，使用 CPU 即可快速运行。
- 不要使用他人的作业，也不要向他人公开自己的作业，否则处罚很严厉，会扣至-100（倒扣本次作业的全部分值）。
- 统一文件的命名：{学号}_{姓名}_hw4.zip

2 Bellman Equation (25pt)

考虑下图所示马尔可夫决策过程（MDP）：衰减系数 $\gamma = 0.5$，大写字母 A、B、C 表示状态，小写字母组合 ca、ab、cb、bc、ba 表示可以采取的动作，正负整数表示采取行为可以获得的奖励，分叉分支上的分数表示转移概率。例如从状态 C 采取动作 ca 有 $\frac{3}{4}$ 概率可以到达状态 A，有 $\frac{1}{4}$ 概率依然在状态 C，两种情况均可以获得 +8 奖励。其他情况下，采取动作一定可以完成状态转移。



1. 写出衰减系数为 γ 的 MDP 中，策略 π 的状态值函数 $V^{\pi}(s)$ 的定义。

2. 写出状态值函数 $V^{\pi}(s)$ 所符合的贝尔曼（Bellman）期望方程。

3. 考虑一个均匀随机策略 π_0（以相同的概率选取所有动作），初始状态值函数 $V_0^{\pi_0}(A) = V_0^{\pi_0}(B) = V_0^{\pi_0}(C) = 0$，请利用 2 中的贝尔曼期望方程，写出上述 MDP 过程中，迭代式策略评估进行一步更新的状态值函数 $V_1^{\pi_0}$。

4. 基于 3 中计算得到的 $V_1^{\pi_0}$，利用贪心法得到确定性策略 π_1。

3 TD(λ) & Eligibility Trace（附加题，10pt）

本题中, 我们将逐步证明 TD(λ) 算法的前向视角（forward-view）和后向视角（backward-view）的等价性。

在 TD(λ) 算法中，我们使用 λ-回报函数

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)},$$

其中 n 步回报

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1}R_{t+n} + \gamma^n V(S_{t+n}).$$

当达到终止步 T 后，所有的后续回报为 0，所以 $G_t^{(T-t)} = G_t^{(T-t+1)} = \cdots$，因此

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}G_t^{(n)} + \lambda^{T-t-1}G_t^{T-t}.$$

在 TD(λ) 算法的**前向视角**中，对于采样轨迹 $S_0, A_0, R_1, \cdots, S_{T-1}, A_{T-1}, R_T$ 中遇到的每一个状态 S_t，我们更新它的价值

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^\lambda - V(S_t)\right).$$

假设此时采用一种 off-line 的更新策略，即：我们预先记录每个状态的价值更新量，在整条轨迹中涉及的所有状态的价值更新总量都被记录完毕后，再对所有状态进行统一的价值更新。

在 TD(λ) 算法的**后向视角**中，我们引入了资格迹的概念。初始状态下，所有状态的资格迹均为 0。在采样轨迹 $S_0, A_0, R_1, \cdots, S_{T-1}, A_{T-1}, R_T$ 中，对于第 t 步，我们遇到状态 S_t，于是更新它的资格迹：

$$E(S_t) = E(S_t) + 1.$$

然后，我们会计算 TD error：

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

并对**所有状态** S 的价值进行更新：

$$V(S) \leftarrow V(S) + \alpha\delta_t E(S),$$

对**所有状态** S 的资格迹进行衰减：

$$E(S) = \gamma\lambda E(S).$$

同样的，我们采用 off-line 的更新策略，在整条采样轨迹处理完毕后再对所有状态进行统一的价值更新。

要证明前向视角和后向视角的等价性，我们可以考察在这两种视角中，对于任意一个状态 s，其价值的更新是否完全一致。

由于在这两种视角中均采用 off-line 的更新策略，我们应该考察在整条轨迹均处理完毕后，每个状态的价值更新总量。

1. 定义

$$I(S_1, S_2) = \begin{cases} 1 & \text{if } S_1 = S_2 \\ 0 & \text{if } S_1 \neq S_2. \end{cases}$$

在后向视角中，状态 s 在第 t 步的价值更新量为 $\Delta V_t^{back}(s) = \alpha \delta_t E(s)$。

请证明：对于整条轨迹，状态 s 的价值更新量

$$\Delta V_{all}^{back}(s) = \sum_{t=0}^{T-1} \Delta V_t^{back}(s) = \alpha \sum_{t=0}^{T-1} I(s, S_t) \sum_{k=t}^{T-1} (\gamma\lambda)^{k-t} \delta_k.$$

2. 在前向视角中，状态 s 在第 t 步的价值更新量为 $\Delta V_t^{for}(s) = I(s, S_t)\alpha \left(G_t^\lambda - V(S_t)\right)$。现在，我们需要将 G_t^λ 展开为用 $R_t, V(S_t)$ 描述的表达式，并得出 $\Delta V_t^{for}(s)$ 使用 $R_t, V(S_t)$ 描述的表达式。（提示：可依次算出每一项的系数）
3. 在前向视角中，对于整条轨迹，状态 s 的价值更新量为 $\Delta V_{all}^{for}(s) = \sum_{t=0}^{T-1} \Delta V_t^{for}(s)$。

请证明：$\Delta V_{all}^{for}(s) = \Delta V_{all}^{back}(s)$.

4 Q-Learning & Sarsa （25pt）

Gymnasium[1]是一套开源强化学习环境。在本题中，你将基于其中的 FrozenLake-v1 环境[2]，用强化学习算法控制小人在冰面移动到目标点，同时避免掉进地图上的冰洞。

冰面是一个 4×4 的网格区域，冰洞的位置固定，出发点和目标点分别在地图的左右上角。特别地，由于冰面十分光滑，当你控制小人往一个方向走时，它有 1/3 的概率成功遵循你的指示，但也有各 1/3 的概率滑向和你期望的前进方向垂直的两个方向（例如，控制小人往右走时，各有 1/3 的概率往上、右、下移动一格）。小人不会滑出地图范围。如果滑入冰洞或者达到目标点，则游戏结束。更详细的设定请参考 FrozenLake-v1 环境官方说明。另外，我们在官方实现上修改了奖励函数（`main.py:9-18`），当掉小人进冰洞时会收到 -1 的奖励值，到达目标点会收到 2 的奖励值，其它时间都会收到 -0.03 的惩罚值，来让小人尽快走向目标点。

本题需要提交实验报告，代码见 `./code/sarsa_Q_learning`。

1. 补充 `./algorithms/QLearning` 函数，**填入 1 行代码**实现 Q-learning 算法；
2. 补充 `./algorithms/Sarsa` 函数，实现 Sarsa 算法；（可参考提供的 Q-learning 算法）
3. 完成两种不同算法迭代步长 `lr` 取值下的对比实验，绘制不同步长下的学习曲线图，并简要分析结果。（提示：`main.py` 的注释中有相关绘图代码）
4. 如果修改 `main.py:17` 处的代码，将每步 -0.03 的惩罚值增大到 -0.3，算法学到的策略会有何不同？请简要分析原因。

[1] `https://gymnasium.farama.org/`

[2] `https://gymnasium.farama.org/environments/toy_text/frozen_lake/`

5 REINFORCE & AC （50pt）

在本题中，你将基于 CartPole-v1[3]环境，利用强化学习算法控制平衡木。本题需要提交实验报告，代码见 `./code/policy_gradient`。

1. 补充 `REINFORCE` 类中的 `learn` 函数，实现 REINFORCE 算法。
2. 补充 `TDActorCritic` 类中的 `learn` 函数，实现 TD Actor-Critic 算法。value 的损失函数已经预先实现了，只需要实现 policy 的损失函数即可。代码中 td_target $= R_{t+1} + \gamma v_\pi(S_{t+1})$。
3. 请绘制**两个模型**的训练曲线，包括训练过程的损失函数的变化和最终奖励值，并分析训练稳定性及收敛效率。由于强化学习的不稳定性，你的结果需要基于至少 3 个种子。

提示：

1. 动手之前，请仔细阅读代码中的注释，确保你已了解问题定义和代码框架。
2. 你可以解除位于 182 行的注释以获取可视化结果，让你看到测试时平衡木的控制结果。
3. 本题已经提供两种 Policy Gradient 算法的代码框架，希望你完成损失函数部分，可以参考论文[4]与课件，REINFORCE 位于第 10 讲课件第 16 页，TD Actor-Critic 位于第 10 讲课件第 24 页，TD Actor-Critic 和 QAC 的核心思想是一致的，区别在于 critic 网络输出的不是 q 值，而是 value。如果你理解了 QAC，那么你应该可以很轻松地完成本次作业。
4. TD Actor-Critic 中，make_batch() 函数已经将所有需要用到的变量转换为 torch.Tensor，你可以直接调用 self.ac.v(self.states) 获取不同状态的价值 $v_\pi(S)$。
5. 训练中出现抖动是正常现象。只要 REINFORCE 在 cartpole-v1 上的训练过程中最终奖励最高能接近 500，就可以获得代码实现的全部分数。但如果 AC 算法在 3 个种子都没有获得最高奖励，也许你应该检查一下你的实现。
6. Pytorch 框架在本题中的用法等价于 numpy，比如可以通过 `torch.mean` 计算均值，通过 `torch.std` 计算方差。在 debug 过程中，如果你无法确定一个 Tensor 的形状，你可以使用 Tensor.shape 获取之。如果你之前没有安装过 Pytorch，推荐通过 conda 安装 cpu 版本，具体命令请参考[5]。
7. pytorch 在计算时会保存计算图，以供 autograd 模块自动求导。因此请注意在计算时不要使用 torch.tensor() 创建中间变量，因为这样创建出的变量是值拷贝，不在计算图上，不会计算梯度。可以使用 torch.cat() / torch.stack() / torch.gather() 创建中间变量。

[3] `https://gymnasium.farama.org/environments/classic_control/cart_pole/`

[4] `https://homes.cs.washington.edu/~todorov/courses/amath579/reading/PolicyGradient.pdf`

[5] `https://pytorch.org/get-started/locally/`