冰壶游戏

问题描述

冰壶游戏是要控制一个半径为1，质量为1 的冰壶，在一个长宽是100×100的正方形球场内移动. 不考虑冰壶的自转. 当冰壶和球场的边界碰撞时，碰撞前后冰壶的速度会乘上回弹系数0.9，移动方向和边界呈反射关系.

我们需要分别操纵x 轴和y 轴的两个力控制冰壶的移动：在x 轴的正或反方向施加5 单位的力；在y 轴的正或反方向施加5 单位的力. 这样一共会有4 种不同的控制动作. 动作可以每1/10 秒变换一次；但在仿真冰壶运动动力学时，仿真时间间隔是1/100 秒. 除了我们施加的控制动作，冰壶会受到空气阻力，大小等于$0.005 \times speed^2$. 假设冰壶和地面没有摩擦力.

在每个决策时刻(1/10 秒)，环境反馈的奖励等于$-d$，其中d 是冰壶和任意给定的目标点之间的距离.为了保证学到的策略能够控制冰壶从任意初始位置上移动到任意目标点，每隔30 秒就把冰壶状态重置到球场内的一个随机点上，同时x 轴和y 轴的速度也随机重置在[-10, 10] 范围内. 与此同时，目标点也被随机重置.

实验平台

Python 3.7.6　NumPy 1.18.1　PyTorch 1.5.0　MatPlotLib 3.1.3 FFmpeg 4.2.2

环境建模

对环境进行建模，代码在 curling.py 中。

状态表示：由冰壶的位置、目标的位置及冰壶移动速度组成的三元组 (position, target, v)

动作表示：0, 1, 2, 3 分别代表 (5, 5), (-5, 5), (5, -5), (-5, -5) 四个动作

Curling 类的主要属性： position （冰壶的位置）、target（目标的位置）、state（冰壶的状态）

Curling 类的主要方法： move （移动冰壶）、 reward （冰壶的回报）、 action （对冰壶采取动作）、 reset （重置冰壶的状态）

实验原理

将每隔30秒当成一次轨迹，则问题是一个 episodic MDPs 问题。更具体地，它是状态连续、动作离散的。因此不是直接使用前五章的知识。

方法一：将连续的空间$\mathbf{S}$划分成非重叠的、相邻的子空间，同一个子空间上的状态具有相同的价值。考虑到冰壶的 $\mathbb{R}$ 和 $\mathbb{P}$ 是确定的，用动态规划法（价值迭代）求解。值函数可以收敛但是冰壶的运动很差，与随机的值函数一样。经过反复查错后，问题出在了这种简单的离散化方法上，离散化后的子空间太大，以至于冰壶在当前状态采取不同的动作后仍在同一子空间中，导致 Q(s, a) 函数的值都一样。

1: 初始化一个函数 V_1 (e.g. $V_1(s) = 0, \forall s \in \mathcal{S}$)
2: **loop**
3: 根据已知的 V_k 计算一个新的函数

$$V_{k+1}(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_k(s') \right), \forall s \in \mathcal{S}$$

4: $k \leftarrow k + 1$
5: **end loop**

方法二：利用多层感知机对Q(s, a)进行逼近，并利用 MC 控制算法进行迭代，结果不收敛。

1: 定义 Q 函数逼近器 $\hat{Q}(s, a, \mathbf{w})$, 初始化 $\mathbf{w}$
2: 提取出 ϵ 贪心策略 $\pi = \epsilon\text{-greedy}(\hat{Q}(\mathbf{w}))$
3: **repeat** {对每个 episode}
4: 根据 π 采样轨迹 $\{s_0, a_0, r_1, s_1, \ldots, s_T\}$
5: **repeat** {对 episode 中每个首次访问的 (s_t, a_t)}
6: 计算回报 $G_t = r_{t+1} + \gamma r_{t+2} + \ldots$
7: 计算损失 $\mathbf{loss} = ||G_t - \hat{Q}(s_t, a_t, \mathbf{w})||^2$
8: 回传损失
9: **until**
10: 更新策略 $\pi = \epsilon\text{-greedy}(\hat{Q}(\mathbf{w}))$
11: **until**

方法三：利用多层感知机对Q(s, a)进行逼近，并利用 Sarsa 控制算法进行迭代，结果不收敛。

1: 定义 Q 函数逼近器 $\hat{Q}(s, a, \mathbf{w})$, 初始化 $\mathbf{w}$,
$\pi = \epsilon\text{-greedy}(\hat{Q}(\mathbf{w}))$, $s_t = s_0$, $t = 0$
2: 采样动作 $a_t \sim \pi(s_t)$ 并执行, 观测 (r_{t+1}, s_{t+1})
3: **loop**
4: 采样动作 $a_{t+1} \sim \pi(s_{t+1})$ 并执行, 观测 r_{t+2}, s_{t+2}
5: 根据 $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ 计算损失

$$loss = ||r_{t+1} + \gamma \hat{Q}(s_{t+1}, a_{t+1}, \mathbf{w}) - \hat{Q}(s_t, a_t, \mathbf{w})||^2$$

6: 回传损失
7: 更新策略 $\pi = \epsilon\text{-greedy}(\hat{Q}(\mathbf{w}))$
8: $t \leftarrow t + 1$
9: **end loop**

方法四：利用多层感知机对Q(s, a)进行逼近，并利用 Q-learning 控制算法进行迭代，结果不收敛。

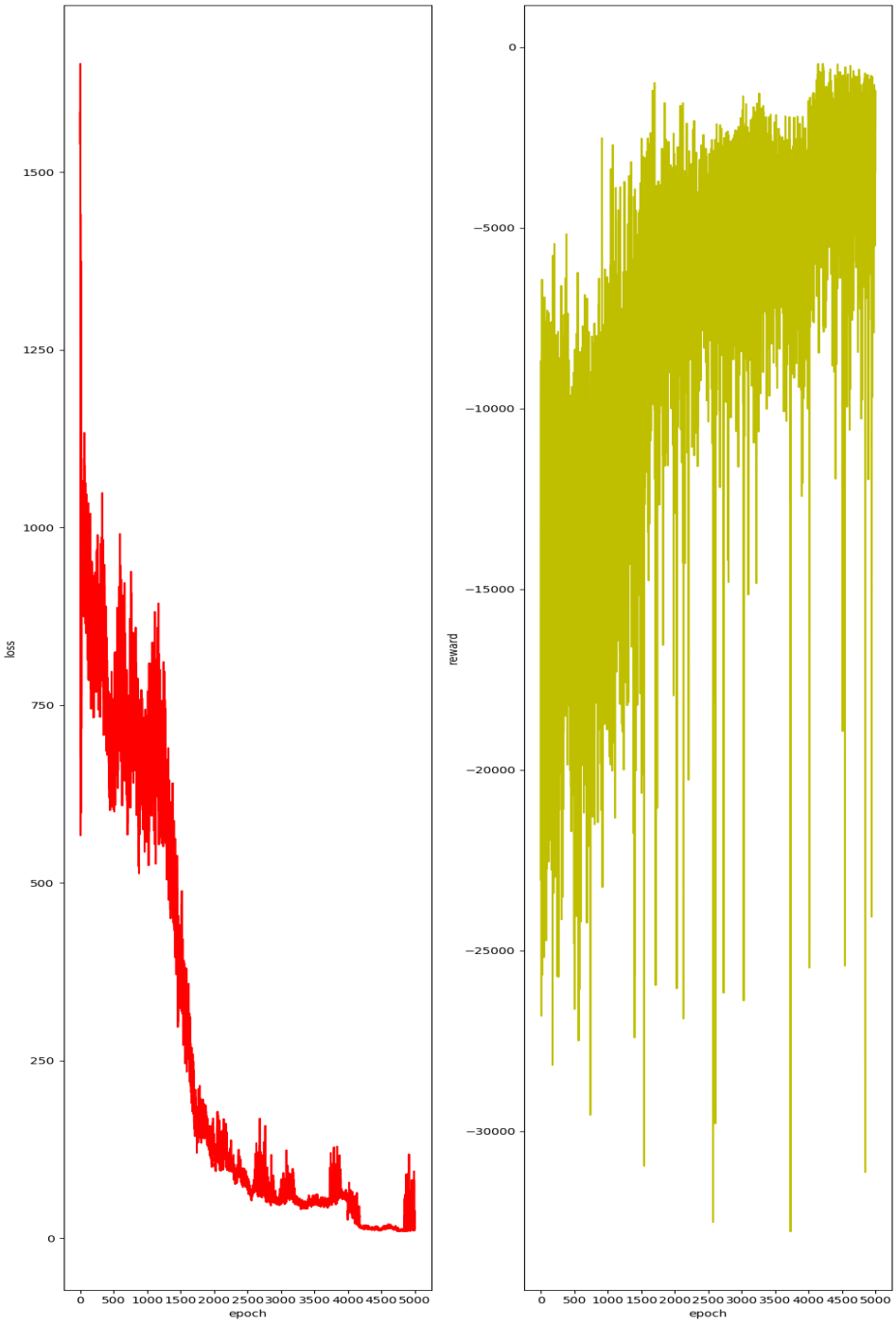
1: 定义 Q 函数逼近器 $\hat{Q}(s, a, \mathbf{w})$, 初始化 $\mathbf{w}$,
$\pi = \epsilon$-greedy$(\hat{Q}(\mathbf{w}))$, $s_t = s_0$, $t = 0$
2: 采样动作 $a_t \sim \pi(s_t)$ 并执行, 观测 (r_{t+1}, s_{t+1})
3: **loop**
4: 采样动作 $a_{t+1} \sim \pi(s_{t+1})$ 并执行, 观测 r_{t+2}, s_{t+2}
5: 根据 $(s_t, a_t, r_{t+1}, s_{t+1})$ 计算损失

$$loss = ||r_{t+1} + \gamma \, max_a \hat{Q}(s_{t+1}, a, \mathbf{w}) - \hat{Q}(s_t, a_t, \mathbf{w})||^2$$

6: 回传损失
7: 更新策略 $\pi = \epsilon$-greedy$(\hat{Q}(\mathbf{w}))$
8: $t \leftarrow t + 1$
9: **end loop**

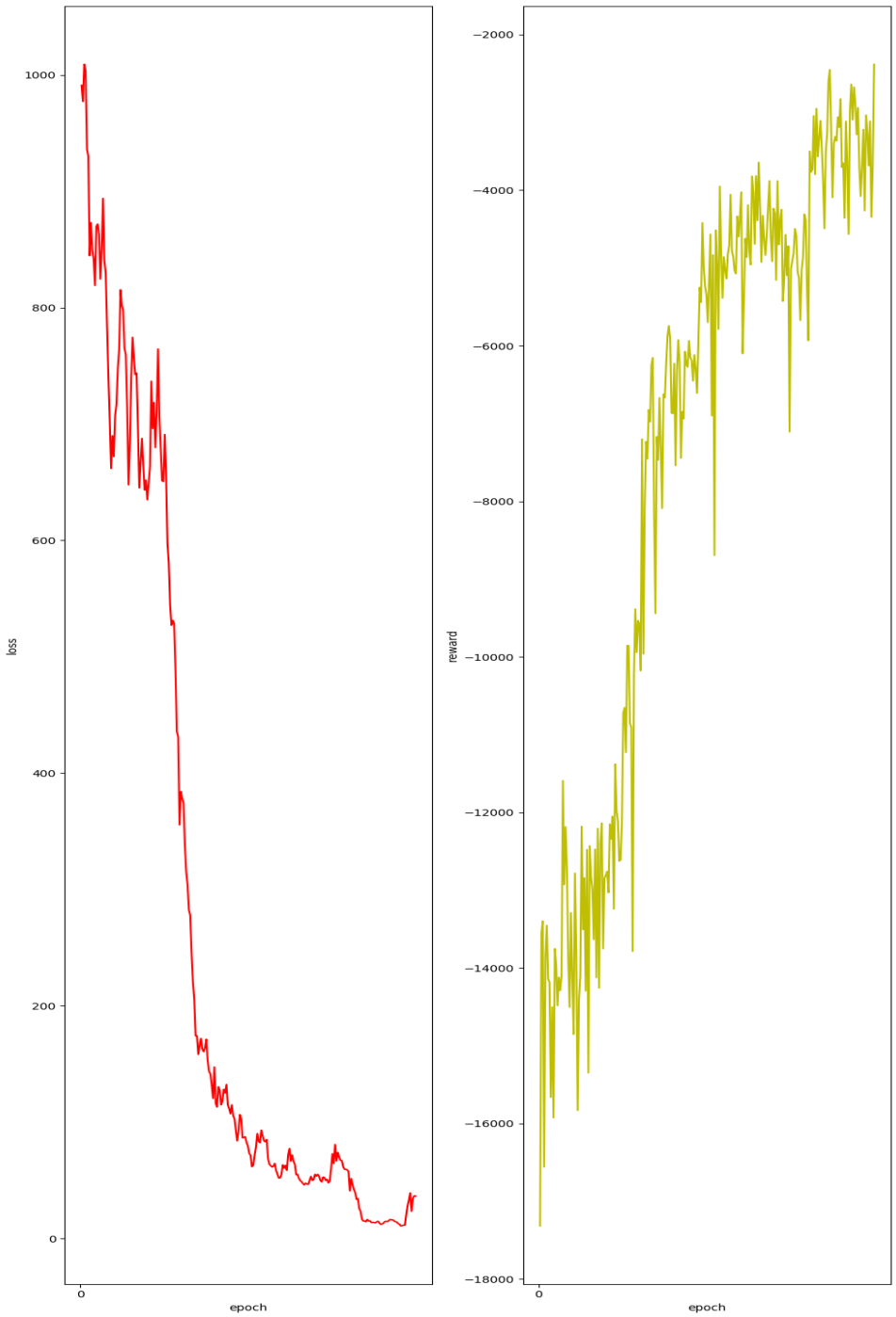
方法五（对方法四的改进）：在仔细研究[1]中的实验分析后，加入了 Experience Replay (ER) ，此时收敛。代码在 dqn.py 中，训练过程保存在 log.txt 中，网络参数保存在 policy.pth 中，demo 为 out.mp4 。

每个episodic对应的loss和reward如图所示：



可以明显看出 loss 随着 epoch 增多而下降，但是 reward 下降不是很明显。

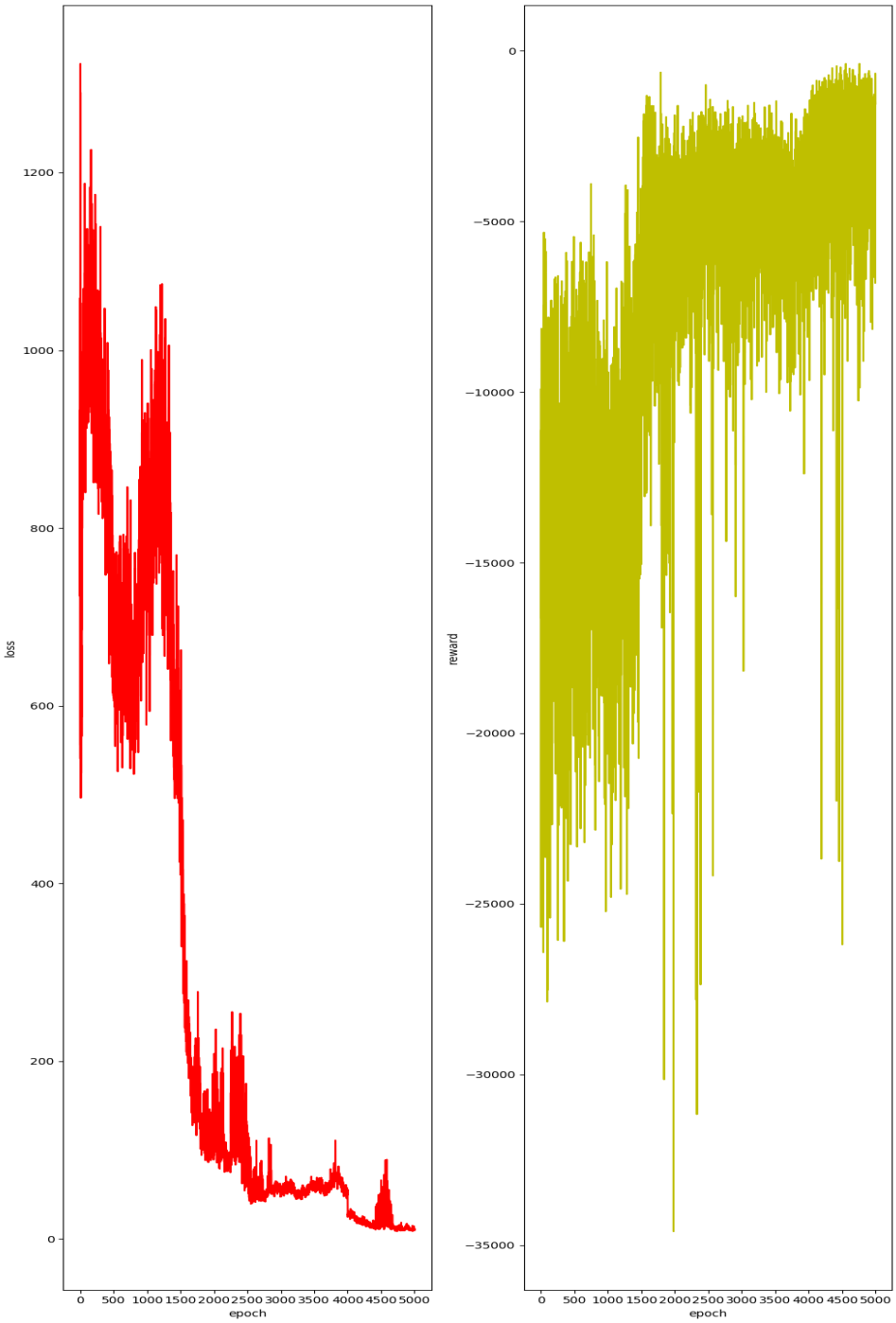
每20个episodics的average loss和average reward如图所示：



非常明显看出 loss 随着 epoch 增多而下降，reward 随着 epoch 增多而上升。

方法六（对方法三的改进）：同方法五加入了 Experience Replay (ER) ，此时收敛。代码在 sarsa.py 中，训练过程保存在 sarsa_log.txt 中，网络参数保存在 sarsa_policy.pth 中，demo 为 sarsa_out.mp4。

每个episodic对应的loss和reward如图所示：



可以明显看出 loss 随着 epoch 增多而下降，但是 reward 下降不是很明显。

实验结果分析

引入Experience Replay (ER)后，解决了数据之间存在强关联性的问题，同时可以 mini-batch 进行训练，降低了方差，有助于 Q 网络训练。

将 Q-learing 的结果与 Sarsa 的结果做对比，10 个 episodics 的 average reward 分别为 -2447.010582779762 和 -3024.4631447815127 。对比 out.mp4

和 sarsa_out.mp4 也可以发现 Q-learning 学出来的结果更好一些。

参考文献

[1] https://www.declanoller.com/2018/11/04/training-an-rl-agent-to-play-puckworld-with-a-ddqn/
[2] Sutton, R. S., and A. G. Barto. Reinforcement Learning: An Introduction. 2018.