强化学习第一次实验作业

> 姓名：陈晖
>
> 学号：2022E8017782004

文件：

- pendulum_Q-learning.py 【倒立摆-Q学习的主文件
- 强化学习第一次作业*v2*陈晖_2022E8017782004.pdf 【作业报告
- self_pendulum_env.py 【修改后的OpenAI公司的仿真文件
- clockwise.png 【仿真所需的图片
- DQN
  - clockwise.png 【仿真所需的图片
  - DQN_model.py 【深度强化学习模型文件
  - pendulum_DQN.py 【深度强化学习解决倒立摆问题
- data
  - result_q_table.csv 【训练好的Q-table

部分版本依赖

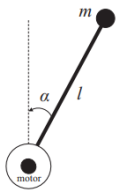
```
pygame==2.3.0
gym==0.26.2
```

题目：

问题 1：倒立摆

倒立摆是将一个物体固定在一个圆盘的非中心点位置, 由直流电机驱动将其在垂直平面内进行旋转控制的系统 (图 1). 由于输入电压是受限的, 因此电机并不能提供足够的动力直接将摆杆推完一圈. 相反, 需要来回摆动收集足够的能量, 然后才能将摆杆推起并稳定在最高点.

(a) 真实系统



(b) 示意图

Figure 1: 倒立摆问题.

研究思路：

pendulum倒立摆模型是经典的强化学习场景，笔者采用Q-learning方法解决该方法。

Q-learning算法过程：

Q-learning是一种基于值函数的强化学习算法，其核心思想是通过学习Q值函数来寻找最优策略。

值函数可以通过如下的更新规则进行更新：

$$Q(s,a) = Q(s,a) + \alpha * [r + \gamma * max(Q(s',a'))–Q(s,a)]$$

其中，Q(s, a)表示在状态s下采取动作a所对应的Q值，

α是学习率，r是奖励，γ是折扣因子，s'和a'是采取动作a之后的下一个状态和动作。

在训练过程中，可以使用ε-greedy策略来进行动作选择，即以一定的概率ε选择一个随机动作，以1-ε的概率选择当前状态下Q值最大的动作。

倒立摆问题解析：

对于动作集：

可以将动作空间离散化成 {−3, 0, 3} 三个动作, 以这三个动作作为动作 集学习最优策略

动作空间：[-3, 0, 3]

对于状态s转的转移：

可通过倒立摆系统连续时间动力学模型，来构造转移函数

Table 1: 倒立摆系统参数

变量	取值	单位	含义
m	0.055	kg	重量
g	9.81	$\mathrm{m/s^2}$	重力加速度
l	0.042	m	重心到转子的距离
J	$1.91 \cdot 10^{-4}$	$\mathrm{kg \cdot m^2}$	转动惯量
b	$3 \cdot 10^{-6}$	$\mathrm{Nm \cdot s/rad}$	粘滞阻尼
K	0.0536	Nm/A	转矩常数
R	9.5	Ω	转子电阻

倒立摆系统连续时间动力学模型是

$$\ddot{\alpha} = \frac{1}{J}\left(mgl\sin(\alpha) - b\dot{\alpha} - \frac{K^2}{R}\dot{\alpha} + \frac{K}{R}u\right) \tag{1}$$

表 1 给出了所有参数的含义和取值. 系统状态包含摆杆的角度和角速度, 即 $s = [\alpha, \dot{\alpha}]^T$. 角度 α 取值范围在 $[-\pi, \pi)$ rad 之间. 其中 $\alpha = -\pi$ 对应摆杆垂直指向下, $\alpha = 0$ 对应摆杆垂直指向上. 速度 $\dot{\alpha}$ 被限制在 $[-15\pi, 15\pi]$rad/s 范围内. 控制动作 (电压)u 被限制在 $[-3, 3]$V 范围内. 采样时间 T_s 选取 0.005s, 离散时间动力学 f 可以根据 (1) 由欧拉法获得

$$\begin{cases} \alpha_{k+1} = \alpha_k + T_s\dot{\alpha}_k & (2) \\ \dot{\alpha}_{k+1} = \dot{\alpha}_k + T_s\ddot{\alpha}(\alpha_k, \dot{\alpha}_k, a_k) & (3) \end{cases}$$

可以通过动作空间中选取动作，带入动力学模型公式，得到更新后的状态

状态是二维向量：【角度，角速度】

对于奖励函数：

函数定义成如下二次型形式

$$\begin{aligned} \mathcal{R}(s,a) &= -s^T Q_{rew} s - R_{rew} a^2 \\ Q_{rew} &= \begin{bmatrix} 5 & 0 \\ 0 & 0.1 \end{bmatrix}, R_{rew} = 1. \end{aligned} \tag{4}$$

折扣因子选取 $\gamma = 0.98$. 选取较高折扣因子的目的是为了提高目标点 (顶点) 附近奖励在初始时刻状态价值的重要性, 这样最优策略能够以成功将摆杆摆起并稳定作为最终目标.

Q表的构造：

可以采用pandas库中的multiindex来构造二维行索引，索引结构为【角度，角速度】，而把动作集构成列索引。

将角度[-π，π)离散化成[-180,180)，间隔为1的状态空间，将角速度[-15π,15π]离散成[-15,15]，间隔为1的状态空间，而动作空间为{-3 ,0 ,3}。

如图所示：

		-3	0	3
angle	angular_velocity			
-180	-15	0.0	0.0	0.0
	-14	0.0	0.0	0.0
	-13	0.0	0.0	0.0
	-12	0.0	0.0	0.0
	-11	0.0	0.0	0.0
...		...	...	...
179	11	0.0	0.0	0.0
	12	0.0	0.0	0.0
	13	0.0	0.0	0.0
	14	0.0	0.0	0.0
	15	0.0	0.0	0.0

第一索引为角度，第二索引为角速度，列索引为动作集，Q表为360·31·3的表结构

可视化展示：

方式一：

可以在控制台打印[角度，角速度]，优点是比较简单，缺点是不够直观

方式二：

openAI公司有倒立摆的仿真系统pendulum，其可以实时渲染倒立摆指针运动状况，但是内部的物理参数与题目中不相符，其输入是扭矩需要改成电压，其输出需要从[sin(θ),cos(θ),角速度]修改成[θ，角速度]，需要对其源码加以修改使用，优点是比较直观，缺点是修改过程比较麻烦

笔者这里两种方式都实现了。

研究内容：

此内容详细介绍代码含义

物理参数处理：

参数意义如备注所示：

```
np.random.seed(2)  # 随机种子，便于代码复现
Angle_N = 60  # 代表角度等分成多少份
Angular_Velocity_N = 40  # 速度等分成多少份

N_Angle_STATES = np.linspace(-math.pi, math.pi, Angle_N)
N_Angle_STATES = np.round(N_Angle_STATES, decimals=4)
N_Angular_Velocity_STATES = np.linspace(-15 * math.pi, 15 * math.pi,
Angular_Velocity_N)  # 离散化角速度
N_Angular_Velocity_STATES = np.round(N_Angular_Velocity_STATES, decimals=4)
U_Voltage_ACTIONS = [-3, 0, 3]  # 离散化可选电压的动作集
EPSILON = 0.9  # 贪心策略参数
Alp = 0.1  # 更新时的学习率
Lambda = 0.98  # 折扣因子
MAX_EPISODES = 100  # 最大轮数
FRESH_TIME = 0.0001  # 刷新时间
ONE_TURN_STEP = 50000

# pendulum参数
m = 0.055  # kg 重量
g = 9.81  # m/s2 重力加速度
l = 0.042  # m 重心到转子的距离
J = 1.91e-04  # kg · m2 转动惯量
b = 3.0e-06  # Nm · s/rad 粘滞阻尼
K = 0.0536  # Nm/A 转矩常数
R = 9.5  # Ω 转子电阻
T_s = 0.005  # 采样频率
```

Q表的构造：

以第一索引为角度，第二索引为角速度，列索引为动作集，构造360·31·3的Q表

```
def build_q_table(n_angle_states, n_angular_velocity_states, u_voltage_actions):
    index = pd.MultiIndex.from_product([n_angle_states,
n_angular_velocity_states])  # 以(角度，角速度)为索引构建Q表
    columns = u_voltage_actions  # 列值为动作集

    # 构造Q表
    table = pd.DataFrame(
        np.zeros((len(n_angle_states) * len(n_angular_velocity_states),
len(u_voltage_actions))), index=index,
        columns=columns
    )
    table.index.names = ['angle', 'angular_velocity']  # 标注索引名
    # print(table)
    return table
```

动作的选择：

使用ε-greedy策略来进行动作选择，以一定的概率ε选择一个随机动作，以1-ε的概率选择当前状态下Q值最大的动作。

```
def choose_action(state: tuple, q_table_: pandas.DataFrame):
    state_actions = q_table_.loc[state]  # 获得当前state对应的action值
    random_rate = np.random.uniform()  # 获得一个随机数
    if (random_rate > EPSILON) or (state_actions.sum() == 0):
        action = np.random.choice(U_Voltage_ACTIONS)  # ε-greedy中探索性选择动作
    else:
        action = state_actions.idxmax()  # 贪心策略选择较大概率的动作

    return action
```

对于状态s转的转移：

对于OpenAI仿真模型展示中，修改了其中代码：

```
    def step(self, u):
        th, thdot = self.state  # th := theta
        dt = self.dt
        u = np.clip(u, -self.max_voltage, self.max_voltage)[0]
        costs = (5 * (angle_normalize(th) ** 2) + 0.1 * thdot ** 2 + u ** 2)

        self.last_u = u  # for rendering

        # costs = angle_normalize(th) ** 2 + 0.1 * thdot ** 2 + 0.001 * (u ** 2)

        newthdot = thdot + self.get_a(th, thdot, u) * dt
        newthdot = np.clip(newthdot, -self.max_speed, self.max_speed)
        newth = th + newthdot * dt

        self.state = np.array([newth, newthdot])

        if self.render_mode == "human":
            self.render()
        return self._get_obs(), -costs
```

根据更新公式：

$$\begin{cases} \alpha_{k+1} = \alpha_k + T_s\dot{\alpha}_k & (2) \\ \dot{\alpha}_{k+1} = \dot{\alpha}_k + T_s\ddot{\alpha}(\alpha_k, \dot{\alpha}_k, a_k) & (3) \end{cases}$$

控制目标是将摆杆从最低点 $s = [\pi, 0]^T$ 摆起并稳定在最高点 $s = [0, 0]^T$. 奖励函数定义成如下二次型形式

$$\begin{aligned} \mathcal{R}(s, a) &= -s^T Q_{rew} s - R_{rew} a^2 \\ Q_{rew} &= \begin{bmatrix} 5 & 0 \\ 0 & 0.1 \end{bmatrix}, R_{rew} = 1. \end{aligned} \quad (4)$$

cost：为奖励的负值

th：为当前角度

thdot：为当前角速度

newth：为更新后的角度

newthdot：为更新后的角速度

```
    def step(self, u):
        th, thdot = self.state  # th := theta
        dt = self.dt
        u = np.clip(u, -self.max_voltage, self.max_voltage)[0] # 防止action不在合理范围内
        costs = (5 * (angle_normalize(th) ** 2) + 0.1 * thdot ** 2 + u ** 2)

        self.last_u = u  # for rendering

        newthdot = thdot + self.get_a(th, thdot, u) * dt
        newthdot = np.clip(newthdot, -self.max_speed, self.max_speed)
        newth = th + newthdot * dt

        self.state = np.array([newth, newthdot])

        if self.render_mode == "human":
            self.render()
        return self._get_obs(), -costs
```

其调用的函数如下：

```
def get_env_feedback_v2(env, action):
    action = [action]
    state, reward = env.step(action)
    state = tuple(state)
    return state, reward
```

倒立摆系统连续时间动力学模型是

$$\ddot{\alpha} = \frac{1}{J}\left(mgl\sin(\alpha) - b\dot{\alpha} - \frac{K^2}{R}\dot{\alpha} + \frac{K}{R}u\right) \tag{1}$$

```
    def get_a(self, th, thdot, u):
        pi_angle = angle_normalize(th)
        m = self.m
        g = self.g
        l = self.l
        b = self.b
        K = self.K
        R = self.R
        J = self.J

        a_pp = (1 / J) * (
                m * g * l * math.sin(pi_angle) - b * thdot - (K ** 2 / R) *
thdot + (
                K / R) * u)

        return a_pp
```

对于控制台打印方式展示角度，角速度，也需要写一个与环境交互的模型：

与OpenAI仿真模式计算方式相同，但涉及了角度的弧度制与角度制的转化。

```
def get_env_feedback(state: tuple, action):
    a_k = state[0]  # 角度
    a_p_k = state[1]  # 角速度/pi

    Reward = - (5 * (angle_normalize(a_k) ** 2) + 0.1 * (a_p_k) ** 2 + action **
2)

    # 获得角加速度
    a_pp = get_a__(a_k, a_p_k, action)

    # 更新角速度
    a_p_k_1 = a_p_k + T_s * a_pp
    a_p_k_1 = np.clip(a_p_k_1, min(N_Angular_Velocity_STATES),
max(N_Angular_Velocity_STATES))

    # 更新角度
    a_k_1 = a_k + T_s * a_p_k_1

    new_state = (a_k_1, a_p_k_1)

    return new_state, Reward
```

```
def get_a__(a, a_, action):
    a = angle_normalize(a)
    a_pp = (1 / J) * (
            m * g * l * math.sin(a) - b * a_ - (K ** 2 / R) * a_ + (
            K / R) * action)  # 计算角加速度
    return a_pp
```

控制台打印函数如下：

```
def update_env(state, episode, step_counter):
    if is_final_state(state):
        interaction = "Episode %s: total_step = %s" % (episode + 1, step_counter)
        print('\r{}'.format(interaction), end='')
        # time.sleep(2)
        print('\n wowow finished!')
    else:

        state = round_state(state)
        interaction = "Episode %s: step = %s" % (episode + 1, step_counter)
        interaction += " angle: " + str(state[0] * 180 / math.pi) + ",
angular_velocity: " + str(
            state[1] / math.pi) + "pi"
        print('\r{}'.format(interaction), end='')
        # time.sleep(FRESH_TIME)
```

其他函数介绍：

round_state：

```
def round_state(state):
    a_k = state[0]
    a_p_k = state[1]

    temp = a_k
    x_ = np.arctan2(np.sin(temp), np.cos(temp))  # 映射到[-pi,pi]

    a_k = get_approximate(x_, states=N_Angle_STATES)

    a_p_k = get_approximate(a_p_k, states=N_Angular_Velocity_STATES)

    return a_k, a_p_k

def get_approximate(val, states):
    idx = np.digitize(val, bins=states)
    if 0 < idx < len(states):
        left = states[idx - 1]
        right = states[idx]
        if np.abs(left - val) < np.abs(right - val):
            idx = idx - 1
```

```
        else:
            idx = idx
    elif idx <= 0:
        idx = 0
    else:
        idx = len(states) - 1
    return states[idx]
```

get_approximate子函数：选择val在states中最近的值

round_state函数：是将一个连续的state状态离散化，取对应离散空间最近的一个值。

angle_normalize(x):

```
def angle_normalize(x):
    return ((x + np.pi) % (2 * np.pi)) - np.pi
```

计算正则化的角度值

is_final_state(state):

```
def is_final_state(state):
    a = state[0]
    a_p = state[1]
    is_mid_a = is_mid(a, N_Angle_STATES)
    is_mid_a_p = is_mid(a_p, N_Angular_Velocity_STATES)
    return is_mid_a and is_mid_a_p


def is_mid(val, states):
    idx_mid = len(states) // 2
    if len(states) % 2 == 0:
        right = states[idx_mid]
        left = states[idx_mid - 1]
        return left < val < right
    else:
        mid = states[idx_mid]
        return val == mid
```

判断当前state是否到达终止条件，由于离散化后，顶点的state不一定是[0,0]，这里取里[0,0]最近的值

实验结果：

实验主函数如下：

def run_pendulum_qlearning(env, is_view=False):

用于训练Q表格，is_view字段可以实时观测倒立摆的状况

```
def run_pendulum_qlearning(env, is_view=False):
    # 构造Q表
    q_table = build_q_table(N_Angle_STATES, N_Angular_Velocity_STATES,
U_Voltage_ACTIONS)

    # 训练MAX_EPISODES个回合
    for episode in range(MAX_EPISODES):
        step_counter = 0
        # 与环境交互
        if is_view:
            state, _ = env.reset()
            state = tuple(state)
        else:
            state, _ = env.reset(is_view=is_view)
            state = tuple(state)

        # 单个回合训练过程
        is_terminated = False
        update_env(state, episode, step_counter)
        while step_counter < ONE_TURN_STEP and (not is_terminated):
            # while not is_terminated:
            state_round = round_state(state)
            # 拿到最大Q值的动作
            action = choose_action(state_round, q_table)
            # 拿到最大Q值的动作对应的Q值
            q_predict = q_table.loc[state_round, action]

            # 与环境交互
            if not is_view:
                state_new, Reward = get_env_feedback(state, action)
            else:
                state_new, Reward = get_env_feedback_v2(env, action)

            # 计算Q-target
            if not is_final_state(state_new):
                q_target = Reward + Lambda *
max(q_table.loc[round_state(state_new)])
            else:
                q_target = Reward
                update_env(state, episode, step_counter)
                is_terminated = True

                # 更新Q表
            q_table.loc[state_round, action] += Alp * (q_target - q_predict)
            state = state_new

            # 打印控制台
            update_env(state, episode, step_counter)
            step_counter += 1

    return q_table

if __name__ == '__main__':
    env = self_pendulum_env.PendulumEnv("human")
```

```
    # 训练时，最好把is_view关闭，因为动画渲染需要时间，影响了整体训练的速度，训练完成后，使用read_q_table(env)进行观察q-table性能
    q_table = run_pendulum_qlearning(env, is_view=False)
    q_table.to_csv('data/result_q_table.csv', index=True, header=True, date_format='%.4f')
```

测试q_table表现状况：

```
def read_q_table(env):
    q_table = pd.read_csv('data/result_q_table.csv', index_col=[0, 1], header=0)
    is_view = True
    state, _ = env.reset(is_view=is_view)
    state = tuple(state)
    step_counter = 0
    is_terminated = False
    while step_counter < ONE_TURN_STEP and (not is_terminated):
        state_round = round_state(state)
        # 拿到最大Q值的动作
        action = choose_action(state_round, q_table)
        action = np.array([action], dtype='float64')
        next_state, reward = env.step(action)
        if is_final_state(next_state):
            is_terminated = True
            env.close()

        state = next_state
        step_counter += 1

    print(q_table)

if __name__ == '__main__':
    env = self_pendulum_env.PendulumEnv("human")
    # q_table = run_pendulum_qlearning(env, is_view=False)
    # q_table.to_csv('data/result_q_table.csv', index=True, header=True, date_format='%.4f')
    read_q_table(env)
```

打印台结果：

```
Episode 6: total_step = 16896
 wowow
Episode 8: total_step = 6787
 wowow
Episode 9: total_step = 2424
 wowow
Episode 11: total_step = 4476
 wowow
Episode 15: total_step = 8218
 wowow
Episode 19: total_step = 36425
 wowow
Episode 21: total_step = 1098
 wowow
Episode 23: total_step = 3968
 wowow
Episode 24: total_step = 1524
 wowow
Episode 26: total_step = 2150
 wowow
Episode 27: total_step = 2688
 wowow
Episode 29: total_step = 25942
 wowow
Episode 30: total_step = 979
 wowow
Episode 32: total_step = 1066
 wowow
Episode 39: total_step = 21649
 wowow
Episode 40: total_step = 583
 wowow
Episode 42: total_step = 2631
 wowow
Episode 43: total_step = 11910
 wowow
Episode 44: total_step = 3901
 wowow
Episode 45: total_step = 8719
```

```
 wowow
Episode 46: total_step = 9773
 wowow
Episode 47: total_step = 2087
```

```
Episode 62: total_step = 1340
 wowow
Episode 63: total_step = 988
 wowow
Episode 64: total_step = 1502
 wowow
Episode 65: total_step = 954
 wowow
Episode 69: total_step = 4241
 wowow
Episode 70: total_step = 2674
 wowow
Episode 72: total_step = 2071
 wowow
Episode 73: total_step = 1859
 wowow
Episode 74: total_step = 1295
 wowow
Episode 75: total_step = 38130
 wowow
Episode 77: total_step = 1306
 wowow
Episode 78: total_step = 1013
 wowow
Episode 82: total_step = 1064
 wowow
Episode 84: total_step = 1433
 wowow
Episode 86: total_step = 1973
 wowow
Episode 89: total_step = 723
 wowow
Episode 90: total_step = 838
 wowow
Episode 94: total_step = 746
 wowow
Episode 95: total_step = 631
 wowow
Episode 96: total_step = 476
```

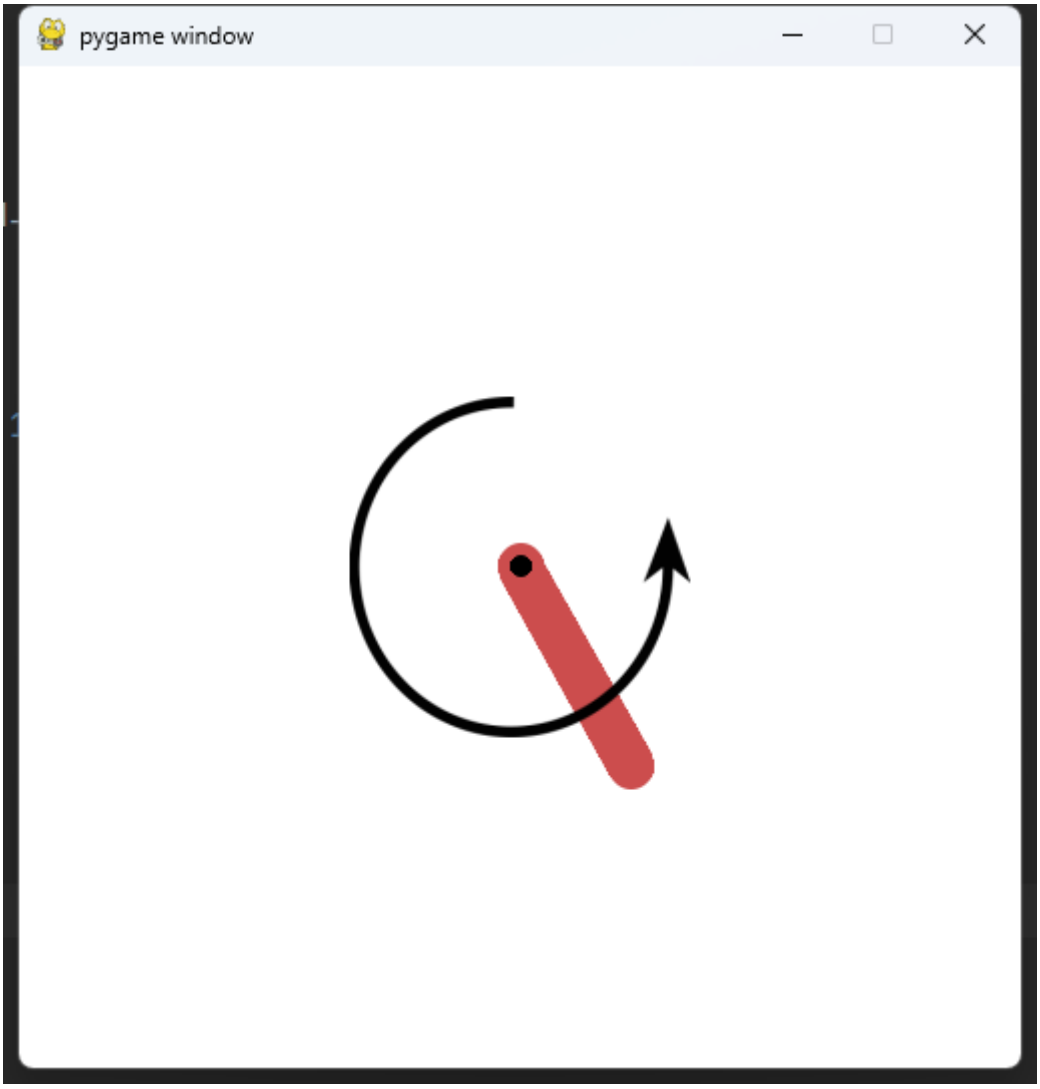
```
 wowow
Episode 98: total_step = 2054
 wowow
Episode 100: total_step = 1315
 wowow
```

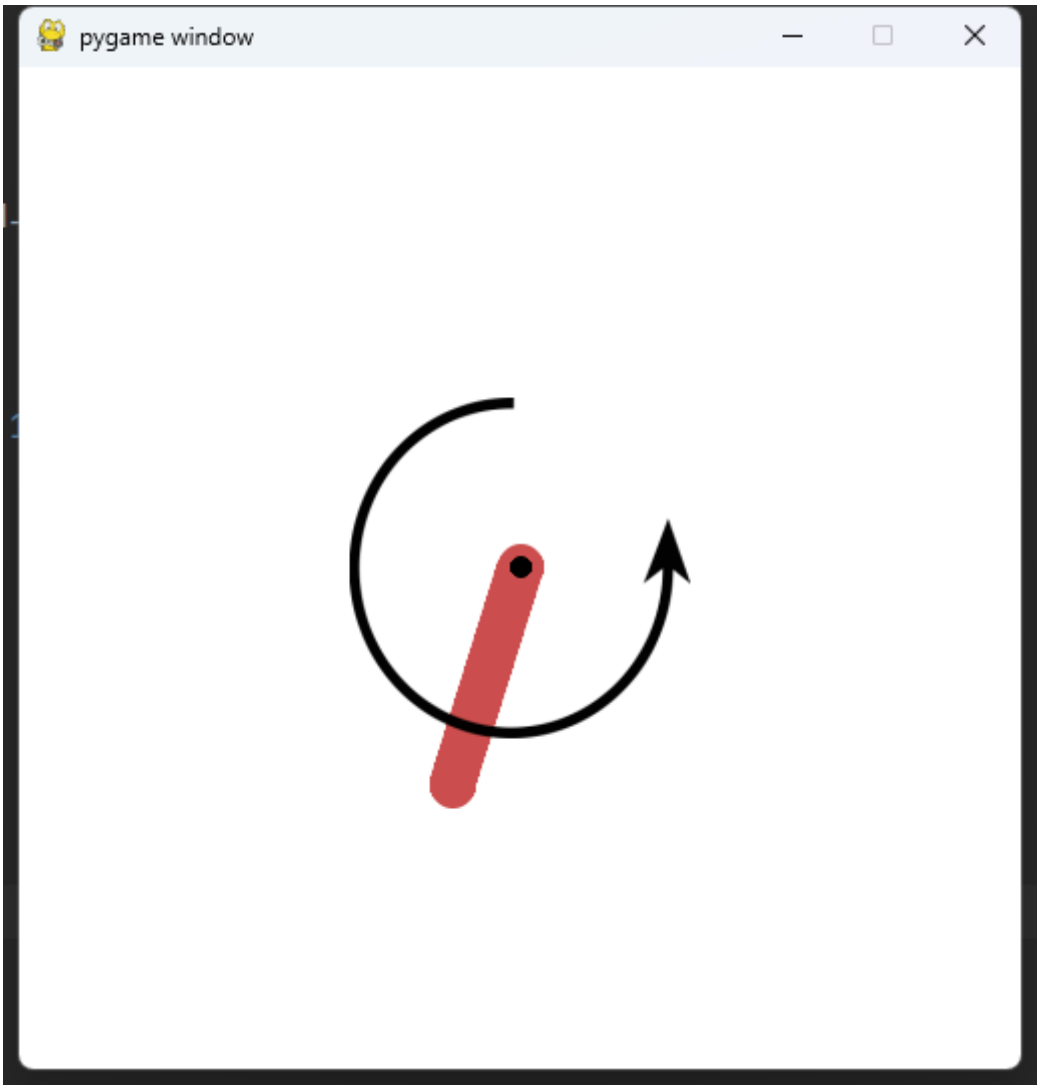
可见一个episode内，agent需要的步数呈整体下降的趋势。

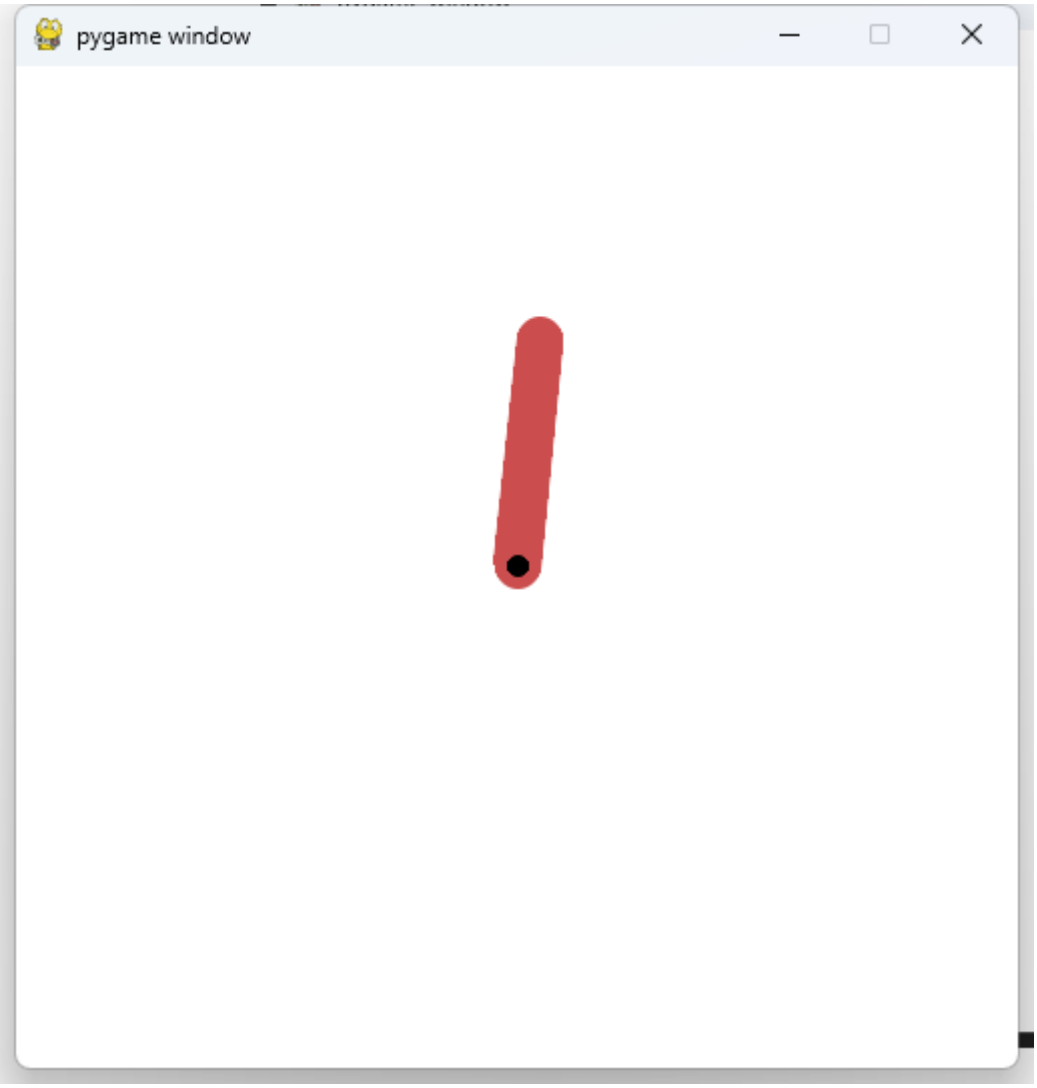
OpenAI仿真结果：

读者可以读取'data/result_q_table.csv'的Q表格，进行渲染

```
if __name__ == '__main__':
    env = self_pendulum_env.PendulumEnv("human")
    # q_table = run_pendulum_qlearning(env, is_view=False)
    # q_table.to_csv('data/result_q_table.csv', index=True, header=True,
date_format='%.4f')
    read_q_table(env)
```




pygame window



分析讨论：

Q-learning是一种基于动态规划的强化学习算法，通常用于解决马尔可夫决策过程 (MDP)问题。

Q-learning的收敛速度和是否能够收敛通常取决于以下几个因素:

1. 学习率 (learning rate) 的选择: 学习率决定了每次代时Q值的更新速度。如果学习率太高，更新幅度会很大，容易导致算法不稳定或震荡。如果学习率太低，则算法收敛速度会很慢。因此，需要根据具体问题选择一个合适的学习率。

2. 探索-利用策略的选择: Q-learning是一种基于贪心的算法，即每次都选择当前状态下Q值最大的动作。如果每次都这样选择，可能会导致算法陷入局部最优解，无法到达全局最优解。因此，需要采用一定的探索策略，例如随机选择动作或按照一定概率选择非最优动作。

3. 状态空间的大小: 状态空间越大，Q-learning需要遍历的状态越多，收敛速度越慢。因此，在状态空间非常大的情况下，需要使用其他方法，如函数逼近.

4. 奖励设计: 奖励是Q-learning的重要组成部分。如果奖励不够合理，可能导致算法无法收敛或收敛速度非常慢。因此，在设计奖励时需要考虑各种因素，例如鼓励探索、奖励正面行为等。

5. 初始Q值的设定: 初始Q值对于Q-learning的收敛速度和稳定性也有影响。如果初始Q值太5高或太低，可能会导致算法在早期阶段出现不稳定或震荡的情况。

对于此次实验：

本人认为有几个原因导致收敛缓慢

1. 离散化取整时，连续的state映射到离散的state时，有精度的丢失
2. Q表状态空间太大，ε-greedy策略需要花很长时间去"寻访"每一个状态-动作队伍，并且使其收敛
3. 奖励设计：对于奖励函数

   控制目标是将摆杆从最低点 $s=[\pi,0]^T$ 摆起并稳定在最高点 $s=[0,0]^T$. 奖励函数定义成如下二次型形式

$$\begin{aligned}\mathcal{R}(s,a)&=-s^TQ_{rew}s-R_{rew}a^2\\Q_{rew}&=\begin{bmatrix}5&0\\0&0.1\end{bmatrix},R_{rew}=1.\end{aligned}\tag{4}$$

   在摆杆在下半圆，即[-pi,-pi/2]U[pi/2,pi]的区域中，电压越小约，R越大，但对于倒立摆模型问题中，在下半圆时，实际情况应该是转动方向和角速度相同时，电压越大约好，这个奖励函数无法直接体现出一点，虽然是可证明最终收敛的，但是会减缓收敛的速度

方法改进：

对于连续空间的强化学习模型，可以采用Deep Q-learning network方法或者函数逼近的方法。

本人用DQN方法继续研究该场景，由于时间不足，暂时还未完成，DQN方式进行倒立摆的强化学习demo如下所示：

```
import torch

# 构建DQN神经网络
class DQN:
    def __init__(self, state_dim=2, action_dim=3, hidden_dim=32, lr=0.001):
        self.model = torch.nn.Sequential(
            torch.nn.Linear(state_dim, hidden_dim),
            torch.nn.ReLU(),
            torch.nn.Linear(hidden_dim, hidden_dim * 3),
            torch.nn.ReLU(),
            torch.nn.Linear(hidden_dim * 3, action_dim)
        )
        self.criterion = torch.nn.MSELoss()
        self.optimizer = torch.optim.Adam(self.model.parameters(), lr=lr)

    def update(self, state, q_target):
        q_predict = self.model(torch.Tensor(state))
        loss = self.criterion(q_predict, torch.Tensor([q_target]))
        self.optimizer.zero_grad()
        loss.backward()  # 进行反向传播
        self.optimizer.step()
```

```
    def predict(self, state):
        with torch.no_grad():
            return self.model(torch.Tensor(state))
```

```
import torch
import numpy as np
from matplotlib import pyplot as plt
import self_pendulum_env

from DQN_model import DQN


# plt.ion()

# 获取DQN模型
def get_model():
    model = DQN(state_dim=2, action_dim=1, hidden_dim=32, lr=0.001)
    return model

# 判断是否到达终止状态
def is_Terminated(state):
    th = state[0]
    thdot = state[1]
    th = np.round(th)
    thdot = np.round(thdot)
    return th == 0 and thdot == 0

# 通过DQN算法进行倒立摆强化学习
def dqn_simple(env, model, episodes, gamma=0.98, epsilon=0.9):
    rewards = []
    for i in range(episodes):
        state, _ = env.reset()
        total_reward = 0
        is_terminated = False

        while not is_terminated:
            random_rate = np.random.uniform()  # 获得一个随机数

            if random_rate > epsilon:
                action = env.action_space.sample()  # ϵ-greedy中探索性选择动作
            else:
                q_values = model.predict(state)  # 贪心策略选择较大概率的动作
                action = [torch.argmax(q_values).item()]
            state_new, reward = env.step(action)
            if is_Terminated(state):
                is_terminated = True

            q_values = reward + gamma *
torch.max(model.predict(state_new)).item()
            model.update(state, q_values)
            state = state_new
```

```
            rewards.append(total_reward)


if __name__ == '__main__':
    env = self_pendulum_env.PendulumEnv(render_mode="human")
    model = get_model()
    dqn_simple(env, model, 50)
```