

第三次报告

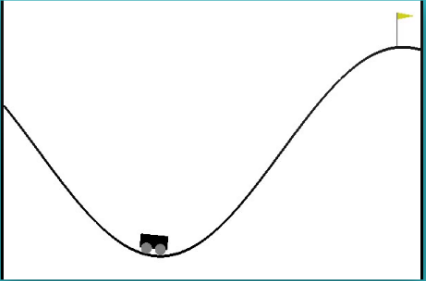
陶云浩 2017201985

MountainCar-v0

A car is on a one-dimensional track, positioned between two "mountains". The goal is to drive up the mountain on the right; however, the car's engine is not strong enough to scale the mountain in a single pass. Therefore, the only way to succeed is to drive back and forth to build up momentum.

This problem was first described by Andrew Moore in his PhD thesis [Moore90].

[Moore90] A Moore, Efficient Memory-Based Learning for Robot Control, PhD thesis, University of Cambridge, 1990.



RandomAgent on MountainCar-v0

问题描述：

动力不足的汽车必须爬上一维小山才能到达目标。

目标位于汽车右侧的山顶上。 如果汽车到达或超出，则剧集终止。

在左侧，还有另一座山。 攀登这座山丘可以用来获得潜在的能量，并朝着目标加速。 在这第二座山顶上，汽车不能超过等于-1 的位置。

输入行为 0 向左，1 不动，2 向右。

奖励：除了爬上山，其余位置均为-1。

一、 Q-learning 解决 MountainCar

与第二次实验中 CartPole 的 Q-learning 解法步骤类似包括：

1. 将状态空间离散化为 Q 表
2. 多次重复游戏，每次基于 Q 表作出选择并对 Q 表进行更新
3. 选择时使用 e-greedy，防止每次都作出相同选择，刚开始探索时更多的选择随机的行动，随着探索的进行，Q 表逐渐完善，开始进行贪心选择
4. 更新时按照一定的学习率对 Q 表的相应位置的值进行更新，公式如下

$$Q[s, a] += \text{learning_rate} * (\text{td_target} - Q[s, a])$$

$$\text{td_target} = R[t + 1] + \text{discount_factor} * \max(Q[s'])$$

具体实现的各函数代码如下：

- 建立 Q 表

```
DISCRETE_OS_SIZE = [Q_TABLE_LEN] * len(env.observation_space.high)

discrete_os_win_size = (env.observation_space.high -
env.observation_space.low) / DISCRETE_OS_SIZE

q_table = np.random.uniform(low=0, high=1, size=(DISCRETE_OS_SIZE +
[env.action_space.n]))
```

- 将状态空间离散化

```
def get_discrete_state (state):

    discrete_state = (state - env.observation_space.low)

    return tuple(discrete_state.astype(int))
```

- e-greedy 选择 action

```
def take_epsilon_greedy_action(state, epsilon):

    discrete_state = get_discrete_state(state)

    if np.random.random() < epsilon:

        action = np.random.randint(0, env.action_space.n)

    else:

        action = np.argmax(q_table[discrete_state])

    return action
```

- 训练模型，完善 Q 表

```
for episode in range(EPISODES):

    # initiate reward every episode

    ep_reward = 0

    state = env.reset()

    done = False
```

```

while not done:

    action = take_epsilon_greedy_action(state, epsilon)

    next_state, reward, done, _ = env.step(action)

    ep_reward += reward

    if not done:

        td_target = reward + DISCOUNT *
np.max(q_table[get_discrete_state(next_state)])

        q_table[get_discrete_state(state)][action] +=
LEARNING_RATE * (td_target -
q_table[get_discrete_state(state)][action])

    elif next_state[0] >= 0.5:

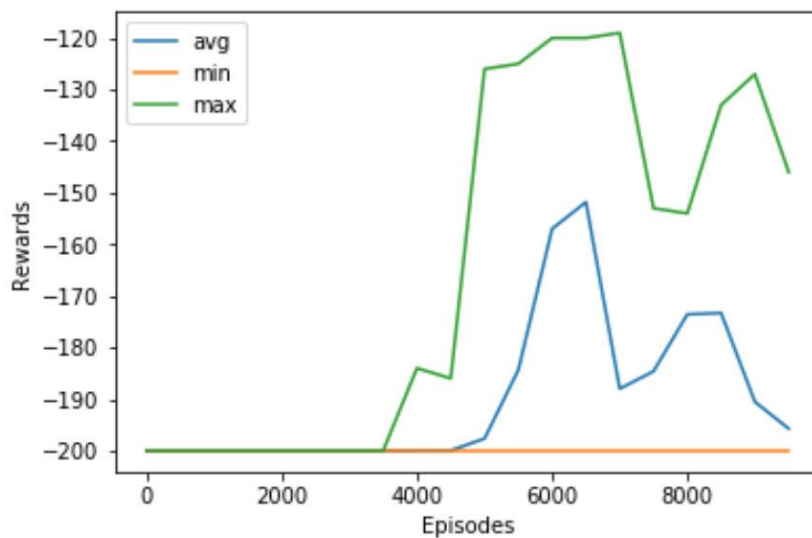
        # print("I made it on episode: {} Reward:
{}".format(episode, reward))

        q_table[get_discrete_state(state)][action] = 0

    state = next_state

```

对上述代码进行训练，训练次数为 10000 次，将每 500 次的平均奖励，最大奖励，最小奖励结果画出来如下：



结果分析：

训练 3000 多次以后开始能够爬上山顶，并且平均需要的步数变少，训练次数到达 6000 以后继续训练，模型效果不再增加，反而有下降趋势。

原因分析：

离散化程度过高，为了节约成本而降低了状态空间数，反而导致结果不够精确，到达瓶颈之后再进行调整反而降低了模型的精确度。

学习率过高，虽然能尽快到达山顶，但到达山顶之后如果想进一步减少操作步数，提高得分的话，过高的学习率反而会导致模型波动性较大，从而造成 6000 次以后的训练得分降低。

实验总结：

本学期主要研究了 Cartpole 和 MountainCar 两类问题模型，使用了随机算法，爬山算法，Q-learning 算法对两类问题进行探究。深入理解了强化学习算法的实现过程以及相关类型的问题求解方法，对机器学习、深度学习算法有了初步的了解，也培养了对人工智能的兴趣。

不足之处：由于精力有限，研究范围较小，涉及的算法不够全面，很多内容还停留在表面。希望以后能更多接触人工智能相关的算法学习和工程开发，对这个领域有更深入的理解。