# 人工智能导论第二次实验报告

**Machine-Learning Agent of MountainCar in OpenAI Gym**

## 1. MountainCar 环境介绍

MountainCar 属于经典控制问题，目标是在尽可能少的步数内把动力不足的车开到山顶（0.5 位置）。起始在-0.6 到-0.4 的随机位置，速度为 0，当到达目标位置或进行了 200 次时，中止操作。游戏中可以根据观测到的车的位置和速度信息，给出行为决策。每进行一步奖励-1，直到达到中止状态。

观测值：位置和速度

| Observation | Min | Max |
|---|---|---|
| position | -1.2 | 0.6 |
| velocity | -0.07 | 0.07 |

行为：三个离散值

| Num | Action |
|---|---|
| 0 | push left |
| 1 | no push |
| 2 | push right |

## 2. 具体实现

### （1） $Q\text{-}Learning$

$Q\text{-}Learning$算法的关键在于如何建立$Q$表，来指导智能体的行动，$Q$表对应$Action$的数值越大，智能体越大概率采取这个$Action$。这里采用$\varepsilon$贪婪方法进行探索-利用困境来更新$Q$表。

**• 算法**

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
    Initialize $s$
    Repeat (for each step of episode):
        Choose $a$ from $s$ using policy derived from $Q$ ($\varepsilon$-greedy)
        Take action $a$, observe $r, s'$
        $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma\, max_a, Q(s',a') - Q(s,a)]$
        $s \leftarrow s'$;
    until $s$ is terminal

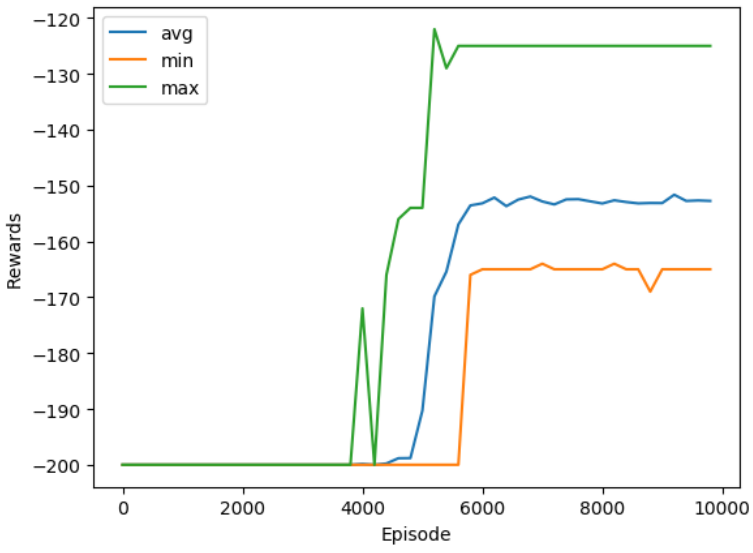
**• 核心训练代码**

```python
#train
for episode in range(EPISODES):
    ep_reward=0
    if episode%SHOW_EVERY==0:
        render=True
    else:
        render=False
    state=env.reset()
    done=False
    while not done:
        action=take_epilon_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:
            q_table[get_discrete_state(state)][action]=0
        state=next_state
```

- 运行截图

```
Episode: 7800 Reward: -152.0
Episode: 8000 Reward: -154.0
Episode: 8200 Reward: -155.0
Episode: 8400 Reward: -153.0
Episode: 8600 Reward: -156.0
Episode: 8800 Reward: -128.0
Episode: 9000 Reward: -155.0
Episode: 9200 Reward: -154.0
Episode: 9400 Reward: -162.0
Episode: 9600 Reward: -155.0
Episode: 9800 Reward: -153.0
```

- 训练结果



### （2） $SARSA$

$SARSA$是当前$S$（状态）$A$（行动）$R$（奖励）与下一步$S'$（状态）$A'$（行动）的组合，是$On\text{-}Policy$算法，自始至终只有一个$Policy$。该算法除了目标值与$Q\text{-}Learning$不同，其余相同。

- 算法

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize $s$
  Repeat (for each step of episode):
    Choose $a$ from $s$ using policy derived from $Q$ ($\varepsilon$-greedy)
    Take action $a$, observe $r$, $s'$
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until $s$ is terminal

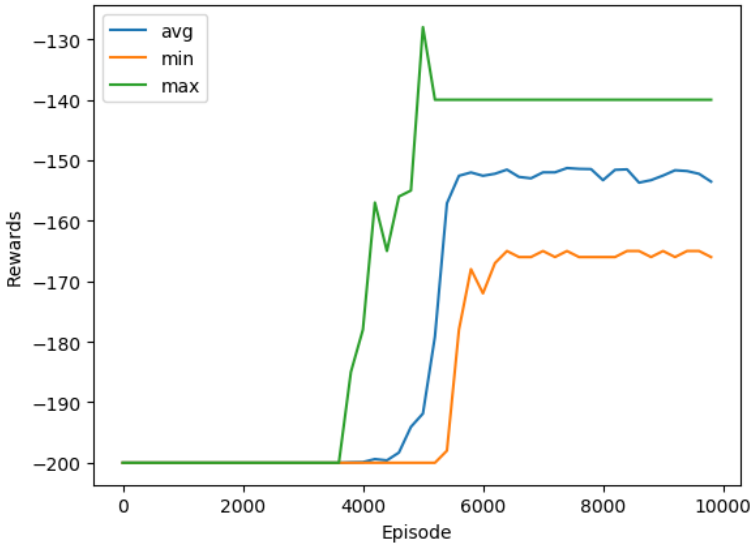
- 核心训练代码

```
#train
for episode in range(EPISODES):
    ep_reward=0
    if episode%SHOW_EVERY==0:
        render=True
    else:
        render=False
    state=env.reset()
    action=take_epilon_greedy_action(state,epsilon)
    done=False
    while not done:
        next_state,reward,done,_=env.step(action)
        ep_reward+=reward
        next_action=take_epilon_greedy_action(next_state,epsilon)
        if not done:
            td_target=reward+DISCOUNT*q_table[get_discrete_state(next_state)][next_action]
            q_table[get_discrete_state(state)][action]+=\
                LEARNING_RATE*(td_target-q_table[get_discrete_state(state)][action])
        elif next_state[0]>=0.5:
            q_table[get_discrete_state(state)][action]=0
        state=next_state
        action=next_action
```

- 运行截图

```
Episode: 7800 Reward: -146.0
Episode: 8000 Reward: -156.0
Episode: 8200 Reward: -144.0
Episode: 8400 Reward: -143.0
Episode: 8600 Reward: -157.0
Episode: 8800 Reward: -157.0
Episode: 9000 Reward: -164.0
Episode: 9200 Reward: -141.0
Episode: 9400 Reward: -159.0
Episode: 9600 Reward: -156.0
Episode: 9800 Reward: -166.0
```

• 训练结果




**（3） *SARSA(lambda)***

该算法引入了衰减系数$\lambda$和$Eligibility\ trace$表（$E$表）。每走一步，更新整个$Q$表和$E$表。

• 算法

Initialize $Q(s,a)$ arbitrarily, for all $s \in S,\ a \in A(s)$
Repeat (for each episode):
$E(s,a) = 0$, for all $s \in S,\ a \in A(s)$
Initialize $S,\ A$
Repeat (for each step of episode):
Take action $A$, observe $R,\ S'$
Choose $A'$ from $S'$ using policy derived from $Q$ ($\varepsilon$-greedy)
$\delta \leftarrow \mathrm{R} + \gamma \mathrm{Q}(\mathrm{S}', \mathrm{A}') - \mathrm{Q}(\mathrm{S}, \mathrm{A})$
$\mathrm{E}(\mathrm{S}, \mathrm{A}) \leftarrow \mathrm{E}(\mathrm{S}, \mathrm{A}) + 1$
For all $s \in S,\ a \in A(s)$:
$Q(s,a) \leftarrow Q(s,a) + \alpha\delta E(s,a)$
$\mathrm{E}(\mathrm{s}, \mathrm{a}) \leftarrow \gamma\lambda \mathrm{E}(\mathrm{s}, \mathrm{a})$
$S \leftarrow S'$; $A \leftarrow A'$;
until $S$ is terminal

• 核心训练代码

```python
#train
for episode in range(EPISODES):
    ep_reward=0
    if episode%SHOW_EVERY==0:
        render=True
    else:
        render=False
    state=env.reset()
    action=take_epilon_greedy_action(state,epsilon)
    e_trace=np.zeros(DISCRETE_OS_SIZE+[env.action_space.n])
    done=False
    while not done:
        next_state,reward,done,_=env.step(action)
        ep_reward+=reward
        next_action=take_epilon_greedy_action(next_state,epsilon)
        if not done:
            delta=reward+DISCOUNT*q_table[get_discrete_state(next_state)][next_action]\
                -q_table[get_discrete_state(state)][action]
            e_trace[get_discrete_state(state)][action]+=1
            q_table+=LEARNING_RATE*delta*e_trace
            e_trace=DISCOUNT*LAMBDA*e_trace
        elif next_state[0]>=0.5:
            q_table[get_discrete_state(state)][action]=0
        state=next_state
        action=next_action
```

- 运行截图

```
Episode: 7800 Reward: -159.0
Episode: 8000 Reward: -156.0
Episode: 8200 Reward: -147.0
Episode: 8400 Reward: -200.0
Episode: 8600 Reward: -200.0
Episode: 8800 Reward: -200.0
Episode: 9000 Reward: -200.0
Episode: 9200 Reward: -154.0
Episode: 9400 Reward: -154.0
Episode: 9600 Reward: -200.0
Episode: 9800 Reward: -150.0
```

- 训练结果