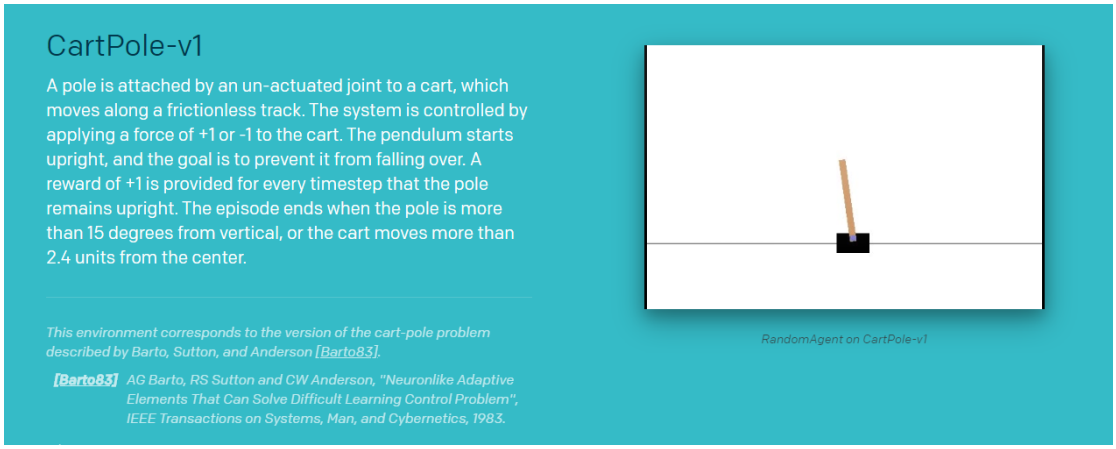
第二次报告

陶云浩 2017201985



问题描述：

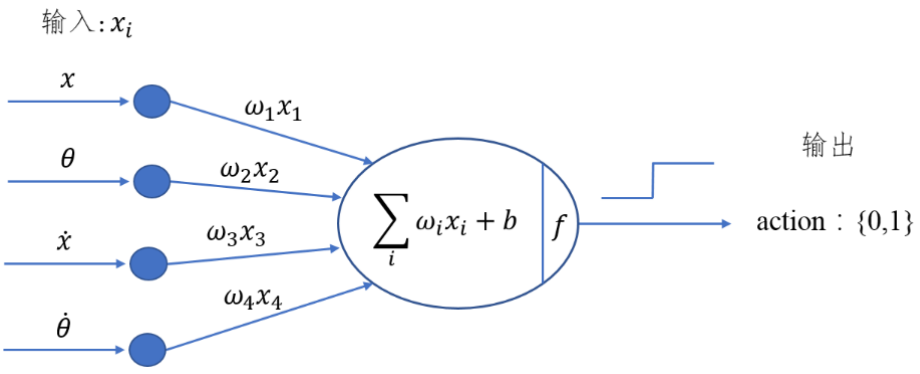
游戏里面有一个小车，上有竖着一根杆子。小车需要左右移动来保持杆子竖直。如果杆子倾斜的角度大于 15°，那么游戏结束。小车也不能移动出一个范围（中间到两边各 2.4 个单位长度）。每个时间点输入 0 表示左移，1 表示右移，返回一个状态，包括小车位置，速度，杆倾斜角，杆转动角速度。保持使杆不倾斜超过 15°，坚持步数即为最终得分。

一、 爬山算法解决 Cartpole 问题

1. 算法描述

为了能够有效控制倒立摆首先应建立一个控制模型。这个控制模型的输入是当前倒立摆的状态（observation），输出为对当前状态做出的决策动作（action）。决定倒立摆状态的 observation 是一个四维向量，包含小车位置、杆子夹角、小车速度和杆子角速度，如果对这个向量求它的加权和，那么就可以根据加权和值的符号来决定采取的动作（action）。

用 sigmoid 函数将这个问题转化为二分类问题，从而可以建立一个简单的控制模型。其模型如下图所示：



通过学习得出一组较优的权值，对当前状态的信息作计算，得出结果。结果的符号为正判定输出为 1，否则为 0。

为了得到一组较好的权值从而有效控制倒立摆，采用爬山算法（hill climbing algorithm）进行学习优化。爬山算法是一种启发式方法，是对深度优先搜索的一种改进，它利用反馈信息帮助生成解的决策。

爬山算法的基本思路是每次迭代时给当前取得的最优权重加上一组随机值，如果加上这组值使得有效控制倒立摆的持续时间变长了那么就更新它为最优权重，如果没有得到改善就保持原来的值不变，直到迭代结束。在迭代过程中，模型的参数不断得到优化，最终得到一组最优的权值作为控制模型的解。

2. 代码及说明

Editor - C:\Users\14744\Desktop\人工智能导论\hill_climbing.py

trans.py | test.py | hill_climbing.py*

```python
import numpy as np
import gym

def get_action(weights, observation):# 根据权值对当前状态做出决策
    wxb = np.dot(weights[:4], observation) + weights[4] # 计算加权和
    if wxb >= 0:# 加权和大于0时选取动作1，否则选取0
        return 1
    else:
        return 0

def get_sum_reward_by_weights(env, weights):
# 测试不同权值的控制模型有效控制的持续时间（或奖励）
    observation = env.reset() # 重置初始状态
    sum_reward = 0 # 记录总的奖励
    for t in range(1000):
        action = get_action(weights, observation) # 获取当前权值下的决策动作
        observation, reward, done, info = env.step(action)# 执行动作并获取这一动作下的下一时间步长状态
        sum_reward += reward
        # print(sum_reward, action, observation, reward, done, info)
        if done:
            break
    return sum_reward
def get_weights_by_random_guess():
    return np.random.rand(5)

def get_weights_by_hill_climbing(best_weights):

    return best_weights + np.random.normal(0, 0.1, 5)

def get_best_result(algo="hill_climbing"):
    env = gym.make("CartPole-v0")
    np.random.seed(10)
    best_reward = 0
    best_weights = np.random.rand(5)

    for iter in range(10000):# 迭代10000次
        cur_weights = None

        if algo == "hill_climbing": # 选取动作决策的算法
            cur_weights = get_weights_by_hill_climbing(best_weights)
        else:
            cur_weights = get_weights_by_random_guess()

        cur_sum_reward = get_sum_reward_by_weights(env, cur_weights)
        print(cur_sum_reward, cur_weights)
        if cur_sum_reward > best_reward:
            best_reward = cur_sum_reward
            best_weights = cur_weights
        if best_reward >= 200:
            break

    return best_reward, best_weights

print(get_best_result("hill_climbing")) # 调用爬山算法寻优并输出结果

```

在 get_best_result 函数中，迭代 10000 次寻找最优权值，每次利用 get_weights_by_hill_climbing 函数对当前权值进行随机调整，基于调整后的参数进行游戏，若得分更高则更换最优权值。

3. 测试结果

```
IPython console
  Console 1/A
(200.0, array([ 0.3633354 ,  0.02279047,  0.59487525,  1.74566816, -0.02239189]))

In [10]: runfile('C:/Users/14744/Desktop/人工智能导论/hill_climbing.py', wdir='C:/Users/14744/Desktop/人工智能导论')
18.0 [ 0.70488267 -0.03980354  0.56854915  0.80006066  0.52474589]
18.0 [0.64660951 0.09703865 0.57388601 0.94911802 0.60038259]
21.0 [ 0.75184676 -0.0631998   0.60391494  0.97843075  0.48971577]
38.0 [ 0.6528818  -0.12662522  0.79379116  1.17808303  0.37784862]
33.0 [ 0.87091464 -0.10757853  0.91995975  1.219849    0.46724439]
39.0 [ 0.65761732 -0.18746777  0.76693635  1.11145283  0.34344744]
47.0 [0.73832251 0.02671277 0.71737453 1.16401658 0.30448279]
33.0 [ 0.83205216 -0.05498465  0.81933608  1.12987294  0.37961093]
32.0 [ 0.70209733 -0.08620036  0.68240348  1.05874392  0.43271046]
24.0 [0.72114198 0.08447857 0.56987704 1.17189542 0.39632416]
111.0 [0.68905278 0.036359   0.65209854 1.14184033 0.19765413]
31.0 [ 0.56628357 -0.1434983   0.5835778   1.00085345  0.32289399]
27.0 [0.73600905 0.06905117 0.83295286 1.00053353 0.44998464]
100.0 [ 0.72378456 -0.05237695  0.54613681  1.17663548  0.04178769]
80.0 [0.58954867 0.29847425 0.6180099  1.12563092 0.24095008]
51.0 [ 0.68860343 -0.15011314  0.59132595  1.40375285  0.26401343]
124.0 [0.83821416 0.0175945  0.7157488  1.30359115 0.18745667]
117.0 [ 0.80660608 -0.16005032  0.77180058  1.50860696  0.20203406]
98.0 [ 0.76326514 -0.01341957  0.59178074  1.24424326  0.14363938]
92.0 [ 0.86635744 -0.00302746  0.88269398  1.19590263  0.21748395]
77.0 [ 0.88851654 -0.09196341  0.56363469  1.32720134  0.25717147]
106.0 [0.81863403 0.02958365 0.79214833 1.27032576 0.18975172]
110.0 [0.84375755 0.0538703  0.7454339  1.44428654 0.20954489]
110.0 [ 0.9372056  -0.05954552  0.76158724  1.47995368  0.12237541]
93.0 [ 0.82394561 -0.07581597  0.71840544  1.17651989  0.2170773 ]
94.0 [ 0.97469083 -0.11283308  0.7543491   1.12769264  0.13240076]
96.0 [ 1.00207672 -0.01378754  0.50705721  1.40766847  0.22049926]
105.0 [0.80579031 0.15269005 0.76308079 1.33565082 0.29576189]
200.0 [ 0.89193983  0.14806433  0.69917038  1.1741322  -0.08840431]
(200.0, array([ 0.89193983,  0.14806433,  0.69917038,  1.1741322 , -0.08840431]))

In [11]:
```

由上述测试结果可以看出，刚开始未对权值进行调整时，得分在 20-40 之间。随着学习的进行和参数的优化，得分不断升高并且很快达到 200 并停止。

若不对算法设置终止条件，即当得分达到 200 时不停止，继续迭代，直到 10000 次全部执行结束，则接近末尾时的测试结果如下：

```
Variable explorer   File explorer   Help
IPython console
  Console 1/A
102.0 [1.21440521 0.15481976 0.89202793 1.70183656 0.1720019 ]
133.0 [1.03003586 0.28839845 0.84086503 1.78362169 0.00721174]
89.0 [0.9793122  0.39015933 0.74035647 1.85825752 0.12949286]
105.0 [1.06985692 0.12025197 0.78232682 1.8492975  0.02649895]
107.0 [0.93576435 0.02457144 0.84414842 1.87343247 0.25000049]
108.0 [0.97520077 0.15875754 0.74768226 1.90807754 0.21689649]
112.0 [0.84810552 0.36052804 0.92124516 1.88260399 0.18556163]
147.0 [ 0.88513179  0.16011858  0.77769376  1.73121595 -0.07920259]
143.0 [0.8649715  0.31965205 0.87983355 1.93078289 0.03416549]
138.0 [ 1.16061386  0.23853349  0.91447212  1.83204508 -0.03850711]
104.0 [1.23549333 0.14933411 0.76926886 1.79596973 0.01690031]
120.0 [0.97023538 0.21111135 0.68741604 1.97903307 0.15262864]
141.0 [ 0.9739595   0.28145282  0.93288143  1.96449008 -0.13594936]
126.0 [1.04195923 0.10249273 0.76450113 1.74440201 0.06455532]
188.0 [1.0871781  0.20289522 1.01449208 1.73644223 0.0076513 ]
200.0 [0.99476012 0.26509652 0.90554581 1.68941893 0.08813526]
200.0 [0.97168177 0.12436771 0.86676376 1.7631864  0.08344292]
105.0 [ 0.92209823  0.17997712  0.82215339  1.84266766 -0.13786351]
87.0 [1.20050282 0.15749784 0.609947   1.93543672 0.10498967]
134.0 [1.13124354 0.30499516 0.8838459  1.84793397 0.05613228]
136.0 [0.97820528 0.33009843 0.67105188 1.92695749 0.01106597]
200.0 [ 1.18276092  0.33840927  0.97042195  1.82830397 -0.04760185]
101.0 [1.0722616  0.2050575  0.6813697  1.99565484 0.05712635]
136.0 [1.02226444 0.34808756 0.67735502 1.79531089 0.0062778 ]
127.0 [ 0.94004494  0.05628069  0.89639091  1.88739432 -0.05583175]
142.0 [0.92463679 0.02355327 0.67188796 1.83805544 0.1161289 ]
107.0 [1.08562063 0.13079735 0.7058608  1.91026694 0.09108611]
102.0 [1.00304593 0.16662719 0.67083334 1.80378389 0.00791652]
194.0 [1.09599038 0.14846868 0.70420955 1.90939002 0.04634706]
93.0 [0.97468444 0.14904311 0.935417   1.80418477 0.17020428]
111.0 [1.03383894 0.11330611 0.70789259 1.98880756 0.15255995]
111.0 [1.12086724 0.20189232 0.97298511 1.90107146 0.21817831]
99.0 [1.135993   0.19053014 0.6429603  1.86871609 0.15548392]
99.0 [0.95378676 0.10370387 0.76526004 1.72477458 0.08810255]
```

4. 结果分析

使用爬山算法得到的权值可以使得分基本稳定在 100-150 的范围内，相较于随机算法，性能有很大的提升。

但爬山算法有几个缺陷，由于权值是随机得到且并不是对每一个操作进行特定的选择，其最终的策略不可能达到全局最优，最多使局部性能得到优化。

另外，action 的选择和 observation 中的参数的关系不可能是简单的线性关系，若两者关系较复杂或存在极端的影响时，可以想见，爬山算法很难得到最优解，甚至不能对解进行优化。

二、 Q-learning 解决 Cartpole 问题

1. 算法描述

Q 矩阵定义：

CartPole 状态是保存在 observation 中的，有 4 个变量，cart 位置和速度，pole 的角度和速度，它们都是连续值，现在要把这些连续值离散化

cart 位置：-2.4 ~ 2.4

cart 速度：-inf ~ inf

pole 角度：-0.5 ~ 0.5 （radian）

pole 角速度：-inf ~ inf

离散化为 6 个 bins，4 个变量，所以有 1296 种 state，action 只有左和右 2 种

因此 Q 矩阵 size 是 1296 * 2

定义 3 个 class：Agent，Brain，Environment，它们的关系如下：

(1) Agent 把当前的 observation 传给 Brain

(2) Brain 把 observation 的状态离散化后，根据 Q 矩阵决定 action，然后把 action 给 Agent

(3) Agent 把 action 用到 Environment，Environment 把 action 反馈的 observation_t+1 和 reward_t+1 返回给 Agent

(4) Agent 把当前 observation_t，action_t，行动后的 observation_t+1，reward_t+1 给 Brain

(5) Brain 更新 Q 矩阵

2. 代码

- 参数设置：

```
import numpy as np
import gym

ENV = 'CartPole-v1'
NUM_DIGITIZED = 6
GAMMA = 0.99
ETA = 0.5
MAX_STEPS = 200
NUM_EPISODES = 500
```

- Agent:

```
class Agent:
    def __init__(self, num_states, num_actions):
        self.brain = Brain(num_states, num_actions)
    def update_Q_function(self, observation, action, reward, observation_next):
        self.brain.update_Q_table(
            observation, action, reward, observation_next)
    def get_action(self, observation, step):
        action = self.brain.decide_action(observation, step)
        return action
```

- Brain

```
class Brain:

    def  __init__(self, num_states, num_actions):
        self.num_actions = num_actions

        self.q_table = np.random.uniform(low=0, high=1, size=(NUM_DIGITIZED**num_states, num_actions))

    def bins(self, clip_min, clip_max, num):
        return np.linspace(clip_min, clip_max, num + 1)[1: -1]

    def digitize_state(self, observation):
        cart_pos, cart_v, pole_angle, pole_v = observation

        digitized = [
            np.digitize(cart_pos, bins = self.bins(-2.4, 2.4, NUM_DIGITIZED)),
            np.digitize(cart_v, bins=self.bins(-3.0, 3.0, NUM_DIGITIZED)),
            np.digitize(pole_angle, bins=self.bins(-0.5, 0.5, NUM_DIGITIZED)),
            np.digitize(pole_v, bins=self.bins(-2.0, 2.0, NUM_DIGITIZED))
        ]

        return sum([x* (NUM_DIGITIZED**i) for i, x in enumerate(digitized)])

    def update_Q_table(self, observation, action, reward, observation_next):
        state = self.digitize_state(observation)
        state_next = self.digitize_state(observation_next)
        Max_Q_next = max(self.q_table[state_next][:])
        self.q_table[state, action] = self.q_table[state, action] + \
            ETA * (reward + GAMMA * Max_Q_next - self.q_table[state, action])

    def decide_action(self, observation, episode):
        state = self.digitize_state(observation)
        epsilon = 0.5 * (1 / (episode + 1))

        if epsilon <= np.random.uniform(0, 1):
            action = np.argmax(self.q_table[state][:])
        else:
            action = np.random.choice(self.num_actions)

        return action
```

- Environment

```python
class Environment:

    def __init__(self):
        self.env = gym.make(ENV)
        num_states = self.env.observation_space.shape[0]
        num_actions = self.env.action_space.n
        self.agent = Agent(num_states, num_actions)

    def run(self):
        complete_episodes = 0
        is_episode_final = False
        for episode in range(NUM_EPISODES):
            observation = self.env.reset()
            for step in range(MAX_STEPS):
                action = self.agent.get_action(observation, episode)
                observation_next, _, done, _ = self.env.step(action)

                if done:
                    if step < 195:
                        reward = -1
                        complete_episodes = 0
                    else:
                        reward = 1
                        complete_episodes += 1
                else:
                    reward = 0
                self.agent.update_Q_function(observation, action, reward, observation_next)
                observation = observation_next
                if done:
                    print('{0} Episode: Finished after {1} time steps'.format(episode, step + 1))
                    break

            if is_episode_final is True:
                break

            if complete_episodes >= 10:
                print('succeeded for 10 times')
                is_episode_final = True
```

3. **测试**

运行以下代码：

```python
test = Environment()
test.run()
```

结果如下：

```
In [1]: runfile('C:/Users/14744/Desktop/人工智能导论/Q-learning.py', wdir='C:/Users/14744/Desktop/人工智能导论')
0 Episode: Finished after 38 time steps
1 Episode: Finished after 27 time steps
2 Episode: Finished after 15 time steps
3 Episode: Finished after 15 time steps
4 Episode: Finished after 17 time steps
5 Episode: Finished after 15 time steps
6 Episode: Finished after 23 time steps
7 Episode: Finished after 31 time steps
8 Episode: Finished after 14 time steps
9 Episode: Finished after 52 time steps
10 Episode: Finished after 26 time steps
11 Episode: Finished after 15 time steps
12 Episode: Finished after 17 time steps
13 Episode: Finished after 20 time steps
14 Episode: Finished after 26 time steps
15 Episode: Finished after 16 time steps
16 Episode: Finished after 77 time steps
17 Episode: Finished after 18 time steps
18 Episode: Finished after 47 time steps
19 Episode: Finished after 11 time steps
```

```
90 Episode: Finished after 179 time steps
91 Episode: Finished after 70 time steps
94 Episode: Finished after 167 time steps
95 Episode: Finished after 101 time steps
96 Episode: Finished after 153 time steps
97 Episode: Finished after 133 time steps
98 Episode: Finished after 158 time steps
99 Episode: Finished after 140 time steps
100 Episode: Finished after 165 time steps
101 Episode: Finished after 154 time steps
102 Episode: Finished after 149 time steps
103 Episode: Finished after 138 time steps
116 Episode: Finished after 23 time steps
117 Episode: Finished after 97 time steps
118 Episode: Finished after 45 time steps
119 Episode: Finished after 61 time steps
120 Episode: Finished after 54 time steps
121 Episode: Finished after 64 time steps
122 Episode: Finished after 115 time steps
123 Episode: Finished after 90 time steps
124 Episode: Finished after 165 time steps
125 Episode: Finished after 148 time steps
126 Episode: Finished after 66 time steps
127 Episode: Finished after 143 time steps
128 Episode: Finished after 182 time steps
129 Episode: Finished after 133 time steps
130 Episode: Finished after 127 time steps
133 Episode: Finished after 167 time steps
134 Episode: Finished after 185 time steps
```

```
464 Episode: Finished after 147 time steps
465 Episode: Finished after 165 time steps
467 Episode: Finished after 131 time steps
468 Episode: Finished after 178 time steps
469 Episode: Finished after 122 time steps
470 Episode: Finished after 198 time steps
471 Episode: Finished after 158 time steps
472 Episode: Finished after 154 time steps
475 Episode: Finished after 167 time steps
476 Episode: Finished after 198 time steps
478 Episode: Finished after 199 time steps
479 Episode: Finished after 170 time steps
482 Episode: Finished after 172 time steps
483 Episode: Finished after 143 time steps
484 Episode: Finished after 164 time steps
485 Episode: Finished after 164 time steps
486 Episode: Finished after 179 time steps
487 Episode: Finished after 148 time steps
488 Episode: Finished after 181 time steps
489 Episode: Finished after 160 time steps
491 Episode: Finished after 176 time steps
492 Episode: Finished after 157 time steps
493 Episode: Finished after 182 time steps
496 Episode: Finished after 180 time steps
497 Episode: Finished after 165 time steps
498 Episode: Finished after 142 time steps

In [2]:
```

4. 分析总结

由测试结果可知，前 20 轮与随机算法的结果大致相同，随着 Q 表的完善和优

化，得分不断提高，到最后基本能维持在 150-200 步。
可见 Q-learning 算法比爬山算法的效率更高，能使得分接近满分。
但 Q-learning 算法也存在缺陷，即当状态不离散时，需要将其离散化，从而导致结果产生偏差。而为了减少这一偏差，就要增加状态个数。随着状态个数的增加，Q 表的大小也大幅增加，对空间和时间的要求都相应增大。