人工智能导论实验报告

2017202124

一、 Rule-based Agent of MountainCar in OpenAI Gym

1. OpenAI Gym

（1） 安装：pip install gym

（2） OpenAI Gym 中的智能体可以通过三种基本方法与环境交互。重置 reset：重置环境并返回观测值；执行 step：在环境中执行一个时间步长，并返回观测值 observation、奖励 reward、状态 done 和信息 info；回馈 render：回馈环境的一个帧，比如弹出交互窗口。

（3） 测试：使用 CartPole-v0 的例子进行测试。

```
import gym
env = gym.make('CartPole-v0')
for i_episode in range(20):
    observation = env.reset()
    for step in range(100):
        env.render()
        print(observation)
        action = env.action_space.sample()
        observation, reward, done, info = env.step(action)
        if done:
            print("Episode finished after {} timesteps".format(step+1))
            break
```

```
[-0.02216001  0.77821409 -0.05124532 -1.22084329]
[-0.00659573  0.58378863 -0.07566219 -0.94464748]
[ 0.00508005  0.77984377 -0.09455513 -1.26011228]
[ 0.02067692  0.58604995 -0.11975738 -0.99847787]
[ 0.03239792  0.7825516  -0.13972694 -1.32624379]
[ 0.04804895  0.58944264 -0.16625181 -1.08034957]
[ 0.05983781  0.39685876 -0.1878588  -0.84410996]
[ 0.06777498  0.20472854 -0.204741   -0.61589124]
Episode finished after 26 timesteps
```

2. 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

3. 算法实现

（1） 随机模式

与 CartPole 的测试代码基本相同。由于获取行为的随机性，影响相互抵消，导致小车一直在低处徘徊，不能到达山顶。

```
#random mode
def test(self,steps):
    env.reset()
    total_reward=0
    for s in range(steps):
        env.render()
        action=env.action_space.sample()#get action randomly
        _,reward,done,_=env.step(action)
        total_reward+=reward
        if done:
            break
    return total_reward
```

（2） 特定规则模式

对于这个实验，最关键的地方在于 action 的选择策略。这里的策略是：当小车向左行驶时，若速度不大且位置不高，需要向左推动，这样可以增加速度；当小车即将到达右边山顶但速度很小时，不推动，因为向右推动也可能无法到达山顶反而会反向到左边使得步数增多；一般来说小车向右行驶时都需要向右推动；其余情况不推动。

```
#specific rule mode
def step(self,steps):
    observation=env.reset()
    total_reward=0
    for s in range(steps):
        env.render()#render the env every step
        action=self.get_action(observation)#get action for car
        #0 push left, 1 no push, 2 push right
        observation,reward,done,_=env.step(action)
        total_reward+=reward
        if done:
            print("Episode finished after {} timesteps. Total reward:{}"
                  .format(s+1,total_reward))
            break
    return total_reward
```

```python
def get_action(self,observation):
    pos=observation[0] #position
    vel=observation[1] #velocity
    if pos<self.goal_position/4 and -self.max_speed/2<vel<0:
        return 0
    elif pos>self.goal_position-0.02 and 0<vel<self.max_speed/10:
        return 1
    elif vel>0:
        return 2
    else:
        return 1
```

• **结果：采用此策略，平均 127 步可以到达山顶。**

```
Average reward of random mode equals to -200.0
Episode finished after 171 timesteps. Total reward:-171.0
Episode finished after 127 timesteps. Total reward:-127.0
Episode finished after 101 timesteps. Total reward:-101.0
Episode finished after 112 timesteps. Total reward:-112.0
Episode finished after 124 timesteps. Total reward:-124.0
Average reward equals to -127.0
```

二、Machine-Learning Agent of MountainCar in OpenAI Gym

1. *Q-Learning*

*Q-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ε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 (ε-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

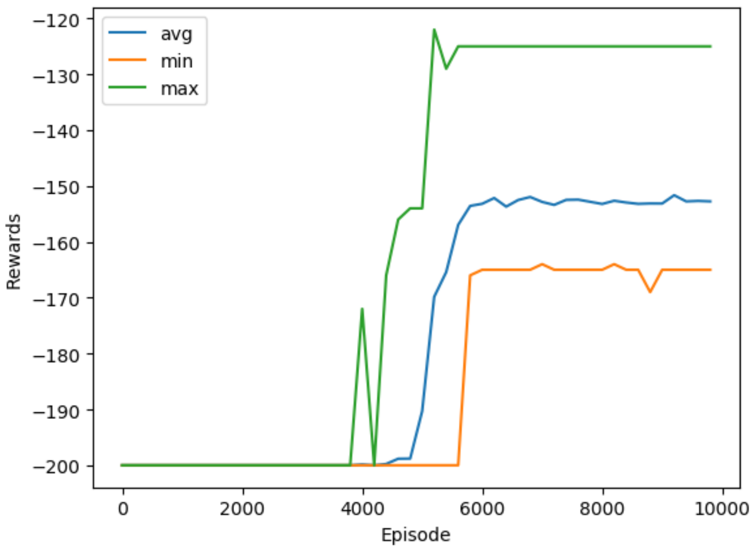
• 核心训练代码

```
#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-Policy*算法，自始至终只有一个*Policy*。该算法除了目标值与*Q-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 (ε-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

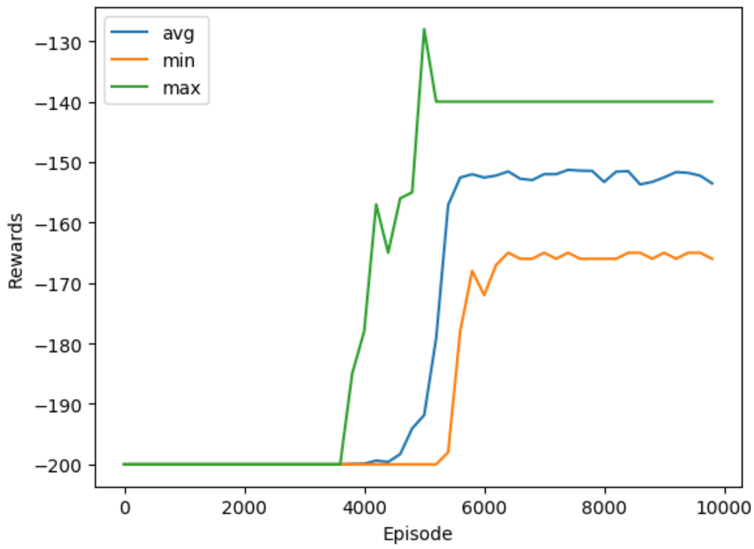
• 核心训练代码

```
#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. $\boldsymbol{SARSA(lambda)}$

该算法引入了衰减系数λ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 (ε-greedy)
        $\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$
        $E(S,A) \leftarrow E(S,A) + 1$
        For all $s \in S,\ a \in A(s)$:
            $Q(s,a) \leftarrow Q(s,a) + \alpha\delta E(s,a)$
            $E(s,a) \leftarrow \gamma\lambda E(s,a)$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until S is terminal

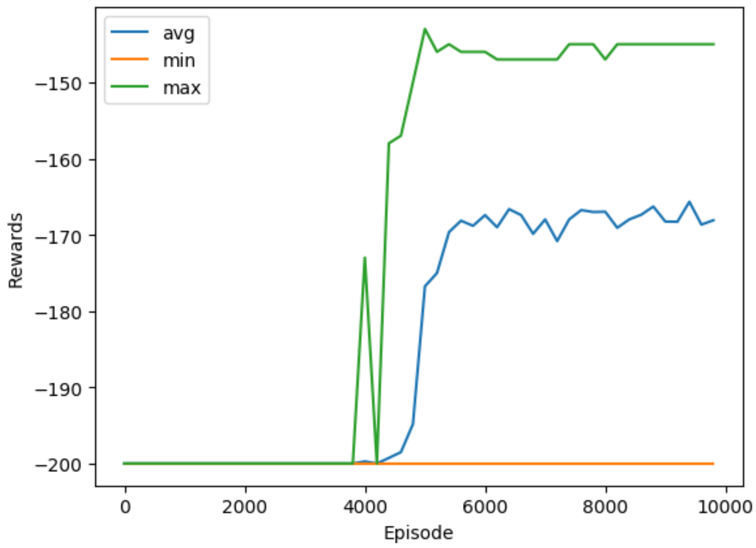
• **核心训练代码**

```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



三、Deep-Learning Agent of MountainCar in OpenAI Gym

1. $DQN(Deep\ Q-Learning)$

DQN不用Q表记录Q值，而是用神经网络来预测Q值，并通过不断更新神经网络从而学习到最优的行动路径。DQN有一个记忆库（$Experience\ replay$）和固定Q目标（$Fixed\ Q\text{-}targets$）。记忆库用来学习之前的经历，通过每步$agent$与环境交互得到的样本储存进记忆网络，要训练时随机拿出一些来训练，从而解决了相关性及非静态分布问题。使用$Q\text{-}targets$是的DQN中出现两个结构完全相同但是参数不同的网络，预测Q估计的网络$MainNet$使用的是最新参数，预测Q现实的神经网络$TargetNet$使用的之前的，一定程度上降低了当前Q值和目标Q值的相关性。

- **算法**

Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For t = 1,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^-\right) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

- **核心代码**

（1） 建立模型

这里使用的是$Keras$序列模型（$sequential\ model$），并设置好参数。

```
def create_model():
    model=models.Sequential()
    model.add(Dense(16,input_shape=(env.observation_space.shape)))
    model.add(Activation('relu'))
    model.add(Dense(16))
    model.add(Activation('relu'))
    model.add(Dense(16))
    model.add(Activation('relu'))
    model.add(Dense(ACTION_SPACE_SIZE))
    model.add(Activation('linear'))
    print(model.summary())
    model.compile(loss='mse',optimizer=Adam(lr=0.001),metrics=['accuracy'])
    return model
```

（2） 智能体类的*train*函数

minibatch：从记忆库中随机取一定数量的样本

current_states：获取当前状态

current_qs_list：在预测网络中查询目标Q值

next_states：获取下一状态

target_qs_list：查询主网络获取目标Q值

```
def train(self,terminal_state,step):
    if len(self.replay_memory)<MIN_REPLAY_MEMORY_SIZE:
        return
    minibatch=random.sample(self.replay_memory, MINIBATCH_SIZE)
    current_states=np.array([transition[0] for transition in minibatch])
    current_qs_list=self.model_prediction.predict(current_states)
    next_states=np.array([transition[3] for transition in minibatch])
    target_qs_list=self.model_target.predict(next_states)
```

开始列举，从未来状态获取新的Q，结束时置为 0，更新当前状态的Q值，将其加入训练数据。

```
for index,(current_state,action,reward,next_state,done) in enumerate(minibatch):
    if not done:
        max_target_q=np.max(target_qs_list[index])
        new_q=reward+DISCOUNT*max_target_q
    else:
        new_q=reward
    current_qs=current_qs_list[index]
    current_qs[action]=new_q
    X.append(current_state)
    Y.append(current_qs)
```

model_prediction.fit：与所有样本做匹配，但仅记录中止状态。

model_target.set_weights：到一定时候用主网络更新目标网络。

```
self.model_prediction.fit(np.array(X),np.array(Y),batch_size=MINIBATCH_SIZE,
                        verbose=0,shuffle=False if terminal_state else None)
if terminal_state:
    self.target_update_counter+=1
if self.target_update_counter>UPDATE_TARGET_EVERY:
    self.model_target.set_weights(self.model_prediction.get_weights())
    self.target_update_counter=0
```

（3） 训练智能体

下面是比较常规的训练过程，除了不用建立Q表外跟Q-$Learning$部分类似。

```
for episode in tqdm(range(1,EPISODES+1),ascii=True,unit='episodes'):
    ep_reward=0
    step=1
    state=env.reset()
    done=False
    while not done:
        if np.random.random()>epsilon:
            action=np.argmax(agent.get_qs(state))
        else:
            action=np.random.randint(0,ACTION_SPACE_SIZE)
        next_state,reward,done,_=env.step(action)
        ep_reward+=reward
        agent.update_replay_memory((state,action,reward,next_state,done))
        agent.train(done,step)
        state=next_state
        step+=1
```

```
ep_rewards.append(ep_reward)
if epsilon>MIN_EPSILON:
    epsilon*=EPSILON_DECAY
    epsilon=max(MIN_EPSILON,epsilon)
if not episode % AGGREGATE_STATS_EVERY or episode==1:
    average_reward=sum(ep_rewards[-AGGREGATE_STATS_EVERY:])\
                /len(ep_rewards[-AGGREGATE_STATS_EVERY:])
    aggr_ep_rewards['ep'].append(episode)
    aggr_ep_rewards['avg'].append(average_reward)
    aggr_ep_rewards['min'].append(min(ep_rewards[-AGGREGATE_STATS_EVERY:]))
    aggr_ep_rewards['max'].append(max(ep_rewards[-AGGREGATE_STATS_EVERY:]))
```

- **运行截图**

（1）模型结构

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 16)                48
_________________________________________________________________
activation (Activation)      (None, 16)                0
_________________________________________________________________
dense_1 (Dense)              (None, 16)                272
_________________________________________________________________
activation_1 (Activation)    (None, 16)                0
_________________________________________________________________
dense_2 (Dense)              (None, 16)                272
_________________________________________________________________
activation_2 (Activation)    (None, 16)                0
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 51
_________________________________________________________________
activation_3 (Activation)    (None, 3)                 0
=================================================================
Total params: 643
Trainable params: 643
Non-trainable params: 0
```

（2）训练过程中

```
  5%|######5
                                    | 47/1000 [15:32<13:14:40, 50.03s/episodes]
```