人工智能导论第三次实验报告

Deep-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

• $\boldsymbol{DQN(Deep\ Q-Learning)}$

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$TargetNe$ 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 $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ 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。

```python
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。

```python
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
```

```python
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]
```