中山大学计算机学院

人工智能

本科生实验报告

（2024学年春季学期）

课程名称：Artificial Intelligence

教学班级	人工智能(22信计+系统结构)	专业（方向）	信息与计算科学
学号	**22336044**	姓名	**陈圳煌**

一、实验题目

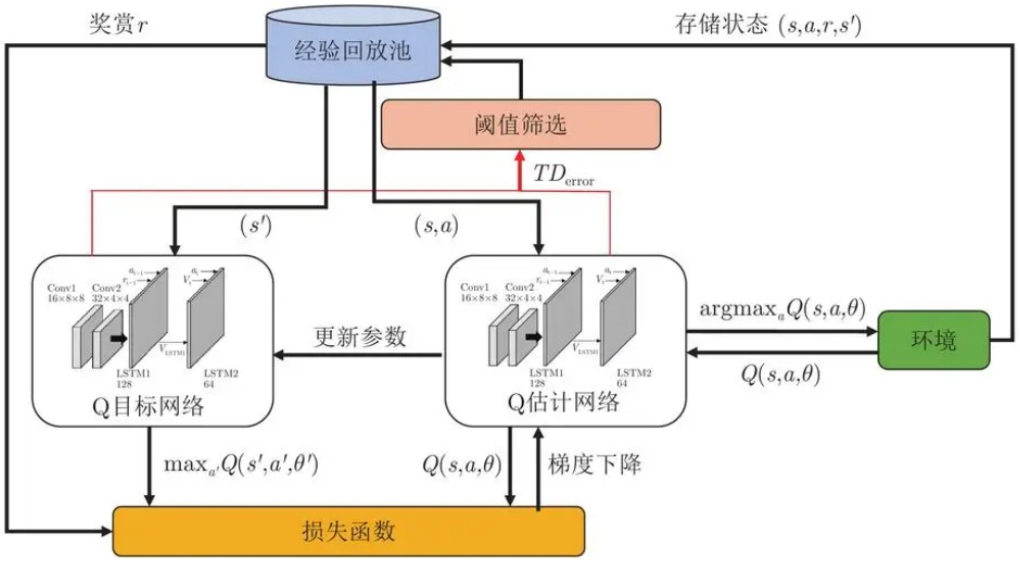
深度强化学习（Deep Q-learning Network，DQN）

二、实验内容

1.算法原理

DQN结合深度神经网络来逼近Q值函数，在实现过程中需要搭建两个网络：一个评估网络，一个目标网络和一个缓冲区用于存储经验回放。在训练过程中，根据智能体与环境交互，选择动作并执行，然后观察下一个状态和获得的奖励，将这些经验（状态、动作、奖励、下一个状态）存储到经验回放缓冲区。接着从缓冲区随机抽样（为了减少数据相关性），使用评估网络和目标网络计算q值和target值，和损失函数loss，用反向传播来更新评估网络的参数。然后不断重复以上过程，DQN最后能够逐渐学习到最佳的行为策略，使智能体在给定环境下能够做出最优的决策。

流程图：



2.伪代码

```
class QNet():
    初始化：输入input_size,隐藏层hidden_size,输出output_size
        self.fc1第一个全连接层（input_size,hidden_size）
        self.fc2第二个全连接层（hidden_size,output_size）

    def forward()前向传播：
        x转换成tensor数组
        两次全连接层
        x=self.fc2(relu(self.fc1(x)))

class ReplayBuffer:
    初始化：缓冲区的最大容量
    def len()返回buffur当前元素个数
    def push(transition)添加新的经验到缓冲区
    def sample(batch_size)从缓冲区随机选取batch_size个经验
    def clean()清空缓冲区

class DQN:
    初始化：
        创建一个评估网络和一个目标网络
        self.eval_net=QNet(……)
        self.target_net=QNet(……)
        self.optim=Adam建立一个优化器
        self.eps作为取动作的概率
        self.buffer=ReplayBuffer(capacity)

    def choose_action(obs):
        依概率选择随机抽取动作或是抽取得分的动作
        if self.eps>（0，1）中的随机数：
            随机选取一个动作
        else:
            根据状态obs选取得分最好的动作

    def store_transition(*transition):
        将经验添加入缓冲区

    def learn()网络的更新
        obs,action,rewards,next_obs,dones=从缓冲区取样
        q评价值=self.eval_net(obs)
        q目标值=self.target_net(actions).max(1)[0]最大得分的动作
        由TD算法计算目标值
        td_target=rewards+γ*(1-dones)*q_target
        计算损失函数loss
        optim.backward()反向传播

主过程：
初始化环境env
agent=DQN(env,input_size,hidden_size,output_size)
for i in 总轮数：
    obs=env
    action=agent.choose_action(obs)
    next_obs,reward=env.step(action)根据动作确定下一步动作
    agent.store_transition(……)存储经验
```

```
    if agent.buffer.len()>256:
        agent.learn()当缓冲区达到一定经验数开始进行学习
```

3.关键代码展示

```
class QNet(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(QNet, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.fc2 = nn.Linear(hidden_size,output_size)

    def forward(self, x):
        x = torch.FloatTensor(x)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

class ReplayBuffer:
    def __init__(self, capacity):
        #deque数据结构存储数据，支持在达到最大容量时自动删除旧元素(最早进入的)。
        #最大容量为capacity
        self.buffer = deque(maxlen=capacity)

    def len(self):
        #返回buffer现有的元素数
        return len(self.buffer)

    def push(self, *transition):
        #添加新的经验到buffer
        self.buffer.append(transition)

    def sample(self, batch_size):
        #从缓冲区随机取batch_size个经验
        transition=random.sample(self.buffer,batch_size)
        #将列表中的元素解压成多个元组并转换成numpy数组
        obs, action, rewards, next_obs, dones=zip(*transition)
        return np.array(obs), np.array(action), np.array(rewards),
np.array(next_obs), np.array(dones)

    def clean(self):
        #清空buffer
        self.buffer.clear()

class DQN:
    def __init__(self, env, input_size, hidden_size, output_size):
        self.env = env
        #两个神经网络：评估网络和目标网络
        # network for evaluate
        self.eval_net = QNet(input_size, hidden_size, output_size)
        # target network
        self.target_net = QNet(input_size, hidden_size, output_size)
        #Adam优化器进行优化
        self.optim = optim.Adam(self.eval_net.parameters(), lr=args.lr)
```

```
        #初始值eps用于贪婪策略
        self.eps = args.eps
        #缓冲区
        self.buffer = ReplayBuffer(args.capacity)
        #均方损失函数
        self.loss_fn = nn.MSELoss()
        #初始化学习步数计数器
        self.learn_step = 0

    def choose_action(self, obs):
        #20%概率随机选择动作（使用贪婪算法后逐渐减小）
        if np.random.uniform() < self.eps:
            action=self.env.action_space.sample()
        else:
            q=self.eval_net(obs)
            action=q.argmax().item()

        return action

    #存储一个经验到缓冲区
    def store_transition(self, *transition):
        self.buffer.push(*transition)

    def learn(self):
        # [Epsilon Decay]
        if self.eps > args.eps_min:
            self.eps *= args.eps_decay

        # [Update Target Network Periodically]
        if self.learn_step % args.update_target == 0:
            self.target_net.load_state_dict(self.eval_net.state_dict())
        self.learn_step += 1

        # [Sample Data From Experience Replay Buffer]
        obs, actions, rewards, next_obs, dones =
self.buffer.sample(args.batch_size)
        actions = torch.LongTensor(actions).view(-1,1)  # to use 'gather' latter
        dones = torch.FloatTensor(dones).view(-1,1)
        rewards = torch.FloatTensor(rewards).view(-1,1)
        #计算当前obs下的评估网络预测的Q值
        # 【eval神经网络结果为两个动作，与actions聚合后得到动作】
        q_eval=self.eval_net(obs).gather(1,actions)#q_eval为256x1
        #下一个obs对应的目标网络的最大Q值，并用detach()从计算图中分离，以避免反向传播对目标网
络的影响。
        q_target=self.target_net(next_obs).max(1)[0].view(-1,1).detach()#q_target
为256x1
        td_target=rewards+args.gamma*(1-dones)*q_target

        loss=self.loss_fn(q_eval,td_target)

        self.optim.zero_grad()#PyTorch中默认梯度会累积,这里需要显式将梯度置为0
        loss.backward()#反向传播
        self.optim.step()

def main():
    env = gym.make(args.env)
```

```
    reward_set=[]
    aver_reward_set=[]
    o_dim = env.observation_space.shape[0]
    #print(o_dim)
    a_dim = env.action_space.n
    #print(a_dim)
    agent = DQN(env, o_dim, args.hidden, a_dim)                          # 初始化
DQN智能体
    for i_episode in range(args.n_episodes):                             # 开始玩游
戏
        obs = env.reset()                                               # 重置环境
        episode_reward = 0                                               # 用于记录
整局游戏能获得的reward总和
        done = False
        step_cnt=0
        while not done and step_cnt<500:
            step_cnt+=1
            #env.render()                                                 # 渲染当
前环境(仅用于可视化)
            action = agent.choose_action(obs)                            # 根据当前
观测选择动作
            next_obs, reward, done, info = env.step(action)              # 与环境交
互
            agent.store_transition(obs, action, reward, next_obs, done) # 存储转移
            # 当buffer满时清空
            # if agent.buffer.len()== args.capacity:
            #     agent.buffer.clean()

            episode_reward += reward                                     # 记录当前
动作获得的reward
            obs = next_obs
            if agent.buffer.len() >= 256:
                agent.learn()                                            # 学习以及
优化网络
        print(f"Episode: {i_episode}, Reward: {episode_reward}")
        reward_set.append(episode_reward)

 aver_reward_set.append(sum(reward_set[len(reward_set)-100:len(reward_set):])/100
)

    plt.plot([i for i in range(len(reward_set))],reward_set)
    plt.title("Reward")
    plt.show()
    print("最多连续：",get500score(reward_set),"局得分达到500分")
    plt.plot([i for i in range(len(aver_reward_set))],aver_reward_set)
    plt.axhline(y=475, color='r', linestyle='--', label='y = 475')
    plt.legend()
    #plt.plot([i for i in range(len(aver_reward_set))],[475 for i in
range(len(aver_reward_set))],c='g')
    plt.title("Average_reward")
    plt.ylim(0,500)
    plt.show()
    print("百局平均中是否有超过475分:",(max(aver_reward_set)>=475))
```

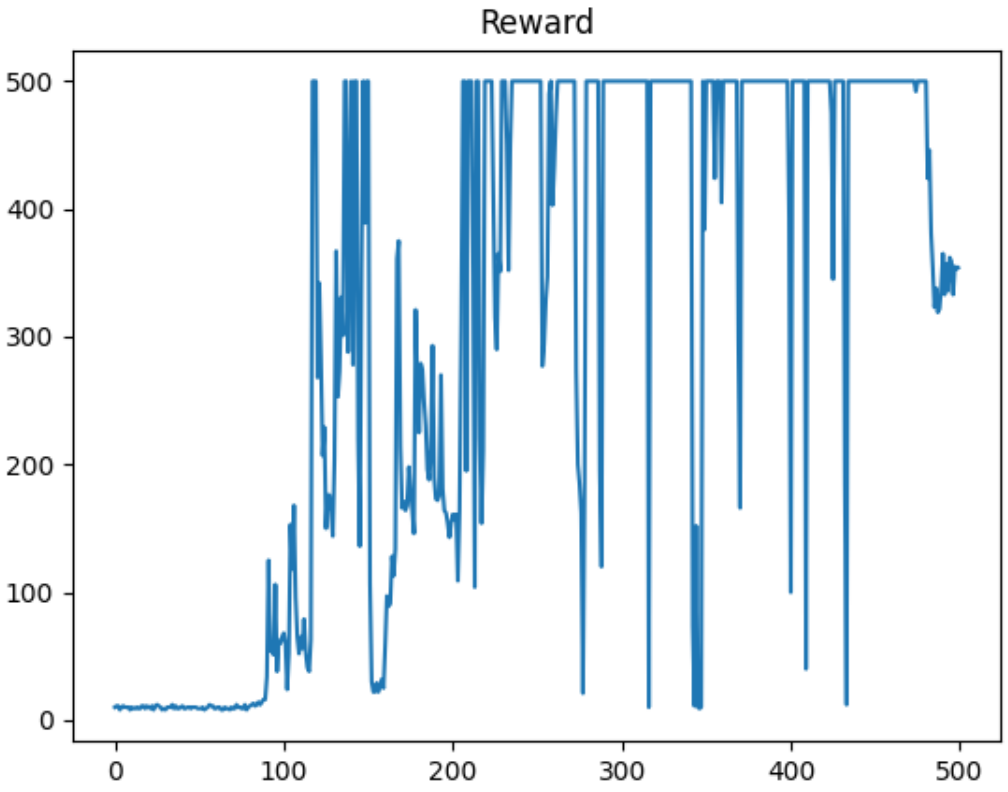
三、实验结果及分析

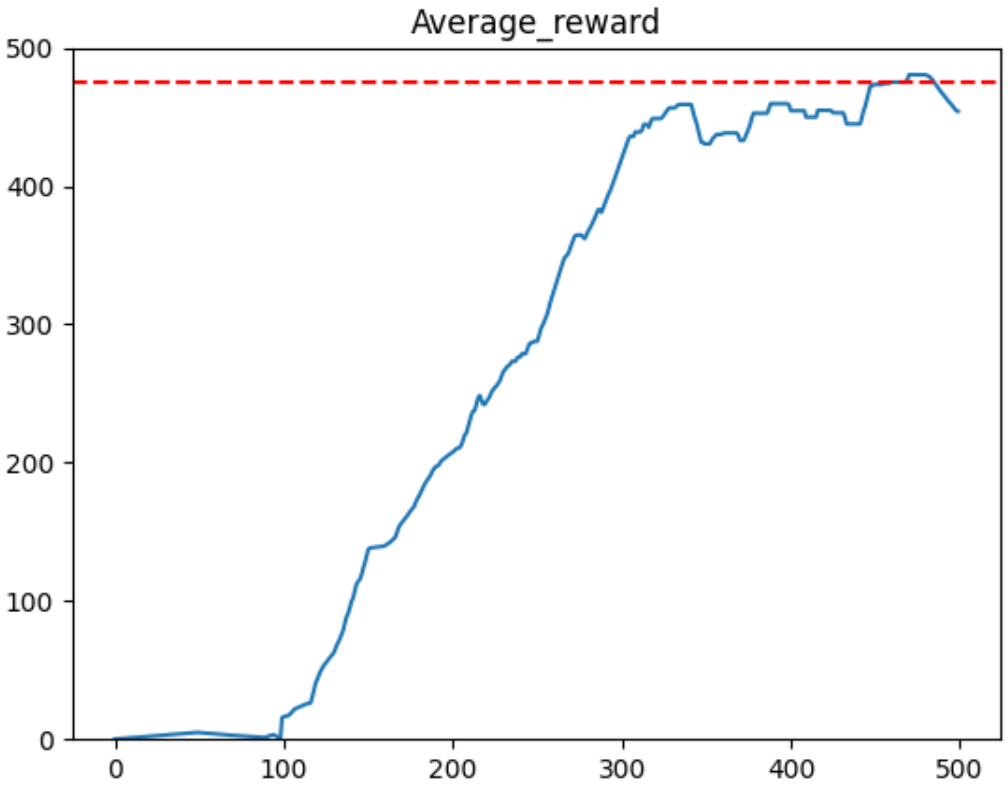
1.结果展示

仅展示效果较优的一次（为result文件夹中的reward9以及average9）

参数设置为：

lr=0.001, hidden=128, capacity=1024, eps=0.08, eps_min=0.05, batch_size=128, eps_decay=0.999, update_target=100，buffer中的经验数大于256时进行学习




Average_reward
500
400
300
200
100
0
0
100
200
300
400
500

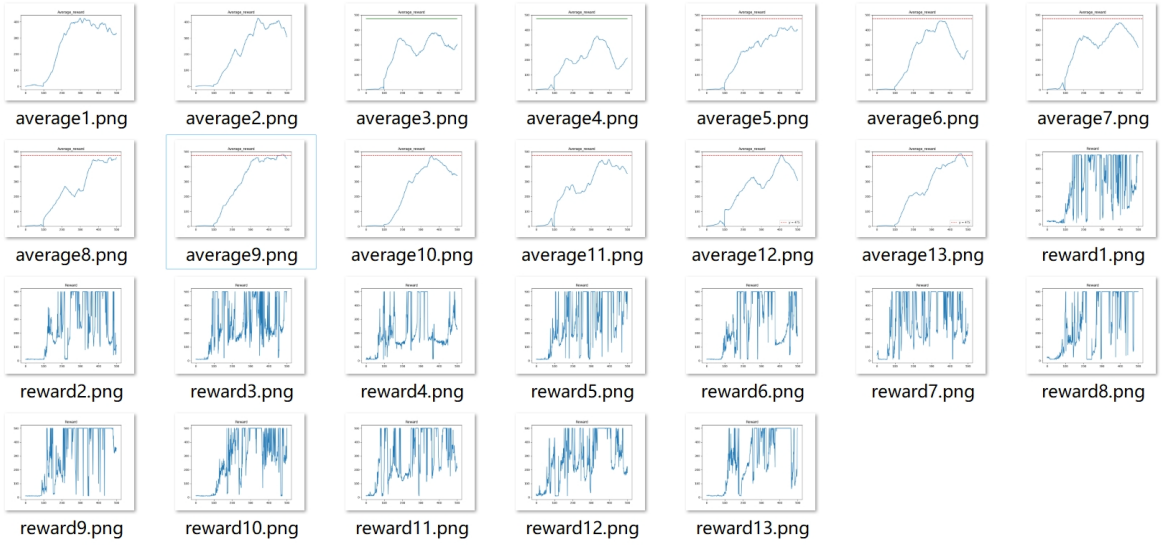
```
Episode: 361, Reward: 500.0
Episode: 362, Reward: 500.0
Episode: 363, Reward: 500.0
Episode: 364, Reward: 500.0
Episode: 365, Reward: 500.0
Episode: 366, Reward: 500.0
Episode: 367, Reward: 500.0
Episode: 368, Reward: 500.0
Episode: 369, Reward: 500.0
Episode: 370, Reward: 500.0
Episode: 371, Reward: 500.0
Episode: 372, Reward: 500.0
Episode: 373, Reward: 500.0
Episode: 374, Reward: 500.0
Episode: 375, Reward: 500.0
Episode: 376, Reward: 500.0
Episode: 377, Reward: 500.0
Episode: 378, Reward: 500.0
Episode: 379, Reward: 500.0
Episode: 380, Reward: 500.0
Episode: 381, Reward: 500.0
Episode: 382, Reward: 500.0
Episode: 383, Reward: 500.0
Episode: 384, Reward: 500.0
Episode: 385, Reward: 500.0
Episode: 386, Reward: 500.0
Episode: 387, Reward: 500.0
Episode: 388, Reward: 500.0
Episode: 389, Reward: 500.0
Episode: 390, Reward: 500.0
Episode: 391, Reward: 500.0
```

```
Episode: 485, Reward: 323.0
Episode: 486, Reward: 338.0
Episode: 487, Reward: 319.0
Episode: 488, Reward: 322.0
Episode: 489, Reward: 339.0
Episode: 490, Reward: 365.0
Episode: 491, Reward: 333.0
Episode: 492, Reward: 357.0
Episode: 493, Reward: 336.0
Episode: 494, Reward: 362.0
Episode: 495, Reward: 359.0
Episode: 496, Reward: 333.0
Episode: 497, Reward: 355.0
Episode: 498, Reward: 353.0
Episode: 499, Reward: 354.0
最多连续： 40 局得分达到500分
百局平均中是否有超过475分： True
```

在实验中，最多可以实现连续40局的得分到达500分，并且能够达到在最近百局的平均reward值能够达到475分，从图像看大约在470局左右达到。

2.评测指标展示及分析

在实验过程中以得分和最近百局平均得分作为指标进行测试，通过调整参数来测试对模型的效果的影响。



前后共记录了13次测试结果，从结果中观察到仅有3次能够实现最近百局平均超过475分（分别为average9、average12、average13图）

进行测试的参数如下（其中.....表示参数与之前的一致）：

第一次：
lr=0.001,hidden=256,capacity=1000,eps=0.1,eps_min=0.05,batch_size=256,eps_decay=0.999,update_target=100, buffer 长度大于 1000 时进行学习
第二次
lr=0.001,hidden=256,capacity=1000,eps=0.1,eps_min=0.05,batch_size=256,eps_decay=0.999,update_target=100, buffer 长度大于 500 时进行学习
第三次
lr=0.001,hidden=256,capacity=1000,eps=0.1,eps_min=0.05,batch_size=128,eps_decay=0.999,update_target=100, buffer 长度大于 250 时进行学习
第四次：
lr=0.001,hidden=256,capacity=1000,eps=0.1,eps_min=0.05,batch_size=128,eps_decay=0.999,update_target=100, buffer 长度大于 250 时进行学习
第五次：
lr=0.001,hidden=256,capacity=512,eps=0.08,eps_min=0.05,batch_size=128,eps_decay=0.999,update_target=100, buffer 长度大于 256 时进行学习
第六次：
lr=0.001,hidden=256,capacity=1024,eps=0.08,eps_min=0.05,batch_size=128,eps_decay=0.999,update_target=100, buffer 长度大于 256 时进行学习
……
第九次：
参数：
lr=0.001,hidden=128,capacity=1024,eps=0.08,eps_min=0.05,batch_size=128,eps_decay=0.999,update_target=100, buffer 长度大于 256 时进行学习
第十次：
参数：
lr=0.001,hidden=128,capacity=1024,eps=0.08,eps_min=0.05,batch_size=128,eps_decay=0.999,update_target=100, buffer 长度大于 256 时进行学习
……
第十二次：
lr=0.001,hidden=128,capacity=1024,eps=0.5,eps_min=0.05,batch_size=128,eps_decay=0.999,update_target=100, buffer 长度大于 256 时进行学习
第 13 次：
lr=0.001,hidden=128,capacity=1024,eps=0.05,eps_min=0.05,batch_size=128,eps_decay=0.999,update_target=100, buffer 长度大于 256 时进行学习（取随机动作的概率固定为 5%）

实验参数合理性：

从参数选择情况可以看到，当选择buffer经验回放缓冲区最大值为1000左右，当缓冲区超过256个经验时进行训练，可以保证有适当的训练次数。而使用deque数据结构，每次缓冲区满时将最早进入的替换，可以保证缓冲区随着更新存储的经验越来越好。同时eps刚开始固定，而随着游戏进行会逐渐变低，减少随机选取动作的情况。

问题以及原因：

但是还是会出现在中间时能够连续达到500分，但是后续得分会降低的情况。

·可能是因为过拟合；

·在后续的结果中依赖之前达到500分的经验而减少了对环境进行有效探索，可能导致了Agent无法发现更好的策略或者适应环境的变化。

优化：

根据[1]中所提及，传统的DQN算法会导致对Q值得过高估计

传统 DQN 优化的 TD 误差目标为

$$r+\gamma \max_{a'} Q_{\omega^-}\left(s', a'\right)$$

而实际上max操作可以分为两部分，首先选取状态s'下的最优动作，接着计算该动作下的价值。当这两部分采用同一套 Q 网络进行计算时，每次得到的都是神经网络当前估算的所有动作价值中的最大值。而神经网络估算的Q值本身会产生误差，在DQN下会导致误差累积，从而我们用以下方法：

$$r+\gamma Q_{\omega^-}\left(s', \arg\max_{a'} Q_{\omega}\left(s', a'\right)\right)$$

用两套神经网络来分别进行选取动作和计算得分。

代码由：

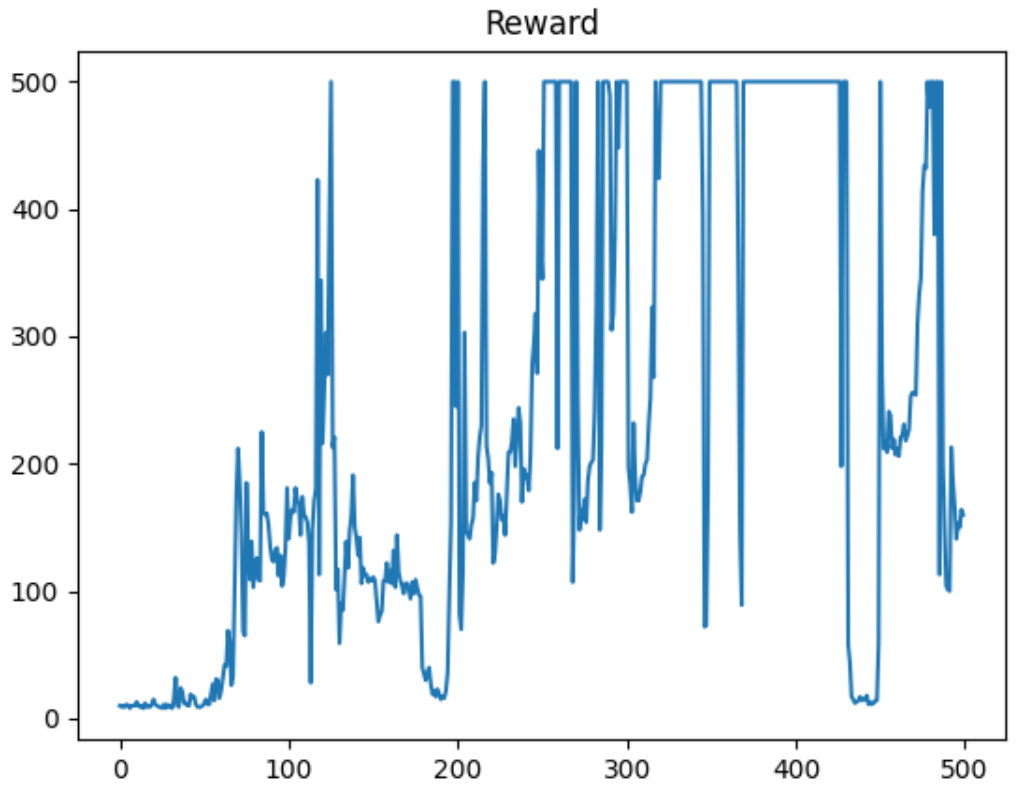
```
q_eval=self.eval_net(obs).gather(1,actions)#q_eval为256x1
q_target=self.target_net(next_obs).max(1)[0].view(-1,1)
#q_target为256x1
td_target=rewards+args.gamma*(1-dones)*q_target
```

修改为：

```
q_eval=self.eval_net(obs).gather(1,actions)#q_eval为256x1
max_action=self.eval_net(next_obs).max(1)[1].view(-1,1)
q_target=self.target_net(next_obs).gather(1,max_action)
td_target=rewards+args.gamma*(1-dones)*q_target
```

得到的结果为（reward_updata1和average_updata1，采用与第九次测试一样的参数，与上面结果进行对比）：

```
Episode: 482, Reward: 380.0
Episode: 483, Reward: 500.0
Episode: 484, Reward: 500.0
Episode: 485, Reward: 113.0
Episode: 486, Reward: 500.0
Episode: 487, Reward: 201.0
Episode: 488, Reward: 149.0
Episode: 489, Reward: 104.0
Episode: 490, Reward: 101.0
Episode: 491, Reward: 100.0
Episode: 492, Reward: 213.0
Episode: 493, Reward: 185.0
Episode: 494, Reward: 166.0
Episode: 495, Reward: 141.0
Episode: 496, Reward: 153.0
Episode: 497, Reward: 150.0
Episode: 498, Reward: 164.0
Episode: 499, Reward: 160.0
最多连续： 58 局得分达到500分
百局平均中是否有超过475分： True
```



与未更新之前相比结果略有提高，能够连续58局达到500分且连续百局评分达475分的位置从470轮提前到了400轮左右，更快到达，表明了优化方法有一定效果。

四、参考资料

[1] [动手学强化学习（八）： DQN 改进算法 - jasonzhangxianrong - 博客园 (cnblogs.com)(https://www.cnblogs.com/zhangxianrong/p/18054304)

[2] week 14 DQN.pdf

[3] Human-level control through deep reinforcement learning | Nature