HW4

一、环境配置

1.1 依赖安装

沿用HW3的环境即可

`pip install gymnasium[Atari]` `pip install gymnasium[accept-rom-license]`下载安装gymnasium以及Atari环境和rom

二、Blank fill

2.1 QNetwork

```
class QNetwork(nn.Module):
    def __init__(self, env):
        super().__init__()
        '''
        定义一个三层神经网络，其中
        输入层：全连接层： 状态空间作为输入，256作为输出，Relu激活函数
        中间层：全连接层：256输入256输出，Relu激活
        输出层：全连接层：256输入，动作空间输出
        '''
        self.fc1 = nn.Linear(env.observation_space.shape[0], 256)
        self.fc2 = nn.Linear(256, 256)
        self.fc3 = nn.Linear(256, env.action_space.n)

    def forward(self, x):
        '''
        前向计算
        '''
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
```

按照提示填入即可

2.2 define Network

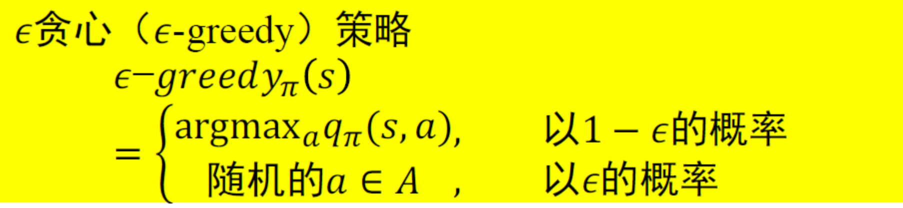
```
'''
    定义两个网络，分别为q_network,以及target_network;
    定义优化器，训练网络
    '''
    q_network = QNetwork(env).to(device)
    target_network = QNetwork(env).to(device)
    target_network.load_state_dict(q_network.state_dict())
    target_network.eval()

    optimizer = optim.Adam(q_network.parameters(), lr=learning_rate)
```

初始化两个相同网络，分别作为评估网络和目标网络

2.3 epsilon-greedy

```
'''
        实现epsilon-greedy算法，epsilon为给定超参
        '''
        if random.random() < epsilon:
            actions = env.action_space.sample()
        else:
            with torch.no_grad():
                obs_tensor = torch.tensor(obs, device=device,
dtype=torch.float32).unsqueeze(0)
                actions = q_network(obs_tensor).argmax(dim=1).item()
```



ϵ贪心（ϵ-greedy）策略

$$\epsilon{-}greedy_{\pi}(s) = \begin{cases} \text{argmax}_a q_{\pi}(s,a), & \text{以}1-\epsilon\text{的概率} \\ \quad \text{随机的}a \in A \quad , & \text{以}\epsilon\text{的概率} \end{cases}$$

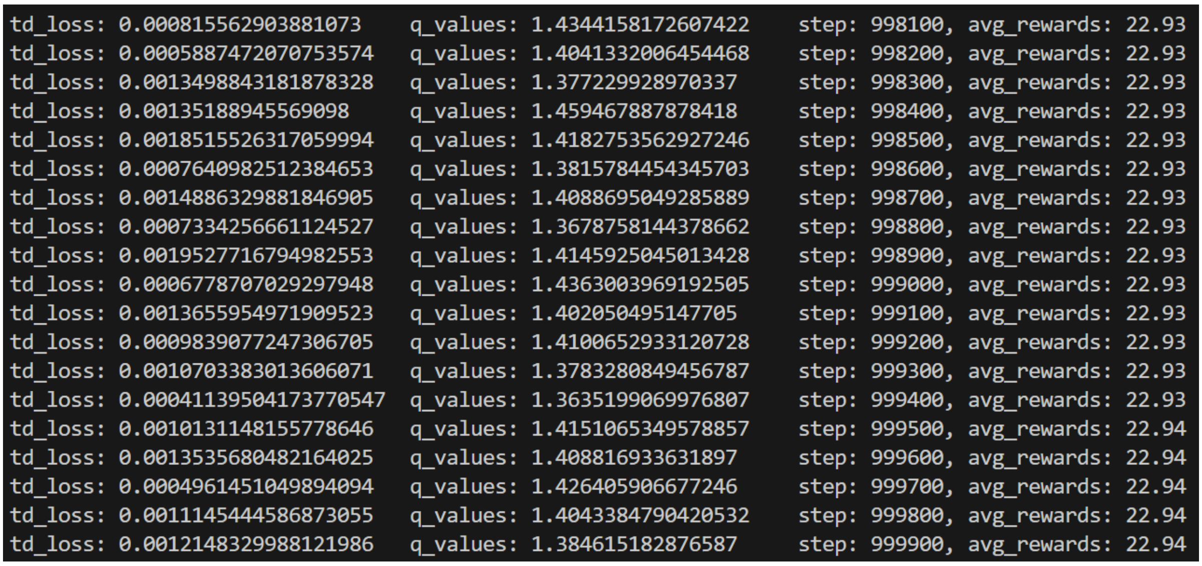
2.4 td_target

```
'''
                计算td_target
                Q(s,a) (old_val)
                '''
                with torch.no_grad():
                    target_max = target_network(next_buffer_obs).max(dim=1,
keepdim=True)[0]
                    td_target = rew + cont * gamma * target_max
                old_val = q_network(buffer_obs).gather(1, act)
```

使用目标网络计算TD：

$$y = r + \gamma Q_{\omega^-}(s', argmax_{a'} Q_{\omega^-}(s', a'))$$

2.5 运行结果（时间步10^6）：



```
td_loss: 0.000815562903881073    q_values: 1.4344158172607422    step: 998100, avg_rewards: 22.93
td_loss: 0.0005887472070753574   q_values: 1.4041332006454468    step: 998200, avg_rewards: 22.93
td_loss: 0.0013498843181878328   q_values: 1.377229928970337     step: 998300, avg_rewards: 22.93
td_loss: 0.00135188945569098     q_values: 1.459467887878418     step: 998400, avg_rewards: 22.93
td_loss: 0.0018515526317059994   q_values: 1.4182753562927246    step: 998500, avg_rewards: 22.93
td_loss: 0.0007640982512384653   q_values: 1.3815784454345703    step: 998600, avg_rewards: 22.93
td_loss: 0.0014886329881846905   q_values: 1.4088695049285889    step: 998700, avg_rewards: 22.93
td_loss: 0.0007334256661124527   q_values: 1.3678758144378662    step: 998800, avg_rewards: 22.93
td_loss: 0.0019527716794982553   q_values: 1.4145925045013428    step: 998900, avg_rewards: 22.93
td_loss: 0.0006778707029297948   q_values: 1.4363003969192505    step: 999000, avg_rewards: 22.93
td_loss: 0.0013655954971909523   q_values: 1.402050495147705     step: 999100, avg_rewards: 22.93
td_loss: 0.0009839077247306705   q_values: 1.4100652933120728    step: 999200, avg_rewards: 22.93
td_loss: 0.0010703383013606071   q_values: 1.3783280849456787    step: 999300, avg_rewards: 22.93
td_loss: 0.00041139504173770547  q_values: 1.3635199069976807    step: 999400, avg_rewards: 22.93
td_loss: 0.0010131148155778646   q_values: 1.4151065349578857    step: 999500, avg_rewards: 22.94
td_loss: 0.0013535680482164025   q_values: 1.408816933631897     step: 999600, avg_rewards: 22.94
td_loss: 0.0004961451049894094   q_values: 1.426405906677246     step: 999700, avg_rewards: 22.94
td_loss: 0.0011145444586873055   q_values: 1.4043384790420532    step: 999800, avg_rewards: 22.94
td_loss: 0.0012148329988121986   q_values: 1.384615182876587     step: 999900, avg_rewards: 22.94
```

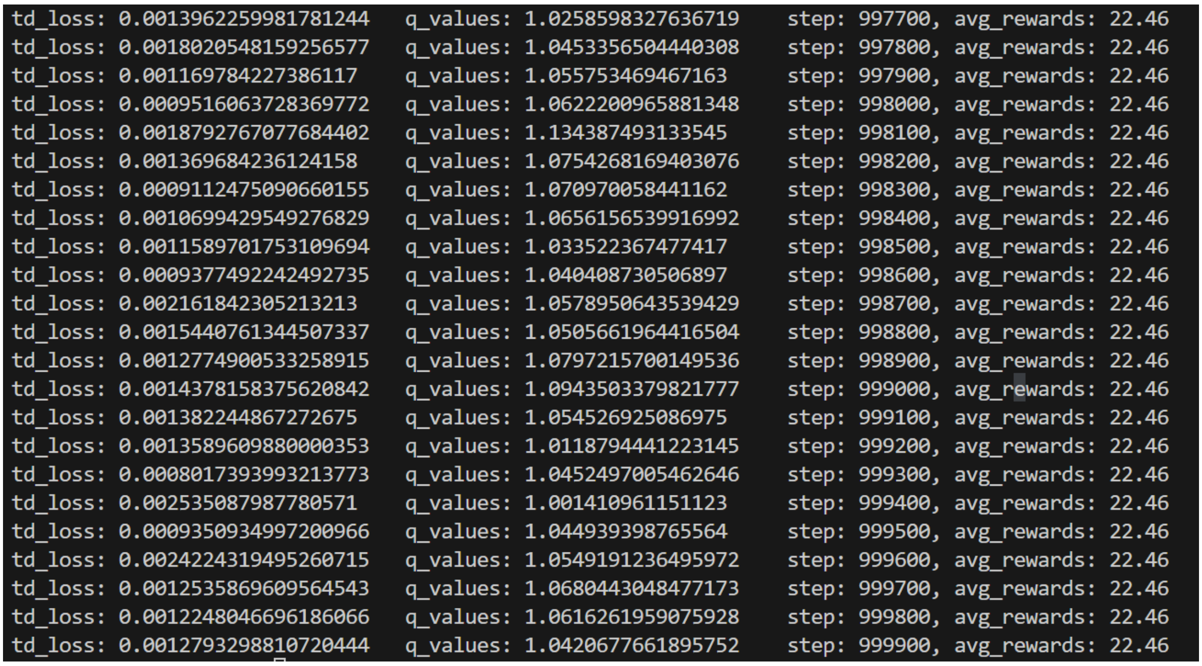
三、DDQN

3.1 td_target

```
with torch.no_grad():
                    next_actions = q_network(next_buffer_obs).argmax(dim=1,
keepdim=True)
                    target_q_values = target_network(next_buffer_obs).gather(1,
next_actions)
                    td_target = rew + cont * gamma * target_q_values
                old_val = q_network(buffer_obs).gather(1, act)![]()
```

和DQN不同的是，action和目标价值的估计是通过两个网络来进行的

3.2 运行结果



```
td_loss: 0.0013962259981781244   q_values: 1.0258598327636719    step: 997700, avg_rewards: 22.46
td_loss: 0.0018020548159256577   q_values: 1.0453356504440308    step: 997800, avg_rewards: 22.46
td_loss: 0.001169784227386117    q_values: 1.055753469467163     step: 997900, avg_rewards: 22.46
td_loss: 0.0009516063728369772   q_values: 1.0622200965881348    step: 998000, avg_rewards: 22.46
td_loss: 0.0018792767077684402   q_values: 1.134387493133545     step: 998100, avg_rewards: 22.46
td_loss: 0.001369684236124158    q_values: 1.0754268169403076    step: 998200, avg_rewards: 22.46
td_loss: 0.0009112475090660155   q_values: 1.070970058441162     step: 998300, avg_rewards: 22.46
td_loss: 0.0010699429549276829   q_values: 1.0656156539916992    step: 998400, avg_rewards: 22.46
td_loss: 0.0011589701753109694   q_values: 1.033522367477417     step: 998500, avg_rewards: 22.46
td_loss: 0.0009377492242492735   q_values: 1.040408730506897     step: 998600, avg_rewards: 22.46
td_loss: 0.002161842305213213    q_values: 1.0578950643539429    step: 998700, avg_rewards: 22.46
td_loss: 0.0015440761344507337   q_values: 1.0505661964416504    step: 998800, avg_rewards: 22.46
td_loss: 0.0012774900533258915   q_values: 1.0797215700149536    step: 998900, avg_rewards: 22.46
td_loss: 0.0014378158375620842   q_values: 1.0943503379821777    step: 999000, avg_rewards: 22.46
td_loss: 0.001382244867272675    q_values: 1.054526925086975     step: 999100, avg_rewards: 22.46
td_loss: 0.0013589609880000353   q_values: 1.0118794441223145    step: 999200, avg_rewards: 22.46
td_loss: 0.0008017393993213773   q_values: 1.0452497005462646    step: 999300, avg_rewards: 22.46
td_loss: 0.002535087987780571    q_values: 1.001410961151123     step: 999400, avg_rewards: 22.46
td_loss: 0.0009350934997200966   q_values: 1.044939398765564     step: 999500, avg_rewards: 22.46
td_loss: 0.0024224319495260715   q_values: 1.0549191236495972    step: 999600, avg_rewards: 22.46
td_loss: 0.0012535869609564543   q_values: 1.0680443048477173    step: 999700, avg_rewards: 22.46
td_loss: 0.0012248046696186066   q_values: 1.0616261959075928    step: 999800, avg_rewards: 22.46
td_loss: 0.0012793298810720444   q_values: 1.0420677661895752    step: 999900, avg_rewards: 22.46
```

可以看到，DDQN减少了过高估计的偏差