强化学习：作业三

黄彦骁 502023370016

November 14, 2023

1 作业内容

在gym Atari环境中实现DQN, Double DQN算法。

2 实现过程

DQN算法伪代码如图1所示。

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 $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    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

Figure 1: DQN算法伪代码

不同于Q-leanring算法，DQN算法引入了经验回放池和目标网络两种策略。经验回放池的主要作用是打破数据之间的关联，从而满足数据独立同分布的假设。而目标网络的作用主要是能够一定程度降低当前 Q 值和目标 Q 值的相关性，从而提高算法稳定性。具体而言，DQN 使用了两个 Q 网络，一个当前网络用来选择动作，更新模型参数，另一个目标网络用于计算目标 Q 值。目标网络的网络参数不参与梯度更新优化的过程，而是每隔一段时间从当前网络中复制过来，即延时更新，这样可以减少目标 Q 值和当前的 Q 值的相关性。具体的实现如图2所示

```python
# How to calculate Q(s,a) for all actions
# q_values is a vector with size (batch_size, action_shape, 1)
# each dimension i represents Q(s0,a_i)
q_values = self.model(s0).cuda()

q_eval = q_values.gather(1, a)

with torch.no_grad():
    q_next = self.target_model(s1).cuda()
    q_target = r + self.config.gamma * q_next.max(1)[0].unsqueeze(1) * (1. - done)

# Tips: function torch.gather may be helpful
# You need to design how to calculate the loss
loss = F.mse_loss(q_eval, q_target)
```

Figure 2: DQN算法实现

之后，实现了DQN的一种变体Double DQN算。Double DQN 引入 Double Q-learning 的思想，不直接在目标网络中找各个动作中最大的Q值，而是先在当前Q网络中先找出最大Q值对应的动作，然后利用选择出来的动作在目标网络里面去计算目标Q值，通过即解耦目标Q值动作的选择和目标Q值的计算这两步，来消除过度估计的问题。DDQN的具体实现如图3所示

3 复现方式

运行DQN算法在code文件夹下运行 `python atari_ddqn.py --train`.
运行Double DQN算法在code文件夹下运行 `python atari_ddqn.py --train --double True`.

```python
q_values = self.model(s0).cuda()

q_eval = q_values.gather(1, a)

with torch.no_grad():
    q_next = self.target_model(s1).cuda()
    if self.double:
        actions = self.model(s1).cuda().max(1)[1].unsqueeze(1)
        q_next = q_next.gather(1, actions)
        q_target = r + self.config.gamma * q_next * (1. - done)
    else:
        q_target = r + self.config.gamma * q_next.max(1)[0].unsqueeze(1) * (1. - done)

# Tips: function torch.gather may be helpful
# You need to design how to calculate the loss
loss = F.mse_loss(q_eval, q_target)
```

Figure 3: Double-DQN算法实现

4 实验效果

在运行DQN算法是出现了算法初期reward一直没有变化的情况，由于算法所需运行时间过长，转而想到实现相关变体可以更快收敛看出效果，于是实现了Double-DQN，结果依然没有变化，reward一直在-20与-21之间波动，没有提升。再次考虑可能是库的版本以及环境存在问题，最开始使用环境为python3.9+gym0.23.1，运行代码存在gym-wrappers的环境报错，将其修改之后可以运行，但仍然会有warning存在。于是重新安装了python3.9+gym0.15.7的版本，用了新的环境之后运行代码不会产生任何报错和waring，可惜结果依然没有变换，如图4所示,浅蓝色为DQN算法，深蓝色为DDQN算法。

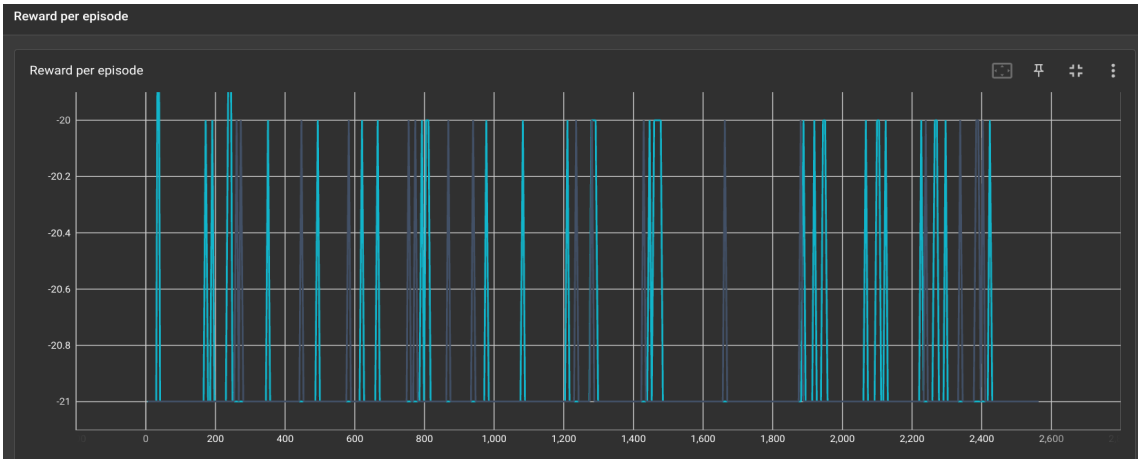

Figure 4: 两种算法效果

5 小结

本次实验采用DQN算法，由于框架实现较为完整，因此只需要实现主体算法部分，代码部分较为简单。但也是由于代码部分较为简单，可调整部分不多，因此寻找reward无法提升的问题时一直没有头绪，加上代码一次训练完成时间过长，所以也没有考虑实现更多的变体来观察效果，希望以后有机会可以对其他变体进行尝试。