中山大学计算机学院

人工智能本科生实验报告

课程名称：Artificial Intelligence

教学班级	网安软工合班	专业（方向）	网络空间安全
学号	20337251	姓名	伍建霖

一、实验题目

Task Implementing DQN、PG（选做）

1. **(coding)** 在`CartPole-v0`环境中实现DQN算法.最终算法性能的评判标准：环境最终的reward至少收敛至180.0.
2. **(optional coding)** 在`CartPole-v0`环境中实现A2C算法.最终算法性能的评判标准：环境最终的reward至少收敛至180.0.

Submission

作业提交内容：需提交一个zip文件，包括代码以及实验报告PDF。实验报告除了需要写writing部分的内容，还需要给出每题的reward曲线图以及算法。

zip文件命名格式: 20220421_张三_实验11；如果需提交不同版本，则命名格式：20220421_张三_实验11_v2等。

二、实验内容

1.算法原理

DQN与QLearning的区别：由于动作和状态是连续的，无法用一个表格来存储，故用一个人工神经网络来代替Q表

DQN是一种off-policy的算法，同时新增了replay buffer和target net的功能

replay buffer，将过去的数据存放在一个列表中，每隔一个batch学习一次

target net，和原来的网络一模一样，将原来的网络作为评估网络，target net作为目标网络。在学习过程中，使用目标网络进行自益得到回报的评估值，作为学习目标。在更新过程中，只更新评估网络的权重，而不更新目标网络的权重。这样，更新权重时针对的目标不会在每次迭代都发生变化，是一个固定的目标。在更新一定次数后，再将评估网络的权重复制给目标网络，进而进行下一批更新，这样目标网络也能得到更新。由于在目标网络没有变化的一段时间内回报的估计是相对固定的，因此目标网络的引入增加了学习的稳定性。

2. 伪代码

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    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 $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

3.关键代码展示（带注释）

```
class QNetwork(nn.Module):
    def __init__(self, input_dim, output_dim, hidden_dim) -> None:
        super(QNetwork, self).__init__()
        self.layer1 = torch.nn.Sequential(
            torch.nn.Linear(input_dim, hidden_dim),
            torch.nn.PReLU()
        )

        self.layer2 = torch.nn.Sequential(
            torch.nn.Linear(hidden_dim, hidden_dim),
            torch.nn.PReLU()
        )

        self.layer3 = torch.nn.Sequential(
            torch.nn.Linear(hidden_dim, hidden_dim),
            torch.nn.PReLU()
        )

        self.final = torch.nn.Linear(hidden_dim, output_dim)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.final(x)
        return x

# learn函数在agentdqn类中
def learn(self, gamma):
        if len(self.replay_memory.memory) < BATCH_SIZE:
            return
        # 随机取batch_size个历史数据
        transitions = self.replay_memory.sample(BATCH_SIZE)
        # 把数据转成设置格式，方便索索引
        batch = Transition(*zip(*transitions))

        states = torch.cat(batch.state)
        actions = torch.cat(batch.action)
```

```
        rewards = torch.cat(batch.reward)
        next_states = torch.cat(batch.next_state)
        dones = torch.cat(batch.done)

        Q_expected = self.q_local(states).gather(1, actions)
        Q_targets_next = self.q_target(
            next_states).detach().max(1)[0]  # 下一步的期望
        Q_targets = rewards + (gamma * Q_targets_next * (1-dones))  # 相当于真值

        # self.q_local.train(mode=True)
        self.optim.zero_grad()
        loss = self.mse_loss(Q_expected, Q_targets.unsqueeze(1))
        loss.backward()
        self.optim.step()

def get_action(self, state, eps, check_eps=True):
        global steps_done
        # 产生随机数，用于和eps对比
        sample = random.random()
        # 如果随机数大于eps，就选择使用DQN网络输出的action，否则随机选择一个动作
        if check_eps == False or sample > eps:
            with torch.no_grad():
                return
self.q_local(Variable(state).type(FloatTensor)).data.max(1)[1].view(1, 1)
        else:
            # 返回动作动作区间里面的随机的一个动作
            return torch.tensor([[random.randrange(self.n_actions)]],
device=device)
```

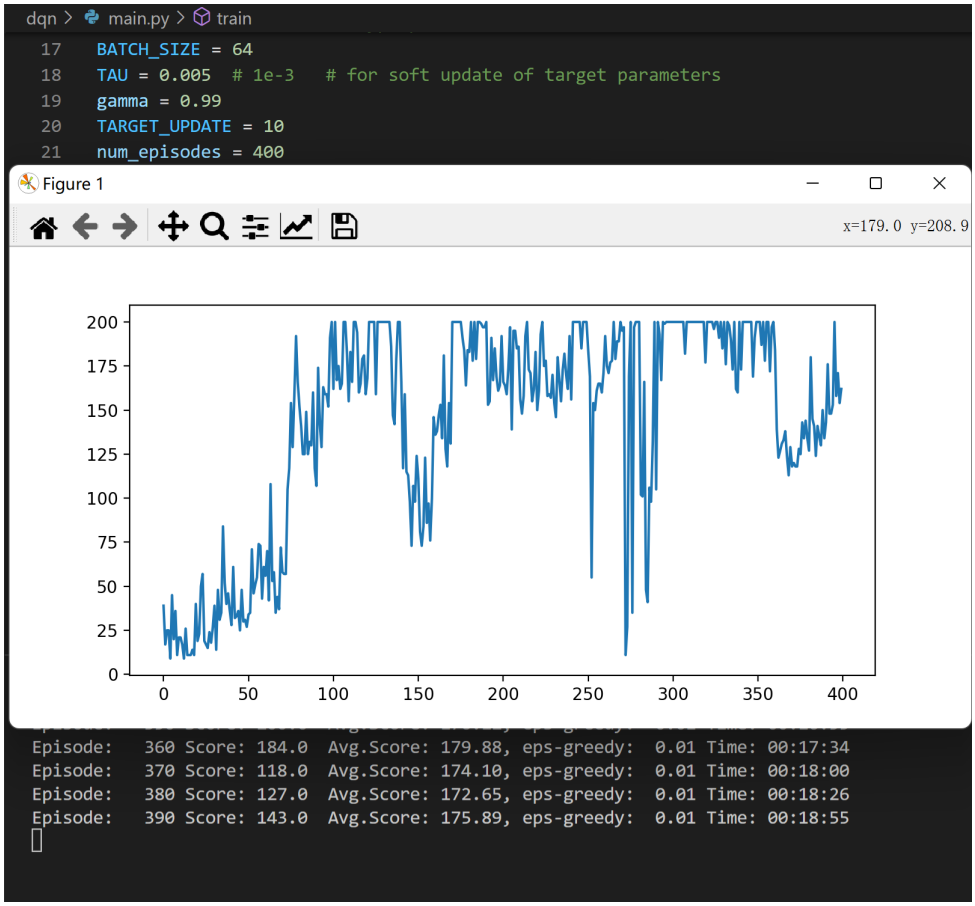
4. 优化

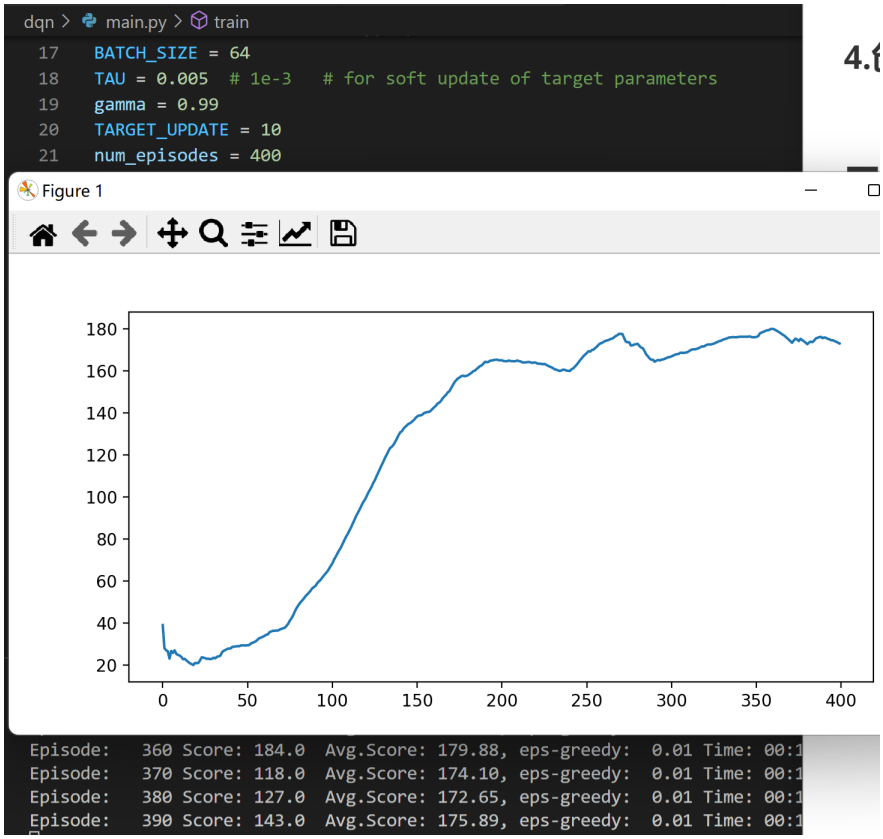
将DQN网络架构定义成三层MLP网络，以获得更好的拟合能力

三、实验结果及分析

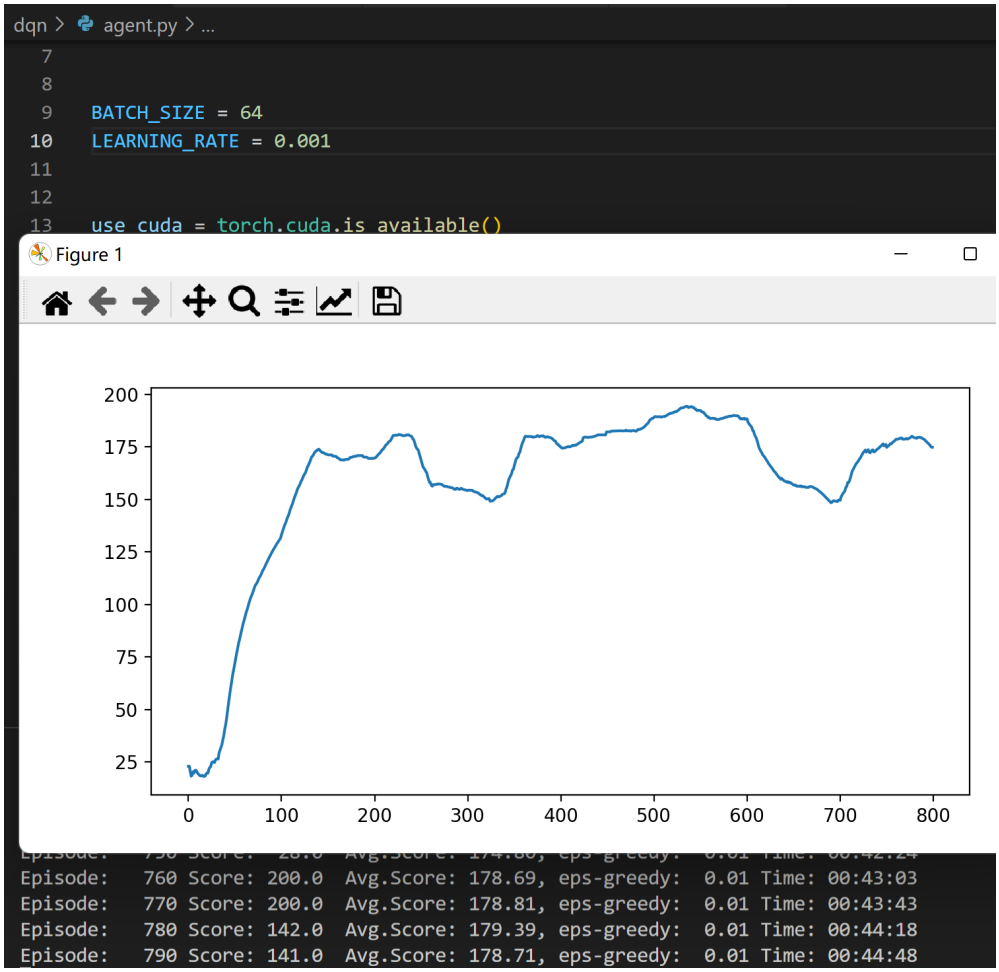
1. 实验结果展示示例（可图可表可文字，尽量可视化）

batch = 64， learning rate = 0.02





batch = 64，learning rate = 0.001



2.评测指标展示及分析（其它可分析运行时间等）

当batch_size为32时，平均得分收敛至150(lr=0.001)，或100(lr=0.01)

当batch_size为64时，平均得分收敛至175(lr=0.02)，或130(lr=0.01)，或在150至183之间来回波动(lr=0.001)

四、参考资料

PPT

dqn/dqn.py at master · tokb23/dqn (github.com)

Deep-Q-Network-Breakout/agent_dqn.py at master · ShanHaoYu/Deep-Q-Network-Breakout (github.com)

【强化学习】Deep Q-Network (DQN) - 知乎 (zhihu.com)

【动手学强化学习（1）DQN】CarPole DQN的实现 - 知乎 (zhihu.com)

PyTorch实现DQN强化学习 - 知乎 (zhihu.com)

【强化学习】Deep Q-Network (DQN) - 知乎 (zhihu.com)