人工智能实验报告

实验8　深度强化学习

陳日康　信息与计算科学　**22336049**

一、实验概述

用 Deep Q-learning Network (DON) 玩 CartPole-v1 游戏，框架代码已经给出，至少需要补充 'TODO' 标记的代码片段。

要求至少完成 500 局（Episodes）游戏内，达成一次：连续 10 局 Reward 值为 500，并展示 “单局 Reward 值” 曲线以及 “最近 100 局的平均 Reward 值” 曲线。

二、算法原理

深度 Q 网络（Deep Q-Network，DQN）是强化学习中的一种算法，用于解决具有高维状态空间的决策问题。其核心思想是结合深度学习和 Q 学习，通过神经网络近似 Q 值函数，从而能够在复杂环境中进行有效的决策。Q 学习是一种值迭代算法，旨在为每个状态-动作对 (s, a) 估计一个 Q 值，表示在状态 s 选择动作 a 后能获得的长期回报的期望值。传统的 Q 学习通过查找表的形式存储 Q 值，但在高维状态空间中，这种方法的存储和计算成本过高。DQN 通过使用深度神经网络来近似 Q 值函数，从而克服了这一限制。

在 DQN 中，核心组件包括经验回放和目标网络。经验回放机制通过存储智能体与环境交互的经验（即状态、动作、奖励、下一状态、是否结束），并在训练时从这些存储的经验中随机采样，以打破经验之间的相关性，提高数据样本的独立性和分布多样性。目标网络是 DQN 的一种稳定化技巧，使用两个网络——评估网络和目标网络。评估网络用于生成 Q 值，而目标网络则用于生成目标 Q 值。目标网络的参数在一定步数之后从评估网络复制，而非每次更新时都改变，这种做法减小了更新的不稳定性。

DQN的训练过程可以描述如下：

1. 初始化评估网络和目标网络的参数，初始化经验回放缓冲区。
2. 在每个时间步，使用 ε-贪婪策略选择动作，即以 ε 的概率选择随机动作，以 1-ε 的概率选择评估网络输出 Q 值最大的动作。随着训练进行，ε 逐渐减小，以保证探索和利用的平衡。
3. 执行动作，观察奖励和下一个状态，将这一经验存入回放缓冲区。
4. 从回放缓冲区随机采样一个小批量的经验，计算每个经验的目标 Q 值。对于非终止状态，目标 Q 值为当前奖励加上下一状态的最大 Q 值乘以折扣因子。对于终止状态，目标 Q 值仅为当前奖励。
5. 使用均方误差损失函数计算评估网络输出的 Q 值与目标 Q 值之间的差异，并通过反向传播更新评估网络的参数。
6. 每隔若干步，将评估网络的参数复制到目标网络。

DQN 通过以上过程不断更新网络参数，使得评估网络能够输出越来越准确的 Q 值估计，从而在复杂环境中实现有效的决策，展现了强大的表现力。

三、流程图

```
开始
|
|---> 导入所需库
|       |
|       |---> 导入Gym、PyTorch、Numpy等库
|
|---> 定义QNet模型
|       |
|       |---> 构建两层线性层和ReLU激活函数
|
|---> 定义ReplayBuffer
|       |
|       |---> 初始化缓冲区
|       |
|       |---> 定义存储和采样方法
|
|---> 定义DQN代理
|       |
|       |---> 初始化eval_net和target_net
|       |
|       |---> 定义epsilon-greedy动作选择
|       |
|       |---> 定义存储经验和学习方法
|
|---> 主训练循环
|       |
|       |---> 初始化环境和代理
|       |
|       |---> 运行多个ϵ
|       |       |
|       |       |---> 重置环境，初始化参数
|       |       |
|       |       |---> 在每个step中
|       |       |       |
|       |       |       |---> 根据ϵ-greedy策略选择动作
|       |       |       |
|       |       |       |---> 环境执行动作，获得新状态和奖励
|       |       |       |
|       |       |       |---> 存储经验到缓冲区
|       |       |       |
|       |       |       |---> 当缓冲区满时，代理进行学习
|       |       |
|       |       |---> 打印每个ϵ的奖励
|
|---> 绘制结果
|       |
|       |---> 绘制每个ϵ的Reward曲线
|       |
|       |---> 绘制Average reward曲线
|
结束
```

四、关键代码展示

`QNet` 类定义了一个简单的神经网络模型，继承 `Module`，包含两个全连接层，每个层之间使用 ReLU 激活函数，并重写前向传播。

```
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)   # 定义第一个全连接层
        self.fc2 = nn.Linear(hidden_size, output_size)  # 定义第二个全连接层

    def forward(self, x):
        x = torch.Tensor(x)       # 将输入数据转换为Tensor
        x = F.relu(self.fc1(x))  # 通过第一个全连接层并应用ReLU激活函数
        x = F.relu(self.fc2(x))  # 通过第二个全连接层并应用ReLU激活函数
        return x
```

`ReplayBuffer` 类实现了经验回放缓冲区，用于存储和采样智能体与环境交互的经验，包含一个列表作为缓冲区，具有固定容量。支持添加经验、随机采样经验和清空缓冲区等操作。

```
class ReplayBuffer:
    def __init__(self, capacity):
        # 初始化经验回放缓冲区和容量
        self.buffer = []
        self.capacity = capacity

    def len(self):
        # 返回缓冲区当前的大小
        return len(self.buffer)

    def push(self, *transition):
        # 如果缓冲区已满，则移除最早的一个
        if len(self.buffer) == self.capacity:
            self.buffer.pop(0)
        # 将新的转换（transition）添加到缓冲区
        self.buffer.append(transition)

    def sample(self, batch_size):
        # 从缓冲区中随机采样一个batch
        transitions = random.sample(self.buffer, batch_size)
        # 解包采样的转换数据
        state, action, reward, next_state, done = zip(*transitions)
        return np.array(state), action, reward, np.array(next_state), done

    def clean(self):
        self.buffer.clear()
```

`DQN` 类实现了深度 Q 网络，结合经验回放和目标网络机制进行强化学习训练。包含评估网络和目标网络，优化器、损失函数、经验回放缓冲区等属性。定义了选择动作、存储经验和学习过程等方法。为强化学习代理，通过与环境交互、存储经验和学习更新网络参数，学习到在不同状态下选择最优动作的策略。

```
class DQN:
    def __init__(self, env, input_size, hidden_size, output_size):
        # 初始化环境和网络
        self.env = env
        self.eval_net = QNet(input_size, hidden_size, output_size)       # 评估网
络
        self.target_net = QNet(input_size, hidden_size, output_size)     # 目标网
络
        self.optim = optim.Adam(self.eval_net.parameters(), lr=args.lr)  # 优化器
        self.eps = args.eps    # 探索概率
        self.buffer = ReplayBuffer(args.capacity)    # 经验回放缓冲区
        self.loss_fn = nn.MSELoss()                  # 损失函数
        self.learn_step = 0                          # 学习步数

    def choose_action(self, obs):
        # 选择动作
        if np.random.uniform() <= self.eps:
            # 随机选择动作
            action = np.random.randint(0, self.env.action_space.n)
        else:
            # 根据评估网络选择最优动作
            obs = torch.FloatTensor(obs).unsqueeze(0)
            action_values = self.eval_net(obs)
            action = torch.argmax(action_values).item()
        return action

    def store_transition(self, *transition):
        self.buffer.push(*transition)   # 存储转换(transition)

    def learn(self):
        # 训练DQN模型
        if self.eps > args.eps_min:
            # 随着训练逐渐减少探索概率
            self.eps *= args.eps_decay

        if self.learn_step % args.update_target == 0:
            # 每隔固定步数更新目标网络
            self.target_net.load_state_dict(self.eval_net.state_dict())
        self.learn_step += 1

        # 从缓冲区采样
        obs, actions, rewards, next_obs, dones =
self.buffer.sample(args.batch_size)
        actions = torch.LongTensor(actions)
        dones = torch.FloatTensor(dones)
        rewards = torch.FloatTensor(rewards)

        # 计算Q值
        q_eval = self.eval_net(np.array(obs)).gather(1,
actions.unsqueeze(1)).squeeze(1)
```

```
        q_next = torch.max(self.target_net(np.array(next_obs)), dim=1)[0]
        q_target = rewards + args.gamma * (1 - dones) * q_next

        # 计算损失并反向传播
        dqn_loss = self.loss_fn(q_eval, q_target)
        self.optim.zero_grad()
        dqn_loss.backward()
        self.optim.step()
```

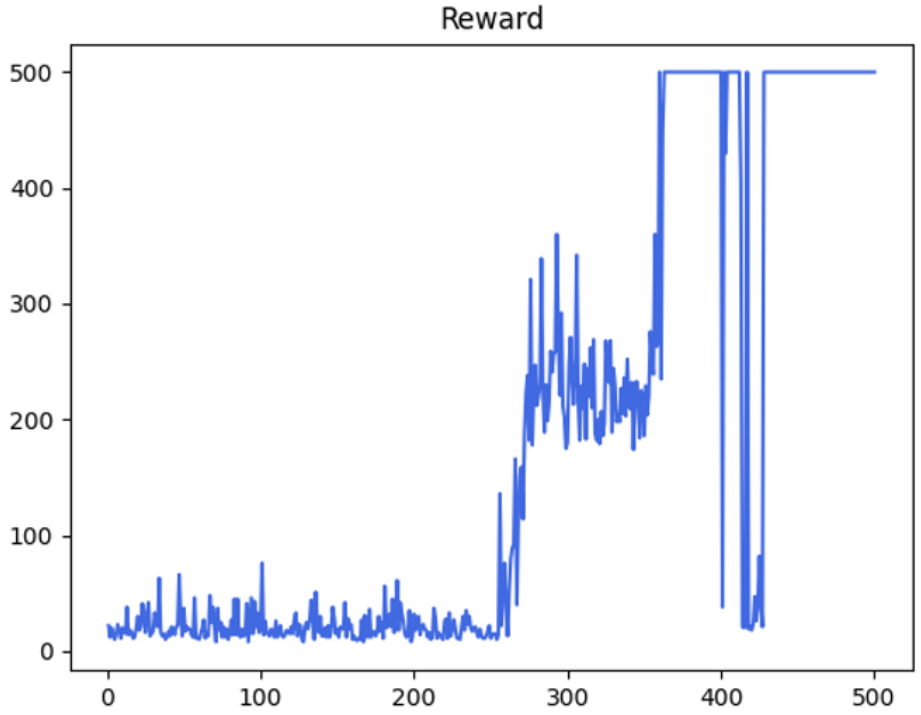
五、实验结果与分析

5.1 实验结果展示示例（实验结果放入 Result 文件夹中）

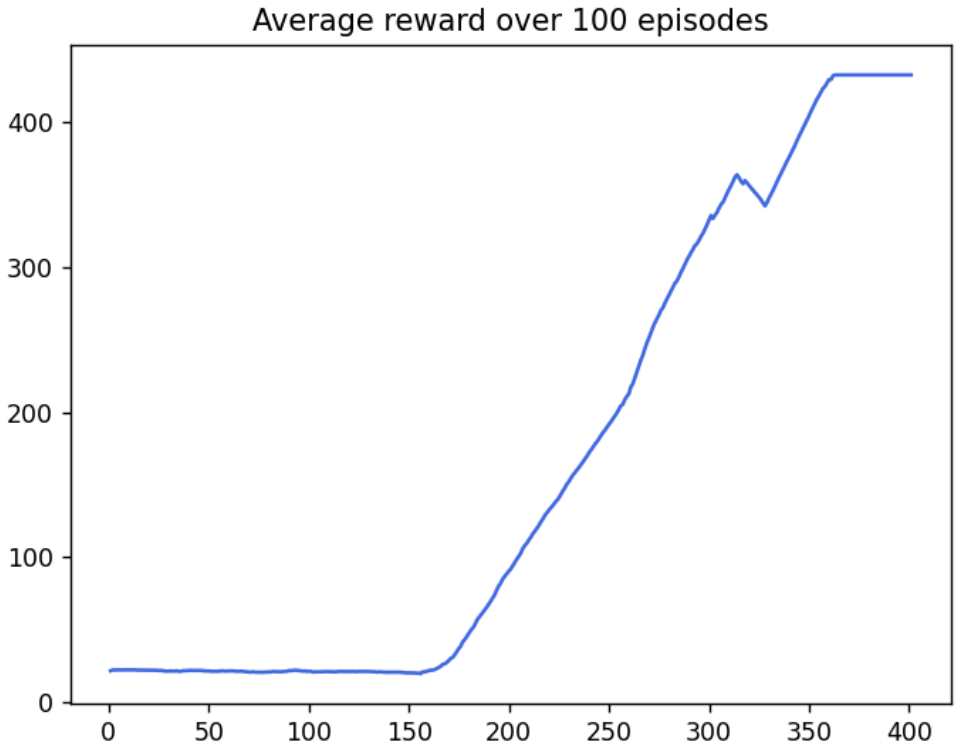
达成连续 10 局 reward 值为 500：

```
Episode: 440, Reward: 500.0
Episode: 441, Reward: 500.0
Episode: 442, Reward: 500.0
Episode: 443, Reward: 500.0
Episode: 444, Reward: 500.0
Episode: 445, Reward: 500.0
Episode: 446, Reward: 500.0
Episode: 447, Reward: 500.0
Episode: 448, Reward: 500.0
Episode: 449, Reward: 500.0
```

Reward 值随训练次数的变化：



近百局 Reward 值平均值：



5.2 评价指标展示及分析

1. 前 200 多局的 reward 值较低，在大约 267 步左右开始上升，这是通过前 200 多步的经验继续学习优化神经网络，使得神经网络可以根据环境值输出较好的结果。
2. 后 200 多步 reward 的值出现了反复升降的现象，但依然仍处于较高的范围。而在 427 步后连续70多步 reward 值步达到 500 的情况，说明经过学习后能连续多次的表现出良好的效果。

六、参考资料

1. https://blog.csdn.net/beiketaoerge/article/details/135611641?ops_request_misc=%257B%2522request%255Fid%2522%253A%252217185944821680018068 4630%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=171859448216800180684630&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_positive~default-2-135611641-null-null.142^v100^pc_search_result_base8&utm_term=DQN&spm=1018.2226.3001.4187