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

学号	23336179	姓名	马福泉

一、 实验题目

实现 DQN 算法

在 CartPole-v0 环境中实现 DQN 算法。最终算法性能的评判标准：以算法收敛的 reward 大小、收敛所需的样本数量给分。 reward 越高（至少是 180，最大是 200）、收敛所需样本数量越少，分数越高。

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 reward 曲线图以及算法细节。

相关代码下载地址：
https://github.com/ZYC9894/2024AI/tree/main/Homework/Experiment

相关环境的说明文档：https://www.gymlibrary.dev/ 或
https://www.gymlibrary.dev/environments/classic_control/cart_pole/

Supplement

我们给出 DQN 在 cartpole 环境的训练曲线图作为参考。

二、 实验内容

1. 算法原理

（2） 主程序模块 (main.py)

① 参数解析(parse 函数):

使用 argparse 模块定义和解析命令行参数
包含训练参数(学习率、批次大小、缓冲区大小等)
包含网络参数(隐藏层大小、更新频率等)
包含探索参数(epsilon 及其衰减)

② 运行控制(run 函数):

对每个随机种子运行独立实验
初始化环境和 DQN 代理
调用训练过程
收集并可视化结果

③ 结果可视化(visualize_results 函数):
计算奖励的标准差
生成两种可视化图表:
所有实验的奖励曲线和标准差曲线
子图布局的详细分析(各实验曲线+标准差)

（2） DQN 代理模块 (agent_dqn.py)

① Q 网络(QNetwork 类):
实现深度 Q 网络的结构
包含两个隐藏层和一个输出层，使用 ReLU 激活函数

② 经验回放缓冲区(ReplayBuffer 类):
使用 deque 实现固定大小的缓冲区
提供 push、sample 和 clean 方法
支持随机采样训练数据

③ DQN 代理(AgentDQN 类):
初始化 Q 网络和目标网络
配置优化器和超参数

- 动作选择(make_action 方法):

实现 ε -greedy 策略

- 训练过程(run 方法):与环境交互收集经验

存储经验到回放缓冲区
定期更新网络参数
实现 epsilon 衰减。

- 网络更新(_update_network 方法):

从缓冲区采样批次数据
计算当前 Q 值和目标 Q 值
使用 MSE 损失更新网络
实现目标网络定期更新

2. 关键代码展示（可选）

（1） 使用 argparse 库来定义和解析运行深度 Q 网络（DQN）训练所需的各种配置参数。它设置了参数的名称、类型、默认值和帮助信息，使得用户可以通过命令行灵活地指定训练配置，如环境名称、网络结构、学习率、折扣因子等。解析完成后，将这些参数打包成一个命名空间对象并返回，以便在程序的其他部分使用。

```python
def parse():
    parser = argparse.ArgumentParser(description="SYSU_AI_DQN")
    parser.add_argument('--train_dqn', default=True, type=bool, help='whether train DQN')
    parser.add_argument('--env_name', default="CartPole-v0", type=str, help='environment name')
    parser.add_argument('--hidden_size', default=128, type=int, help='neural network hidden size')
    parser.add_argument('--lr', default=0.001, type=float, help='learning rate')
    parser.add_argument('--lr_min', default=0.0001, type=float, help='minimum learning rate')
    parser.add_argument('--lr_decay', default=0.9, type=float, help='learning rate decay factor')
    parser.add_argument('--lr_decay_freq', default=500, type=int, help='learning rate decay frequency')
    parser.add_argument('--gamma', default=0.95, type=float, help='discount factor')
    parser.add_argument('--n_frames', default=50, type=int, help='number of frames to train')
    parser.add_argument('--batch_size', default=128, type=int, help='batch size')
    parser.add_argument('--buffer_size', default=4000, type=int, help='replay buffer size')
    parser.add_argument('--update_target_freq', default=10, type=int, help='target network update frequency')
    parser.add_argument('--epsilon', default=1.0, type=float, help='initial exploration rate')
    parser.add_argument('--epsilon_min', default=0.0001, type=float, help='minimum exploration rate')
    parser.add_argument('--epsilon_decay', default=0.996, type=float, help='exploration rate decay factor')
    parser.add_argument('--seeds', nargs="+", type=int, default=[1111, 2222, 6179], help='random seeds')
    parser.add_argument('--log_dir', default="./result", type=str, help='directory for storing results')

    args = parser.parse_args()
    return args
```

（2） run 函数：

初始化奖励记录列表：用于收集所有实验的奖励数据。

循环执行实验：对每个随机种子，执行以下步骤：

创建环境实例，用于与智能体交互。

设置当前实验的随机种子，以确保结果的可重复性。

初始化 DQN 智能体，并根据命令行参数配置其属性。

如果指定训练 DQN，则调用智能体的 train 方法进行训练，并收集训练过程中的奖励记录。

收集实验结果：将每次实验的奖励记录添加到结果列表中。

结果可视化：调用 visualize_results 函数，将收集到的实验结果进行可视化处理，并将图表保存到日志目录中。

```python
def run(args):
    os.makedirs(args.log_dir, exist_ok=True)
    experiment_results = []  # 存储所有实验的奖励记录
    # 对于每个种子，运行一次完整实验
    for current_seed in args.seeds:
        # 创建环境
        env = gym.make(args.env_name, render_mode="rgb_array")
        # 设置种子
        args.seed = current_seed

        # 创建并训练DQN智能体
        agent = AgentDQN(env, args)

        if args.train_dqn:
            agent.train()
            episode_returns = agent.historical_returns  # 获取训练过程中的奖励记录
            experiment_results.append(episode_returns)

    # 可视化结果
    visualize_results(experiment_results, args.log_dir)
```

（3） QNetwork 类

- 初始化（__init__ 方法）：

定义了三个全连接层（nn.Linear）：

layer1：输入层，将环境的观测状态（input_size）映射到隐藏层（hidden_size）。

layer2：第二个隐藏层，进一步处理来自第一个隐藏层的特征。

head：输出层，将第二个隐藏层的输出映射到动作空间的大小（output_size），即每个可能动作的 Q 值。

这些层在初始化时被注册为模型的子模块，PyTorch 会自动管理它们的参数。

- 前向传播（forward 方法）：

定义了数据通过网络的前向传播过程：

输入数据首先通过 layer1，然后应用 ReLU 激活函数，增加网络的非线性表达能力。

接着，经过 layer2 和 ReLU 激活函数，进一步处理特征。

最后，通过 head 层输出每个动作的 Q 值。

这个方法在每次调用模型进行预测时被自动执行。

```python
class QNetwork(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(QNetwork, self).__init__()
        self.layer1 = nn.Linear(input_size, hidden_size)
        self.layer2 = nn.Linear(hidden_size, hidden_size)
        self.head = nn.Linear(hidden_size, output_size)

    def forward(self, inputs):
        hidden1 = torch.relu(self.layer1(inputs))
        hidden2 = torch.relu(self.layer2(hidden1))
        return self.head(hidden2)
```

（4） ReplayBuffer 类

- 初始化：设置缓冲区的容量（buffer_size），即最多可以存储的经验数量。

创建一个 deque 对象作为存储结构，它是一个双端队列，可以高效地从两端添加或删除元素。设置其最大长度为缓冲区容量，以确保缓冲区不会无限增长。

- 获取长度（__len__ 方法）：

返回缓冲区当前存储的经验数量。

- 添加经验（push 方法）：

将新的经验（如状态转移元组）添加到缓冲区中。如果缓冲区已满，旧的经验将被自动替换。

- 采样（sample 方法）：

从缓冲区中随机抽取指定数量（batch_size）的经验样本。这些样本用于训练神经网络，以更新智能体的策略。

```python
class ReplayBuffer:
    def __init__(self, buffer_size):
        self.capacity = buffer_size
        self.storage = deque(maxlen=buffer_size)

    def __len__(self):
        return len(self.storage)

    def push(self, *transition):
        self.storage.append(transition)

    def sample(self, batch_size):
        return random.sample(self.storage, batch_size)

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

（5） AgentDQN 类

篇幅问题，展示伪代码

```
初始化智能体:
    设置环境和参数
    设置随机种子
    选择设备(CPU 或 GPU)
    初始化经验回放缓冲区
    初始化 Q 网络和目标 Q 网络
    初始化优化器
    初始化训练参数
训练智能体:
    对于每个帧:
        重置环境状态
        对于每个 episode 直到达到最大帧数:
            如果未结束:
                根据当前策略选择动作
                执行动作并观察结果
                将经验存储到缓冲区
                如果缓冲区足够大:
                    从缓冲区中采样一批经验
                    更新 Q 网络
```

```
                    如果达到更新频率:
                        更新目标 Q 网络
                    更新探索率
                如果结束:
                    记录奖励并重置环境状态
            每 10 个 episode 打印一次进度和平均奖励
    选择动作:
        如果测试或探索率大于随机数:
            选择具有最大 Q 值的动作
        否则:
            随机选择动作
    更新 Q 网络:
        从缓冲区中采样一批经验
        计算当前 Q 值和目标 Q 值
        计算损失并更新 Q 网络参数
    调整学习率:
        根据衰减因子更新优化器的学习率
```

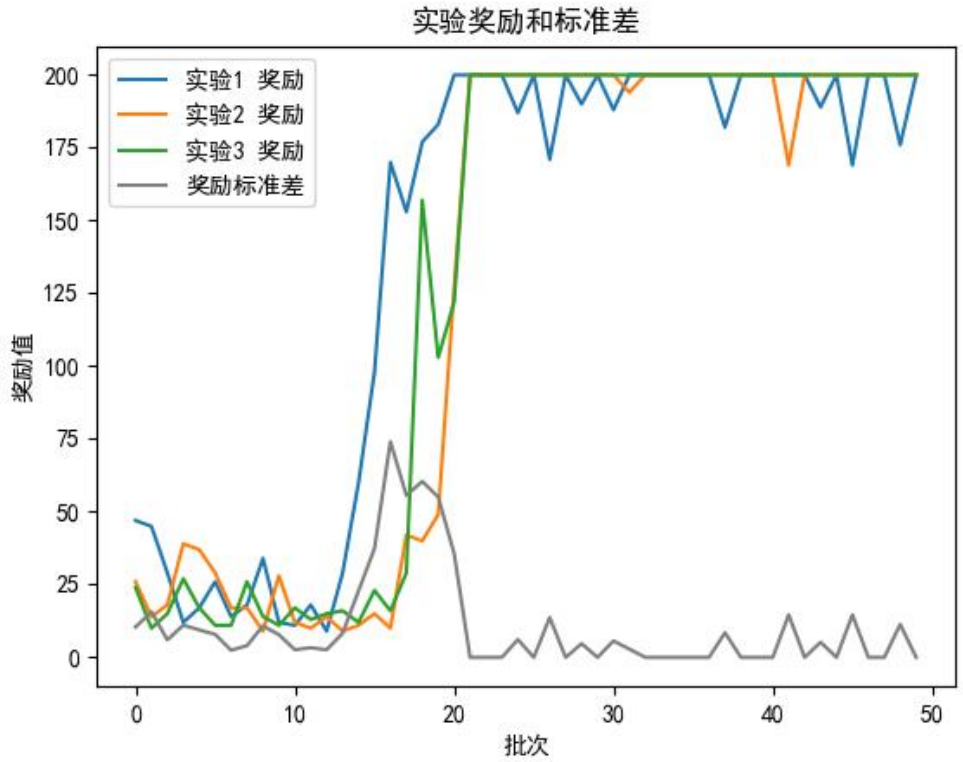
3. 创新点&优化（如果有）

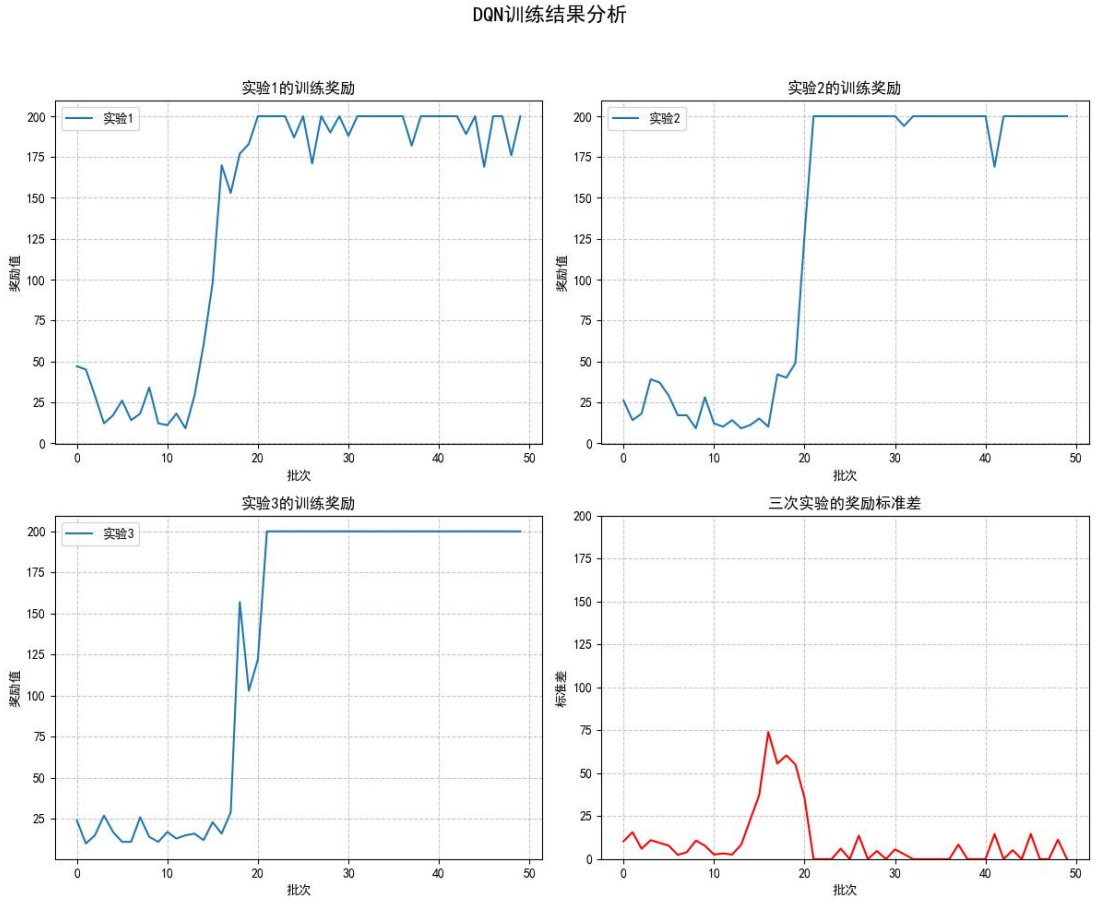
- 学习率衰减：实现了学习率衰减机制，有助于在训练后期进行更精细的优化。
- 随机种子设置：设置了随机种子，确保了实验结果的可重复性。
- 奖励记录和可视化：记录了训练过程中的奖励，并实现了结果的可视化，便于分析和调试。
- 参数化配置：使用 argparse 库来解析命令行参数，使得实验配置灵活且易于调整。

三、 实验结果及分析

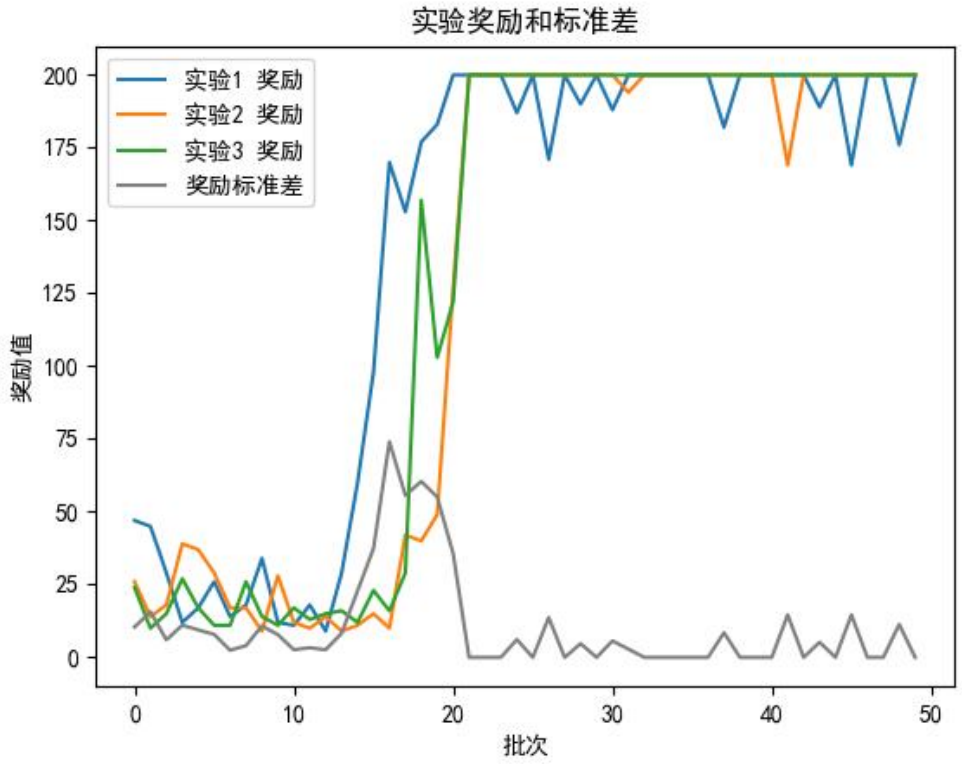
1. 实验结果展示示例（可图可表可文字，尽量可视化）

```
Rewards: [[ 12.  31.  18.  48.  16.  25.  24.  17.  16.  19.  17.  33.  41.  34.
   41.  54. 155. 162. 191. 175. 190. 182. 158. 190. 175. 200. 167. 146.
  174. 200. 174. 181. 178. 200. 200. 167. 200. 184. 177. 159. 193. 180.
  190. 184. 150. 175. 169. 197. 164. 149.]
 [ 20.  14.  14.  32.  12.  19.  42.  13.  22.   9.  14.  17.  11.  24.
   24.  34.  10.  32.  58. 200. 200. 200. 200. 200. 200. 200. 200. 200.
  200. 200. 200. 200. 200. 200. 200. 200. 200. 200. 200. 200. 200. 200.
  200. 200. 200. 200. 200. 200. 200. 200.]
 [ 26.  28.  45.  27.  73.  10.  15.  12.  24.  20.  15.  56.  62.  44.
  200. 200. 200. 200. 200. 200. 200. 200. 200. 200. 200. 200. 200. 200.
  200. 200. 200. 200. 200. 200. 200. 200. 200. 200. 200. 200. 200. 200.
  200. 200. 200. 200. 200. 200. 200. 200.]]
```

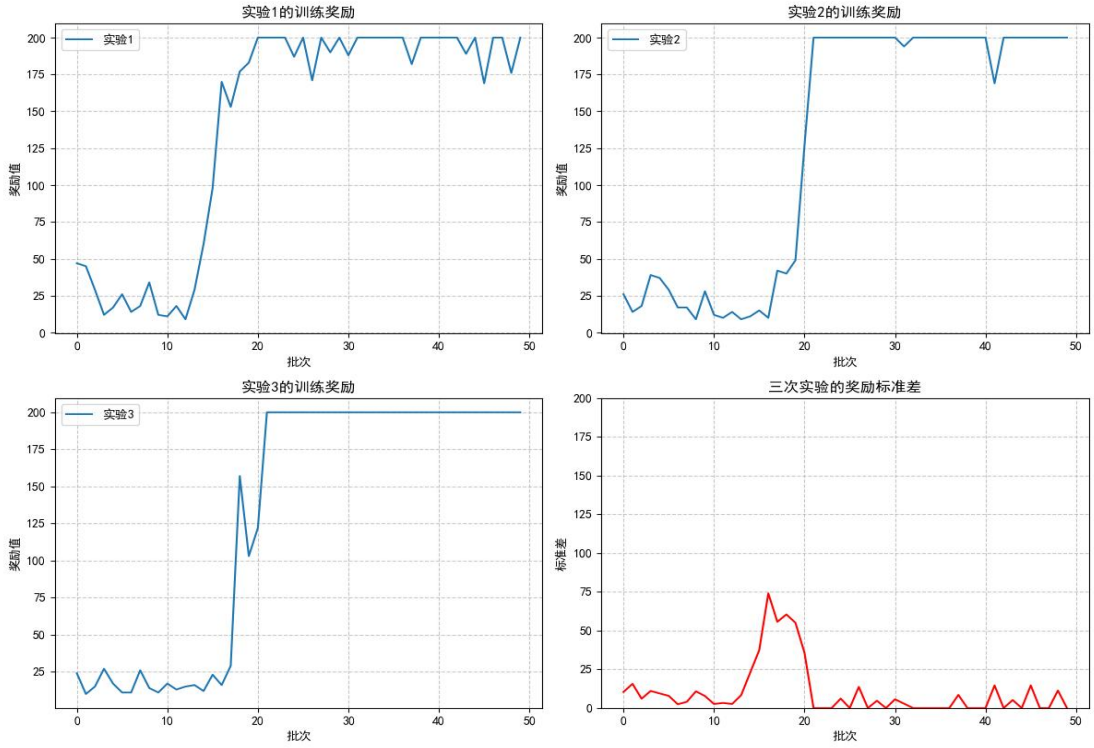

实验奖励和标准差
实验1 奖励
实验2 奖励
实验3 奖励
奖励标准差
奖励值
批次


DQN训练结果分析
实验1的训练奖励
实验1
奖励值
批次
实验2的训练奖励
实验2
奖励值
批次
实验3的训练奖励
实验3
奖励值
批次
三次实验的奖励标准差
标准差
批次

2. 评测指标展示及分析（机器学习实验必须有此项，其它可分析运行时间等）



DQN训练结果分析



可以看到实验 1-3 结果较为一致

- 收敛的 reward 大小：实验的奖励值最终稳定在 200。
- 收敛所需的样本数量：实验在大约第 20 个批次后开始显著提高，并在第 30 个批次左右达到稳定状态。
- 三次实验的奖励标准差：从标准差图表可以看出，实验 2 在中期有较大的波动，这可能表明算法在该阶段的稳定性稍差。实验 1 和实验 3 的标准差相对较小，表明算法在这些实验中的表现更为稳定。
- 总体评估

算法性能：所有实验的算法性能都很好，奖励值都达到了 180 以上。且达到了最大奖励 200。
收敛速度：所有实验都在大约 30 个批次后收敛，收敛速度较快。
稳定性：实验 1 和实验 3 的表现更为稳定，实验 2 在中期虽然有较大的波动，但是从三次实验的奖励标准差来看，实验较为稳定

四、 参考资料

1. 部分 debug，代码优化建议参考了 ai 大模型的建议。

2. 阅读参考了以下文章：

深度强化学习——DQN 算法原理-CSDN 博客

强化学习 4：DQN 算法-CSDN 博客

通俗讲解深度强化学习经典算法——DQN_dqn 算法-CSDN 博客