强化学习吃豆人DQN实现

——基于深度Q网络(DQN)的Atari游戏智能体 实验报告

1. 实验概述

本实验实现了深度 Q 网络(Deep Q-Network, DQN)算法，用于训练智能体玩 Atari 游戏平台中的吃豆人(Ms. Pac-Man)游戏。通过将深度学习与强化学习相结合，我们成功训练了一个能够从原始游戏画面像素中学习策略的智能体，玩吃豆人稳定在 4000-6000 分，有些轮次还跑出 **6700** 分。
实验基于 Gymnasium 环境，使用 PyTorch 框架构建和训练神经网络。

2. DQN算法原理

2.1 Q-Learning 回顾

传统 Q-Learning 算法维护一个 Q 表，用于存储每个状态-动作对的价值估计。其更新公式为：

$$\boldsymbol{Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]}$$

其中：

- $Q(s,a)$ 是状态 s 下采取动作 a 的估计价值
- α 是学习率
- r 是即时奖励
- γ 是折扣因子
- $\max_{a'} Q(s',a')$是下一状态的最大 Q 值

2.2 DQN 核心创新

DQN 算法对传统 Q-Learning 进行了两项关键改进：

1. **深度神经网络函数逼近**：使用卷积神经网络处理高维状态空间（如游戏画面），将状态映射到各个动作的 Q 值。
2. **经验回放(Experience Replay)**：存储智能体的经验 (s,a,r,s')到回放缓冲区，随机采样进行学习，打破样本间的相关性。
3. **目标网络(Target Network)**：维护一个单独的目标 Q 网络，周期性地从当前 Q 网络复制参数，稳定训练过程。

2.3 算法流程

1. 初始化回放缓冲区 D 和参数 θ
2. 对每个 episode：
    - 获取初始状态 s
    - 对每个时间步 t：
        - 基于 ε-greedy 策略选择动作 a

- 执行动作 a，获得奖励 r 和下一状态 s'
- 将经验(s,a,r,s')存入回放缓冲区 D
- 从 D 中采样小批量经验
- 计算目标值 $y = r + \gamma \cdot \max_{a'} Q(s',a';\theta^-)$
- 进行梯度下降优化$(y - Q(s,a;\theta))^2$
- 每 C 步更新目标网络参数 $\theta^- \leftarrow \theta$

3. 实验实现

3.1 神经网络架构

本实验中，DQN 的核心是一个卷积神经网络，负责从输入游戏画面中提取特征并估计各个动作的 Q 值：

```
class QNetwork(nn.Module):
    def __init__(self, env):
        super().__init__()
        self.network = nn.Sequential(
            #nn.Conv2d(4, 32, 8, stride=4),
            # 第一个卷积层：输入 4 通道(4 帧叠加)，输出 32 通道，卷积核 8×8，步长 4
            nn.Conv2d(4, 32, kernel_size=8, stride=4),
            nn.ReLU(),

            # 第二个卷积层：输入 32 通道，输出 64 通道，卷积核 4×4，步长 2
            nn.Conv2d(32, 64, kernel_size=4, stride=2),
            nn.ReLU(),

            # 第三个卷积层：输入 64 通道，输出 64 通道，卷积核 3×3，步长 1
            nn.Conv2d(64, 64, kernel_size=3, stride=1),
            nn.ReLU(),

            # 展平卷积层输出
            nn.Flatten(),

            # 全连接层：卷积输出维度为 7×7×64=3136
            nn.Linear(3136, 512),
            nn.ReLU(),

            # 输出层：对应动作空间大小
            nn.Linear(512, env.single_action_space.n),
        )

    def forward(self, x):
        return self.network(x / 255.0)
```

这一网络架构与原始 DQN 论文中用于 Atari 游戏的网络结构一致。

3.2 环境预处理

为了使游戏画面更适合网络处理，实验中采用了以下预处理步骤：

- **NoopResetEnv**：游戏开始时执行随机数量的空操作
- **MaxAndSkipEnv**：跳帧，每 4 帧选择一帧处理
- **EpisodicLifeEnv**：游戏生命结束作为 episode 终止条件
- **FireResetEnv**：某些游戏需要按 Fire 键开始
- **ClipRewardEnv**：将奖励裁剪为{-1, 0, 1}
- **ResizeObservation**：调整画面大小为 84×84
- **GrayScaleObservation**：转换为灰度图
- **FrameStack**：将连续 4 帧堆叠作为状态输入

3.3 经验回放实现

```
rb = ReplayBuffer(
    args.buffer_size,
    envs.single_observation_space,
    envs.single_action_space,
    device,
    optimize_memory_usage=True,
    handle_timeout_termination=False
)
```

使用大小为 100 万的回放缓冲区存储和采样经验，解耦状态转移的时间相关性。

3.4 训练流程

训练过程中的关键步骤：

1. **探索策略**：使用线性衰减的 ε-greedy 策略，从 1.0 降至 0.01
2. **经验存储**：每步交互将(s, a, r, s')存入回放缓冲区
3. **批量学习**：每 4 步从缓冲区采样 32 个样本进行训练
4. **目标计算**：使用目标网络计算 TD 目标 $r+\gamma \max_{a'} Q(s',a')$
5. **网络更新**：使用 MSE 损失更新 Q 网络参数
6. **目标网络更新**：每 1000 步软更新目标网络

4. 实验结果与分析

4.1 训练过程

训练过程中，智能体学习效果显著:

- 初始阶段：前几十万步，随机行为，得分仅几百分
- 中期阶段：一两百万步，开始学习基本策略，得分有所提高，能稳定玩到一两千分
- 后期阶段：掌握游戏规则，知道主动吃强化豆然后吃幽灵；得分稳定提升至四千到六千多分

4.2 训练表现

训练过程中记录了以下关键数据:

- 最高得分：6700
- 平均表现：训练后期稳定在 4000-6000 分范围
- 完成的 episode 数：显著增加。五百万步可达 18000，一千万步可达 32600.

4.3 实验中遇到的挑战

1. 游戏本身存在随机性，需要更频繁地保存结果，以减少随机因素影响
2. 训练初期表现不稳定，得分波动较大
3. 训练后期学习曲线出现波动，表明探索与利用的平衡仍有优化空间
4. GPU 资源利用不够充分，存在性能优化空间

5. 结论与改进方向

5.1 结论

本实验成功实现了 DQN 算法并应用于 Ms. Pac-Man 游戏，验证了深度强化学习在复杂视觉控制任务上的有效性。智能体能够从原始像素输入中学习游戏策略，并取得较好的得分表现。

5.2 改进方向

1. **算法改进：**
    - 实现 Double DQN 减少 Q 值过估计
    - 添加优先级经验回放(Prioritized Experience Replay)
    - 尝试 Dueling DQN 架构
2. **训练优化：**
    - 优化批大小和环境数量提高 GPU 利用率
    - 调整学习率和目标网络更新频率
    - 延长训练时间获得更好性能
3. **可视化改进：**
    - 实现实时学习曲线绘制
    - 自动保存最高分视频
    - 添加注意力可视化分析智能体决策

通过本实验，我理解了 DQN 算法的核心思想和实现细节，为进一步研究和改进深度强化学习算法奠定基础。