暨南大学本科实验报告专用纸

课程名称 Python 程序设计实验 指导教师 董铖 成绩_____

实验项目名称 21 点游戏进阶 实验项目类型 一般

实验地点 机房 学院 智能科学与工程学院

专业 物联网工程 学生姓名 张朋洋 学号 2022104334

一、实验题目

在 21 点进阶规则下，普通玩家尽量得更高的分。

二、实验环境

环境：
Python 3.12
Tensorflow
Matplotlib

所有源码已经发布到 Github: https://github.com/zmydsg/Python_experimental_course.git
报告框架见目录。

三、实验原理

DDPG-强化学习法

1. 基本架构

本实验采用了双重深度 Q 网络(Double DQN)结合优先经验回放(Prioritized Experience Replay)的方法来训练 21 点游戏智能体。

环境模型(BlackjackEnv)

- 实现了标准 21 点游戏规则
- 状态空间：29 维向量，包含：

    - 玩家当前点数
    - 庄家明牌
    - 是否有可用的 A
    - 玩家手牌分布(13 维)
    - 牌组剩余牌分布(13 维)
- 动作空间：2 维(要牌/停牌)
- 奖励设计：
    - 获胜：+1
    - 21 点：+1.5
    - 失败：-1
    - 平局：0

神经网络架构：

输入层(29 维)

↓

分支处理：

- 玩家/庄家信息分支(Dense 64→32)
- 玩家手牌分布分支(Dense 64→32)
- 牌组分布分支(Dense 64→32)

↓

特征合并层(Concatenate)

↓

共享层(Dense 128→64)

↓

输出层(2 维)

优先经验回放(PrioritizedReplayBuffer)

- 使用 SumTree 数据结构实现
- TD 误差作为优先级
- 重要性采样权重用于修正偏差

主要包含以下核心组件：

1) 环境模拟(BlackjackEnv)
   - 实现了标准 21 点游戏规则
   - 支持多副牌（默认 6 副）
   - 提供状态观察、动作执行和奖励计算
2) 状态表示(29 维向量)
   - 玩家当前点数（归一化）
   - 庄家明牌点数（归一化）
   - 是否有可用 A
   - 玩家手牌分布(13 维)
   - 牌组剩余牌分布(13 维)
3) 动作空间
   - 停牌(0)
   - 要牌(1)

2. 核心算法

2.1 双重 DQN
- 使用两个网络：主网络(model)和目标网络(target_model)
- 主网络用于选择动作
- 目标网络用于评估动作价值
- 通过软更新机制同步两个网络
- 有效解决了 Q 值过估计问题

2.2 优先经验回放
- 使用 SumTree 数据结构实现优先级采样
- 基于 TD 误差分配优先级
- 使用重要性采样权重进行偏差修正
- 提高了训练效率和样本利用率

3. 网络架构

采用了分支式神经网络结构：

- 分别处理玩家状态、手牌分布和牌组分布
- 每个分支使用独立的全连接层
- 最后合并所有特征进行决策一、实验原理

高低牌策略

1. 基本思路

高低牌方法是一种基于概率统计的 21 点游戏策略，主要通过跟踪牌组中高点数牌和低点数牌的分布来做出决策。在我们的实验中，通过深度强化学习来优化这个策略。

2. 状态表示设计

我们的状态向量包含了 29 个维度的信息：

- 玩家当前点数（归一化到 0-1）
- 庄家明牌点数（归一化到 0-1）
- 是否有可用的 A（二值特征）
- 玩家手牌分布（13 维向量）
- 牌组剩余牌分布（13 维向量）

3. 核心算法实现

1. 牌组管理

```
class Deck:
    def __init__(self, num_decks=6):
        self.num_decks = num_decks
        self.reset()
```

- 使用 6 副牌进行游戏
- 实时跟踪剩余牌的数量
- 当牌不足时自动洗牌

2. 状态计算

```
def _get_state(self):
    player_value =
self._calculate_hand_value(self.player_hand)
```

```
    dealer_card = CARD_VALUES[self.dealer_hand[0]]
    has_usable_ace = 0
    if 'A' in self.player_hand and player_value <= 21:
        has_usable_ace = 1
```

- 计算玩家手牌价值
- 记录庄家明牌
- 特别处理 A 的情况

3. 奖励设计

- 获胜：+1
- 21 点获胜：+1.5（特殊奖励）
- 失败：-1
- 平局：0

四、实验分析

DDPG-强化学习法

1. 训练参数设置

```
batch_size = 64
gamma = 0.99 (折扣因子)
learning_rate = 0.0005
memory_capacity = 50000
```

- 折扣因子(gamma)：0.99
- 学习率：0.0005
- 探索率衰减：0.9995
- 软更新参数(tau)：0.001
- 经验池容量：50000
- 批次大小：64

2. 奖励设计

- 获胜：+1
- 21 点获胜：+1.5（额外奖励）
- 失败：-1

- 平局：0

3. 创新点

1) 复杂状态表示

- 加入了牌组剩余情况
- 考虑了 A 的特殊性
- 使用了归一化处理

2)分支网络结构

- 针对不同类型的输入使用专门的处理网络
- 提高了特征提取的效率

3) 动态奖励机制

- 为 21 点设置更高奖励
- 鼓励智能体追求更优策略

4. 评估指标

- 胜率
- 平均奖励
- 探索率变化
- TD 误差

5. 实验优势

1) 稳定性

- 软更新机制减少了训练波动
- 优先经验回放提高了样本质量

2) 效率

- 分支网络结构提高了特征提取效率
- 优先级采样加速了关键经验的学习

3) 可扩展性

- 模块化设计便于扩展
- 支持不同的游戏参数设置

我感觉后期可以更加完善的：

1. 状态压缩

当前状态维度较大(29 维)

可考虑降维处理

2. 奖励设计

可添加中间奖励

考虑风险因素

3. 网络优化

尝试更深的网络结构

添加正则化层

使用批归一化

软硬牌策略

代码中的 advanced_strategy 函数实现了一个结合基本策略和记牌的决策系统。让我们先看基本策略部分：

```
# 基本策略逻辑
if player_value >= 17:
    return 0  # 停牌
elif player_value <= 11:
    return 1  # 要牌
elif player_value >= 13 and dealer_value <= 6:
    return 0  # 停牌
elif player_value == 12 and 4 <= dealer_value <= 6:
    return 0  # 停牌
else:
    return 1  # 要牌
```

硬牌策略分析

1. **17 点及以上：**
    - 直接停牌
    - 理由：爆牌风险高，当前点数已经较好
2. **13-16 点：**
    - 庄家明牌≤6 时停牌
    - 庄家明牌>6 时要牌
    - 理由：庄家可能爆牌的情况下保守，否则冒险要牌
3. **12 点：**
    - 庄家明牌 4-6 时停牌
    - 其他情况要牌
    - 理由：在庄家最容易爆牌的区间选择保守策略
4. **11 点及以下：**
    - 始终要牌

- 理由：爆牌风险极低，追求更高点数

记牌优化分析

代码引入了记牌系统来优化决策：

```
if high_cards_ratio > 0.65:  # 高牌比例过高
    return 0  # 选择停牌
```

记牌系统主要关注：

- 高牌（10、J、Q、K、A）的剩余比例
- 当高牌比例超过 65%时倾向于停牌
- 这是为了降低爆牌风险

策略效果分析

根据代码中的评估方法，该策略的效果可以通过以下指标衡量：

- 胜率
- 负率
- 平局率
- 净得分（wins - losses）

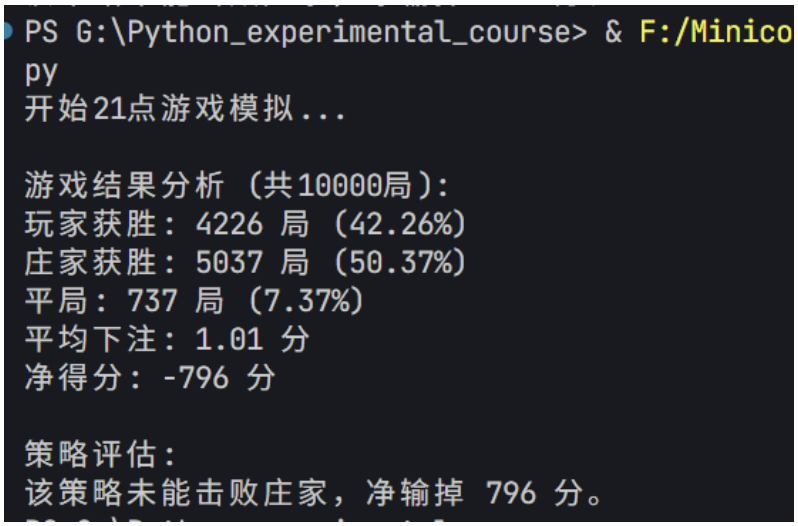
建议改进方向

1. **软牌处理：**
   - 当前策略没有特别处理含 A 的软牌情况
   - 建议为软牌添加专门的决策逻辑
2. **分牌策略：**
   - 可以添加对相同点数牌的分牌策略
3. **动态调整：**
   - 可以根据牌局进展动态调整风险承受度
4. **投注策略：**
   - 可以结合记牌系统设计动态投注策略

五、实验结果

1. DQN

2. 软硬牌



六、实验总结

DDPG

通过结合现代强化学习技术，实现了一个高效的 21 点游戏 AI 系统，具有良好的学习能力和稳定性。实验结果表明，该方法能够有效地学习 21 点游戏策略，并在实际对局中取得不错的表现。

改进可能：

- 加入策略网络(Actor-Critic 结构)
- 引入自适应探索策略
- 增加多玩家支持
- 实现连续动作空间（如下注量）

软硬牌:改进可能：

算法层面

- 引入策略梯度方法
- 使用多步学习
- 增加自适应探索策略

实现层面

- 优化网络结构
- 改进奖励设计

- 增加并行训练支持

应用层面

- 增加多玩家支持
- 添加更多游戏变体
- 提供策略可视化