机器学习-蒙特卡洛树游戏应用报告

——报告人：魏少杭 学号：20373594 班级：204211 班

一、蒙特卡洛树原理简述

1. 简述：

蒙特卡洛树（MCTS）基于蒙特卡洛思想，是一种通过采样、模拟一系列动作、状态，以达到估计各个可能状态下的价值，从而为未知环境下的游戏状态提供动作选择参考的方法。

2. 适用条件：

（1）作为**启发式算法**，其适用于任何除基本规则以外的任意博弈游戏应用，**不需要任何外部给定的领域知识、策略等辅助决策。**

（2）**更加适合具有更大分支因子的博弈游戏**：因为 MCTS 在搜索阶段利用的置信区间上限算法能够保证其不是完全无差别枚举所有的结点，而是以非对称的方式去扩展搜索下一状态。这样能够将时间成本集中于更关键的树/结点部分。

3. 关键思想的理解：

（1）**蒙特卡洛方法思想**：利用**随机采样样本值来估计真实值**。该思想理论基础**为中心极限定理**，当独立随机采样时样本数量趋近无穷大时，可以认为真实值就是平均样本值。

（2）**置信区间上限算法**：$V_i + C \cdot \sqrt[2]{\frac{ln(N)}{n_i}}$，**公式左边部分理解易错，其含义**是对**选择第 i 个子状态后的预期价值**，这个价值是**针对当前状态下**的价值，**这部分将在后面的游戏代码实现中具体说明**。公式右边部分 N、n_i分别表示当前状态、第 i 个子状态被访问的次数，C 为正常数；**整个公式右边意义**是**第 i 个子状态被发现(explore)的潜力**，若第 i 个子状态被搜索次数少，则会导致第 i 子状态价值估计不准确的情况或作为本身价值较高的状态但被忽略掉，为了避免此问题出现则设置了**探索潜力的项**作为衡量一个子状态被选择的优劣性的补充。

二、蒙特卡洛树搜索在游戏中的步骤实现：

（一）游戏环境介绍

我设计了两种经典的使用蒙特卡洛树搜索方法解决的动态博弈游戏：1.五子棋游戏 2.井字棋游戏。他们的棋盘大小可以任意改动（一般五子棋 size≥5 而井字棋 size≥3）。

井字棋游戏的规则是：对战的双方（无论人类玩家还是 AI 玩家）随时可以观察当前棋盘状态，轮流下棋；若有某玩家率先满足其棋子从横向、纵向、斜对角和反斜对角线方向排满棋盘，则该玩家胜出；若当棋盘摆满了所有的棋子，但无玩家率先达到前述要求，则达成平局。

五子棋游戏的规则与井字棋类似，不同的地方只是胜利的要求不同；对于一个至少 5*5 大小的棋盘，若某玩家率先满足横向、纵向、斜对角、反斜对角线方向连续排 5 个棋子则获胜。

（二）游戏步骤实现

1.构建环境

代码文件为 Game.py。定义了 Game 类、State 类。

①Game 类：

这个类主要描述了游戏界面形状大小、当前状态、轮数等属性；还有更新游戏状态、绘制游戏界面的功能。

A.__init__：定义棋盘大小；初始化棋盘为全 0，表示空的棋盘；定义画棋盘状态图的规则；初始化游戏当前状态；初始化游戏的当前轮数。

B. update_game_state：给定一个下棋的位置，更新游戏状态为下一个状态；轮数加 1；并绘制游戏界面（主要通过 print 到终端来实现）；

C. get_ui：使用 utils.py 中的绘制游戏界面的函数 render_game_ui 来绘制当前环境。

②State 类：

这个类作为所有游戏局面的描述，包括当前局面棋盘的样子、当前需要下棋的玩家（的编号）。其功能服务于前述的游戏的状态和蒙特卡洛搜索树的结点建立。

A.__init__：设置当前状态棋盘是什么样子，接下来由哪个玩家来落子。

B.__eq__：判断两个 State 类是否相等：这个方法要重写，是因为在使用的时候需要比较不同 id 下的两个状态实例是否物理意义相等。

C.get_possible_actions：获取当前状态下可行的所有动作。实现方法是扫描当前状态的棋盘，看 player 可以下棋的位置有哪些。

D.get_new_state：对于当前状态下，给定 player 下棋位置，创建新的状态实例。

E.judge_result：根据当前状态的棋盘棋子的分布，再结合当前游戏类型（五子棋还是井字棋）来判断是否当前状态为终止状态，返回判断结果、赢家。终止状态即为平局或有赢家。若为平局或未达到终止状态，则赢家返回值为 None。

2.蒙特卡洛树-AI 玩家的定义：

代码文件为：AI_MCTS.py

①Node 类：

这个类就是蒙特卡洛树的结点类。每一个结点实例需要挂载一个状态属性；需要存储它的父节点，以方便反向传播过程使用；需要保存未执行的动作，以供扩展操作使用；需要保存子节点，以供搜索操作时进行 UCB 算法搜索；需要保存每一个结点的累计价值 all_value；需要保存当前结点的累计访问次数 visit。

此外，还定义了一些获取状态、平均价值、更新当前节点的累计价值、累计访问次数的方法，以及反向传播中对结点更新当前状态的方法 update、判断当前节点是否为终止状态对应的叶子结点 isTerminal、判断当前状态所有子节点已经全部展开的方法 isAllExpanded、更新未尝试过的动作的方法 update_untried_actions。

②MCTS 类：

这个类就是一棵蒙特卡洛搜索树。需要定义遍历过程中的描述。其主要关注蒙特卡洛搜索树在游戏过程中的当前节点 cur_node 和根节点 root_node。在 MCTS 类中还定义了当 MCTS 实例作为玩家时选择动作的方法 choose_action、蒙特卡洛搜索算法思想的主要方法，即采样预测方法 Sample_Estimate 及其具体 4 个步骤的相关方法。

A.__init__：初始化当前节点、根节点。

B.choose_action：

当 AI 玩家需要作出决策时，传入游戏当前的状态，找到 MCTS 树已经构建的结点中与当前棋局状态匹配的结点。

其次，我们要以当前节点为子树根节点进行采样估计，通过蒙特卡洛方法，通过若干轮模拟估计当前子树根节点的价值。模拟的次数与游戏状态空间大小为正相关。然后仅根据 exploit 原则在当前节点下选择最优动作并得到新的结点，这个结点转化为对手马上面临的状态的结点，返回当前 AI 作出的落子位置。

C.Sample_Estimate：采样预测，是蒙特卡洛树搜索算法的核心。如下图。

```
# 第一步：UCT方法搜索，得到需要扩展的点
expand_node = self.UCT_search()
# 第二步：通过对需要expand的节点进行模拟/随机游走，
winner = self.Simulation(expand_node)
# 第三步：通过叶子结点的值，反向传播
self.BackPropagation(expand_node, winner)
```

D.UCT_search 就是按照置信区间上限算法进行搜索。对于当前节点的子节点全部被扩展过，则根据 UCB 公式计算选择最优结点。否则要扩展一个新的结点，作为搜索的叶子结点。

```python
def UCT_search(self):
    # 每次搜索的时候，从当前节点开始搜索，直到搜索到一个可以expand的叶节点
    node = self.cur_node
    # 如果已经搜索到叶子结点，直接返回
    while not node.isTerminal():
        # 1.如果全部扩展，根据置信区间上限公式选择最优结点
        if node.isAllExpanded():
            node, _ = self.UCB_selection(node) # UCB exploit+explore
        # 2.如果没有全部扩展，进行expand操作
        else:
            node = self.expand(node)
            return node
    # 返回叶子结点
    leaf_node = node
    return leaf_node
```

E.UCB_selection：根据置信区间上限算法计算，并挑选当前节点的子节点中最优的结点。可以**分为 exploit 部分（价值估计）和 explore 部分（探索潜力）。**这里需要注意的是，相邻两级结点分别挂载的状态是属于己方、对手的状态，但是结点的价值函数均表示结点状态本身的收益。对于下棋这种零和博弈游戏，当己方要搜索对自己最优的下一个状态结点的时候，exploit 的价值估计部分不能直接套用子节点的价值，因为子节点的价值是对手的收益，我们要己方收益最大化，则必须要对手收益最小化，**所以遍历子节点计算 UCB 时，需要对子节点的价值求负值，以表示最大化己方价值收益。**

```python
for child in node.childs.values():
    exploit = - child.get_ave_value()
    explore =  C * np.sqrt(2 * np.log(node.visit) / child.visit)
    ucb = exploit + explore
```

F. expand：给定一个结点，从结点还未扩展过的子节点中选择一个新的结点。

G. Simulation：随机游走模拟。从给定的结点开始（即被扩展结点）进行模拟，直到终止状态。返回赢家玩者（编号 1/2 或者 None）。

H. BackPropagation：反向传播过程。被扩展的结点开始反向传播至游戏初始根节点，根据 winner 编号与沿线中每一个结点对应状态下的玩家编号进行比较，如果相等，则价值加 1，如果不等，则价值减 1，如果赢家是 None，则价值为 0。这个方法实现**使用了递归。**

```python
# 1.更新visit
cur_node.visit += 1
# 2.更新values
if winner:
    cur_node.all_value += 1 if winner == cur_node.state.player else
else:
    pass

if cur_node.parent != None:
    return self.BackPropagation(cur_node.parent, winner)
```

3.人类玩家的定义：代码文件为 Human.py。主要为 Human 类。通过 input 指定格式的内容，通过正则表达式解析出落子的位置，作为人类对游戏的操作。

4.绘制棋盘：代码文件为 tuils.py。主要应用了 print 函数来实现对齐、画图等。其中 player1（AI）下棋为'o'，player2（人类或 AI）下棋为'x'，空位画'.'。

5.运行流程：代码文件为 main.py。**运行开始时，需要选择是否要 AI 与 AI 对抗还是人与 AI 对抗。若 AI 与 AI 对抗，则用 MCTS 类实例化两个不同的 AI。人类玩家用 Human 类实例化。**

不断运行直到结束，先获取当前状态，再根据状态获取下棋位置，最后根据当前状态与动作更新 game 的状态，并获取当前的界面。更新 game 后要交换下棋人。如果下棋结束，需要根据赢家输出下棋结果。

三、结果分析

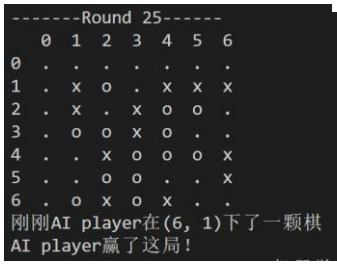
游戏 1.1：AI 对 AI 玩五子棋游戏，棋盘 7*7。游戏具体过程见“机器学习作业说明.pptx”Page1。最终结果是下图左上角 Round13 结果。AI1 赢得了此次游戏。但理论上讲两个同样算法下 AI 玩五子棋应该打成平局，我分析这可能是因为每轮采样次数不够多（仅有 5000 次）。耗时较长。

游戏 1.2：AI 对人类玩家玩五子棋游戏，棋盘 7*7。游戏具体过程见“机器学习作业说明.pptx”Page2-4。由于我自己在 Round22 疏忽下错棋，AI 就能打败我，说明训练的效果还是很理想。每轮采样次数为 2500。

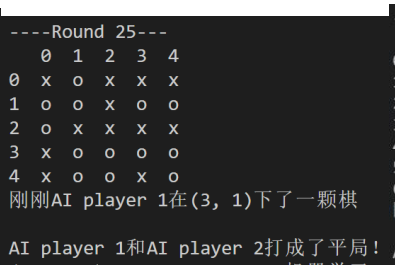
游戏 2.1：AI 对 AI 玩井字棋游戏，5*5 棋盘。见 PPT 第 5-6 页。结果平局。

游戏 2.2：AI 对人类玩井字棋游戏，7*7 棋盘，见 PPT 第 7 页。结果平局。说明 AI 已经在该游戏中具有对人类下的棋子有拦截的作用，且 AI 能够自行规划其需要打成的目标。

游戏 1.1



游戏 1.2



游戏 2.1

游戏 2.2

四、亮点

1.**实现了两种经典的零和动态博弈游戏**，能够很好地对 MCTS 算法基本理论进行核心理论实践和对实现细节的把控。

2.**实现了人类玩家与 AI 交互**，人类玩家能够体验到与自己写出来的 AI 玩家进行游戏对抗的乐趣。

3.还**尝试了 AI 与 AI 之间的对抗游戏**，游戏能完全托管、自动化进行。

4.Game.py 中实现的游戏可以变换为其他类似的博弈游戏，AI 玩家、人类玩家已经定义好了，且**高度抽象化，可以将 MCTS 模块移植到其他游戏上**去。

5.通过以上的游戏解释可以看出由 **MCTS 算法实现的 AI 在游戏中的对抗能力较强**。

五、展望

1.该算法中 MCTS 类在进行模拟时设定了每轮进行一定的采样次数上限，经过观察，随着游戏进行，游戏状态的可行状态空间变小，而从游戏状态到终止状态的各个结点访问次数增长得越来越快，这说明当前游戏状态的结点之后的结点的价值估计已经较为准确。为了更快收敛，**可以用类似于可变学习率的思想，让采样次数上限随着游戏进行的轮数进行合适速度的减小。**

2.五子棋/井字棋游戏中，若棋盘为 7*7 则状态空间已经非常大了，对应所需的每轮采样上限次数需要很多，且模拟的过程采用的是随机走子模拟，这直接导致 AI 作出决策之前的速度很慢。为了加快速度，可以**考虑如 AlphaGo 的快速走棋网络用来代替随机模拟策略**。

3.使用**策略网络**估计动作选择概率，减少分支因子；使用**价值网络**，估计胜利概率，减小蒙特卡洛树的深度。以上各种网络的使用目的都是为了**降低 AI 运行的时间成本**。