中国科学院大学
University of Chinese Academy of Sciences

计算博弈课程设计报告

（2021-2022 秋季学期）

报告题目 五子棋智能博弈相关技术调研与 AI 设计

学生姓名 解润芃 学号 202118014628004

学生姓名 倪子懿 学号 202128014628018

指导教师 兴军亮、徐波

学位类别 直博、学硕

学科专业 模式识别与智能系统

研究方向 类脑与认知计算、语音增强

研究所（院系） 自动化研究所

填表日期 2021/11/20

目录

五子棋智能博弈相关技术调研与 AI 设计

解润芃，倪子懿

2021 年 11 月

一、 项目简介

此为《计算博弈原理与应用（2021 年-2022 年秋季学期）》的课程设计报告，本文将对智能博弈棋牌类游戏的相关方法进行调研，并从中选取完美信息零和博弈的五子棋游戏进行实现。

我们主要分别调研了以极大极小搜索为起步的 Alpha-Beta 搜索、进一步结合哈希 Zobrist 算法的 Alpha-Beta 搜索、纯蒙特卡洛树搜索方法、结合深度学习神经网络的蒙特卡洛树搜索（AlphaGo 系列工作）、基于模型的树搜索（MuZero 模型）这五种方法，从相关技术的主要原理与思想、算法描述两个方面进行介绍；并对上述方法进行实验，成功训练并结合前期的调研改进了相关程序以解决五子棋游戏问题；最后给出上述方法下的实验结果。

我们对几种方法得到的实验结果进行比较并从多个角度观察，分析我们改进实验后存在的问题和不足。初步的实验证明我们的改进取得了一定的效果。

1.1 本文结构

本文将先对五子棋这种游戏进行简介，然后介绍目前位置主流的五子棋博弈相关的技术，并对其中的极小极大值搜索、蒙特卡洛树搜索，以及基于强化学习的方法，例如 AlphaGo 等模型进行更详细的阐述。

在方法部分，我们将对本次项目主要使用的三种模型进行详细的分析，我们将给出对应的算法、模型的特点和一些实现的细节。这三个模型是：基于 Alpha-Beta 剪枝和 Zobrist 置换表技术的 MinMax 搜索；AlphaZero 方法；MuZero 方法。

在实验及结果部分，我们将关注模型的训练以及对应的训练结果。我们将对训练情况和实际的对局结果进行一个详细的分析，并对各种模型的特性和性能进行讨论。

在讨论部分，我们将主要分析各个模型的差异和与它们结构特征之间的关联、这些模型如何从结构上体现出它们的特征以及可能改进的部分。

在总结部分，我们将对整个项目进行回顾，并对我们的调研以及测试结果进行总结。

二、 研究背景

2.1 五子棋

棋类游戏作为一种完美信息博弈，可以被定义为一种交替的马尔科夫博弈 [7]，其以下几个部分组成：1）一个状态空间 S，用以表示当前行棋的玩家及其所面临的状态；2）一个动作空间 A，用以表示其在任何状态下所能做出的合法动作集合；3）一个状态转移方程 $results()$，定义在给定状态在获得一定的输入后的后续状态集合，即 $s \leftarrow result(s, a)$；4）奖励函数 $r()$ 描述了玩家 i 在某个状态 s 下所获得的奖励 $r^i(s)$。特别的，对于两人的零和博弈，奖励函数满足 $r^1(s) = -r^2(s)$。

五子棋作为一种棋类游戏，是一种两人零和博弈。游戏过程包含两个玩家和一个固定大小的棋盘。游戏从空白的棋盘开始。规定执黑棋方先行在棋盘中央横纵线的交叉处（即天元处）放下自己的一颗棋子；随后执白棋方在任意交叉处放下自己的一颗棋子，随后黑方行棋，如此反复直到游戏胜利或者棋盘上没有空位可以行棋。游戏胜利条件为先行获得在横线、竖线或者斜线上连续的 5 颗己方的棋子。若直到游戏结束都不存在胜方，则称为这局游戏和棋。标准的五子棋棋盘有纵横线各 15 条，即具有 225 个落子点。

五子棋发展至今，人类玩家以及总结出了一些规律和定式，例如“一子双杀”，即在通过一步行棋获得两个可以赢棋的线路，从而取得胜利。为了形成“一子双杀”局面，有“两头空”、“边二空”等众多的行棋策略。

2.2 五子棋博弈的研究现状

五子棋具有一定大小的搜索空间和策略性，但同时其复杂程度不如围棋等一些搜索空间更加庞大的棋牌游戏，使得各种棋类游戏的方法即可以在这个博弈问题中实现，又不对实验的条件过分苛刻，使其作为了一个较为理想的研究对象。在本部分我们将对一些经典的五子棋博弈技术及方法进行介绍，而对于我们实际在实验部分中实现的模型及实验的细节描述，将分别放在方法部分和实验部分进行详细的介绍。

2.2.1 极小极大值搜索

最优的价值函数可以通过极小极大值算法（又称 MinMax 搜索）递归计算。但如果不加限制的搜索的搜索空间过大，因此一般会用一个近似的价值函数替代最终的奖励对博弈进行截断。基于 Alpha-Beta 剪枝的深度优先 MinMax 搜索在许多棋类游戏上取得了超越人类的表现。

在围棋领域，MinMax 搜索没有取得很好的效果；然而五子棋较围棋具有更小的搜索空间，使用 MinMax 搜索能够取得不错的性能。因此，对于五子棋等搜索空间较小的棋类游戏，使用 Alpha-Beta 剪枝的 MinMax 搜索算法的棋类程序仍然是一种十分有竞争力的技术方案。

考虑五子棋博弈问题的形式化：五子棋游戏中的每一个状态为棋盘上棋子的摆放情况，玩家每一次行棋构成了一条可行的路径，使博弈从一个节点转移到另一个节点，在行棋前考虑行棋的过程实际是一次完美信息博弈。将这些节点和与之相连的路径连接，可以获得一棵巨大的博弈树。在这个树中，从根节点为空棋盘开始，奇数层表示 $Player_1$ 行棋后的棋盘状态，偶数层表示 $Player_2$ 行棋后的棋盘状态。从而这个问题可以使用 MinMax 搜索解决。

尽管五子棋的状态空间大小远小于围棋问题，但其空间仍然很大，不能获得一个完整的博弈树 [10]。因此需要使用启发式函数对于当前的局面进行一个价值评估，提前判断子集获胜和失败的情况，使我们知道哪一个分支的走法最优。对每一次搜索，设定一个固定的最深搜索深度，如果一次搜索抵达了这个深度或者已经有某方获得胜利，则调用启发式函数获得对于当前节点的价值评估。

遍历这颗博弈树就可定下如何选择分支：$Player_1$ 走棋的层我们称为 MAX 层，这一层玩家要保证自己利益最大化，那么就需要选分最高的节点；$Player_2$ 走棋的层我们称为 MIN 层，这一层玩家要保证自己的利益最大化，那么就会选分最低的节点。

2.2.2 基于 Alpha-Beta 剪枝的 MinMax 搜索

现在考虑基础的 MinMax 算法的复杂度。假设每一步存在 50 个后续节点，思考到第四层，那么搜索的节点数就是 $50^4 = 625$ 万，并随着层数的增加呈指数级增大。而 Alpha-Beta 算法就希望在不影响搜索结果的条件下，通过提前结束子节点的搜索来降低搜索的复杂度。

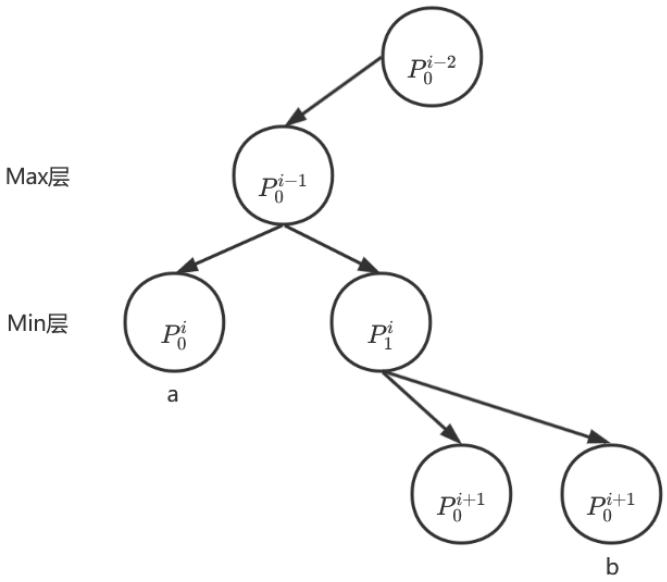


图 2.1　MinMax 示例

我们注意到（如图 2.1），存在这样的情况：如果一个 Max 层节点 P_0^{i-1} 的子节点 P_0^i 返回了一个值，设为 a；同时对于正在搜索的子节点 P_1^i（即 Min 层），其存在一个子节点 P_o^{i+1} 返回的值为 b，假设 $b < a$，则对于子节点 P_1^i 来说，其希望获得一个最小值，该最小值的最大值不会大于 b。因此对于节点 P_0^{i-1}，其选择 P_0^i 一定优于选择 P_1^i，因此对于 P_1^i 节点的子节点可以立刻停止，继续寻找其它的子节点。将这种情况抽象出来，并加以使用，以提前结束对于一些分支的搜索，就可以得到 Alpha-Beta 剪枝的 MinMax 搜索 [4]。

正如前面给出的例子，Alpha-Beta 剪枝算法是一种安全的剪枝策略，其不会对棋力产生负面影响，同时可以大幅度缩减博弈树的规模。其中 Alpha 指代我方搜索到的最大值，Beta 指代对手搜索到的最小值。它的基本依据是：棋手不会做出对自己不利的选择。依据这个前提，如果一个节点明显是不利于自己的节点，那么就可以直接剪掉这个节点。$Player_1$ 如果发现这一步是对对方更有利的，则会拒绝走该步。同理，如果 $Player_2$ 发现某一步策略对 $Player_1$ 更有利，其不会选择这一步。使用传统 Alpha-Beta 剪枝策略的 MinMax 搜索算法见算法1。

Algorithm 1 Alpha-Beta Search

function ALPHA-BETA-SEARCH(state) **returns** an action
    v ← MAX-VALUE(state, -∞, +∞)
    returns the action in ACTIONS(state) with value v
function MAX-VALUE(state, α, β) **returns** a utility value
    if TERMINAL-TEST(state) **then return** UTILITY(state)
    v ← $-\infty$
    for each a in ACTION(state) **do**
        v ← MAX(v, MIN-VALUE(RESULT(s, a), α, β))
        if v ≥ β **then return** v
        α ← MAX(α, v)
    return v
function MIN-VALUE(state, α, β) **returns** a utility value
    if TERMINAL-TEST(state) **then return** UTILITY(state)
    v ← $+\infty$
    for each a in ACTION(state) **do**
        v ← MIN(v, MAX-VALUE(RESULT(s, a), α, β))
        if v ≤ α **then return** v
        β ← MIN(β, v)
    return v

2.2.3 蒙特卡洛树搜索

蒙特卡洛方法（Monte-Carlo method）[4] 是统计学理论的经典方法，其广泛地被应用于金融学、经济学、物理学等众多领域。蒙特卡洛树搜索（Monte-Carlo Tree Search，MCTS）[1]，是一个迭代的，最佳优先树搜索过程。在 MinMax 搜索方法中，每一步需要对后续状态空间进行穷举，这一步骤尽管使用 Alpha-Beta 剪枝进行优化，MinMax 构建的决策树仍然非常庞大；并且这种方法能够抵达的深度有限，仅能够使用启发式的方式获取状态的价值，这是非常依赖于先验知识的。蒙特卡洛树搜索则可以解决这两个问题：1）使用采样代替完全决策树的构建；2）使用快速模拟结果取代对于状态的启发式函数估计。其目标是帮模型们判断应该采用什么样的动作，从而实现长期受益最大化。

蒙特卡洛树搜索具有四个主要阶段：选择（Selection），扩展（Expansion），模拟（Simulation）和回溯（Backpropagation）。

选择 选择步使用 UCT（Upper Confidence Bound Apply to Tree）算法获取当前潜在的最优走法 [3]。其中考虑了对于已经走过该点的根据经验获得的估计价值，以及这个走法的模拟次数。我们希望如果一个棋路，根据经验被认为有越高的可能性获胜，则我们越倾向于走它；此外我们希望模型能够探索新的走法，即若一个棋路被走过的次数越少，我们越倾向于尝试它。我们借此获得 UCT 的价值函数。

扩展 对与一个节点，如果我们多次访问，认为它是有价值的且并未游戏结束，则可以扩展该节点，使后续的模拟可以朝这个方向继续拓展。

模拟 为了获得这个选择是否有效，选择随机走子结束这一局游戏。

回溯 通过模拟，我们可以获得这一局游戏“预期”的情况，从而对此前 UCT 的估计进行修改，使它更为准确。

通过重复执行这些阶段，MCTS 每次都会在一个节点未来可能的动作序列上逐步构建一棵搜索树。在这个树中，每一个节点都代表游戏的一个当前局面的确定状态，而节点之间的连线则表示从一个状态到下一个状态的动作。

在每局游戏过程中，每一步落子前，蒙特卡罗树搜索都会模拟游戏多次——就像人类思考的方式一样，通过模拟游戏的发展方向，观察每一步可以落子的位置是否会导致最终胜利。然后回溯到当前，选择最有可能获胜的落子。

2.2.4 AlphaGo 等深度强化学习方法

DeepMind 研究人员研发了 AlphaGo 系列的强化学习算法。2014 年著名的 AlphaGo[6] 的诞生。2015 年，AlphaGo 围棋击败围棋冠军李世石。AlphaGo 使用了蒙特卡洛树搜索的原理，将游戏的其余部分规划为决策树。但微妙之处在于添加一个神经网络来智能地选择动作，从而减少未来要模拟的动作数量。

AlphaGo 的神经网络从一开始就以人类玩游戏的历史进行训练，2017 年 AlphaGo 被不使用人类历史经验、可以自我博弈驱动的 AlphaGo Zero 取代。这是第一次该算法

Algorithm 2 Monte-Carlo Tree Search

```
function MONTE-CARLO-PLANNING(state) returns a move
    Repeat
        SEARCH(state, 0)
    until Timeout
    return BEST-ACTION(state, 0)
function SEARCH(state, depth) returns a utility value
    if TERMINAL-TEST(state) then return 0
    if IS-LEAF(state; d) then return EVALUATE(state)
    action ← SELECT-ACTION(state, depth)
    (nextstate; reward) ← SIMULATE-ACTION(state, action)
    q ← reward +γ SEARCH(nextstate, depth+1)
    UPDATE-VALUE(state; action; q; depth)
    return q
```

在没有围棋策略先验知识的情况下学会了自己和自己玩。它使用 MCTS 进行自我博弈，然后通过神经网络的训练从这些部分进行学习。

AlphaZero 是 AlphaGo Zero 的一个版本 [8]，在这个版本中，所有与 Go 游戏相关的技巧都被删除，因此它可以推广到其他棋盘游戏。

2019 年 11 月，DeepMind 团队发布了新的、更通用的 AlphaZero 版本，名为 MuZero[5]。其特殊性在于其为基于模型的搜索，这意味着算法对博弈有自己的理解。在 MuZero 之前，动作对游戏的影响是硬编码的，而 MuZero 从未知动态环境学习模型（概率分布），构建环境，然后再根据所学的 model 进行规划。因此，MuZero 摆脱了对于环境的依赖，可以像发现新游戏的人一样来通用地玩任何游戏。

三、 方法

3.1 基于 Zobrist 置换表技术的 MinMax 搜索

3.1.1 Zobrist 置换表技术

对于 Alpha-Beta 剪枝的 MinMax 搜索方法，可以用 Zobrist 置换表技术进一步提高算法性能。Zobrist 算法 [9] 是一种有效的应用于棋类游戏的哈希算法。其可以将棋局映射为一个哈希值；并且在游戏开始后，只需要很小的计算量就可以对任意落子和提子后棋局的哈希值进行更新。

Zobrist 哈希算法的数学原理非常简单。首先对于棋盘给定一个映射矩阵 $zobrist \in R^{M\times M\times 2}$（$M$ 是棋盘的宽度），这个矩阵的每一个值对应了棋盘对应位置上摆放的是黑

棋还是白棋。$zobrist$ 矩阵使用随机整数填充作为初始化。此外对于空棋盘（比如五子棋），设定初始的哈希值 $h = 0$。对每一次落子（例如，坐标：(x, y)，下黑子），就将哈希值与该子对应的矩阵中的值异或一次，即 $h \leftarrow h \oplus zobrist[x][y][0]$。如果我们希望撤销这一次落子，则再异或一次，即 $h \leftarrow h \oplus zobrist[x][y][0]$ 即可。由于 $A \oplus B \oplus B = A$，则前后两次异或可以保证操作后的哈希值与落子前完全相同。

Algorithm 3 Zobrist

```
function INITIATE-ZOBRIST() returns a 3-d matrix
    // fill a table of random numbers/bitstrings
    table := a 3-d array of size length × width × 2
    for i from 1 to length do //(loop over the board)
        for j from 1 to width do //(loop over the pieces)
            for k from 1 to 2 //(loop over the pieces)
                table [i] [j] [k] ← random_bitstring()
    return table
function STEP(pos, table) returns hashcode h
    h ← h XOR table[pos]
    return h
```

这使得每一个棋局都可以获得几乎唯一的哈希值表示；并且每下一步棋，仅需要执行一次异或操作，这使得搜索过程中需要撤销落子回到父节点的运算时间复杂度小。并且在搜索过程中，只需要记录一个当前棋盘的哈希值，空间复杂度为 $O(1)$。

3.1.2 具体实现

对于仅使用 Alpha-Beta 剪枝的 MinMax 搜索，很多时候会有重复的搜索，即几种走法只是顺序不同，但最终走出来的局面一样。然而对于这种情况，算法由于没有记录之前的情况，仍然需要再搜索一次。而在使用 Zobrist 置换表技术后，对于任何一个棋面，只需要进行一次打分，将其存入散列表中，第二次就可直接从散列表中读出估计的值。这个方法对于搜索深度深、待搜索状态空间大的情况尤为有效。

特别的，对于 MinMax 搜索，如果在两个不同的深度搜索到了同一节点，其中一个节点仍能够朝更深的博弈树搜索，这种时候我们不希望使用此前更浅层的搜索结果代替这次搜索结果，所以除了记录落子信息外，我们还需要记录落子时当前的搜索深度，例如 $h \leftarrow h \oplus zobrist[x][y][0] \oplus depth$。并且记录的该深度信息同样是顺序无关的，所以能够满足我们的需求。

在 MinMax 搜索的具体实现中，这里使用了一个小技巧，在标准的 MinMax 搜索中，需要分别设定 Max 层的搜索函数和 Min 层的搜索函数，但这两个函数的具体细节颇为相似，只是一个希望获得极大值，而另一个希望获得极小值。则在实现中，我们将从子

Algorithm 4 Zobrist-Enhanced Min-Max Search

```
function ZOBRIST-ENHANCED-MIN-MAX-SEARCH(state) returns an action
    table ← INITIATE-ZOBRIST()
    record ← EMPTY-SET()
    v ← MAX-VALUE(state, -∞, +∞, depth, 0)
    returns the action in ACTIONS(state) with value v
function SEARCH(state, α, β, depth, h) returns a utility value and the hashcode
    if TERMINAL-TEST(state) or depth = 0 then return UTILITY(state)
    v ← −∞
    for each a in ACTION(state) do
        h ← STEP(POS(a), h) ⊕ (depth+1)
        if h in record then return record.GET(h)
        v ← MAX(v, -SEARCH(RESULT(s, a), -α, −β), depth-1, h))
        ADD((h, v),record)
        h ← STEP(POS(a), h) ⊕ (depth+1)
        if v ≥ β then return v
        α ← MAX( α, v)
    return v
```

层获得的估计值取反，则可以自然地保证这个要求；同时，为了保证 Alpha-Beta 的语义正确，在向下一层传入 Alpha 和 Beta 两个值时，同样需要取反。

3.2 AlphaZero 方法

3.2.1 思想及原理

由 Deepmind 提出的 AlphagGo 是棋牌类智能 AI 技术的突破性的一个进展，其使用了大量人类专业选手比赛数据，应用了诸多技巧，在围棋上获得了巨大成功。在 AlphaGo 之后，为了降低先验知识的影响，AlphaZero 不再使用人类棋手比赛数据作为训练数据，也不再使用手工设计的特征和快速走子网络，从规则出发完全通过自我博弈学习，使用最新的深度模型结构大幅提升性能。因此，我们选取了一个基于 AlphaZero 的代码实现框架在五子棋上进行实验，并对其存在的问题进行分析，并加以改进优化。

与 AlphaGo 不同，AlphaZero 中仅使用了一个神经网络，同时获得策略和价值估计，代替了 AlphaGo 中的策略网络和价值网络。策略为 MCTS 提供一个先验概率，让 MCTS 优先搜索对当前选手而言更可能获胜的路径，也就是说基于当前策略进行采样，而不是随机采样；价值估计的作用在于搜索到叶子节点的时候，若游戏没有结束，则以该值进行回溯更新。单纯的 MCTS 在搜索到非游戏结束的叶子节点的时候会进行模拟，通过

随机走子直到游戏结束，用这个采样结果来估计基于当前局面当前选手的得分期望。

以某种棋局状态出发进行多次 MCTS 搜索以后，就可以依据各个子节点的访问次数构造一个概率分布，依据该概率分布来决定真正应该在何处落子（这和 MCTS 中的走子策略是相同的），同时该概率分布也可以作为网络训练策略的监督信号。每局棋结束以后，就可以知道该轮对弈在每个棋局状态下的最终胜负，该得分将作为训练价值估计的监督信号，从而对神经网络模型进行更新。

3.2.2 算法及其神经网络结构

AlphaZero 核心使用 MCTS 方法完成对于下一步的估计以及走子。

在每一步走子中，完整执行 MCTS 方法的选择，扩展，模拟和回溯步骤。特别的，在其进行模拟的步骤中，如果成功地在叶子节点抵达了游戏结束，则使用游戏的胜负结果作为对于叶子节点的价值估计；反之使用 AlphaZero 的神经网络对于价值进行估计。在第 i 步落子前，由于 MCTS 方法的模拟，会产生对于下一步走子的概率分布 P_i，模型按照概率分布随机选择下一步执行。同时对于当下所在的状态，AlphaZero 的神经网络方法可以给出其预计的本状态玩家获胜的概率 u_i 以及预计走子的概率分布 P_i'，分别作为价值和策略的估计。其中 $P_i, P_i' \in [0,1]^{w\times w}$，$w$ 为棋盘的宽度和长度。

在一局游戏完成后，一共走了 S 步，则我们可以获得三个序列：模型实际模拟中的走子概率分布 $\{P_i\}, i \in [1,S]$、模型预测的走子概率分布 $\{P_i'\}, i \in [1,S]$ 和估计的获胜概率 $\{u_i\}$。同时获得这一句游戏实际的胜负情况，设其中胜利方走子下标集合为 W，败方走子下标集合为 L。AlphaZero 希望能够令预估的获胜概率和实际的情况尽量符合，同时估计的走子概率和实际 MCTS 通过尝试获得的实际走子概率相近，于是获得了策略网络的损失函数。

$$\begin{aligned} loss &= loss_{probs} + loss_u \\ &= -\frac{1}{w^2}\sum_{i=1}^{S} P_i \cdot P_i' + \frac{1}{w}(\sum_{i\in W}(1-u_i)^2 + \sum_{i\in L} u_i^2) \end{aligned} \tag{3.1}$$

在 AlphaZero 中，使用一个独立的神经网络（图3.1）充当策略网络和价值网络，代替原本 MCTS 随机走子至游戏结束，和 AlphaGo 中原本的策略网络和价值网络函数两个独立的神经网络，使得原本的模型更加简单。这个模型接受一个大小为 $w \times w \times 4$ 的张量作为输入，每一层分别记录当前棋局己方走子情况、对方走子情况、上一步走子、当前哪一方走子四种不同的数据。其中代表“当前哪一方走子”的数据通过一个全为 1 或者 0 的矩阵表示。经过三层卷积层，获得特征图，然后分别经过不同的全卷积层及全连接层获得大小为 $w \times w$ 和 1×1 两个输出，前者表示模型估计的策略（即走子的概率分布），后者表示当前棋局下当前玩家的胜率。

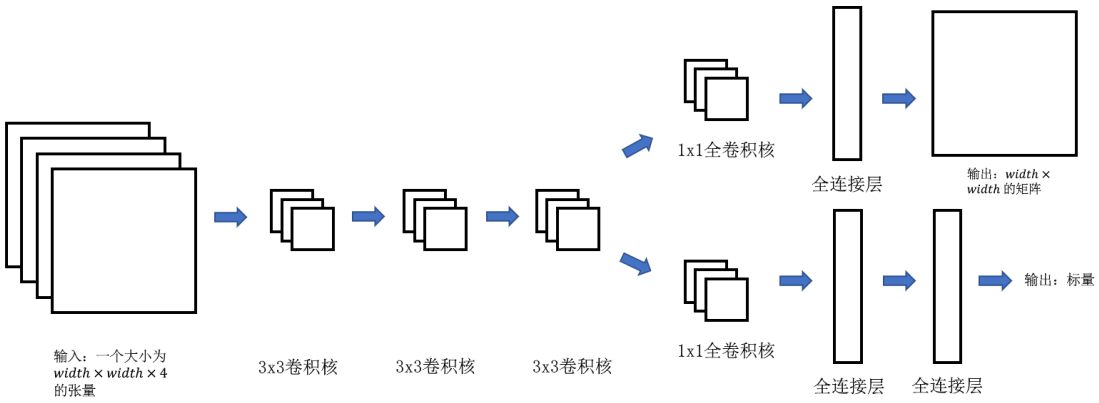


图 3.1　AlphaZero 使用的策略及价值网络

3.3　MuZero 方法

3.3.1　思想及原理

MuZero 是用于棋盘游戏（国际象棋、围棋等）和 Atari 游戏的最先进的强化学习算法，作为 AlphaZero 的继承者，但并没有动态环境的信息。MuZero 是通过学习建立环境的模型，并使用仅包含用于预测奖励、价值、政策和过渡的有用信息的内部表示。MuZero 接近于价值预测网络。

其主要思想是从基于过去的观测值以及未来的行为，对于给定的步中的每一个步，通过学得的与真实环境所对应的模型在每个时间步进行预测，使其所给出的未来每一步的价值和奖励都接近真实环境中的值。

MuZero 包括三个相互联系的组件用于表示（representation），动态（dynamics），和预测（prediction）。其中表示函数以对真实环境的观察作为输入，将初始化的隐层状态作为输出，并对初始化根节点隐层状态 s^0 进行编码 $s^0 = h_\theta(o_1, \ldots, o_t)$。动态函数以前一个隐藏层状态和候选动作作为输入，以当前即时奖励和隐层状态作为输出。其可以表示为 $r^k, s^k = g_\theta(s^{(k-1)}, a^k)$。这里 g 为确定函数。预测函数采用类似于 AlphaZero 的联合策略和价值神经网络，以隐藏状态作为输入，同时输出策略估计和价值估计：$p^k, v^k = f_\theta(s^k)$。

在与环境交互时，Muzero 仍然是通过蒙特卡洛树进行动作的选择，根据访问节点次数对搜索策略进行采样。环境接受这个动作，给出新的观察和奖励。交互的轨迹会存储在一个缓存中，在训练时进行采样。初始时，表示函数会将之前的观察作为输入，然后模型递归展开 K 步。在每一步中，动态函数将上一步的隐状态和当前步的真实动作作为输入。训练时，三个组件的网络参数会使用时间传播反向算法进行联合训练，预测策略分布，价值函数，和奖励值。

从相同的真实状态开始，通过抽象马尔可夫决策过程（Markov Decision Process，MDP）的 K 步轨迹的累积报酬与真实环境中 K 步轨迹的累积报酬进行匹配，价值等价

（value equivalence）的实现有效保证了 MuZero 模型可以学习出对真实环境的理解。

3.3.2 MuZero 方法和 AlphaZero 方法的差异

在以下几个部分 AlphaZero 仍会利用人类知识规则：搜索树中的状态转移，搜索树中状态的可获得性，搜索树中的提前中止条件（当搜索树达到已知的结束状态时，不再进行搜索，直接返回结束状态的值）。而这些在 MuZero 中都被一个隐式的神经网络模型所代替，这进一步减少了对人类知识的需求。使得模型具有更高的灵活性和更加出色的泛化能力。

四、 实验及结果

在方法部分，我们对使用 Zobrist 置换表和 Alpha-Beta 剪枝的 MinMax 算法、AlphaZero 方法及 MuZero 方法都进行了详细的介绍。在本部分，我们将分别对这三种方法，以及蒙特卡洛树搜索法进行测试加入作为参照。测试主要关注每一步计算所需的时间以及和实验者对弈的结果，并将对弈的步骤记录加以分析。随后我们将令各个方法互相对弈，获得它们之间的性能差异。

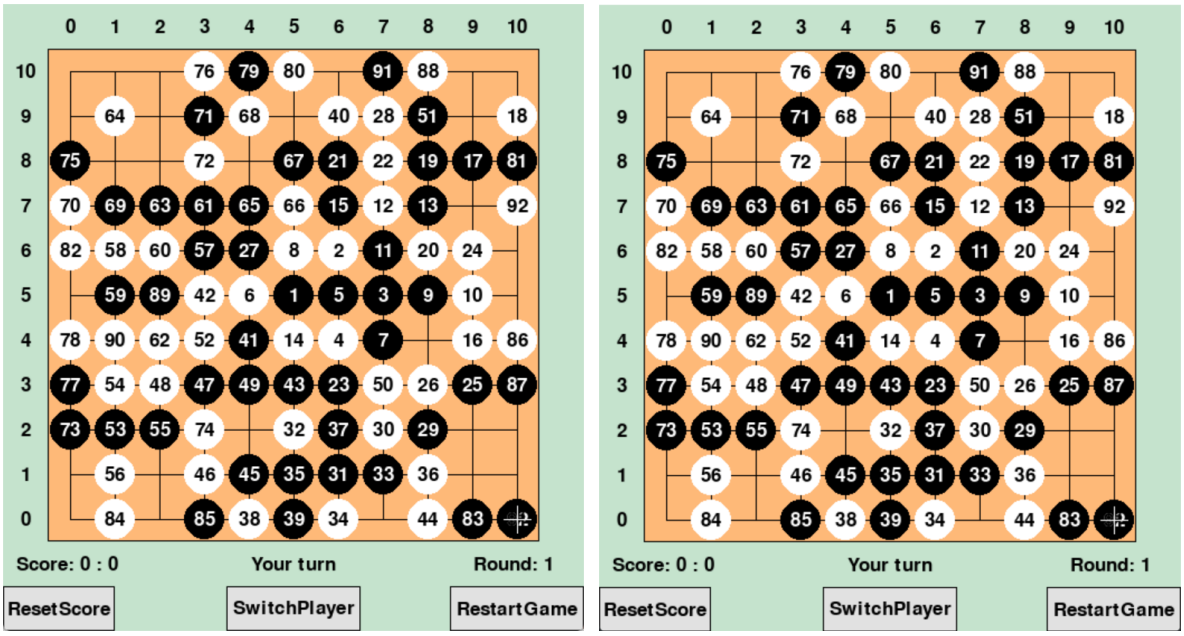
4.1 实验设置

本实验所有模型统一使用 python 版本为 3.7，其中 AlphaZero 的 PyTorch 版本为 1.10.0，MuZero 的 Pytorch 版本为 1.6.0。使用的 CPU 为 i7-11800H，GPU 分别为笔记本机载 GTX3060（6GB 显存，95W 功耗版本）和所里的服务器 GP102 [TITAN X]（12G 显存）。

4.2 基于 Zobrist 置换表技术的 MinMax 搜索

模型主要参照 https://blog.csdn.net/marble_xu/article/details/90647361 实现。

4.2.1 Zobrist 置换表技术不影响 MinMax 搜索方法的正确性



(a) Alpha-Beta 剪枝的 MinMax 搜索　　(b) 使用了 Zobrits 置换表技术的 MinMax 搜索

图 4.1　不使用 Zorbrist 置换表的 MinMax 搜索和使用了的算法结果对比

图4.1为人类玩家分别和 MinMax 搜索方法和使用 Zobrist 置换表技术的 MinMax 搜索对弈的记录，其中人类玩家先与 MinMax 搜索方法的智能体对弈一次，记录所有的走法，然后按照该走法与使用 Zobrist 置换表技术的智能体对弈。可以看到两个算法运行的结果相同，这说明使用 Zobrist 置换表并不会显著影响 MinMax 搜索的正确性（尽管有极小的概率会出现哈希值相同）。故在后续与其它模型的对照实验中将直接使用使用了 Zobrist 技术的 Max-Max 搜索算法。

4.2.2 MinMax 搜索的效率与搜索深度的关系

搜索层数	**使用 Zobrist 置换表技术** 平均搜索时间	**MinMax 搜索** 平均搜索时间	**减少搜索比例**
3	0.088s	0.101s	8.38%
4	1.09s	1.56s	15.6%
5	1.25s	1.54s	17.8%
6	7.36s	10.55s	27.1%

表 4.1　Zobrist 置换表技术对于算法效率的影响

表4.1中数据为使用不同最大搜索深度的 MinMax 搜索，与使用 Zobrits 置换表技术后的 MinMax 搜索用时比较。其中平均搜索时间指多次对局每步搜索花费的搜索时间。该数据通过使用 MinMax 搜索和使用 Zobrist 置换表技术的 MinMax 搜索对局 20 次取平均获得，其中 10 次为 MinMax 搜索先行，其余 10 次为使用置换表技术的 MinMax 搜索先行。表格中，减少搜索比例指使用 Zobrist 置换表中的结果代替重复求取节点启发式函数值的情况占所有搜索次数的总数。事实上我们发现，当搜索深度为 5 时，以同样的 MinMax 算法为对手，MinMax 搜索已经找到了先手获胜的方式，并使游戏结束在该位置；在搜索深度小于等于 4 时则会平局。

从表4.1中可以发现：1）平均每步搜索时间随搜索深度的增加而显著上升；2）使用 Zobrist 置换表技术的 MinMax 搜索花费的时间明显小于不使用该技术的算法；3）随着最大搜索深度的增加，减少搜索比例随之增加。

4.3 AlphaZero 方法

4.3.1 实验方法及训练细节

模型主要参照 https://github.com/junxiaosong/AlphaZero_Gomoku 实现。AlphaZero 方法使用标准的自博弈训练方法：令玩家 1 和玩家 2 均为 AlphaZero 模型，每一步按照在“方法”部分中详细描述的算法获得，在每一局游戏结束之后储存数据并开始下一轮游戏。其中每一次比赛可以获得一组序列的数据。在总数的数据量达到 512 组时，从所有已经存储的数据中随机选取 512 个数据作为一批，使用梯度下降更新一次 AlphaZero 中使用的深度卷积神经网络的参数。

模型使用 Adam 优化器 [2]，设定初始学习率为 $2e^{-3}$，并根据学习的结果动态地调整学习率。

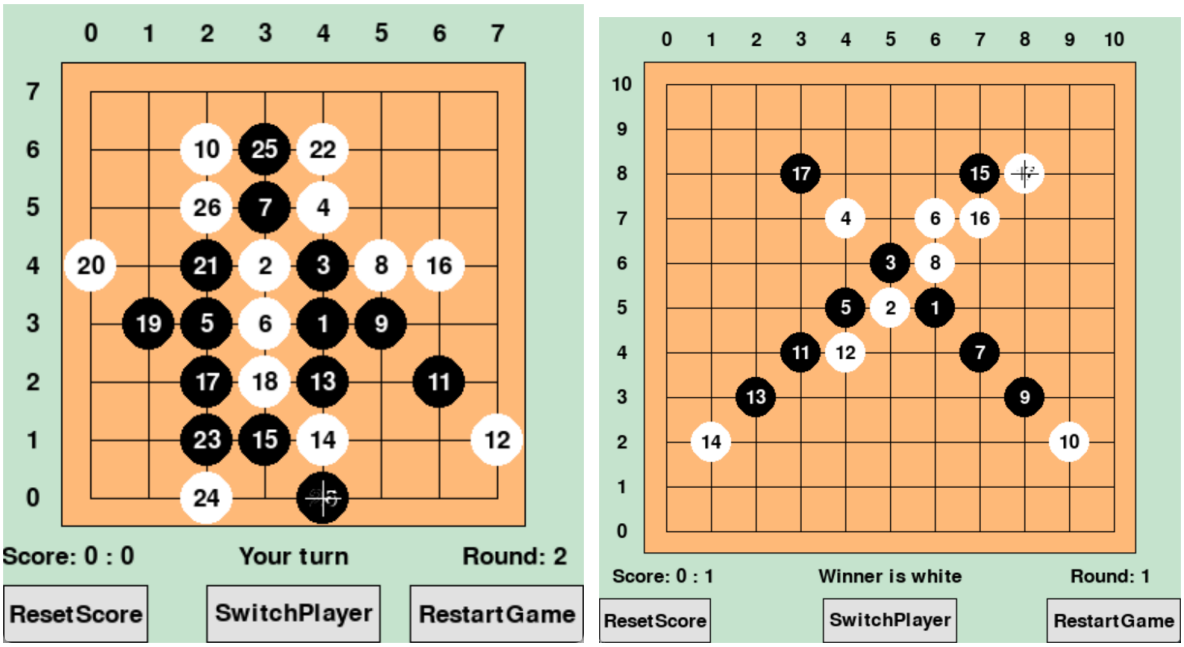
训练使用的棋盘大小处于设备限制，设计棋盘大小为 11×11，连成 5 颗棋子后胜利。模型在训练过程中每 50 次梯度下降后和 MCTS 方法进行 10 次对弈（双方先手各五次），考察其胜利情况。训练成果较理想时，提高 MCTS 中试探的次数，以获得更强的 MCTS 模型，然后继续进行对比。当出现一个胜率高于此前的模型时，就存下这组数据。

4.3.2 训练结果

我们训练了 8×8，11×11 两种大小棋盘的 AlphaZero 模型。其中 11×11 棋盘大小的，按照上述方法训练至 2000 批次时，模型获得了较强的性能，总训练耗时约 10 小时，获得最后的模型。其中 11×11 棋盘大小的，按照上述方法训练至 3000 批次时，总训练耗时约 24 小时，获得最后的模型（该模型表现尚不理想）。

其中 8×8 棋盘由于很容易造成和局，因此即使是在 AlphaZero 先行的时候仍无法取得胜利。同时存在一些策略上的明显疏忽。如图4.2(a) 所示，其中智能体执黑棋，人类玩家执白棋。在智能体完成步 21 时，这已经是人类玩家必输的局面。人类玩家没有

选择堵上 21 上方的位置，而是堵住了 7 右上方的位置。此时智能体保持了最优的操作走 23 的位置，人类玩家在 24 拦截。此时智能体没有选择走 21 上方的位置取得胜利，而是选择在别处行棋 25。这无疑是智能体的一个失误。



(a) 8 × 8 人机对弈　　(b) 11 × 11 人机对弈

图 4.2　AlphaZero 人机对弈示例

同时我们注意到在 AlphaZero 自博弈训练中，会出现步数由大变小、又由小变大的过程。其中步数最小是为 9，即其中先手五步取得胜利的情况。由于在面对理智的对手时，五子棋一定不能五步及以内取胜，这说明智能体在步数减小的过程中逐渐学习到了胜利的条件，呈现出攻击性，尽快达成胜利条件，而没有防守性（见图4.2(b)）。

4.4　MuZero 算法

4.4.1　实现方法及训练细节

基于 https://github.com/werner-duvaud/muzero-general 的框架，构建 MuZero 模型，将它应用到五子棋游戏上。这里我们系统地学习了 MuZero 的相关原理，并配置相应强化学习的基础环境，调通程序代码。由于未及时申请到云端计算资源，训练资源不足的困难明显。调通的原始完整 MuZero 模型在所里的 GP102 [TITAN X]（12G 显存）的单卡上训练会使得 GPU 使用爆满，尤其初始 11 × 11 大棋盘上，在前期自博弈（self-play）过程中，收敛速度较慢，前 1000 个 epoch 的结果 Loss 值仍维持在 10 以上。选取保存 5 小时内运行得到的最好模型，在 11 × 11 的棋盘上尝试人机对打，该过程十分漫长，显示平均每次搜索返回最优决策需要的时间约为 12.10s，显然最终得到的该模型并非最理想结果。

出于上述客观原因，我们对 MuZero 原有模型的神经网络部分进行了调整和压缩，随后将其重新部署在五子棋游戏上，并尝试了多种不同的超参数上，与其他几种方法进行了比较。

我们采用将五子棋设置为 7×7 的小棋盘，Self-Play 过程中在 MCTS 的最大探索步数设置为 49，未来移动的采样轮数为 100，并将所有的全连接层改为残差网络，设置 batch size 为 128。与 AlphaZero 中的原理相类似，输入残差网络的特征里每一层分别记录当前棋局己方走子情况、对方走子情况、上一步走子、当前哪一方走子四种不同的数据，因此我们在这里直接设置输入的 channel 数为 4；我们进一步将 MuZero 模型实验里把三层卷积层组成的基础网络也改味残差结构的 18 层网络，将 dynamic network, prediction network 中预测策略，奖励和价值的隐层数量设置 16，并将学习率由 0.002 增大为 0.02 等，上述操作大幅度地减小探索空间和实际运算需求量，实验结果显示损失函数取得了更快的下降，收敛速度明显加快。

特别地，考虑到减少搜索步骤下，为同时保证模型的多样性和随机性，调整增加迪利克雷噪声的时机：在根节点选择动作时添加，而不是在扩展树节点时添加。

核心函数之一 SelfPlay ：调整为单一线程运行游戏，从 ShareStorage 中获得最新的网络，然后存储到 replay-buffer。

• play_game() 函数，使得每一步中使用基于蒙特卡罗树搜索的动作玩一场游戏；

• select_action(node, temperature) 由实际访问的次数和通过游戏相对应的 visit _softmax _temperature _fn(self, trained_steps) 函数返回的参数 temperature，决定下一步动作。temperature 参数可以更改访问次数分布，以确保随着训练的进行，动作选择变得更贪婪。其值越小，越有可能选择最佳行动（即访问次数最多的动作）。

核心函数之一 DiagnoseModel ：理解学习到的模型。

• Muzero 使用自己学到的模型而非真实环境进行游戏，但每步仍然执行一次 MCTS 计算，虚拟的轨迹存在 get_virtual_trajectory_from_obs()；

• Trajectoryinfo() 存下的真实的轨迹相关信息；

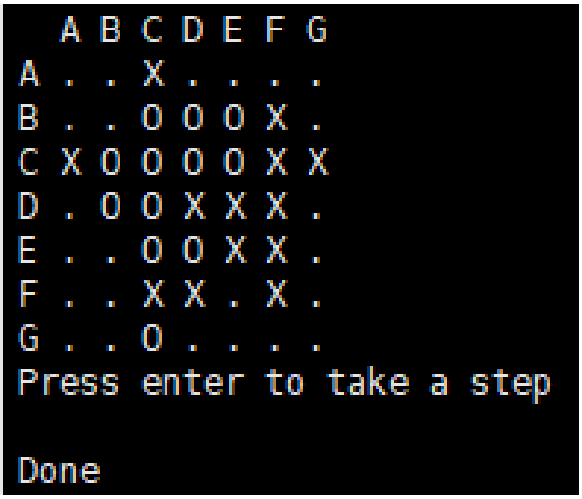
• 在 compare_virtual_with_real_trajectories() 汇总 virtual 和真实的 trajectory，MuZero 先使用其模型，而不是使用环境进行游戏。然后，MuZero 根据先前的轨迹确定最佳轨迹，但在真实环境中执行，直到达到真实环境中不可能的动作。这里默认 K 步为棋盘 $n \times n$ 的总格数，即将棋盘完全下满时。

4.4.2 训练结果

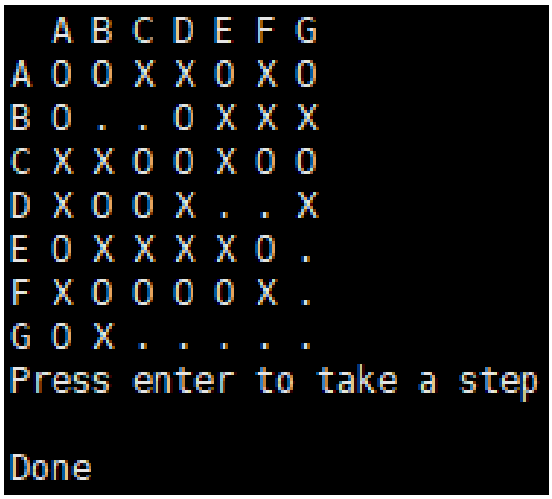
在未充分训练的的 11×11 棋盘下，人机对打，MuZero 作为先手，对局 10 次。此时由于 MuZero 未完全收敛，最终情况是 MuZero 赢 7 局，输 3 局。此时平均平均每步对方计算决策的时间为：2.81s。

随后在缩小后的 7 × 7 棋盘下重复人机对打，MuZero 作为先手，对局 10 次。可能由于棋盘很小，最终情况是 MuZero 6 局赢，4 局平。但此时平均每步对方计算决策的时间仅为：0.79s。

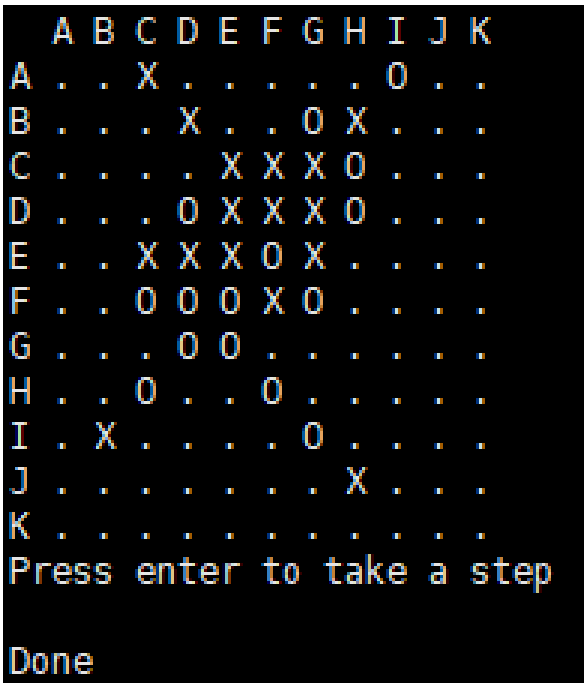
下图4.3为 11 × 11 和 7 × 7 棋盘下 MuZero 赢局，平局和输局情况下的可视化界面，‘x’代表 MuZero 模型一方棋子，‘o’代表己方棋子。



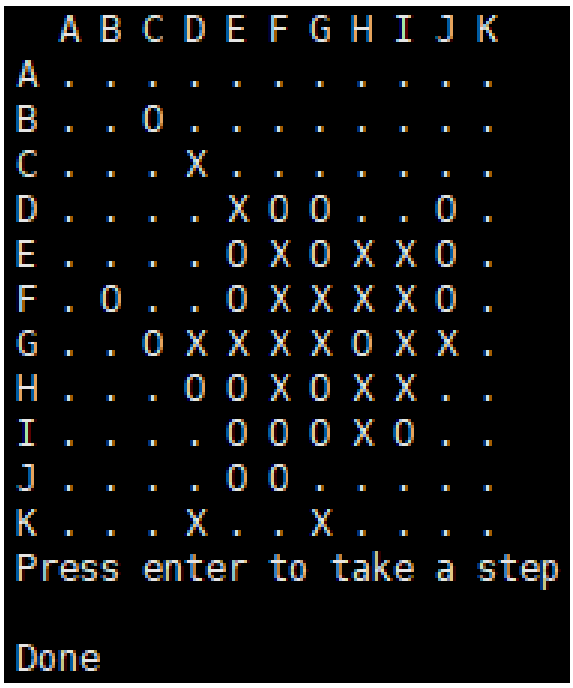
(a) MuZero 在 7 × 7 棋盘上的赢局



(b) MuZero 在 7 × 7 棋盘上的平局



(c) MuZero 在 11 × 11 棋盘上的赢局



(d) MuZero 在 11 × 11 棋盘上的输局

图 4.3　Muzero 棋局结果示例

而且训练得到 MuZero 模型常常为了探索更多空间，除了 MuZero 已学到作为先手会落到中元位置外，其它下棋位置更倾向于分散布局。MuZero 的强大之处在于自发学习模型构建环境的理解，在于其泛化性和通用性。在小棋盘上，在有限棋局上其作用未

能充分显现，同时运行速度明显变慢；而且其训练资源非常有限的情况下，训练得到模型未最优完全收敛。

通过相应结构、搜索策略和参数的调整和改进，可以在保证一定的性能（胜率）的情况下，训练和最终计算速度都明显提升，得到了足够令人满意的结果。

4.5 模型间对比

4.5.1 胜负结果

我们采取 MCTS 方法（试探次数分别为 2000、4000）作为 Baseline 模型，与加入 Zobris 策略的 MinMax 方法（搜索深度分别为 4 层、5 层），AlphaZero 方法（试探次数为 800）进行对比。其中每种方法与其它方法对弈 10 局，各执 5 次先手。所有测试在 11×11 的棋盘上进行。由于 MuZero 方法在 11×11 棋盘上的训练情况一般，故不参加本节的对比。

	MCTS2000	MCTS4000	MinMax-4	MinMax-5	AlphaZero
MCTS2000	-	1:4	0:5	0:5	1:4
MCTS4000	4:1	-	0:5	0:5	2:3
MinMax-4	5:0	5:0	-	5:0	4:1
MinMax-5	5:0	5:0	5:0	-	0:5
AlphaZero	5:0	5:0	5:0	5:0	-

表 4.2 各种智能体对弈结果

表格4.2中每一格分号左侧为横轴代表智能体为先手，横轴代表智能体的胜利次数，分号右侧为纵轴智能体代表的胜利次数。例如 (2,1) 格中的 4:1 代表 MCTS4000 先手并胜利了 4 局，MCTS2000 胜利了 1 局，这和 (1,2) 格中的情况（MCTS2000 先手）是不同的。对于模型自博弈，先后手没有区分。如果胜利的次数总和小于 5，则意味着这些局数是平局。

从上表中可以看出蒙特卡洛树搜索方法在尝试次数在 2000 或者 4000 情况候，性能都不如 MinMax 方法。尝试次数为 4000 的蒙特卡洛树搜索性能优于尝试次数为 2000 的。MinMax 算法之间搜索的最大搜索深度的性能在 11×11 棋盘上没有显著的差异，谁先手则谁获得胜利。对于 AlphaZero 方法，则其稳定优于蒙特卡洛树搜索方法，而与 MinMax 算法水平相当，获胜的情况取决于谁先获得先手。

4.5.2 时间效率

暂且忽略上述几种模型是在 CPU 版本上运行，而 MuZero 模型是在 GPU 上加载；在这里我们进一步比较上述几种方法与 MuZero 方法，在 8×8 和 11×11 棋盘上的平均

每步的决策时间。

其中 AlphaZero 中的 MCTS 树 playout 次数为 800，该参数与前面对局测试中的设置保持一致；该平均每步的决策时间结果通过计算一轮完整对局的总时间除以所走步数得到，棋盘的统一下棋的对象为 MCTS2000。

由下表可以看到，MCTS 搜索需要较多次数的尝试，其所需要的时间最久，在 11×11 棋盘上高达近 20s 的决策时间显然不符合在线部署要求；相比之下使用深度学习代替 MCTS 的 AlphaZero 方法，其效率相对纯 MCTS 有明显改善；调整探索空间和结构的 MuZero 经过相对充分训练后在 8×8 棋盘上表现良好，但由于训练问题，其在 11×11 棋盘上的十几秒的决策时间则非正常，为不可用状态；意料之中，Min-Max 搜索可以快速返回决策，尤其是在加入 Zobrist 技术后，Min-Max 搜索时间得以进一步减少。

	MCTS2000	MCTS4000	MinMax-4	MinMax-5	AlphaZero	MuZero
8×8	5.92	11.70	0.52	2.65	1.95	1.08
11×11	18.40	36.00	0.72	3.38	3.47	12.10

表 4.3　各种智能体平均决策时间 (s)

综合上述各智能体的对弈结果和时间效率来看，Min-Max 搜索能够以较快的运行速度保持不错的性能，似乎仍然是解决在线五子棋博弈问题的最优之选。

五、　讨论

在之前的实验中，我们可以看到基于启发式函数的 MinMax 算法在解决棋盘大小为 11×11 五子棋问题中具有十分良好的性能：不仅显著优于纯蒙特卡洛树搜索（2000 次及 4000 次尝试），而且略优于使用深度神经网络的 AlphaZero 模型（尽管这可能是由于训练没有完成完成）。但 MinMax 存在以下缺陷：1）其搜索复杂程度和最大搜索深度成指数关系，即使我们可以使用 Zobrist 置换表技术减少其计算时间，仍不能避免在搜索深度过深时搜索时间太长的问题。这个问题使得其尽管可以在五子棋问题上具有良好性能，却不能被应用于一些需要更深的搜索深度的棋类问题，例如围棋。2）依赖人的先验和直观。MinMax 搜索中最重要的用以剪枝的依凭是启发式的价值函数，这个价值函数在状态空间不大的五子棋中可以较好地被总结，但是在一些更复杂的问题中，找到有效的启发式函数是困难的，并且限制了智能体的能力。

对于上述问题，我们可以考虑使用轻度的机器学习方法对其加以缓解，例如将计算启发式函数的各种棋形的得分参数化，通过自博弈获得数据集，并对得分进行一个拟合，从而在保留先验知识的同时可以一定程度地脱离人类的先验知识获得更好的结果。

相较于 MinMax 搜索方法，AlphaZero 方法和 MuZero 方法都不依赖于人的先验知识，并且可以通过变动 MCTS 方法中尝试的次数，线性地调控每一步搜索需要的时间，

而且该时间也相对更短。这使得使用这类算法的智能体思考时间长度更加可控，也能更好地应用于其它不同种类的棋类任务。但基于强化深度学习的智能体，训练的过程时间长且更加的复杂而不可控，例如不同的训练样本权重的调整、学习率的调整都会显著地影响模型的性能。而由于深度网络更多时候仍然是一个黑箱，尽管我们可以直观地对一些问题进行解决，却不能够确定是否这些措施可以奏效。

例如在我们训练的过程中，出现了 AlphaZero 模型极具攻击性的问题（见图4.2(b)），这样的模型希望先手能够快速获得胜利、尽量减少步数，而放弃了后手，尽管这类模型显然表现出了对于五子棋规则的理解，但却并不是最优的。我们可以尝试人为地减少这些对局在训练时被采样的频率，或者加大噪声，鼓励模型向更多的走法探索（例如调整 UCT 算法的参数）。

六、 结论

在本项目中，我们对五子棋已有的智能博弈技术进行了一定程度的讨论，对当下流行的 MinMax 搜索和一些基于 AlphaGo 的强化学习方法都进行了深入的了解，并对这些模型及其所使用的基本概念、建模方法和对应的算法都进行了详细的介绍。

从这些主流方法中，我们选择了 MinMax 方法、纯蒙特卡洛树搜索方法、AlphaZero 方法和 MuZero 方法进行更加仔细的学习和实践。针对 MinMax 搜索方法重复搜索节点比例大、搜索时间问题长的问题，我们在五子棋问题中实现了棋类博弈中常用的 Zobrist 置换表技术，并通过测试确定了 Zobrist 置换表技术的正确性和有效性。其次，我们调试并运行 AlphaZero 的代码，并针对不同棋盘大小的模型能力都进行了简单的测试，对其训练过程的特点和训练方式都有了一定的理解。随后在 MuZero 部分，我们主要完成了这个泛用模型的五子棋上的实现，并对其进行了训练及简单的测试，通过相应模型结构和参数的调整，使其能够在相应较短的时间内时间较好的实验结果。最后我们对所有模型进行了人机对弈测试和模型间测试，通过现有所展示的结果，我们认为我们训练得到的 AlphaZero 方法能力和 MinMax 搜索方法的水平相当，这两种方法均优于纯蒙特卡洛树搜索方法；已训好的 MinMax，AlphaZero 与 MuZero 模型都达到了超越非专业棋手的智能水平。

对未来工作的展望上，我们希望继续解决目前在 AlphaZero 实现上观察到的存在问题，并希望进一步联动经过充分训练的 MuZero 模型和其余几个模型进行对打。同时考虑到 AlphaZero 和 MinMax 都是在 CPU 上运行，而 MuZero 是在 GPU 上训练得到的模型结果，后面将主要尝试如何在给定相对同时间与同计算资源的情况下进行更为全面、公平的性能比较，并给出较为统一的分析结果。

七、 分工情况

解润芃（自动化所，202118014628004）同学代码部分主要负责 MinMax 算法的改进及测试，AlphaZero 算法的调试、训练及测试。论文撰写部分主要负责 AlphaZero 相关内容，如蒙特卡洛树搜索和 Zobrist 置换表技术。完成了部分实验结果、讨论及结论部分的撰写。

倪子懿（自动化所，202128014628018）同学代码部分主要负责 MuZero 算法的实现、训练及测试。论文撰写部分主要负责研究背景部分，以及 MuZero 算法相关内容，如 MuZero 算法细节。完成了部分实验结果、讨论及结论部分的撰写。

两人共同完成了前期的调研工作，后期可视化 UI 界面的设计修改和最终代码与文档的整理完善。

参考文献

[1] Hendrik Baier and Mark Winands. Time management for monte carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, 8:301–314, 09 2016.

[2] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[3] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. volume 2006, pages 282–293, 09 2006.

[4] Nicholas Metropolis et al. The beginning of the monte carlo method. *Los Alamos Science*, 15(584):125–130, 1987.

[5] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[6] David Silver, Aja Huang, Christopher Maddison, Arthur Guez, Laurent Sifre, George Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 01 2016.

[7] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy

Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. 12 2017.

[8] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–359, 10 2017.

[9] Albert Zobrist. A new hashing method with application for game playing1. *ICGA Journal*, 13:69–73, 06 1990.

[10] 林华. 基于 self-play 的五子棋智能博弈机器人. Master's thesis, 浙江大学, 2019.