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

五子棋智能博弈相关技术调研与 AI 设计

解润芃 202118014628004、倪子懿 202128014628018

1 摘要

此为 2021 年计算博弈的课程设计报告，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。

2 极大极小值搜索

棋类游戏作为一种完美信息博弈，通常可以定义为一种交替的马尔科夫博弈[1]。在这些博弈中由以下几个部分组成：一个状态空间，表示当前玩家及其所面临的状态，一个动作空间，给定在任何状态下所能做出的合法动作，一个状态转移方程，定义在给定状态和随机输入后的后续状态，以及最终的奖励函数描述了玩家i在某个状态 s 下所获得的奖励r_s^{i}。对于两人的零和博弈$\mathrm{r}^1(s) = -r^2(s) = r(s)$。博弈最终的结果是在游戏结束时，从当前玩家的角度的最后的奖励$\mathrm{z_t} = \pm r(s_T)$。策略 p(a|s) 是定义在合法动作上的概率分布，价值函数是如果所有的玩家都根据策略 p 来进行动作选择的期望结果输出。最优的价值函数可以通过极大极小值搜索（或等价的负值最大搜索）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 剪枝的深度优先极小极大搜索在许多棋类游戏上取得了超越人类的表现。

基于树的搜索方法是棋类游戏智能设计的重要工具，在 AlphaGo 以前的四十年里，最强的棋类游戏均基于 Alpha-Beta 剪枝的极小极大搜索。虽然在围棋上极大极小搜索没有取得很好的效果，但五子棋相对于围棋，搜索空间要小得多，使用极大极小值方法能够取得不错的性能。已有工作也表明，基于传统的 MCTS 搜索的棋类程序比基于 Alpha-Beta 搜索的棋类程序性能要差的多，并且基于神经网络的 Alpha-Beta 搜索在性能和速度上同样无法和基于人工设计评价函数的 Alpha-Beta 搜索相比。因此，对于五子棋等搜索空间较小的棋类游戏，Alpha-Beta 搜索的棋类程序仍然是一种十分有竞争力的技术方案。

2.1 主要原理与思想

2.1.1 基本 Min-Max 搜索

五子棋虽有各种走法，而实际上把每一步的走法展开，其就是一棵巨大的博弈树。在这个树中，从根节点为 0 开始，奇数层表示电脑可能的走法，偶数层表

示玩家可能的走法。极大极小值搜索基于此可在五子棋博弈树上应用。

假设电脑先手，那么第一层就是电脑的所有可能的走法，第二层就是玩家的所有可能走法，以此类推。若假设平均每一步有 50 种可能的走法，那么从根节点开始，往下面每一层的节点数量是上一层的 50 倍，当进行 4 层思考，即电脑和玩家各走两步时，该博弈树的最后一层的节点数为 50^4 = 625 万个。我们可以通过递归来遍历这一棵树。

搜索需要一个评估函数对当前整个局势作出评估，返回一个分数，使我们知道哪一个分支的走法最优。规定对电脑越有利，分数越大，对玩家越有利，分数越小，分数的起点是 0。遍历这颗博弈树就可定下如何选择分支：

电脑走棋的层我们称为 MAX 层，这一层电脑要保证自己利益最大化，那么就需要选分最高的节点；玩家走棋的层我们称为 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。

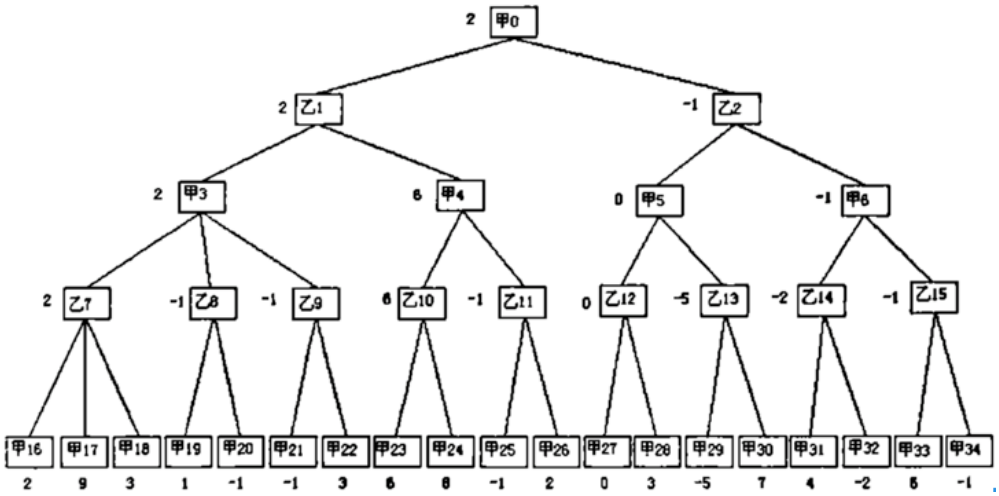

图 1：极大极小值搜索

此图中甲是电脑，乙是玩家，那么在甲层的时候，总是选其中值最大的节点，在乙层的时候，总是选其中最小的节点。

而每一个节点的分数，都是由子节点决定的，因此我们对博弈树只能进行深度优先搜索而无法进行广度优先搜索。深度优先搜索用递归非常容易实现，主要工作是完成一个评估函数，该函数需要对当前局势给出一个比较准确的评分。

2.1.2 基于 Alpha-Beta 剪枝的 Min-Max 搜索

出于对计算量的考虑，MIN-MAX 方法一定需要配合 Alpha-Beta 剪枝策略 Alpha 和 Beta 分别指的是 MAX 和 MIN 节点。因为平均一步考虑 50 种可能性的话，思考到第四层，那么搜索的节点数就是 50^4 = 625 万，一般性能的电脑上一秒钟能计算的节点不超过 5 万个，在实际运行中最好一步能控制在 5 秒以内。

在思考层数设置上，至少为 4 层，否则只看眼前利益，棋力会非常弱。具体设置可以根据对胜算和时间效率进行折中考虑，如果能进行 6 层思考基本可以达到对战普通玩家有较高胜率的水平，如果能达到 8 层或以上的搜索，对普通玩家就有碾压的优势，可以做到 90%以上胜率。

Alpha-Beta 剪枝算法是一种安全的剪枝策略，也就是不会对棋力产生任何负面影响。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。AI 发现这

一步是对玩家更有利的，则拒绝走该步。同理，如果玩家走了一步棋发现其实对 AI 更有利，玩家必定不会走这一步。

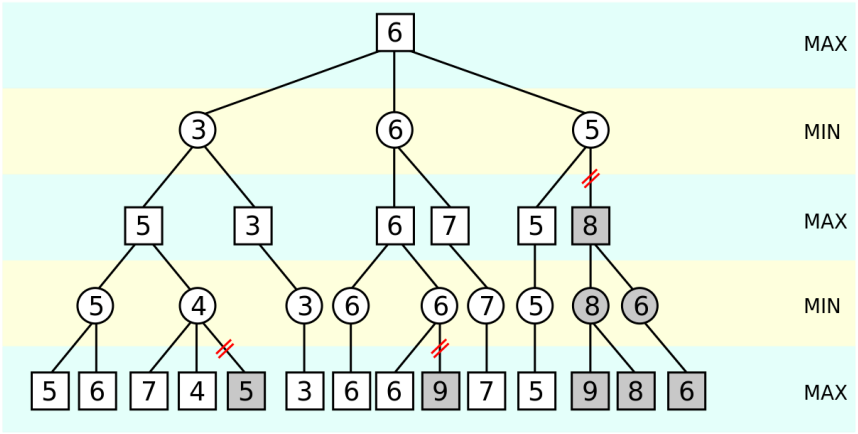

图 2：Alpha-Beta 剪枝

2.1.3 基于 Zobrist 算法优化的 Alpha-Beta 搜索

对于传统的 Alpha-Beta 剪枝来说，可以用置换表技术，即一种特殊的哈希表提高算法效率。Zobrist 算法就是一种非常有效的将局面映射为一个独特的哈希值的游戏局面评估的快速哈希算法，其由 Abert L.Zobrist 在 1970 年发表的 A New Hashing Method with Application for Game Playing 论文中提出，该算法为棋盘上每一个位置的所有可能状态赋予一个绝不重复的随机编码，通过对不同位置上的随机编码进行异或计算的方式来实现在极低冲突率的前提下将复杂的棋局编码为一个整数类型哈希值的功能。

仅使用 Alpha-Beta 剪枝的 Min-Max 搜索过程，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。因此，其实只需要对一次的局面结果进行打分，然后缓存下来，第二次就可直接使用，保存下来以备后续使用的棋盘信息就是置换表。

Zobrist 哈希算法的数学原理非常简单，就是利用异或操作的数学特性:(A^B)^B=A，棋盘的一个唯一的标识就是棋盘的所有状态的异或值，每下一步棋，执行一次异或操作。棋盘的每一个状态实际对应着一个随机数。

Zobrist 算法的主要实现步骤：

将棋局转换为 HashCode。初始化两个 Zobrist[M][M] 的二维数组，其中 M 是五子棋的棋盘宽度。设置的两个数组一个表示黑棋，一个表示白旗。上述数组的每一个都填上一个随机数，一般最好是 64 位。初始键值也设置一个随机数。

要把 HashCode 转换为地址存到 Hash 表中。把 hashcode 转换为地址存到 hash 表中简单的办法就是除表的大小取余数。每下一步棋，则用当前键值异或 Zobrist 数组里对应位置的随机数，得到的结果即为新的键值。如果是删除棋子，则再异或一次。存储每步对应的深度和 Min/Max 值。

2.2 算法描述

2.2.1 基本 Min-Max 搜索

负值最大化形式的极大极小值搜索方法的伪代码如下：

Algorithm 1 Min-Max Decision search

```
function MINMAX-DECISION(state) returns an action
    returns argmax_{a∈ACTION(s)} MIN-VALUE(RESULT(state,a))
function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for each a in ACTION(state)do
        v ← MAX(v, MIN-VALUE(RESULT(s, a)))
    return v
function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for each a in ACTION(state)do
        v ← MIN(v, MAX-VALUE(RESULT(s, a)))
    return v
```

2.2.2 基于 Alpha-Beta 剪枝的 Min-Max 搜索

使用传统 Alpha-Beta 剪枝策略的极大极小值搜索伪代码如下：

Algorithm 2 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 ← −∞
    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 ← +∞
    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 基于 Zobrist 算法优化的 Alpha-Beta 搜索

Zobrist 算法的伪代码描述如下，用于辅助 alpha-beta 剪枝，对于重复棋局面状态可以减少重复计算，提升效率：

Algorithm 3 Zobrist

```
function initialize ZOBRIST()
    // fill a table of random numbers/bitstrings
    table := a 2-d array of size length×width
    for i from 1 to length do //(loop over the board)
        for j from 1 to width do //(loop over the pieces)
            table [i] [j] := random_bitstring()
function hash(board)
    h :=0
    for i from 1 to length do
        if board [i] ≠empty
            j :=the piece at board[i],above
            h :=h XOR table[i] [j]
    return h
```

2.3 程序实现

- 核心函数 get_move ()：这个函数根据当前的状态，寻找最合适的落棋点（第一步总是下在中央）。
- findBestChess ()，search (), __search () 函数用来寻找最好的一步，它要求对于棋盘的每一个空位都进行搜索。
    - evaluate ()：以棋盘上的所有点为中心，向各个方向应用 analysisLine () 函数，生成一个包含有己方各个连续己方长度的记录表，这个记录表最终可以得到对方和己方的得分差（计算方法见 getPointScore ()）。
- getLine ()：获取某一行或者某一列或者某一斜线的所有棋子。
- analysisLine ()：对于通过 getLine () 获得的某一队列棋子，计算这一行中对方棋子的长度，将其分为 XMMMMX, XMMMMP 这样的情况。
    - 如果上述获得的 score 绝对值已经高于 5，说明出现了连续的 5 个，则停止搜索并回溯。
    - 否则则生成一组可以执行的步（棋盘上的所有空位。要求相邻范围已经有下过的棋子。）
        - 这一步骤搜索必胜的步骤（如果存在）并记住。
    - 对上述生成的步进行递归的搜索，并搜索深度为超参，这里执行$\alpha-\beta$剪枝。对 max 和 min 函数都增加一个 alpha 和 beta 参数。在 max 函数中如果发现一个子节点的值大于 alpha，则不再计算后序节点，此为 Alpha 剪枝；在 min 函数中如果发现一个子节点的值小于 beta，则不再计算后序节点，此为 Beta 剪枝。
- （需要补充 Zobrist 部分函数）

2.4 实验结果

3 AlphaGo 系列工作

DeepMind 研究人员研发了 AlphaGo 系列的强化学习算法。2014 年著名的 AlphaGo 的诞生。2015 年，AlphaGo 围棋击败围棋冠军李世石。

AlphaGo 使用了蒙特卡罗树搜索（MCTS）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，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。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 可以像发现新游戏的人一样来通用地玩任何游戏。

3.1 主要原理与思想

3.1.1 纯蒙特卡洛树搜索

蒙特卡洛树搜索，是一个迭代的，最佳优先（best-first）树搜索过程。其目标是帮我们计算出到底应该采用什么样的动作，可以实现长期受益最大化。最佳优先，意味着搜索树的扩展（expansion）是依赖于搜索树中的价值评估（value estimates）。

蒙特卡洛树搜索具有四个主要阶段：模拟，选择，扩展和回溯。通过重复执行这些阶段，MCTS 每次都会在一个节点的在未来可能的动作序列（action sequences）上逐步构建一棵搜索树。在这个树中，每一个节点都表示一个未来状态，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。

3.1.2 Alpha Zero

由 Deepmind 提出的 AlphagGo[2] 是棋牌类智能 AI 技术的突破性的一个进展，在围棋上获得了巨大成功，其使用了大量人类专业选手比赛数据，应用了诸多技巧 [5]。因此，在 AlphaGo 之后为解决这些问题而提出的 Alpha Zero[3] 不再使用人类棋手比赛数据作为训练数据，不再使用手工设计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 Alpha Zero 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。

Alpha Zero 综合权衡了对利用和探索的考虑，核心思想是用一个神经网络搭配 MCST 进行搜索和优化，。神经网络的输入为相对于当前玩家棋盘局面，输出为在各个地方落子的概率（即 policy）和当前局面对于当前玩家而言最终的得分期望（即 value）。

policy 的作用是为 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；value 的作用在于搜索到叶子节点的时候，若游戏没有结束，则以 value 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展开（Monte-Carlo Rollouts），进行一次路径采样，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再何处落子，同时该概率分布也可以作为网络训练 policy 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 value 的监督信号。

从以上算法可知，这是一个不断根据 network 的输出以 MCST 进行对弈采样，然后把对弈结果再拿来更新网络的参数，这样一个不断迭代过程。

3.1.3 Muzero 模型

MuZero 是用于棋盘游戏（国际象棋、围棋等）和 Atari 游戏的最先进的 RL 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[4]。

其主要思想：MuZero 包括三个相互联系的组件用于表示（representation），动态（dynamics），和预测（prediction）。给定上一个时刻的隐状态，动态函数会生成一个当前的 reward 和一个新的隐状态，预测函数根据当前的隐状态进行价值网络与策略网络的计算。初始的隐状态是表示函数根据之前的观察给出的。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（observation）和奖励（reward）。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，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 BPTT 进行联合训练，对三个量预测策略分布，价值函数（value），和奖励值（reward）。

MuZero 针对 AlphaZero 进行比较：AlphaZero 利用规则的主要部分包括 1. 搜索树中的状态转移，2. 搜索树中状态的可获得性，3. 搜索树中的提前中止条件（当搜索树达到结束状态时，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。

3.2 算法描述

- 蒙特卡洛树：这棵树的每一个节点都代表游戏的一个当前局面的确定状态。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。然后选择最有可能获胜的落子。MCTS 伪代码如下：

Algorithm 4 Monte-Carlo Tree Search

```
function MonteCarloPlanning(state)
    Repeat
        search(state, 0)
    until Timeout
    return bestAction(state, 0)
function search(state, depth)
    if Terminal(state)then return 0
    if Leaf(state; d) then return Evaluate(state)
    action := selectAction(state, depth)
    (nextstate; reward) := simulateAction(state, action)
    q: = reward +γ search(nextstate, depth+1)
    UpdateValue(state; action; q; depth)
    return q
```

- AlphaZero：对当前局面状态进行评估，使用 MCTS 和 DRL 这两种方法的组合来选定最佳的落子位置。

    - 在对弈落子时，先是通过模拟游戏走势来进行预判，从而了解哪些位置最有可能在未来成为一个“好”位置，也就是多看几步，我们使用 MCTS 来完成这个游戏预判动作。
    - 然后预判时我们结合深度强化学习训练出的神经网络模型来判断可行的落子位置是“好”还是“坏”，这些落子会导致游戏是胜是负或是平局。

- Muzero：学出一个与真实环境所对应的 Dynamics Model，使其所给出的未来每一步的 value 和 reward 都接近真实环境中的值；基于所学环境模型，在无法与真实环境交互过多的情况下进行规划。

    - 将从真实环境中获取的状态，通过编码 representation function 转换为抽象状态空间的隐藏状态。
    - 该隐藏状态通过前一个隐藏状态和假设的下一个动作进行循环迭代；每一个 step，隐藏状态通过执行一次 MCTS 搜索到的下一个动作。
    - 在抽象的状态空间中，去学习 Dynamics Model 和 value prediction，对每一个隐藏状态上的策略都进行预测，得到 Policy Prediction Network。
    - 隐藏状态下，在抽象状态空间中训练的 Dynamics Model 以及 Policy Prediction Network，可以在初始的隐藏状态以及执行未来 k 步后，**对这未来 k 步的 value 和 reward 的预测**，与真实环境中通过搜索的 value 以及观察到的 reward **尽可能的相近**。

3.3程序实现

3.3.1 纯蒙特卡洛树搜索

。

3.3.2 Alpha Zero

第一步：模拟并获取比赛结果

- 核心函数之一是 start_self_play()：这个函数反复地自我下棋，获得不同的状态路径和落子概率下的胜利情况。
  - 在游戏结束/棋盘放满前不断重复：
    - 首先对于当前结点，棋盘上的所有可以落子的点获得一个落子的概率分布（get_move_probs()）。重复执行_n_playout 手，可以获得积累的下棋概率。
    - _playout() 函数
    - 这个函数贪婪地选择节点直到叶子节点（UCB 策略）
    - 如果这个叶子节点时刻，游戏结束了，则记录胜利者，并更改叶子节点数值。
    - 否则扩张（expand()）这个节点，并将该 DL 网络的估计数值作为当前节点的数值
    - 这里使用 DL 网络预测的 v 代替了常规 MCTS 算法中随机抽取步骤并抵达游戏结束的估计值。
    - expand() 函数
    - 对于当前节点可以落子的所有其它节点，将其加入自己的子节点
    - update_recursive()
    - 更新了 visited_times 等状态用于计算 UCB。
    - 然后根据获得的落子概率，随机获得一个移动的方向，并执行之。
  - 如果游戏结束
    - 确定当下的胜利者和路径中的所有状态（包括蒙特卡洛树的结构以及每一个节点时获得的落子概率分布）
  - 由于上述的过程是对称的，所以可以额外获得一份数据作为竞争者的模拟。

第二步：更新策略

- 首先我们现在获得了一组状态，以及这些状态下的落子概率分布，以及这些状态最后是否获胜的情况。
- DL 模型可以对某一个状态进行一个估计，即当前状态的分值以及其估计的下棋的概率分布。
- 损失函数分为两部分：
  - 状态的值的估计损失，与是否获胜比较；
  - 估计的落子概率和此前记录的访问次数的落子概率进行比较。

相应的调整：

3.3.3 Muzero 模型

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 step，通过一个带有参数θ的模型μ_θ，在每个时间步 t 进行预测。模型预测了策略网络、价值函数、即时奖励这三个量，即拿到过去的观测数据，编码到当前的隐藏状态，然后再给定未来的动作，就可以在隐藏状态空间中进行规划了。

模型预测了 3 个量：

3.4 实验结果

4 几种方法的结果比较

- 胜负结果：
- 时间效率：

5 总结、展望和分工

参考文献

[1] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[2] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[3] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[4] 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. *arXiv preprint arXiv:1911.08265*, 2019.

[5] 兴军亮. 计算博弈原理与应用第十讲 *AlphaGo* 技术演化史. 2021.