目 录

摘 要

长期以来，人机博弈就是一个相当具有挑战的领域，棋类游戏中的人机博弈一直以来都是人机博弈中的一个大的方向。如何实现一个通用的媲美人类玩家甚至超越人类玩家的算法就成为了一个热门的研究方向。在棋类游戏中如何评估当前局面给出下一步最优的落子位置就成为了问题的关键，然而由于大多数棋类问题本身的复杂度，即使是计算机也无法通过简单的暴力穷举的方式来解决问题。由此衍生了许多优化算法，但是遇到如围棋这种问题复杂度极高的问题时，即使人类专业的不断努力，穷尽一切人类经验，实现的算法的棋力往往较为低下，很难击败人类专业玩家。这种算法的实现需要该领域的专家的协同开发无疑对开发者有了更高的要求，并且此类算法几乎没有迁移性，对于其他棋类问题通常需要重新编写。随着深度学习热潮的重现，以 AlphaGo 为代表强化学习的应用使得这个问题迎来了新的转机。

本文主要是采用蒙特卡洛搜索树与残差神经网络实现的一个可在小规模硬件设施上短期训练一个拥有较强棋力的五子棋 AI。参考 AlphaGo Zero 原始论文《Mastering the game of Go without human knowledge》实现的一个在五子棋游戏上的复现，实现过程中采用相应的原创性方法进行改进，使其算法更加适应项目需求并最终取得的较好的效果。MCTS 部分使用 C++编写的带虚拟损失的树并行版本的 Python 扩展，训练管道与神经网络部分均使用 Python 编写。

关键词：强化学习；并行蒙特卡洛搜索树；残差神经网络

Abstract

Man-machine game has been a challenging field for a long time. Man-machine game in board games has always been a major direction of man-machine game. How to realize a general algorithm that is comparable to or even superior to human players has become a hot research direction. In board games, how to evaluate the current situation and give the best position for the next move has become the key to the problem. However, due to the complexity of most board games, even computers can't solve the problems through simple violent exhaustive methods. From this, many optimization algorithms have been derived, but when it comes to a problem with extremely high complexity such as Go, even if human professionals make continuous efforts and exhaust all human experiences, the chess power of the algorithms realized is often low, and it is difficult to beat human professional players. The realization of this algorithm requires the collaborative development of experts in this field, which undoubtedly requires higher requirements for developers. Moreover, this kind of algorithm has little migration, and other chess problems usually need to be rewritten. With the resurgence of deep learning upsurge, the application of reinforcement learning represented by AlphaGo makes this problem usher in a new turning point.

This paper mainly uses Monte Carlo search tree and residual neural network to realize a short-term training of a gobang AI with strong chess ability on small-scale hardware facilities. Referring to the reproduction of gomoku game realized in the original paper “Mastering the Game of Go Without Human Knowledge of AlphaGo Zero”, the corresponding original method is adopted to improve the algorithm in the process of realization, so that it can better meet the project requirements. In the end, the MCTS part is extended by Python, a parallel version of the tree with virtual loss written in C++, and the training pipeline and neural network part are both written in Python.

KeyWords: Reinforcement Learning; Parallel Monte Carlo Tree Search; Residual Neural Network

第一章　绪 论

1.1 研究背景与意义

棋类游戏中的人机博弈算法一直以来都是一个研究的热门领域，然而由于其较指数级别的问题复杂度，使得开发一个媲美人类玩家的算法变得及其的困难。早在十八世纪初，欧洲有制作国际象棋的机器的记录。近代以来，机器游戏的主要研究对象仍然是围棋、象棋、五子棋等棋类游戏。2006 年，匈牙利的 L.Kosis 和 C.Szepprsvari 创新性地提出将 UCT 算法应用到围棋游戏程序中，这被视为围棋游戏中划时代的事件，改变了以往围棋研究一直依赖基于专家知识的静态评价函数的局面。因此，后来一些学者将 UCT 算法应用于围棋游戏之前称为“传统计算机围棋时代”，之后称为“现代计算机围棋时代”，这表明了 UCT 算法思想带来的巨大变化[1]。

“现代计算机围棋时代”之后，研究者对计算机游戏的研究开始转向 UCT 算法，其中蒙特卡洛树搜索算法是研究最广泛的算法之一。虽然早在 1973 年扑克二十一点博弈中就使用了蒙特卡洛树搜索算法，但蒙特卡洛模拟在这个信息不完全、随机性强的博弈中相对容易实现，而对于信息完全的博弈则很难实现[2]。直到 2015 年，Michael Bowling 等人在《科学》中提到，他们在扑克游戏中改进的 CFR 算法已经取得了很好的效果。2016 年 3 月，谷歌公司 DeepMind 研发机构开发的 AlphaGo 智能围棋程序大获成功，以 4:1 的比分成功击败世界级围棋选手李世石，真正证明了算法的实力和潜力。AlphaGo 系统结合了蒙特卡罗树搜索算法和策略值神经网络模型。神经网络的复杂性非常大。它依靠巨大的计算能力在与其他程序的竞争中几乎 100%获胜。2017 年 5 月，经过调整和完善，该程序仅使用神经网络和蒙特卡洛搜索树，再次以 3:0 的成绩赢得了世界冠军柯洁。

机器游戏的另一个研究对象是五子棋。五子棋起源于古代中国。随着这种棋类的发展，其规则也在不断规范和完善。特别是在正式比赛中增加了许多限制性的规则，原来的十九行棋盘变成了现在的十五行棋盘。1988 年，成立国际五子棋连珠联赛，成功举办五子棋国际锦标赛。经过不断的发展，五子棋已成为标准化的国际游戏和世界智力体育奥林匹克竞赛的固定项目。五子棋虽然种类不多，但由于棋路丰富多变，五子棋在机器游戏领域具有很高的研究价值和普适性，吸引了众多学者参与研究，取得了丰富的研究成果。但总的来说，与其他类型的机器游戏相比，现阶段五子棋的大部分人机游戏系统仍停留在传统的游戏类型评估中，并且还存在自学习能力不足和游戏执行延迟等问题，仍需进行深入研究。

综合计算机博弈的发展历程,特定局面下的博弈树搜索方法和局面评估方法

是其研究重点，这两部分内容设计的优劣程度往往直接决定了博弈系统博弈水平的高低。

1.2 本文的研究内容

本文主要是对于常规棋类博弈算法存在的种种问题做出的一些算法上的改进，提出了一种在棋类人机博弈领域类的一种简单高效、易于迁移的、收敛更快算法框架。本文后续行文中使用的棋力博弈中五子棋这一游戏载体设计的算法，但是其实算法设计过程中并没有涉及到除了游戏基础规则外的其他任何人类先验知识，所以可以非常轻松的迁移到其他棋类游戏当中。本文总结将前人在棋类博弈问题上的算法经验，以五子棋这一具有其他棋类典型特征的棋类作为研究对象，重点围绕棋局的局面评估与蒙特卡洛搜索树算法展开了相关的研究工作，提出了基于策略价值网络的局面评估模型与基于蒙特卡洛搜索树的落子决策模型并且基于五子棋博弈问题创造性的提出 p-对立策略选择算法，最后编写并训练了整个算法模型，验证了算法的有效性。

1.3 论文结构

本文通过改进目前现存的基于蒙特卡洛搜索树与卷积神经网络算法提出的一种更加简单、易于训练的算法模型，文章主要介绍了一下目前棋类博弈算法领域的研究意义与发展背景，国内外的发展现状，以及算法框架中各个重要部分的实现原理。大体可以分为七章：

第一章 绪论。主要介绍了论文的研究背景与意义，该领域内国内外的研究现状以及本文的研究内容。

第二章 相关概念和基础理论介绍。对本研究相关的基本概念、基础理论、方法进行了重点介绍。

第三章 蒙特卡洛搜索树的构建。主要分析对比目前存在的博弈树搜索算法之前的优劣，分析并行蒙特卡洛搜索树的实现方式的差异，基于目前常规的蒙特卡洛搜索树算法的改进。

第四章 价值策略网络的构建。主要是阐明了一种基于策略优化的双头残差神经网络，从包括神经网络的输入、网络结构与原理、参数初始化方式等多个方面进行全面的介绍。

第五章 训练管道搭建与项目部署。主要是通过数据生成过程中的并行化问题，生成数据增强、训练模型评估策略、训练管道中的不同超参数调整比较以及 GPU 服务器的项目部署方式展开。

第六章 项目实现与性能评估。主要讲解项目的总体框架与实现过程中的一

些细节问题，从实现高性能的并行 MCTS 的 Python 库和模型的最终性能评估展开。

第七章 总结与展望。对本文的主要研究内容以及贡献进行了总结，并对未来的相关研究工作进行了展望。

第二章　相关概念与理论介绍

2.1 五子棋基本游戏规则和棋型

2.1.1 五子棋基本游戏规则

想较与其他棋类游戏，五子棋的游戏规则相当简单。棋局大小为 15*15，游戏双方执黑白子，黑子先行，依次可在棋盘中空位点落子，当落子使得棋盘中横竖斜角四个方向中构成五子同色相连即为胜利。

在五子棋的发展过程中，人们发现往往先行方的胜率要高于后行方，通过某些特别的开局方式先行方通常可以在对局中获得较大的优势最终赢得棋局，渐渐人们就开始研究在五子棋中是否有先行必胜的走法。1992 年 Victor Allis 通过编程证明不带禁手的五子棋，黑必胜的，在朴素游戏规则下，对于黑方有较大的先手优势。为了平衡黑白方的游戏平衡，我们对于黑方引入了禁手规则。一子落下，如果黑棋同时形成两个或两个以上的活三、活四、冲四，或者一子落下超过五连的，这个落子位置叫做禁手。现在正式比赛时采取的“三手交换”、“五手两打”以及指定开局，开局形状均以“星”和“月”命名。采用指定开局办法的比赛均采用 26 种开局，包括斜指开局和直指开局，如恒星局、流星局、斜月局、寒星局、花月局、游星局等。

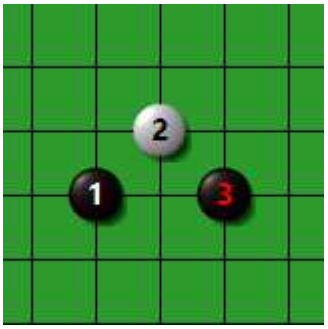

图 2. 1 恒星局

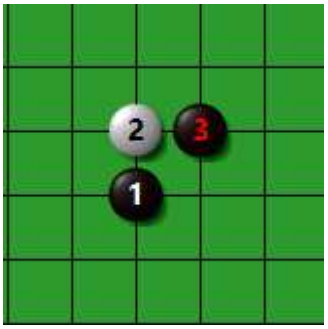

图 2. 2 花月局

2.1.2 五子棋棋型

五子棋的基本棋型大体有以下几种：连五，活四，冲四，活三，眠三，活二，眠二。

连五：任意同一方向下五颗同色棋子相连。

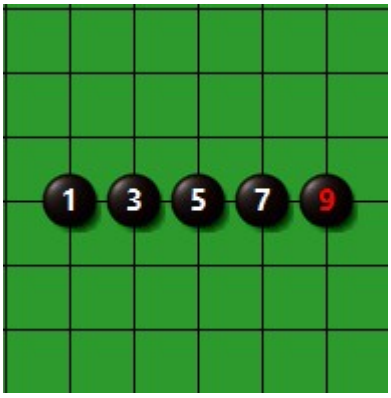

图 2. 3　连五

活四：有两个连五点（即有两个点可以形成五），在这种情况下，单纯防守无法阻止其连五。

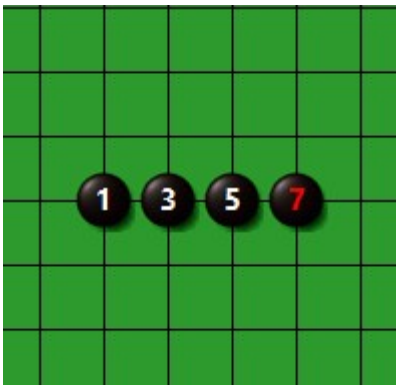

图 2.4 活四

冲四：仅有有一个连五点的四子相连，相对比活四来说，冲四的威胁性就小了很多，因为这个时候，对方只要跟着防守在那个唯一的连五点上，冲四就没法形成连五。

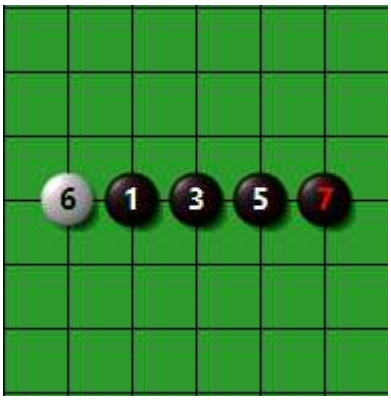

图 2.5 冲四

活三：可以形成活四的三相连，活三棋型是进攻中最常见的一种，因为活三之后，如果对方采取措施防守，将在下一步形成活四棋型。在没有更好的进攻落

子的情况下，需要对其进行防守，以防止其形成活四棋型。

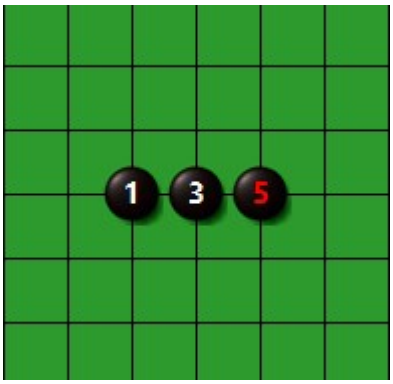

图 2.6 活三

眠三：只能够形成冲四的三子相连，眠三棋型的威胁远不如活三，通常即使不去主动防守，下一步也仅仅是形成冲四棋型。

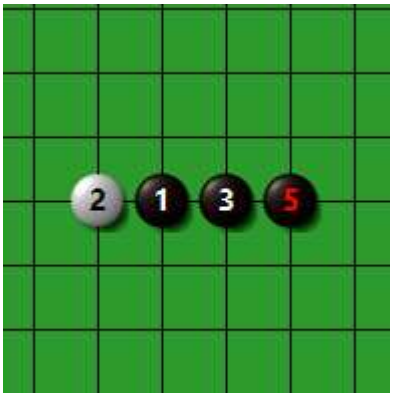

图 2.7 眠三

活二：能够形成活三的二子相连，活二棋型是非常重要的，尤其是在开局阶段，我们形成较多的活二棋型的话，后续连续的形成活三棋型到达绝杀局面。

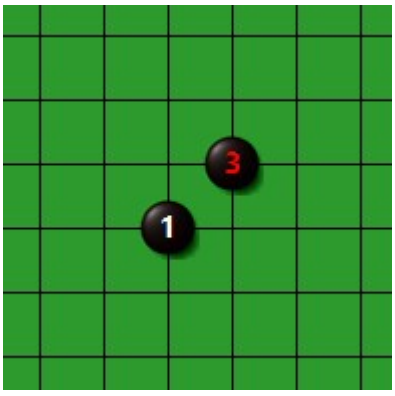

图 2.8 活二

眠二：能够形成眠三的二子相连。

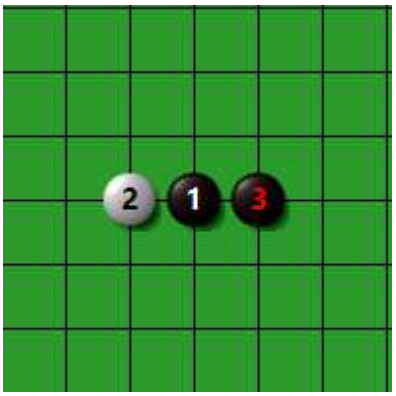

图 2.9 眠二

2.1.3 人机博弈的计算复杂度

人机博弈就是运用博弈论的知识，赋予计算机与人博弈的能力。计算机博弈研究的主要目的是让计算机作为博弈的一方，通过一些复杂的算法去学会人类的博弈思维，既要理解博弈的基本规则，也需要对当前局面有一定的评估能力，据此来进行博弈决策。计算机博弈一直是近年来人工智能领域的热门话题，其中棋类博弈平台是计算机博弈研究和实践领域的一个重要实践平台，通过棋类博弈实践来深入算法技术研究，进而将其应用到更多的领域中。

在计算机科学中，一个算法的计算复杂度或简单的复杂度就是运行这个算法所需要的资源量，特别是时间（CPU 占用时间）和空间（内存占用空间）需求。由于运行一个算法所需的资源量通常随输入规模的大小而变化，因此复杂度通常用函数 f(n)表示，其中 n 是输入量的大小，时间复杂度通常表示为对一个输入值长度所需基本操作（通常是加法操作或者乘法操作）的数量。我们假设基本操作在一台计算机上只占用一个不变的时间量（比如 1 纳秒），而在另一台计算机上运行时，只根据一个常量因子进行改变（比如 k*1 纳秒）。空间复杂度通常表示为算法对一个输入值长度所需的内存量。计算复杂度是用来衡量计算机处理问题所需资源量，也是描述博弈问题的一个标准。在计算机博弈领域中，由于每种棋的复杂度不同，关注点主要有两个方面，分别是状态空间复杂度和博弈树复杂度

从逻辑方面把博弈双方对战的着法抽象成一个数据结构即博弈树，状态空间就是由博弈树的根节点扩展生成的节点的结合,在逻辑上的表现形式就是树。在复杂的棋型中，博弈树可以扩展成无法想象的庞大。树中的节点就是一个个的状态，树枝就是状态之间转换的动作，状态空间的一条路径就是一系列的动作序列连起来的状态序列。博弈树复杂度简单来说就是所建立博弈树的叶子节点的多少，节点太多，博弈树扩展和回溯困难，极大影响效率。表 2.1 为各类经典棋类的复杂度对比情况。

表 2.1 常见棋类的计算复杂度

棋种	棋盘大小	博弈树复杂度（10 做底数）	状态复杂（10 做底数）
五子棋	15×15	70	105
六子棋	19×19	188	172
十九路围棋	19×19	300	172
中国象棋	10×9	150	48
国际象棋	8×8	123	46
国际跳棋	100	31	21

2.2 无人类经验的强化学习

强化学习是机器学习的领域之一。深受行为心理学的影响,主要研究智体怎样在情境中做出不同的行为,从而最大程度的获得累积激励。激励学习系统主要由智能体、环境、状态、动作、奖励等构成[3]。当智能体进行完一个行为之后,状态就会切换到一种新的环境,而针对于这种全新的状态环境就会给予激励信号，然后,智能体按照新的状态和环境所反馈的奖励,并根据相应的策略进行新的动作。上述程序是智能体与周围环境利用姿态、动作、奖赏进行互动的方法。智能体经过强化练习,能够了解自身处于何种状况下,需要做出怎样的动作促使自己获取最大奖赏。

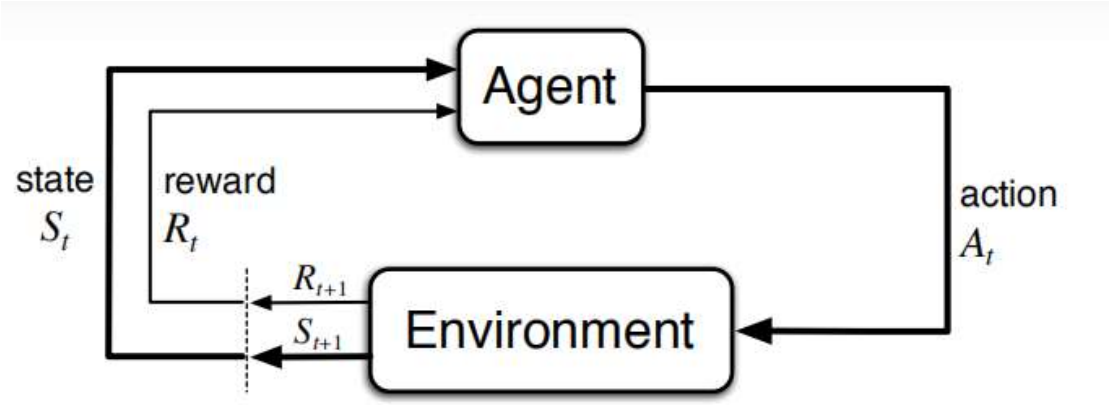

图 2.10 强化学习训练示意图

现在的深度学习通常使用人类专家的数据去训练一个和人类专家类似的模型。但是专家数据经常是昂贵的、不稳定的或者不可用的，即使是可用的数据，但训练出来的模型可能也只能和人类专家相似，而无法超越其学习的对象，在强

化学习中，智能体只知道可执行动作，并不知道其他任何信息，仅仅依靠与环境的互动与每次的动作后的奖励来学习。缺乏先验知识代表着智能体需要从零开始学习。我们将这种从零开始学习的方法称作纯强化学习[3]。也就是本文所使用的无人类经验的强化学习。

2.3 残差神经网络

在卷积神经网络中,如果我们的输入是图片的三维信息,卷积的神经网络就是一种信息提取机,由基本的特点中逐渐提取到了一个更加抽象的特点,网络的层次更高通常意味着通过这能够获取到的更为丰富的信息,并且从更深的网络中获得的特点就更抽象了,也更便于实际的训练[4]。对于一般的卷积式神经网络,如果简单的加大了网络的深度,很容易产生梯度弥散甚至爆炸。对于这个问题的解决方法一般是正则初始化和中间的正则化层，但是这会导致另一个问题，网络退化问题，随着网络层数的增加，在训练集上的准确率却趋近于饱和，甚至是下降[5]。

假设某神经网络的输入是 X，期望的输出函数是 H(X)，则 H(X)就是神经网络期望拟合的复杂潜在映射，但是经过学者反复的试验发现这个映射学习难度比较大。如果将学习的目标设计为残差函数 F(X) = H(X)- X,则原先的期望映射就是 F(X) + X,这里只要 F(X)= 0 就构成了一个恒等映射 H(X)= X。通过这种方式将网络学习的目标改为拟合残差函数，学习起来就会更容易，训练的负担也会减轻。在深度残差网络中，深度残差网络在网络模型中设置了跃层连接，为从 x 到 y 的直接通路，而且深度残差网络中的跃层连接没有权值，因此传递 x 后每个网络块只需学习残差函数 F(X),网络稳定易于学习[5]。

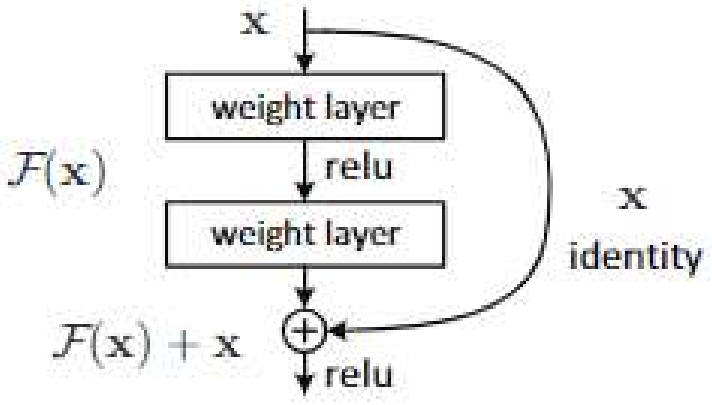

图 2.11 残差连接

第三章　蒙特卡洛搜索树的构建

3.1 博弈树搜索算法

3.1.1 极大极小算法

极大极小算法是计算机博弈理论的经典算法,它由香农在 1950 年首次提出[6]。它是一种找到失败的最大可能性中的最小值的算法,也就是基于对方的最优选择来最小化对手的收益,通常通过递归来进行实现。该算法常用于博弈双方完全对立的问题当中，也即是问题本身不存在一个对博弈双方均最优的解。具体的算流程如下：

（1） 由当前出发向下一层依次随机执行一个合法动作。

（2） 对终端局面进行评估，得到该路径下的最终局面分数。

（3） 反向传播，从所有终端局面出发直到最初的出发结点，每一层分别选择对于该层结点的最优终端局面评估值作为该层结点的实际值。

算法的实际模拟过程如下图所示。

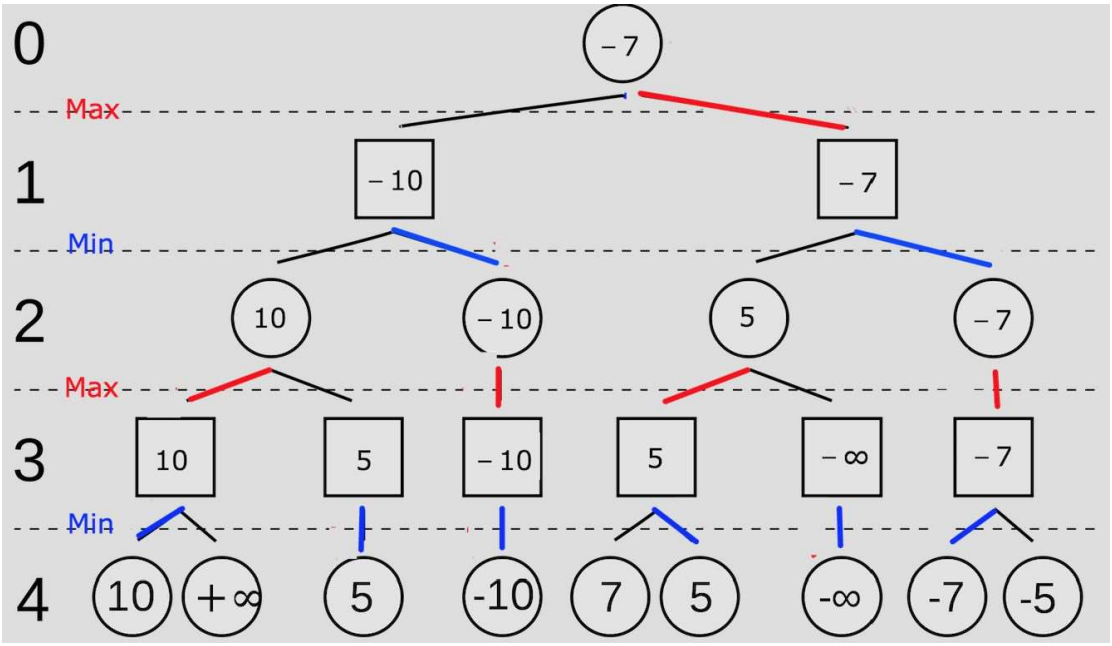

图 3.1 极大极小算法示意图

3.1.2 a-β 剪枝算法

a-β 剪枝算法是由极大极小算法改进而来，能够较好的解决极大极小算法中的不必要的计算问题。该算法利用深度优先遍历的剪枝原理，使得不必将博弈树完全展开,从而加快算法的搜索效率。a-β 剪枝算法利用 alpha 和 beta 参数来优

化极大极小算法的搜索过程，在搜索开始时，alpha 表示 MAX 层收益最高的局面，初始值设为负无穷，在遍历过程中如果其下一层的 MIN 层的评估结点中包含了小于 alpha 值的结点，那么该分支后续结点均可以不用继续搜索计算，直接裁剪掉不会影响最终的计算结果，如图 3.2 所示，该过程就是 α 裁剪。同理 Beta 表示表示 MIN 收益最低的值，初始值设为正无穷，在遍历过程中如果其下一层的 MAX 层的评估结果中包含了大于 beta 值的结点，那么该分支后续结点均可以不用继续搜索计算，直接裁剪掉不会影响最终的计算结果，如图 3.3 所示，该过程就是 β 裁剪。

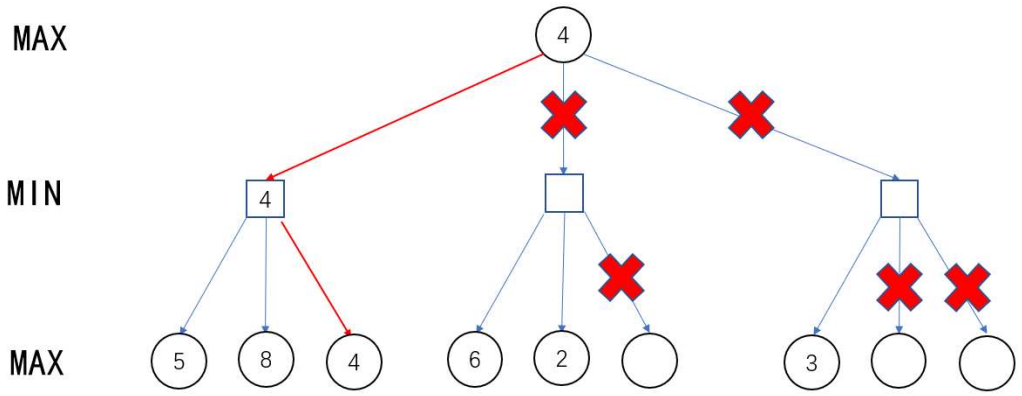

图 3.2 α 裁剪

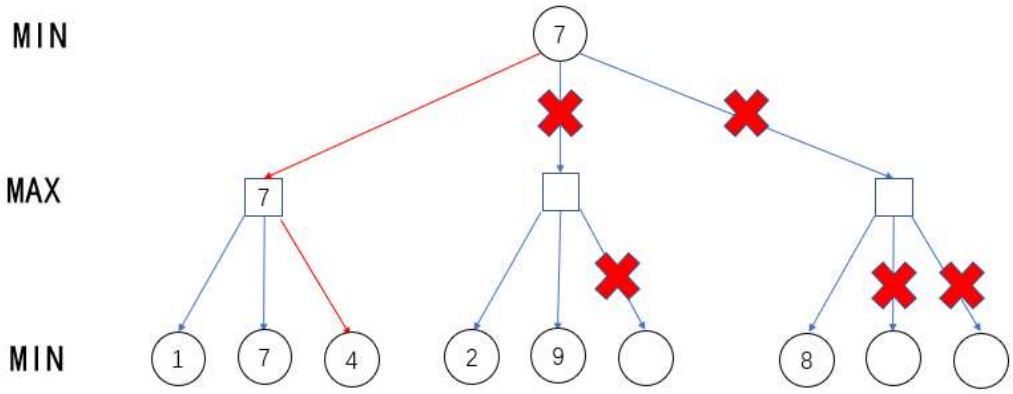

图 3.3 β 裁剪

3.1.3 蒙特卡洛搜索树

蒙特卡洛树搜索算法是一种用于决策的启发式搜索算法，其算法原理主要基于蒙特卡洛方法，是一种以概率统计理论为指导的数值计算方法,又被称为计算

机随机模拟方法。蒙特卡洛树搜索通过蒙特卡洛抽样方法逐步建立和拓展博弈树，在树内一般采用贪心算法，树外采用随机策略[7]。由于结合了随机模拟的一般性和博弈树搜索的准确性，使得更有可能成为最优着法的分枝获得更多的搜索机会，在有限的时间内使用有限的资源提高搜索的准确率[8]。蒙特卡洛树搜索算法分为四步，分别是选择、扩展、模拟和反向传播，具体流程如下：

(1)**选择** 从根节点开始，无再可拓展结点后，通过 UCT 函数计算选择一个最有潜力的子结点，即 UCT 值最大的结点，UCT 函数的定义如下：

$$\mathbb{UCT}(v_i, v) = \frac{Q(v_i)}{N(v_i)} + c\sqrt{\frac{\log(N(v))}{N(v_i)}}$$

其中 Q 值表示当前结点的模拟获胜次数，N 值表示当前结点的总模拟次数[9]。

(2)**扩展** 扩展是对可拓展的节点进行的，即随机添加一个新的子节点。

(3)**模拟** 模拟是对上一步扩展出来的子节点进行一次模拟游戏，双方随机下子，直到分出胜负。

(4)**反向传播** 从扩展出来的子节点向上回溯，更新所有父节点的Q、N参数，即获胜次数和被访问次数。

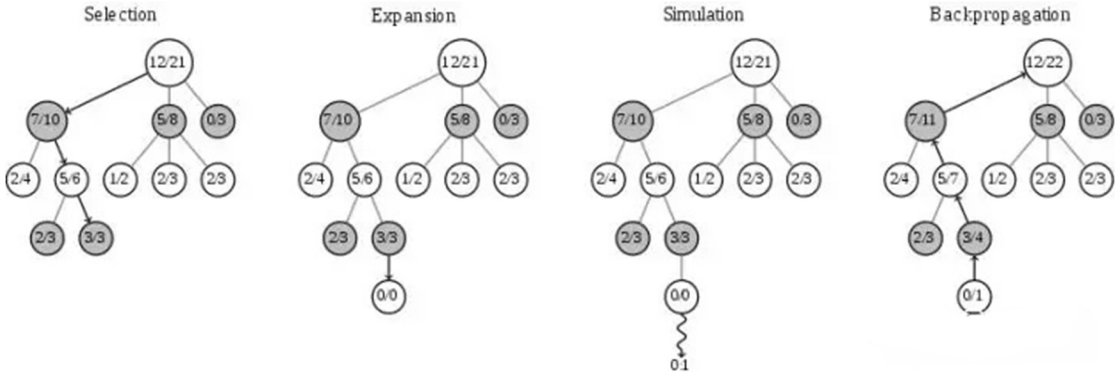

图 3.4 蒙特卡洛搜索树流程图

3.2 并行蒙特卡洛搜索树

MCTS 中每个模拟的独立性意味着该算法是一个很好的并行化目标。并行化的优点是可以在给定的时间内执行更多的模拟，并且可以利用多核处理器的广泛可用性。然而，并行化带来了一些问题，例如在单个搜索树中组合来自不同来源的结果，以及线程的同步[10]。蒙特卡洛搜索树的并行方式主要分为三种：

(1) 叶并行，即在叶子结点扩展示进行并行。

(2) 根并行，即直接使用进程或线程创建多个不同的树，在不同的树中同时执行搜索。

(3) 树并行，即多个线程在同一个树中进行并行，每个线程在树的不同部分执行搜索。

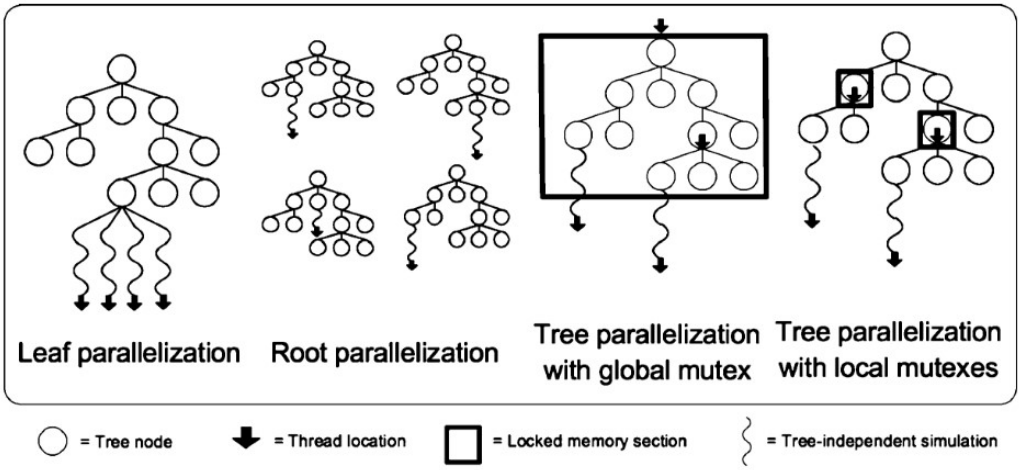

图 3.5 蒙特卡洛搜索树并行方式

3.2.1 叶并行

传统非并行化的蒙特卡洛搜索树在模拟过程中是在一个当前局下的随机过程展开进行的一盘随机落子的自对弈博弈，最后该点处的胜败将由上述的这一次随机博弈的最终结果决定。该点的模拟博弈在整个搜索过程可能会遇到多次，但是其实每一次的前置的落子的点并没有发生改变，也即是这一步是具有并行性的。我们完全可以不用线性的每次选择到该点时才进行一次模拟。当该结点被选择过一次时，我们并行的进行多次模拟，最终的结果进行累计的反向传播，这样在数值上我们最终模拟得到的最终该点可能的胜负率会更加准确，也就是其实我们是在叶子结点出进行的并行，最终的效果是增强每一步的反向传播数值的可靠性，增加了其棋力的稳定性，但是并没有增加其模拟量。

3.2.2 根并行

根并行的蒙特卡洛搜索树是其并行算法中最为简单明了的一个方式，它的原理其实就是对当前局面下进行并行的建立多个蒙特卡洛搜索树，最后取这些搜索树中的最优解作为当前局面下的最优决策落子点，在并过程中不同线程建立的蒙特卡洛搜索树直接其实是完全的独立不想干扰的，不同线程建立的蒙特卡洛搜索树之间的差异其实是在于模拟过程中的随机走子过程导致的，但是由于不同线程间的蒙特卡洛搜索树没有相互的信息传递，所以在实际的运行过程中其实是有部分的重复计算量的，但是这种并行方式的好处也非常明显，实现起来比较简单，而且由于树间不想干扰所以实现过程中并不需要使用到锁，效率上非常高，而且

不同于叶并行，根并行的方式是真正意义上的增加了其模拟量，采用这种并行结构，落子决策的棋力将更强，但是缺点也同样明显，根并行方式实现过程中需要更大的内存需求，并行过程中单步棋力的提升近似线性，但是对内存的增长要求同样也是线性的，并且由于只在根处并行，所以其实上有大量重复的非优落子点浪费了部分的算力，并没有将全部的算力用在最该用的地方。

3.2.3 树并行

树并行的蒙特卡洛搜索树是上述三种并行方式中最为复杂的，也是本文中采用的实现方式。由于是树内的并行，所以需要处理大量的线程间的通信问题，在原理上其实它是结合了叶并行与根并行，也即是我们摈弃在单一的叶子结点处做并行提高落子棋力的稳定性或者在根处做并行，增加其模拟次数来提高棋力的办法。我们进行树内的完全并行，也即是在一个蒙特卡洛搜索树中，我们同时的存在其中的选择、扩展、模拟、反向传播四个过程，做到树内的完全并行化。但是由于这四个过程的天然串行性，直接的简单并行是行不通的。首先遇到的一个问题就是如何并行，我们的每次选择其实都是在反向传播后才能得到更新，但是模拟和反向传播过程非常耗时。如果非要等到反向传播过后才能进行选择，那么其实整个算法的并行度是非常低的，因为在多数情况下树并行总是在重复的选择同一个结点进行扩展和模拟。为了使得选择过程中即使是结点没有进行反向传播更新，我们也能选择更多样的结点，所以在这里引入了一个称为虚拟损失的结点变量。当某个结点被选择一次时其虚拟的损失量将增加[10]。反向传播时会被复原，但是我们在进行利用 UCT 算法计算最有选择结点时会将其损失量进行考虑，也就是说在理论上其实这个虚拟损失量就是反向传播过程中结点访问次数的前置体现。树并行的蒙特卡洛搜索树相较于传统的串行版本的蒙特卡洛搜索树而言，在落子决策稳定性以及棋力本身都有很大的提升，灵巧的使用原子变量和锁可以实现一个相当高效的版本，这一部分较为复杂，我将其放在详细讲解。

3.3 改进的蒙特卡洛搜索树

3.3.1 前置访问次数

在上述的树并行版本的蒙特卡洛搜索树中我们提到过虚拟损失变量的意义，它主要是使得在并行的选择过程中，不用等待反向传播更新参数后才能选择新的的结点，而是在计算结点 UCT 值进行选择时减去虚拟损失量与虚拟损失系数的乘积，使得结点的悬着更加的多样。我们说过这个虚拟损失量其实就是相当于方向传播过程中结点访问次数的前置体现，但是他其中其实存在一个人为设置的参

数系数，这种硬性附加的惩罚包含有人了的先验经验，在实际的开发过程中效果可能并不完美，而我们知道反向传播过程中的这个访问量其实在模拟过程中就已经被引入了。也就是其实我们在拓展的过程中就可以提前更新这个结点的访问量了，而不用等到反向传播时更新。这种提前更新的好处是，我们可以不用使用到虚拟损失变量就能达到多样性选择的目的，通过引入额外的统计量会取得了更好的勘探开发折衷[11]。

3.3.2 p-对立策略选择

1. 强化学习中的局部最优问题

同传统的深度学习相比，强化学习的“样本效率”较低，并且即使定义了一个较好的 reward,深度强化学习也总是容易陷入局部最优。

经典的探索—利用困境，它是困扰强化学习的一个常见问题：你的数据来源于你当前的策略，但如果当前策略探索得过多，你会得到一大堆无用数据并且没法从中提取有效信息。而如果你进行过多的尝试（利用），那你也无法找到最佳的动作。

对于解决这个问题，业内有几个直观而简便的想法：内在动机、好奇心驱动的探索和基于计数的探索等。这些方法中有许多早在 20 世纪 80 年代以前就被提出，其中的一些方法甚至已经被用深度学习模型进行过重新审视，但它们并不能在所有环境中都发挥作用[3]。

2. 五子棋 AI 训练过程中的问题

正如前面所言，长期以来，深度强化学习中的一个问题的就是如何解决训练过程中的局部最优。通常而言，与传统深度学习相比由于其样本的利用率较低，强化学习中的局部最优问题往往更为棘手，这其中会浪费大量的计算资源，在我们本文这个五子棋 AI 训练过程中同样也遇到了相同的问题。起初我们的五子棋智能体的预测最优落子位置是趋于随机落子的，AI 认为棋盘上的每一个空位的落子概率应该都是近似的，所以在起初的训练过程中，每一盘棋局基本都要落子至近满盘才能分出胜负。此时的 AI 自对弈博弈过程中还没有理解游戏规则，近乎与随意的落子，然而随着训练过程的深入，在 AI 自对弈 300 局后，每一盘结束时对局的回合数都在缩短。我们可以发现 AI 逐步的明白了游戏的规则，只有在五子连线的情况下才会取得胜利，我们具体分析发现智能体对于特定局势的每个落子空位给出的落子概率逐渐变得有选择性，而不是简单的随机。所以可以发现其实是在训练过程中 AI 慢慢学会了规则，他会在个别的更容易胜利的位置进行落子，但是随着自对弈的进一步进行，我们的智能体陷入了一个局部最优的恶性循环当中了由于机器硬件有限，无法保存足够多的历史样本数据，以及单一训

练的缘故，AI 训练到后期容易落入一个局部最优局面即 AI 只攻不防。而此时由于没有足够多的数据或者其他的训练模型生成的数据将 AI 从中解救处理，AI 容易长时间陷入局部的恶性循环当中：9 步结束，后手方认为无论怎么下后手都必输，AI 各下一行不防对面，这样必然是后手输，而这个结果又反过来印证了它认为的后手必输。

3. 算法原理

在传统上解决此种基于蒙特卡洛搜索树的强化学习所遇到的局部最优问题多是使用的增加 playout 的次数与减少狄利克雷噪声的参数来来解决。说到底其实是希望通过使得噪声点分布更加尖锐来整加落子的多样性与思考深度[12]，希望 AI 能通过自身去跳出这个局部最优。虽然多数情况下我们通过上述优化以及增加训练时长，最终是可以从这个局部最优状态逃离出来，但是这种方法本身来说只是治标不治本，只是起到了一个缓解作用。并且由于强化学习对于样本的利用率较低，而整个训练过程中其实样本的生成过程是占用了其中绝大多数时间的，这样我们的代理智能体在很长一段时间中将算力用于无意义的重复数据的生成当中，这无疑不是浪费了大量的算力。

回到五子棋问题本身，这里存在一个简单的先验经验，即对于五子棋游戏而言，本方胜算不大时，对方的下一步最优落子点其实对于当前方是隐性的强防御手，即在五子棋游戏对方的下一步最优落子点应该作为当前方最优落子集合中的一个元素。当我们给本文的神经网络引入此归纳偏执后，便可以极大的提升智能体的防守能力，但是当什么时候引入这个归纳偏执就变得相当重要。首先如上文所言，不引入该归纳偏执，那么智能体在整个训练过程中将陷入只攻不防的局部最优。如果我们在智能体认为以极大概率要败时，就完全使用对方下一步的最优落子点作为当前的落子点，那么智能体的整体行棋风格又会趋于保守，将很难学习到一些险招。

综上分析我们得出一个结论上述归纳偏执是有使用的必要的，第二频繁的使用该归纳偏执的预测效果并不好。故本文中提出一种基于概率的归纳偏执---p-对立策略选择，即当 AI 某一步 Value 头返回的价值低于一定的阀值时（AI 认为大概率要败），此时我们让他以一个概率 P 将下一步落子落于对手的下一步最优的落子点，这样就可以防止 AI 只攻不防，明知道会输还下某个位置和要输时就乱下的问题。并且这个阀值与概率 P 是在训练过程中通过生成棋谱的状况进行动态的调整的，所以其实一方面我们使得 AI 生成的棋谱落子上不会过于的激进，只攻不防，另一方面也不会让其过于保守，AI 同时也会学习到通过连续冲四，双连三等对方必防定式的走法达到胜利。

3.4 本章总结

本章从博弈树搜索算法出发，依次的介绍了博弈树的基本概念，并且讲解了传统的博弈树算法与其优化策略，然后引入了本文中使用的蒙特卡洛搜索树，讲解了其基本的原理，并且分析了蒙特卡洛搜索树并行的三种常用方式与其优缺点，最后我们引入了本文使用的并行蒙特卡洛搜索树的两个改进的算法策略，并对其进行了详细的说明，为我们后文进一步展开奠定了基础。

第四章　价值策略网络的构建

4.1 网络输入输出数据结构

在深度学习中，神经网络的构建大题上分为三个主要部分，即输入层、隐藏层、输出层，整个神经网络部分通常看做黑盒，我们并不关心机器是如何从输入层到输出层的映射，我们只用知道深度神经网络几乎可以拟合所有的未知函数。对于具体的问题，输入层提供是应该是包含预测信息所需的原始数据，输出层就应该是我们要想得到的预测数据，所以要想明确输入数据结构，我们要先明确对于本文中的神经网络需要的预测数据。

4.1.1 网络的输出数据结构

我们最终需要得到的是给定一个当前的棋面状态，我们的神经网络可以返回出当前棋盘中各点的落子概率，并且由于蒙特卡洛搜索树的模拟过程中传统是基于一个随机走子的策略最终直到一方胜利为止。正如上一章所分析所分析过的这种基于随机走子的策略会使得最终 AI 棋力的不稳定，但是如果不进行这种随机的走子模拟我们就很难去对当前的局面优劣进行一次客观可分析。结合原始的博弈树算法，例如在极大极小值算法中，我们其实是人为的定义了一个专家系统，这个局面分析函数利用了许多人类的对于该棋力游戏的先验经验对各个固定的局面进行了一次打分，这其中的缺点在上文中已经提及过这里就不再进行进一步的展开了。现在当我们使用到深度神经网络时，使用神经网络去拟合这个之前的这个局面评估函数，就不需要为每个棋类游戏单独的编写局面评估函数，我们只需要定义好一个奖励函数，然后通过强化学习，使得神经网络自行的训练学习到这个局面函数。也就是说我们的预测输出还应该有个当前局面的评估返回值，这个返回值可以是在当前局面下，目前落子方的获胜概率，更加上述的讲解，所以我们的深度神经网络的返回值应该有两个，以 15*15 的标准五子棋棋盘而言，神经网络应该有个双头输出，一个是 225 维的向量，其中的每个值即使该点的落子概率，概率总和是 1，另一个是一个标量，代表当前落子方的获胜概率，范围是 [-1,1][12]。

4.1.2 网络的输入数据结构

首先最基本的要想表示一个棋盘的目前状态，我们需要表示出现在棋盘中黑子和白子的落子位置，但是仅仅这个数据是无法包含提供我们预测分析需要的全部信息的。正如上文提到过的，在没有禁手规则下，五子棋中先手是有极大的优

势的，那么同样的局面，但是落子方是否是先手方就变得尤为的重要，且这个信息我们是无法通过简单的局面信息来推导得出的，所以最基本的我们需要提供一个 3*15*15 的张量数据。前两层分别是黑子与白子在该点处是否有落子，有则为 1，无则为 0，最后一层表示当前落子方是否是先行方，如果是则全部为 1，否则全部为 0。

理论上上述的输入数据已经包含了我们预测所需的全部的隐性信息，但是在本文的后续实验的训练过程中发现，在这种输入数据下，神经网络的损失的收敛输的非常的缓慢。结合人类的五子棋对弈中的分析落子点的方法后，我们发现对于棋类游戏中，其实绝大多数点是并不适合我们落子的，我们通常考虑的落子点都在于博弈双方对弈时前几步的落子点附近，这种方式其实在原始的基于人类经验编写的棋力算法中就有利用到过，这种方式背称为是启发式搜索方式[10]。那么其实在通过使用神经网络进行预测分析中，由于博弈双方最近的落子点也是无法通过简单当前局面进行分析得到的，但是这个信息对于我们最终的预测结果显然是有很大的帮助的，所以我们也可以引入这个信息。对于五子棋游戏而言，游戏胜利的条件是五子连线，所以通常而言最近 6 步双方的落子信息就足够表示整个博弈过程中需要的前置落子位置了，所以最终来看，我们最后的输入层应该是一个 13*15*15 的张量数据，其中前 12 层分别是双方近 6 步的棋面状况，最后一层表示当前落子方是否是先行方。更具体而言就是前 12 层数据分别依次展现近 6 步的双方的落子形成的棋谱，其中一个状态下的棋谱有 2*15*15 的张量数据表示，其中分别是当前落子方的棋子在棋盘中的位置和对方棋子在棋盘中的位置，有落子则为 1，无落子或者是对方的落子则为 0，最后一层表示如果当前落子方是先行方，则整个 15*15 的张量数据全部为 1，否则全部为 0。

4.1.3 输入数据结构的隐性问题

在一开始我的网络输入近六步黑子与白子的落子情况外加一个层表示当前的落子方颜色，确实这种表示方式已经足够表示棋盘的当前局面信息，但是在后续的训练过程中极其坎坷，非常容易遇到梯度消失的问题，而且网络的泛化能力很弱。在人工调试对局时发现几乎同样的局面，只是双方颜色交换，AI 在一种颜色下的表现尚可，但是另外一种情况下确有问题，这个情况让我意识到，起初的这个网络输入是有问题的。因为参考 AlphaGo Zero 的原论文[12]，由于五子棋的棋谱也是可以对称翻转的我们对数据进行了一个加强，使得其生成 8 中相同的同等的数据，而对于我上述这种网络输入来说，虽然只是颜色的翻转，但是对于网络输入来说这只两种完全相反的局面。而在实际的对局过程中其实同样的对局数据颜色翻转的概率并不大，导致 AI 反转颜色的对局预测不尽相同，并且我个人

认为对于完全相反的颜色棋谱输入，最后一层的当前颜色信息对最后的结果有非常大的直接影响。换句话最后网络输入的最后一层信息占比的权重远大于其他层，间接的导致了上述问题的出现，而参考原始论文，发现其实对于输入数据的描述是当前落子玩家的棋子分布与对手的棋子分布外加一个当前落子方是否是先手方，其实在输入数据包含的信息量上这两种方式并没有区别，但是后者包含的信息明显是更加利于网络去学习的。而且也不会遇到上述中单纯翻转双方颜色就导致棋力发生明显变换的情况，因为其实翻转双方颜色其实在输入数据上的改变只是最后一层，而前面十多层其实并没有变换，也就是说其实通过一个简单输入数据方式的改变，使得网络中同时学到了一个颜色翻转情况的预测。这种细微的差别其实如果单纯读论文时比较难注意到，只有真正去复现时遇到问题是才会知道这个问题。

4.2 神经网络的结构与设计原理

4.2.1 神经网络结构

本文中算法只采用了一个神经网络，即策略-价值网络，它包含有两个输出头，分别输出当前局面下的棋盘中每个位置是最优落子点的概率与当前棋盘局面的一个评估值，其主体使用的是一个残差神经网络，包含总共有 17 个卷积层，由 7 个残差块组成，其中还包含若干个全连接层，具体结构如下图所示[12]：

图 4.1 网络输入层

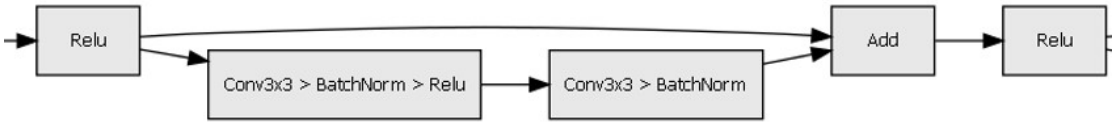

图 4.2 残差层

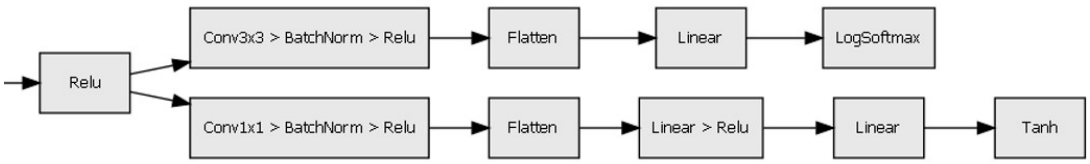

图 4.3 双头输出层

本文的神经网络的输入数据结构在上文中已经讲解过了，这里只按上文中所示结构将数据传入输入层即可，因为输入数据本身就是二值矩阵数据，所以无需进行归一化处理，接下来输入数据将进行三步处理最终得到我们需要的输出值[12]。

（1）将棋盘数据输入卷积层,这里的卷积运算使用了 256 个大小为 3*3.步长为一的卷积运算核用来获取五子棋盘特征.。为防止边缘数据的流失,对于棋盘数据,先对输入数据进行了边缘大小为一的零扩展处理,当卷积运算完成后获得尺寸为 256*15*15 的训练数据,当我们数据完成归一化处理之后,再通过 Relu 激活训练函数。

（2）此阶段由 7 个残差块组成,数据在残差块中的每个卷积层都需进行边长为 1 的零扩展，这阶段的卷积运算使用 256 个大小 3 *3,步长为 1 的卷积核。

（3）深度残差网络输出的数据分别传入策略层与价值层，策略层先将数据进行边缘为 1 的零扩展，再用 2 个大小为 3*3 步长为 1 的卷积核进行数据的降维，数据归一化处理后将其通过全连接层映射到 225 维的向量，该向量通过 softmax 激活函数转化为当前局面下每个位置是最优棋盘落子点的概率，价值层先将数据进行边缘为 1 的零扩展，并用 1 个大小为 1*1 步长为 1 的卷积核进行数据降维，数据归一化后使用一个 flatten 层和全连接层先数据从 1*15*15 映射到 256，在通过另外一个全连接层将数据映射为一个标量，最后通过 tanh 激活函数将数据范围换化到[-1,1]，这个标量代表的就是当前局面下的预测胜率，1 代表必胜，-1 代表必败[12]。

4.2.2 设计原理

1. 为什么使用卷积神经网络

对于棋类人机博弈 AI 而言，我们在上文中已经展开分析了，我们需要对于给定的当前局面信息给出下一步最优的落子策略，对于棋类游戏来说，棋面的局势是存在定式的，这些定式的组合排列直接是影响整局的双方的胜负。对于本文所研究的五子棋而言，这些定式就是在第二章中所介绍的五子棋的基本棋型，所以我们希望使用的神经网络可以识别出棋局中的这些棋型结构，并且从中分析出合适的落子位置，其实这种思考方式和人类的落子策略相当接近。深度学习在图像处理领域中一个常用的网络层就是我们本文中所使用到的卷积神经网络层，相对于传统的全连接层，卷积神经网络的优点在于它有效的利用了视觉领域中的两个有效的归纳偏执---局部性与空间不变性[4]。局部性指的是空间位置上的元素的联系性或者相关性与他们在空间位置的远近有关系，更具体来说，就是在图片中离着越近的两个元素他们的相关联性就越大，空间不变性指的是在欧几里得几

何中，平移是一种几何变换，表示把一幅图像或一个空间中的每一个点在相同方向移动相同距离。比如对图像分类任务来说，图像中的目标不管被移动到图片的哪个位置，得到的结果（标签）应该是相同的，这就是卷积神经网络中的平移不变性。平移不变性意味着系统产生完全相同的响应（输出），不管它的输入是如何平移的 。平移同变性意味着系统在不同位置的工作原理相同，但它的响应随着目标位置的变化而变化。卷积神经网络的这两个归纳偏执对于识别图像数据中的固定特征有着相当大的帮助，这使得卷积神经网络在我们本文中棋面中的棋型上便有着相当不错的效果，在本文整个神经网络框架中卷积神经网络的作用就相当于是在输入的棋面数据中做棋型特征提取，为后续的输出头通过更好的特性去学习，并且由于卷积神经网络本身是通过权重共享的稀疏连接，相比于全连接层，卷积神经网络在训练上需要的计算量更小，更加利于我们中小型硬件设备的后续训练。

2. 为什么使用残差神经网络

在本文中的第二章中就已经讲解过了残差神经网络的好处，这也也不再进行赘述，其实最初版本的 AlphaGo 在价值策略网络的隐藏层中使用的就是纯粹的卷积神经网络的重复堆叠，但是正如在残差神经网络中所提到过的，过深的深度神经网络在训练的过程中会遇到梯度弥散与爆炸导致网络的问题，在这个过程中我们需要耗费大量的时间去微调神经网络中的超参数，并且更为致命的问题在于网络的退化上[5]。使用残差神经网络可以非常容易的规避上述的这些问题，让我们的神经网络的最终性能更好。

3. 为什么使用双头输出层

在 AlphaGo 中价值网络和策略网络时分开的，我们在训练过程中需要训练两个网络，但是其实价值网络和策略网络的输出在某种意义上是有很强的相关联性的。他们都同时需要对于当前局面的数据的特征进行提取，并且如果智能体对于某个落子策略有很高的置信度，那么他对于当前局面的获胜概率同理应该有一个较为准确的估计，反之如果智能体对于所有的落子策略的置信度都近乎相同时，它对于当前局面的获胜或者落败的判断同样就无法有个精确的估计，所以在顶层的特征信息的提取上，这两个神经网络应该是近似相同的。并且共用同一个神经网络无疑是节约了单独训练两个神经网络的算力了，使得整个网络更加的精简，也利于训练[12]。

4.3 损失函数定义

上文提到过在本文中的神经网络部分使用的是一个基于残差神经网络双输出头的网络结构，所以我们的损失函数部分能需要同时的对这两个输出层的最终

结果进行优化，首先策略输出层返回的是棋盘中每个落子位置的概率，我们利用的是 softmax 函数进行坐标位置概率的映射，我们可以视为分类问题，因此对于策略网络使用的损失函数是常用的交叉信息熵函数。价值输出层返回的是当前局面价值的评估值，对于这种具体数值的预测问题我们通常使用的损失函数就是均方损失函数。最终我们的损失函数的数学定义就是如下：

$$l = (z - v)^2 - \pi^T \log\, p + c \parallel \theta \parallel^2$$

其中 z 是预测的当前局面的获胜或者落败的概率，v 是实际最终获胜或落败的结果，胜为 1，败为-1，π 是当前局面实际的每个位置的最优落子概率，p 是当前局面预测的每个位置的最优落子概率，c 是正则项系数，θ 模型参数[12]。

4.4 本章总结

本章从本文使用的价值策略网络的构造出发，依次讲解了网络输入和输出的数据结构与为什么使用这样的输入数据，以及训练过程中在输入数据结构上遇到的问题进行了详细的分析并且提供了相应的解决办法，然后就网络结构部分进行进一步的展开讲解，完整的讲解了本文所用到的网络结构并且分析了其中的设计原理，最后根据本文中使用的网络的结构的需要输出数据相应的进行分析的到为什么使用该损失函数。

第五章　训练管道的搭建与项目部署

5.1 神经网络训练流程

在本文的之前章节中已经详细的讲解过程了蒙特卡洛搜索树与价值策略网络的构建细节，这一部分将详细的讲解训练管道的搭建与项目部署的相关内容。其中的训练管道的搭建是连接蒙特卡洛搜索树与价值策略网络的中间模块，整训练过程是首先由自对弈模块，使用蒙特卡洛搜索方法预测落子每一步对进行自对弈数据的搜集，最终得到最终的胜负信息。然后我们之前搜集到的数据作为神经网络的输入数据与实际的理论的输出数据传输给神经网络部分进行训练学习，通过反向传播更新了模型参数后，再次利用这个新的生成模型进行下一轮的数据生成，以此循环往复得到最终的模型。在训练的过程中每经历一个固定轮次的模型参数更新后我们就进行一个新老模型直接的自对弈评估，当新模型对战旧模型的胜率到达一定阀值后，我们就讲新模型替代旧模型，成为当前的最优模型，大体的训练流程如下图所示[12]：

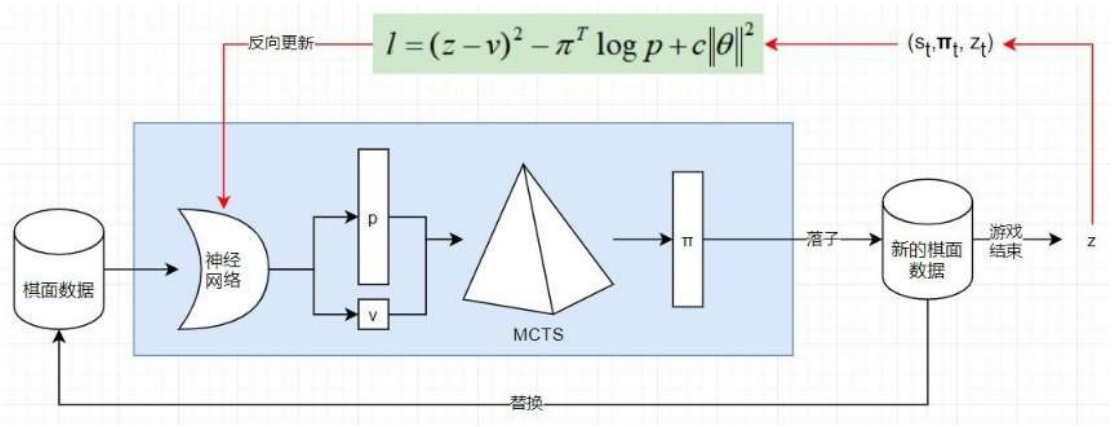

图 5.1 模型训练流程图

在模型训练过程中我们用于数据生成的模型并非是最优模型，而是最新的模型，虽然在直觉上我们认为最优的模型生成的训练数据更好。这样根据利于模型的收敛，然而在我们实际训练过程中其实发现使用最新的模型和最优模型在最后的收敛效果上并没有太大差别，但是使用最新的模型在训练上模型收敛速度上要快与使用最优模型。在原理上，本人猜测应该是使用最新的模型也是一种探索策略[9]，通常来说最新模型的性能应该是不会比最优模型差多少的。如果使用最优模型，我们直接学习到的新参数，将在下一次的最优模型评估中才能得到更新，也就是在这期间其实模型的生成数据并没有得益与我们的训练更新的参数，数据生成的多样性不如使用最新模型。

5.2 数据生成与增强

5.2.1 数据生成

输入数据的生成较为简单就是通过搜集对弈过程中黑白棋子形成的落子棋面数据作为输入，这个详细结构再第四章中做过详细的介绍，输出的实际值 v 就是该对局最终的实际胜负结果，胜为[1],负为[-1]也可以较为容易的得到结果，关键的困难点就在于实际每个局面的落子策略输出 π 上，接下来我将详细的讲解在本文中落子策略 π 是如何得到的。

（1） 我们从当前的局面出发建立上文所提到过的改进后的蒙特卡洛搜索树，并在其上进行 1000 次模拟。

（2） 我们使用蒙得卡落搜索树根结点下一结点的访问次数计算 π，落子完成后，搜索树可以被重用，被选中的边的子结点又变成根结点，上面的统计值继续使用，未被选中的分支会被舍弃。具体公式如下：

$$\pi(a \mid s_0) = N(s_0, a)^{1/\tau} / \sum_b N(s_0, b)^{1/\tau}$$

其中 τ 是一个温度参数，控制探索的程度，τ 越大，探索的能力越强。

（3） 保证落子的多样性，在对局的前 18 步，温度 τ =1，在之后的棋局局面中 τ 设置为无穷小，并且我们在先验概率中添加狄利克雷噪声 $P(\vec{s}, \vec{a}) = (1-\epsilon)p_{\vec{a}} + \epsilon\eta_{\vec{a}}$，其中 $\eta \sim \mathrm{Dir}(0.06)$ 且 $\epsilon = 0.25$，这个噪声保证所有的落子可能都会被尝试[12]。

（4） 根据第 3 步中得到的添加狄利克雷噪声的 π 中各点的概率，依照概率现在其中的一个落子位置作为我们当前的实际落子位置，然后返回（1），直到最终游戏结束。

在上述的数据生成过程中我们其实注意到了在第二步中并没有使用结点的 Q 值去计算 π，其实这里是为了使得模拟过程中对于异常值更加问题，我们使用的是蒙特卡洛搜索树中根节点下一结点中访问次数进行计算，因为如果一个结点被探索的次数很多，说明该结点下的子结点一定含有很多“高招”。

5.2.2 数据增强

数据增强让有限的数据产生更多的数据，增加训练样本的数量以及多样性提升模型鲁棒性。神经网络需要大量的参数，许许多多的神经网路的参数都是数以百万计，而使得这些参数可以正确工作则需要大量的数据进行训练，但在很多实际的项目中，我们难以找到充足的数据来完成任务。随机改变训练样本可以降低

模型对某些属性的依赖，从而提高模型的泛化能力[13]。

在本文中其实在数据的生成部分通过添加狄利克雷噪声的方式就是数据增强的一种方式，然而这对于我们训练的五子棋 AI 模型来说还不够。强化学习本身对于样本数据的利用率就比较低，我们因此就需要跟多的数据对模型进行喂养，但是整个训练框架中数据的生成部分非常耗时，尽可能的减少数据的生成量就是加快整个训练过程的关键。对于五子棋游戏而言，整个棋面数据通过对称选择变换是不影响最终游戏结果的，所以我们具有一个非常简单的数据增强思路，我们可以对于输入数据 s 和策略输出数据 π 进行对称变换与旋转变换。这样一次模拟对局就可以得到 8 份训练数据，这样极大的增加了可训练数据的量。

5.3 模型训练

在整个模型训练过中我们训练数据部分保存在一个缓冲队列当中，缓冲队列中最多保存近 8 万对局训练数据，后续新生成的数据将替换最初生成的数据，依次往复。本文中训练时采用的优化器是 Adam[14]，训练时通过从缓冲队列中随机的选择 15*512 条训练数据，我们采用小批量梯度下降的方式进行训练[15]，批量大小为 512，并计算本次训练前后模型的落子策略输出和价值输出与实际值间的 kl 散度值以及解释方差作为模型训练效果的零时参考，并且整个训练过程中我们的学习率采用退火算法[14]，学习率退火比例如下表所示：

表 5.1 学习率退火比例

自对弈生成棋谱步数	强化学习率
训练初始化（前 100 局）	0.01
0~20	0.01
20~60	0.001
＞ 60	0.0001

5.4 最优模型评估器

在强化学习的模型训练过程中，模型的性能并非随着训练时间线性增长，所以为了得到更好模型参数，在训练过程中我们需要保存当前训练过程中的最优模型。在本文中是在每 250 次自对弈数据生成、训练后进行一次模型的评估，模型评估由 10 次新旧模型的自对弈对局组成，考虑到无禁手五子棋先行有较大的优势，所以这 10 次对弈过程中，新旧模型分别 5 次先行，并且我们在最优模型评

估时的自对弈时不在使用探索策略，整个对局过程中温度参数 τ 始终趋于无穷小，也就是每一步我们都尽可能的落子于模型认为的最优落子点，10 局对弈结束后统计这个对局当中新模型的胜利，当达超过胜率阀值 0.55 时，也即是获胜 6 局及以上时我们将旧最优模型替换为当前最新模型，得到新的最优模型[12]。

5.5 GPU 云服务器平台部署

AlphaGo Zero 在训练数据的生成过程中使用了近 5000 个 TPU,对于本文中训练的五子棋 AI 而言，虽然五子棋游戏的问题规模的复杂度上要远低于围棋加之我们上文所提到的优化方式的运用，在本人的配置的 GTX 1060 的个人电脑上训练 5 天后也能得到一个不错的模型，但是如果需要得到一个足够优秀的模型，普通个人电脑上训练的时间成本还是太大了，所以本文最终的模型训练时使用的是 GPU 云服务器。

出于效率考虑对于数据生成的蒙特考虑搜索树部分我们采用的是 C++编写，当迁移至服务器时，我们需要将其转化为 Liunx 下 Python 可以调用的库函数。这里我们使用的 Pybind11 这个轻量级的库方便的将 C++代码封装为 Python 可以调用的接口，这一部分的细节将在下一章处进行展开讲解，最终使用的是一台搭载 GTX 3090 显卡的服务器进行训练，经过大约一天训练后，在没有任何人为经验的输入下，AI 已经学会了诸如梅花、斜三阵、八卦阵等等五子棋的定式。最后执黑高胜率，普通玩家几乎无法战胜，执白,在非黑棋必胜开局下，以极高胜利战胜普通玩家，由于五子棋在无禁手规则下理论上先手必胜，所有理论上只要经过足够的时间训练，可以做到职黑以 100%胜率战胜任何玩家，但是由于硬件限制这里并没有训练过长的训练。

```
Iter: 194 Current Time: Sun Feb 20 07:33:19 2022
itre194 id:1 time consumption:95.99511790275574  step number:13
itre194 id:2 time consumption:116.86012697219849  step number:18
itre194 id:3 time consumption:111.81832575798035  step number:15
itre194 id:4 time consumption:118.60226774215698  step number:16
itre194 id:5 time consumption:125.53042531013489  step number:18
itre194 id:6 time consumption:123.55110311508179  step number:19
itre194 id:7 time consumption:128.5763294696808  step number:21
itre194 id:8 time consumption:132.70683693885803  step number:23
itre194 id:9 time consumption:143.4509916305542  step number:29
itre194 id:10 time consumption:134.80047225952148  step number:31
avg_step:20.3 lr:0.01
```

图 5.2 训练过程中棋谱数据生成

```
iter:28 epoch:1,loss:1.2321627140045166 policy_loss:0.8657213449478149 value_loss:0.36644142866134644
iter:28 epoch:2,loss:1.2699470520019531 policy_loss:0.9152098894119263 value_loss:0.35473719239234924
iter:28 epoch:3,loss:1.0691221952438354 policy_loss:0.7477402091026306 value_loss:0.3213820159435272
iter:28 epoch:4,loss:1.1866803169250488 policy_loss:0.8263208866119385 value_loss:0.36035943031311035
iter:28 epoch:5,loss:1.2082264423370361 policy_loss:0.8552279472351074 value_loss:0.3529985547065735
iter:28 epoch:6,loss:1.184553623199463 policy_loss:0.8498311042785645 value_loss:0.3347225785255432
iter:28 epoch:7,loss:1.1216181516647339 policy_loss:0.7837230563163757 value_loss:0.33789509534835815
iter:28 epoch:8,loss:1.1475034952163696 policy_loss:0.7931182980537415 value_loss:0.3543851673603058
iter:28 epoch:9,loss:1.1953990459442139 policy_loss:0.8713521957397461 value_loss:0.3240468502044678
iter:28 epoch:10,loss:1.2321821451187134 policy_loss:0.8713065385818481 value_loss:0.3608756363391876
iter:28 epoch:11,loss:1.290703535079956 policy_loss:0.9067487716674805 value_loss:0.3839547336101532
iter:28 epoch:12,loss:1.3949096202850342 policy_loss:1.0135043859481812 value_loss:0.3814052939414978
iter:28 epoch:13,loss:1.2895214557647705 policy_loss:0.937630295753479 value_loss:0.3518912196159363
iter:28 epoch:14,loss:1.3099350929260254 policy_loss:0.9648768901824951 value_loss:0.3450581431388855
iter:28 epoch:15,loss:1.3598201274871826 policy_loss:1.0026190280914307 value_loss:0.35720109939575195
kl:0.4384434904662555 evar_old:0.5927075262230089 evar_new:0.6501349397980676
```

图 5.3 神经网络训练

```
itre125 time consumption:66.04737162590027  step number:22 result: Loss
cotest 1 result: Win  step num:18 time consumption: 82.47081589698792
cotest 2 result: Win  step num:20 time consumption: 83.67930555343628
contest 3 result: Loss  step num:17 time consumption: 85.37532472610474
contest 4 result: Loss  step num:17 time consumption: 88.58889842033386
cotest 5 result: Win  step num:18 time consumption: 108.5327799320221
cotest 6 result: Win  step num:23 time consumption: 118.90201091766357
cotest 7 result: Win  step num:23 time consumption: 124.3768539428711
cotest 8 result: Win  step num:23 time consumption: 105.52527952194214
cotest 9 result: Win  step num:23 time consumption: 115.95338582992554
cotest 10 result: Win  step num:23 time consumption: 106.67594408988953
******************************contest end******************************
our current model total Win:8 Draw:0 Loss:2
current update threshold:0.55 current model win rate:0.80
we get new best model
```

图 5.4　最优模型评估器

5.6 本章总结

本章大体的介绍了整个神经网络的训练流程，承上启下的连接了前文中所建设的蒙特卡洛搜索树与价值策略网络，详细的讲解了网络训练过程中的一些细节点，包括训练数据的生成与增强、模型训练、最优模型评估器的设计细节问题，最后讲解了本文中提到的五子棋 AI 是如何在 GPU 云服务器上的到训练的，至此为止文本中的主题部分已经全部介绍完毕。

第六章　项目实现与性能评估

6.1 项目结构

本文是基于 PyTroch 和 LibTorch 框架进行开发实现的五子棋落子模型的训练，开发语言上使用的是 C++和 Python 进行混合开发，利用面向对象思想进行系统的设计与实现。C++语言几乎零开销抽象、极高的信念及运行效率在本文中核心性能模块的开发中起到了不可替代的作用，Python 语言具有可移植性高、开发效率高等特点，另外其具有非常丰富的库函数，极大的加快了本文整个算法框架的编写过程。PyTorch 和 LibTorch 是目前较为成熟的深度学习框架，由于它简单易用高效的特点，在目前的学术界得到了广泛的使用。本文中实验的相关环境配置如下表所示：

表 6.1 实验硬件环境

硬件环境	配置
内存	16GB
硬盘	1TB
CPU	Intel(R) Core(TM) i7-8750H
GPU	GeForce RTX 1060
服务器 GPU	TITAN XP
显卡驱动	NVIDIA_SMI 472.12
操作系统	Windows 10
编程语言	C++ 、Python
开发工具	Virtual Studio 2019、PyCharm 2022
深度学习框架	Pytorch 1.10、Libtorch 1.10

实验代码共包含 10 个 C++文件和 5 个 Python 文件，文件结构如图 6.1 和 6.2 所示。其中 C++代码部分，APV_MCTS 文件为并行蒙特卡洛搜索树代码、ThreadPool 文件为线程池代码、NeuralNet 文件为神经网络预测接口代码、GomokuBoard 文件为棋盘类代码、CppLibs 文件为 Python 运行库生成代码、human 文件为人机对弈代码、test 文件为自对弈代码。Python 代码部分 checkpoint 文件为生成的训练数据、best_model、current_model 为当前模型与最优模型，config 文件为训练管道参数文件、policy_value_net 文件为神经网络代码部分、train 文件为训练管道部分代码、Scripts_function 文件为模型性能评估脚本部分代码、test 文件为模型测试部分代码。

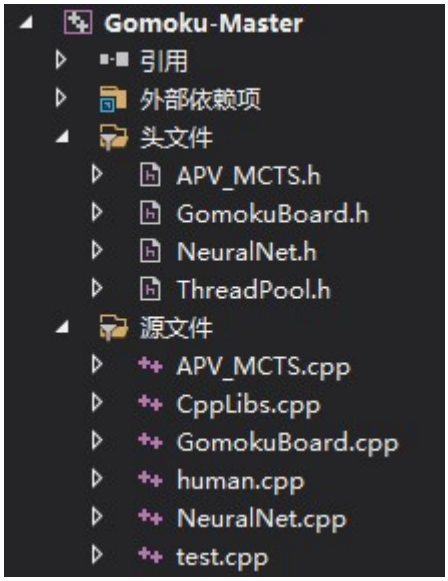

图 6.1 项目 C++文件结构

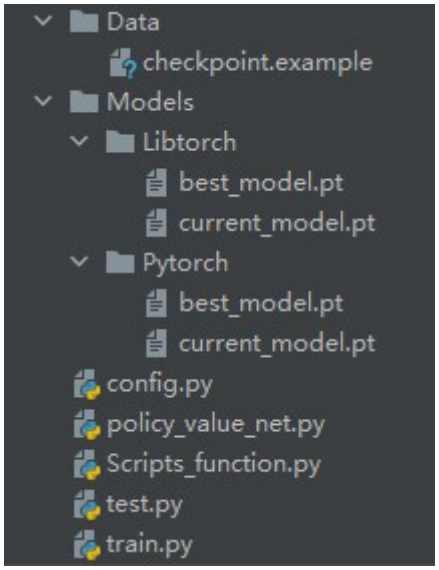

图 6.2 项目 Python 文件结构

6.2 实现高性能并行 MCTS 库

在上文中提到过整个模型的训练过程中主要时间消耗就在于训练数据的生成上，训练数据的生成快慢取决于蒙特卡洛搜索树模拟过程中每一步的时间消耗，所以最终提高整个模型的收敛速度的关键就在于实现一个高性能并行的 MCTS 库。由于 Python 语言中全局解释锁（GIL）的存在，所以 Python 中的多线程是真正意义上操作系统级的多线程。这也就意味这同一时间点只能由一个 Python 线程运行，这样如果使用 Python 实现蒙得卡洛搜索树就没有办法完全的利用到多核处理器的优势，并且 Python 语言本身在运行效率上往往就要低于其他的静态语言，最后综合开发效率与运行效率考虑，我们最终使用了 C++进行这一部分程序的编写。下面将对于这一部分的细节部分进行简单阐述。

蒙特卡洛搜索树结点的定义部分，每个结点中都包含有父节点指针 parent、子节点列表 children、叶子结点标志 is_leaf、孩子结点锁 children_lock、数据锁 data_lock、拓展锁 expand_lock、前置访问量 n_unobserved、访问量 n_visited，结点 Q 值 q_sa，结点选择概率 p_sa、虚拟损失量 virtual_loss,其中前置访问量、访问量、虚拟损失均为原子变量，每个结点都可以执行四种基本操作，选择、拓展、反向传播以及返回结点价值。在上述定义过程中其实可以方向在蒙特卡洛搜索树模拟过程中需要经常改变的变量我们都将其定义为了原子变量，这是为了避免在并行过程中反复加锁解锁带来的性能消耗，children_lock 的使用主要是在结点选择过程中保证整个选择过程中孩子结点中的数据不会被更新，导致选择时不同结点的不公平。这里锁的使用避免了孩子结点值的复制耗时，数据锁的使用原

理类似，只是主要是对于避免 Q 值更新时线程的切换导致其计算时新旧数据的不统一，拓展锁作用主要是避免同一结点被同时拓展导致错误，因为在本文中实现的蒙特卡洛搜索树只是在叶子结点处才可以进行拓展，但是拓展过程是比较耗时的，在拓展过程中，此结点不再是叶子结点，所以在多线程下我们需要保证一个结点只能被拓展一次。

蒙特卡洛搜索树的模拟过程中我们实际上并没有使用原始随机走子策略去的到最终的结果，而是在这个叶子结点处通过上文中的价值策略函数进行的一次预测，但是在本文中一次落子位置的计算需要的模拟量就是 1000 次，也就是需要 1000 次的价值策略函数的预测，神经网络的模拟预测其实就是进行的多次的矩阵乘法，矩阵乘法本身是可以并行运算的，但是由于我们实际并行过程中这些预测请求的异步性，导致对于 CUDA 加速的神经网络来说，实际上我们的程序还没有真正的并行化。这是因为 LibTorch 的预测执行实际上受限制于 Default CUDA Steam，默认是串行的，这也会导致多线程被阻塞。在本文中我们最终使用的办法是.合并预测请求。这里我们使用的方法是用缓冲队列合并多个预测，一次性推送到 GPU，这样就防止了 GPU 工作流的争用导致线程阻塞。

为了使得我们使用 C++编写的高性能并行 MCTS 可以在我们的 Python 中得以简单的调用，这里我们需要去为 Python 封装一下调用接口。这里我们使用的 pybind11, pybind11 是一个轻量级的仅头文件库，它在 Python 中公开 C++ 类型，反之亦然，主要用于创建现有 C++ 代码的 Python 绑定。它的目标和语法类似于 David Abrahams 的优秀 Boost.Python 库：通过使用编译时自省来推断类型信息，从而最大限度地减少传统扩展模块中的样板代码。编写好接口程序后我们可以编程生成一个 Python 可调用的 pyd 文件或者 so 文件，之后在 Python 中就可以像调用普通的 Python 库函数一样的调用我们上述实现的 MCTS 库，这样我们就既保留 C++高性能并行的优势，又结合了 Python 简单高效的开发优势。

6.3 自动化性能评估脚本

验证本文中采用的蒙特卡洛搜索树和神经网络结合的博弈系统的棋力强度，我们将其与传统的基于 α - β 裁枝算法、纯蒙特卡洛搜索树设计博弈系统对弈，并且微调不同参数比较其胜率[16]。对于每组不同的参数，本文模型与其他传统算法依次先手后手 25 场自对弈实验，即每组参数共进行 50 次实验。在整个实验过程中我们本文模型使用的单步模拟次数均为 1500 次，具体情况如下表所示。

表 6.1 本文模型与基于α－β裁枝算法的对弈情况

模拟深度（层）	胜（局）	败（局）	平（局）
2	50	0	0
4	50	0	0
6	50	0	0
6+8(VCT)	43	6	1

表 6.2 本文模型与基于纯蒙特卡洛搜索树算法的对弈情况

模拟次数（次）	胜（局）	败（局）	平（局）
2000	50	0	0
3000	50	0	0
5000	50	0	0
10000	37	13	0

同基于α-β裁枝算法的对弈过程中，仅仅当其模拟深度达到6层并使用8层“算杀”时才有机会战胜本文训练的模型。详细分析时发现，其获胜的6盘均为其先行将对局拖入中盘后获胜，这里分析主要原因还是在于模型训练时间不足导致的。模型对于中盘的数据学习量太少，导致进入中盘后棋力下降，这个通过更多时间的训练应该是可以解决的。在同样是纯基于蒙特卡洛搜索树的对局中纯蒙特卡洛搜索树算法使用的并非是随机走子策略，而是采用我们在α-β裁枝算法中使用的基于专家系统的评估函数进行当前局面的评价。同样的，只有当基于纯蒙特卡洛搜索树算法的单步模拟次数达到 10000 次时，才战胜了本文训练的模型，单步模拟 10000 次的基于纯蒙特卡洛搜索树的算法在单步耗时上近乎等同于6层模拟+8层算杀的α-β裁枝算法，但是我们发现最终胜场数是要比起高了一倍多，所以我们可以知道这种基于蒙特卡洛搜索树的算法在性能上是要强于前者的，而本文所使用的模型单步模拟次数要远低于这种纯粹的蒙特卡洛搜索树，最后的胜率却要高不少，所以我们可以知道蒙特卡洛搜索树+神经网络策略通常意义上是要优于采用简单编写的专家系统的评估函数+蒙特卡洛搜索树的。

特别的在本章的我们利用 Python 控制脚本将最终设计实现的五子棋博弈模型与目前五子棋领域内最强引擎—弈心进行对弈分析评估最终的结果，弈心是第13届、14届、15届、16届、17届、18届、19届 Gomocup 冠军。2017年，弈心成为首个在公开比赛中战胜人类顶尖棋手的人工智能程序。其设计的局面评估函数采用是专家系统，而并非本文中所使用的神经函数。故在无禁手规则下，其

先行胜率近乎 100%，在训练硬件有限的条件下，我们最终的模型并没有达到最优。我们采用单步模拟 3500 次的本文模型战采用职业四段棋力的弈心时，即使是先行仍然是败多胜少，但是即使如此，通过短时间的训练最终训练的模型有机会战胜通过精心设计专家系统的五子棋的国际冠军程序仍然说明了本文中方法的巨大潜力。在足够的硬件与时间支持下，完全战胜弈心并非是不可能的事。下图是本文模型对弈弈心时的训练情况与部分棋谱。

```
Iter: 19 Type: Yixin vs Gomoku AI Current Time: Wed May  4 00:35:07 2022
iter:19 epoch:1,loss:1.0513683557510376 policy_loss:0.7439512014389038 value_loss:0.3074171543121338
iter:19 epoch:2,loss:1.1152632236480713 policy_loss:0.7563371658325195 value_loss:0.35892605781555176
iter:19 epoch:3,loss:1.0286749601364136 policy_loss:0.6867560148239136 value_loss:0.3419189453125
iter:19 epoch:4,loss:1.1048448085784912 policy_loss:0.811957597732544 value_loss:0.29288721084594727
iter:19 epoch:5,loss:1.0211361646652222 policy_loss:0.7086600661277771 value_loss:0.31247609853744507
iter:19 epoch:6,loss:1.206592321395874 policy_loss:0.8491332530975342 value_loss:0.35745906829833984
iter:19 epoch:7,loss:1.0088306665420532 policy_loss:0.6616334319114685 value_loss:0.3471972346305847
iter:19 epoch:8,loss:1.1434861421585083 policy_loss:0.8319312930107117 value_loss:0.31155481934547424
iter:19 epoch:9,loss:1.1400007009506226 policy_loss:0.7935263514518738 value_loss:0.3464743196964264
iter:19 epoch:10,loss:1.223632574081421 policy_loss:0.8536872267723083 value_loss:0.36994534730911255
iter:19 epoch:11,loss:0.946407675743103 policy_loss:0.6472097039222717 value_loss:0.2991979718208313
iter:19 epoch:12,loss:1.0711015462875366 policy_loss:0.7374212741851807 value_loss:0.33368027210235596
iter:19 epoch:13,loss:1.2484170198440552 policy_loss:0.9025249481201172 value_loss:0.345892071723938
iter:19 epoch:14,loss:1.0994055271148682 policy_loss:0.7618708610534668 value_loss:0.33753466606140137
iter:19 epoch:15,loss:1.0743820667266846 policy_loss:0.7469169497489929 value_loss:0.3274651765823364
kl:0.522673753426757 evar_old:0.642921972264088 evar_new:0.7107737273114154
itre19 time consumption:92.1235032081604  step number:27 result: Win
```

图 6.3 模型与弈心对弈训练

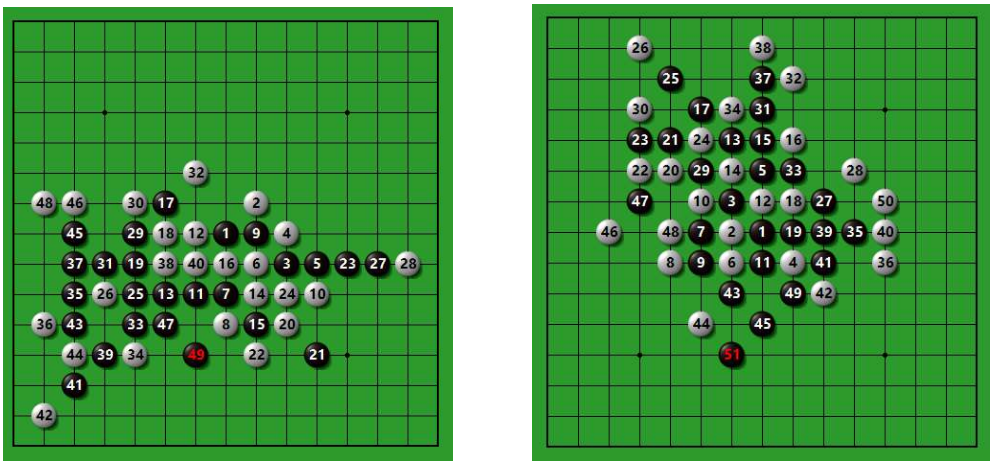

图 6.4 模型与弈心的部分对弈棋谱（模型执黑）

6.4 本章小结

本章节首先介绍了本文的实验环境、代码文件结构、模型的评价方法以及实验设置的参数，并对影响本文五子棋算法模型性能的重要参数进行了优化。基于 α - β 剪枝算法以及纯蒙特卡洛搜索树五子棋算法与本文基于蒙特卡洛树与深度神经网络的五子棋算法模型通过比赛的方式进行对比分析，最后对比目前基于专

家系统设计的最强五子棋引擎弈心，分析了模型的不足点以及原因。

第七章　总结与展望

7.1 研究工作总结

本文在总结前人在棋类博弈问题上的算法经验，以五子棋这一具有其他棋类典型特征的棋类作为研究对象，重点围绕棋局的局面评估与蒙特卡洛搜索树算法展开了相关的研究工作，提出了基于策略价值网络的局面评估模型与基于蒙特卡洛搜索树的落子决策模型并且基于五子棋博弈问题创造性的提出了 p-对立策略选择算法，构建了原型系统，最后编写并训练了整个算法模型，验证了算法的有效性。主要的研究工作总结如下：

(1) 对棋类博弈系统的发展历程、计算机博弈搜索算法和局面评估算法，以及本文主要研究对象五子棋计算机博弈的相关概念、研究相关的基本概念、基础理论、方法进行了重点介绍。

(2) 开展了决策落子算法优化研究。从传统的博弈树搜索算法出发，分析了传统算法的原理、差异，引出并行蒙特卡洛搜索树并提出了针对于五子棋的改进方法，结合深度学习，提出了使用策略价值网络替代传统的专家系统的改进措施，并采用强化学习的训练方式使得不需要任何人类的经验，通过模型自行对弈、学习的方式进行训练。

(3) 开展实现了本文中提出的原型算法模型，利用该训练模型，围绕系统自我训练能力、自我学习能力和系统棋力进行.了多组对比实验。通过实验验证，证明了论文所提出的改进模型和优化算法的有效性，系统具有较好自我训练、自我学习能力,随着自我训练过程的持续进行其棋力不断提升，能够开展人机对战。

7.2 研究工作展望

本文的核心仍然在蒙特卡洛搜索树，本质上仍然和传统的方法一致。传统的蒙特卡洛仿真搜索树算法的关键之所于搜索空间的缩小,原理上是通过缩小搜索广度和降低搜索深度。而本文的亮点,就是通过使用深度网络的方式,来有效地减小蒙特卡洛仿真搜索空间大小。也可以从二种方面考虑,其中策略网络在蒙特卡洛树上扩展是缩小搜索的广度,而价值值网络在蒙特卡洛树上棋盘局面评价则是降低了搜索的深度。再进一步,这二种优化方向的重点就是使用深度网络来实现函数模型拟合。策略网络用于模型拟合策略函数,而价值网络则用来模型拟合价格函数。按照传统的方式,可以通过对单一的输入特征的线性组合实现回归模型拟合,而深度神经网络则能够通过更多的元数据、更有效的方式实现函数模型拟合。但实际上,在这里的深度学习已经解决了一般的树搜索过程的启发式评分方

法问题。当然,这篇报告的亮点还包括了深度学习与强化知识的创新方法,不过它都是为蒙特卡洛搜索树的效率服务的。而且总体而言,还是摆脱不了蒙特卡洛搜索树。一些可能的改进方向，例如,方法是不断调整蒙特卡洛仿真的树,采用不同的技术在蒙特卡洛四个方面继续精细化处理,包括尝试采用其他的神经网络结构,采用优化强化的方法,试图导入其他的神经网路模块或改变已有的神经网路模块等。还有一个学者研究了不依赖于蒙特卡洛仿真搜索树的新方法,其问题在于蒙特卡洛仿真搜索树并不是完美,而且其本质上也就是可以暴力枚举的,因此只有当更智能的人在进行枚举搜索时,对某些更复杂棋局,才无法合理的减少查找距离,从而导致求解工作量很大,甚至发生了误差,文献论文[8]提到了使用深度交替神经网络 DANN 和长期评估 LTE 来代替蒙特卡洛搜索。

致　谢

历经三个月的沉淀也终于将这篇论文写完，对于这篇论文，首先要感谢我的导师吴海涛老师，给予了我很大的指导和支持，对论文的不足之处一一指点。同时感谢 4 年来各位专业老师的教导，开阔了我的视野，让我学习到了很多知识。

四年一瞬，聚散有时，感谢我的大学舍友们，四年的朝夕相处对我友善和包容，感谢四年同窗的同学们，能与我一起度过四年的学习生活，感谢我所遇到的挚友们，在孤单寂寞时与我一起度过。四年时光，很幸运承蒙你们给予的善意，陪我度过整个青春。

只言片语，落笔至此，思绪繁绕。再次衷心感谢过去时光中所有给我人生带来感动和启迪的师长、同学、朋友和亲人们，我也将继续前进，奔赴人生的下一站，希望自己对得起这一路的真诚，希望自己对得起所学和所遇。

最后，引用一句莎士比亚的名言共勉。

凡是过往，皆为序章。

参考文献

[1] 李昊.五子棋人机博弈算法优化研究与实现.2020.大连海事大学,MA thesis.

[2] 蒋帆.基于蒙特卡洛树搜索的德州扑克 AI 算法改进方法研究.2020.东南大学,MA thesis.

[3] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. nature, 2015, 518(7540): 529-533.

[4] LeCun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE, 1998, 86(11): 2278-2324.

[5] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C].Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.

[6] Shannon C E. Programming a Computer for Playing Chess[M].Computer Chess Compendium. Springer, New York, NY, 1988: 2-13.

[7] Guo X, Singh S, Lee H, et al. Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning[J]. Advances in neural information processing systems, 2014, 27.

[8] Wang J, Wang W, Wang R, et al. Beyond Monte Carlo tree search: Playing go with deep alternative neural network and long-term evaluation[C].Proceedings of the AAAI Conference on Artificial Intelligence. 2017, 31(1).

[9] David Silver et al. “Mastering the Game of Go with Deep Neural Networks and Tree Search” (2022).

[10] Browne, Cameron B., et al. "A survey of monte carlo tree search methods." IEEE Transactions on Computational Intelligence and AI in games 4.1 (2012): 1-43.

[11] Liu, Anji, et al. "Watch the unobserved: A simple approach to parallelizing monte carlo tree search." arXiv preprint arXiv:1810.11755 (2018).

[12] Silver, David, et al. "Mastering the game of go without human knowledge." nature 550.7676 (2017): 354-359.

[13] Buslaev A, Iglovikov V I, Khvedchenya E, et al. Albumentations: fast and flexible image augmentations[J]. Information, 2020, 11(2): 125.

[14] Kingma D P, Ba J. Adam: A method for stochastic optimization[J]. arXiv preprint arXiv:1412.6980, 2014.

[15] Ruder S. An overview of gradient descent optimization algorithms[J]. arXiv preprint arXiv:1609.04747, 2016.

[16] 王钦.基于深度强化学习的五子棋算法研究.2019.重庆大学,MA thesis.