# 棋牌类智能博弈技术调研及五子棋上的实现

赵云龙 202018014628044、李曦云 202018020428004

December 2020

## 1 引言

在本次课程设计中，我们对棋盘类智能博弈技术进行了调研，并选取了经典的五子棋问题进行了实现。我们以极大极小搜索起步，调研了以 Alpha-Beta 剪枝为代表的多种剪枝搜索策略、基于蒙特卡洛树的搜索、基于模型的树搜索，然后选取 baseline 模型加以训练，多个角度观察存在的问题和不足，然后结合前期的调研给出改进方案并实验，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（alternating Markov games）[3]。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 (reward function) 描述了玩家 (player)$i$ 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 $r^1(s) = -r^2(s) = r(s)$。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 $z_t = \pm r\left(s_T\right)$。策略 $p(a \mid s)$ 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

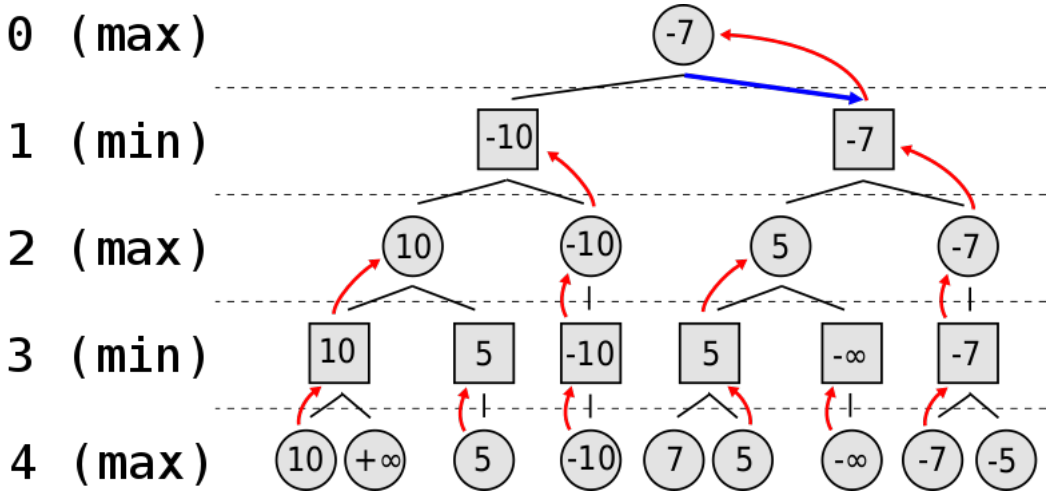


图 1: 极大极小值搜索

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 极大极小搜索算法描述

极大极小（MinMax）值搜索方法是一种深度优先的博弈树搜索算法。极大极小值搜索的基本思想就是选择最好，避免最差。极大极小搜索的算法流程为：构建一棵博弈树，树的每一层为 MAX 层和 MIN 层的交替，且根节点为 MAX 层。本方下棋的时候是 MAX 层，在 MAX 层的节点要选取得分最高的节点，对手下棋的时候是 MIN 层，在 MIN 层的节点要选取得分最低的节点。MAX 层节点的得分是其子节点的得分的最大值，MIN 层节点的得分是其子节点得分的最小值。图 1 是一个极大极小搜索树的例子。负值最大化的方法将极大极小值搜索在对方层的得分取负值，统一对得分取最大值即可。而每一层进行递归调用的时候也就是切换为对方层的时候，因此只需要对得分取负值即可。

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** 极大极小搜索

```
1: function MinMax(node, depth)
2:     if depth==0 then
3:         return evaluation score of node
4:     end if
5:     生成 Node 的子节点集合 V
6:     score ← −∞
7:     for childnode in V do
8:         score = max(score, - MinMax(childnode, depth-1))
9:     end for
```

10: **return** score
11: **end function**

## 2.2 alpha-beta 搜索算法描述

极大极小值搜索如果对所有子节点都进行搜索，搜索空间会随着搜索深度的增加成指数型增长，因此需要对搜索树进行剪枝。alpha-beta 搜索是一种极大极小搜索进行剪枝的方法，图 2 为 alpha-beta 搜索的一个示意图。

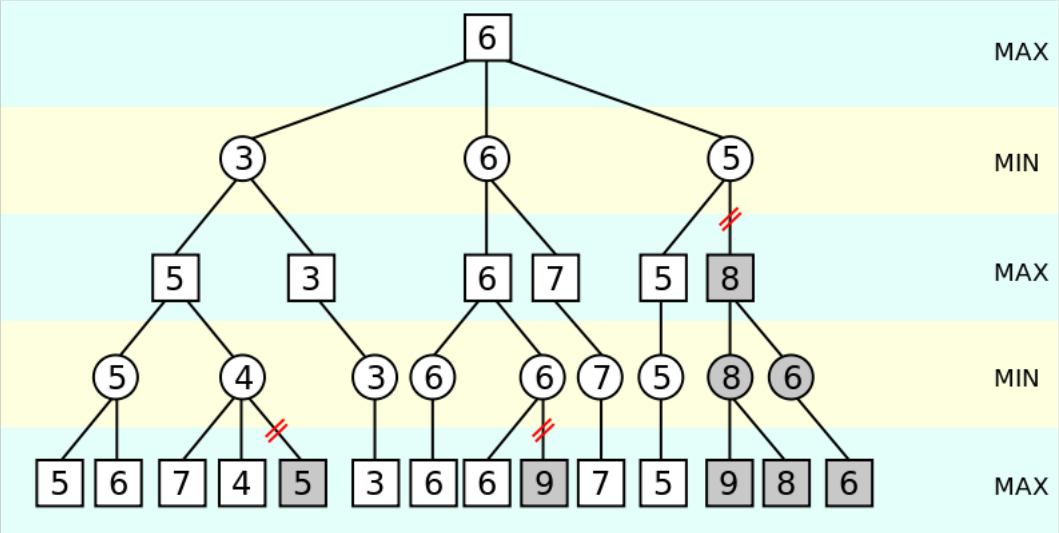


图 2: alpha-beta 搜索

对于每个搜索节点，alpha 代表该节点当前所取得的最好的结果，beta 值代表父节点当前的 alpha 值，如果 alpha 已经大于 beta 值，说明当前遍历的最好的结果已经比父节点最好的情况要好了，父节点可以决定选择该节点了，后续分支不需要遍历了。负值搜索形式的 alpha-beta 搜索伪代码如下所示：

**Algorithm 2** alpha-beta 搜索

o

1: **function** MinMax(node, depth, alpha, beta)
2: **if** depth==0 **then**
3: **return** evaluation score of node
4: **end if**
5: 生成 Node 的子节点集合 V
6: $score \leftarrow -\infty$
7: **for** childnode in V **do**
8: score = max(score, - MinMax(childnode, depth-1,-beta,-alpha))
9: **if** $score > alpha$ **then**

```
            alpha ← score
            if alpha ≥ beta then
                break
            end if
        end if
    end for
    return score
end function
```

## 2.3 评价函数及实现

极大极小算法的评价函数的设计与代码的实现部分，参考了 csdn 博客 `https://blog.csdn.net/marble_xu/article/details/90647361`。极大极小搜索的价值奖励是在叶子节点处进行计算，再反传回根节点进行选择。我们需要人工设计评价函数对叶子节点的所处的状态进行打分，在本实现中所使用的评价函数所打分的棋面为以下几种：连五，活四，冲四，活三，眠三，活二，眠二。关于这几种棋面的具体接受可以参考 http://game.onegreen.net/wzq/HTML/142336.html。实现中，需要注意的一点是要对评价函数中各种棋面的分值进行恰当的设置，保证更具优势的棋面出现的棋面比相对差的棋面的得分要高。在本实现中，采用的一个合适的得分设置为：连五为 100000 分，活四为 10000 分，眠四为 1000 分，活三为 100 分，眠三为 10 分，活二为 8 分，眠二为 2 分，计算得分时对这些棋面的出现的组合也可以设置不同的分数组合，人工评价函数的效果对极大极小搜索的效果的影响比较大。

此外，为提高速度和性能，在实现中对搜索还做了一定的限制。首先，只考虑对周围有落子的空位进行搜索，并对当前位置进行评分，当搜索层数过高时且超过最大搜索限制时，只对评分高的位置进行搜索。如果为先手下棋，则在“天元”位置（棋盘中心位置）落第一子。

## 2.4 实验结果

极大极小搜索的五子棋程序能够取得不错的成绩，进行人机对战测试，已很难击败机器程序。下图为一次对弈的棋面，执黑棋的是机器程序。

与 Alpha Zero 基线程序（即下一部分的 alpha zero 实现）进行对战，在 $8 \times 8$ 的棋盘上进行五子连珠的机器对弈，和在 $6 \times 6$ 的棋盘上进行四子连珠的机器对弈，无法分出胜负。分别对弈 10 局，均达成平局 10 局。主要原因可能是 $8 \times 8$ 和 $6 \times 6$ 的棋盘都过小了，很容易形成没有胜者的局面。在小棋盘上，无法检验极大极小搜索和 alpha zero 的棋力。

因此，我们选取了另一个大棋盘上与经过充分训练的 Alpha zero 程序实现进行对战，在 $11 \times 11$ 的棋盘上，Alpha zero 为先手的情况下，Alpha zero 获得 10 局全胜。在极大极小搜

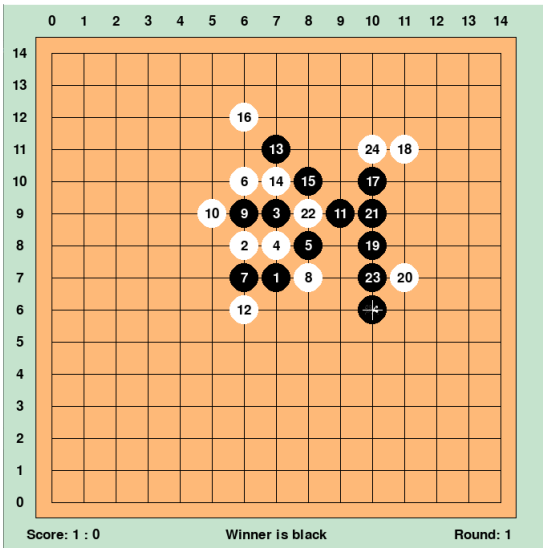


图 3: 极大极小搜索对弈棋面

索为先手的情况下，alpha zero 比极大极小搜索为胜 7 场，平 3 场。在 $15 \times 15$ 的棋盘上，Alpha zero 为先手的情况下，Alpha zero 比极大极小搜索为胜 9 场，输 1 场。在极大极小搜索为先手的情况下，alpha zero 比极大极小搜索为胜 6 场，输 3 场，平 1 场。我们能够看到，尽管 Alpha Zero 没有人工设计的评价函数的指导，通过自博弈学习，就能够取得比极大极小搜索方法更好的性能。

# 3 AlphaGo 系列工作

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，但其不足之处是使用了大量人类专业选手比赛数据，此外还有快速走子网络使用了人工设计的特征，学习过程复杂，用了诸多技巧 [5]。

因此，在 AlphaGo 之后为解决这些问题而提出的 AlphaZero[4] 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。

我们选取的 baseline 方法模型里的基网络为简单的 3 层卷积网络，可能会导致一些不足，具体在实验结果部分加以展示。我们借鉴 Alpha zero 原论文中使用的更复杂更深的网络结构，且为了避免梯度消失，我们使用了残差结构的神经网络结构。另外，考虑棋盘的对称性，我们借鉴 AlphaGo zero 的数据增强，在训练数据增强和深度网络评估时都对一个棋局数据进行八倍增强。与 Alphazero 相同，我们尝试只保存当前不断更新的一个模型，baseline 模型的自我对弈数据由之前所有模型中最好的一个模型产生，不去做额外的模型评估和选择，以及在先验策略中增加噪声来提高模型探索能力。此外我们对这个代码做了 UI 界面，

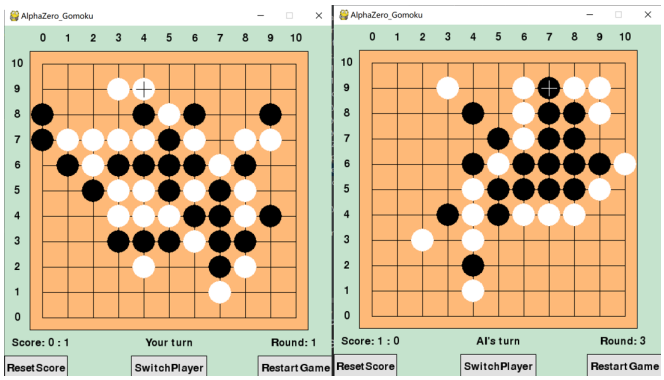


图 4: baseline 模型

支持 AI 和 AI 对弈的界面展示以及 AI 和人的对弈界面展示，支持查看每步棋的轮数这样可以更好的分析棋局。

## 3.1 实验部分

在这一部分我们的 baseline 模型为这个开源实现参见，

`https://github.com/junxiaosong/AlphaZero_Gomoku`。在这个代码里包含几部分组成，五子棋游戏部分棋盘部分、深度强化学习和蒙特卡洛树搜索，我们的实验开展在 11G Tesla K80，由于前期服务器资源限制和我们自身的原因，截止在提交报告前训练轮数还不够多，尤其是在模型收敛效果，我们的实验思路是从 AlphaGo zero、Alphazero 里借鉴 idea，进行消融实验，逐个验证我们的想法的可行性，我们的实验 setting 与 baseline 相似，优化器为 Adam 优化器。

## 3.2 实验结果

我们对实验结果进行分析，观察分析 alphazero 的学习效果。在固定的下法上，我们能够发现其在执先手时，在 $6 \times 6$，$8 \times 8$，$11 \times 11$，$15 \times 15$ 的棋盘上，alpha zero 都学会了在天元，即棋盘中心位置落第一子。另外尽管五子棋较于围棋简单很多，在我们的实验里，同样观察到了类似 AlphaGo 下围棋的现象，强化学习学到的五子棋下法一定程度上已经脱离了人类下棋的方法，下图为人和 baseline 模型对弈的情况，七次对弈人只赢了一局，可以看出 baseline 模型已经具有了不错的五子棋能力，如下图，搜索树的每轮搜索路径数 *playout* 为 400，其中白棋为 baseline 模型，黑棋为人。

我们观察 baseline 模型发现 baseline 模型不会堵边，搜索树的每轮搜索结点数 *playout* 为 400，体现在下图，下图为两个 baseline 模型对弈的情况，对于左边界 (0,5)，白棋在第 20 步没有堵边，而黑棋在第 21 步同样没有选择落子在左边界，其实已经四子相连，直到第 29 步黑棋才落子在这里，说明这里很可能是 baseline 模型视线盲区，不仅是出现已经有明显的必胜的落子的走法时，选择不落在那里，而且最开始的下棋先后手都集中在中心区，没有多

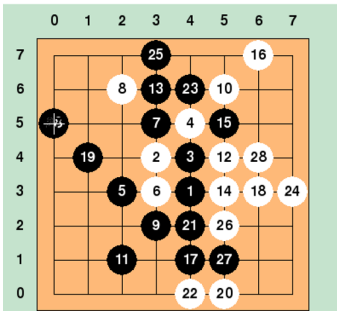

图 5: baseline 模型存在的问题

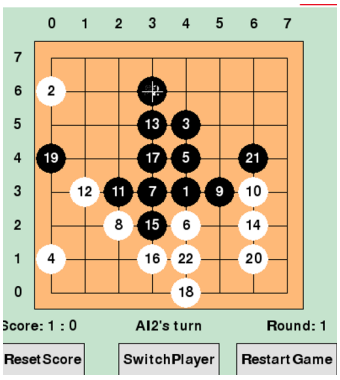


图 6: playout 增大解决这个问题带来的新问题

样化的走法，因此我们希望通过增加模型复杂性提高模型学习能力来解决这一问题，我们认为当模型结构更复杂之后可以有更好的效果，因此引入残差结构。

但是把 playout 调大到 800 也就是下每一步棋搜索路径更多、花费时间更长之后明显解决了这个问题，如下图的第 19 步，模型的下棋风格也有了变化，模型开始尝试刚开始从边缘开始下，但是又出现了新问题，比如在如下棋局中 18 和 20 步棋都没有去堵非常明显的中间那条直线，这是一个很大的问题，当然可能因为模型训练的还不够，从这个角度出发，我们猜测更复杂的结构有更好的能力解决这些不足，因此我们把基础网络改成了 18 层残差结构的卷积网络。

在我们的实验里展示了我们的修改可以使得模型在小棋盘 6*6 上取得更好更快的收敛结果，大棋盘上经过足够的训练收敛之后应该也可以取得更好的结果，实验效果表明虽然我们仍然打不过原作者提供的 baseline 模型，但是相同时间下的训练效果远好于 baseline 模型，单卡训练一天与纯搜索树模型的对弈结果是下 10 局赢 9 局输 1 局，而在 8*8 和 15*15 棋盘上同样取得了不错的结果，仅单卡训练一天就可以达到与纯搜索树模型的对弈赢 5 输 5 的效果。

### 3.3 实现

策略网络和价值网络通过共享网络模型来完成，策略网络用于评估当前棋局下每一个点下一步落子概率，损失函数为对数似然误差，价值网络则返回当前棋局下最终的胜负预测概率，损失函数为均方误差函数，价值网络和策略网络共享基础网络部分，网络输出采用多任务的方式，一个任务为策略网络的目标，一个任务为价值网络的目标。价值网络和策略网络共享前三个卷积层组成的基础网络，策略网络在额外的一个卷积层进行降维之后再经过一个线性层，原因是输入特征里包含四个棋盘，分别是自己已经走的棋局、自己上一步走的棋、对手已经走的棋局和当前是先手还是后手的棋局，这也是输入 channel 为 4 的原因，而价值网络则是一个额外的卷积层和两个全连接层。

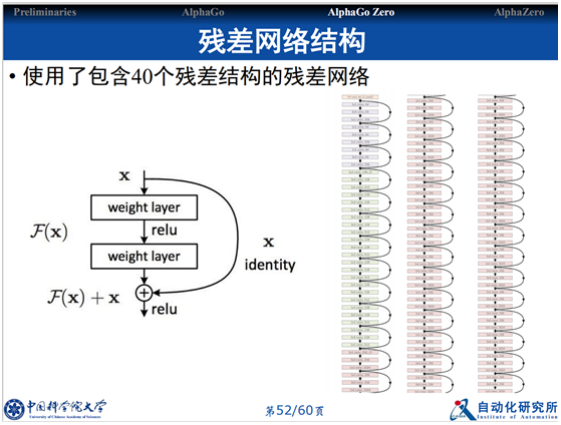


图 7: 残差结构的基础网络

如图所示，AlphaGo Zero 里的一个重要创新点是残差网络结构，使用了包含 40 个在我们进一步的实验里把三层卷积层组成的基础网络改成残差结构的 18 层网络，初步实验结果显示损失函数取得了更快的下降。

## 4 MuZero 调研：

AlphaZero 不需要训练数据，只通过自博弈的方式就取得了能够超过人类水平的棋类游戏智能。在我们的五子棋的实现例子中，alpha zero 不需要像极大极小值搜索那样设计人工评价函数，但取得了超越极大极小值搜索的性能。Alpha Zero 宣称不需要人类知识指导，也不需要人类的训练数据，就能够取得目前最好的性能。但实际上，Alpha Zero 在进行博弈树搜索时，是有人类设计的知识存在的，对于不同的游戏也有相应的规则的应用。基于树搜索的 planning 方法在棋类任务中获得了成果，但都依赖于完美的模拟和规则知识。而在真实世界中，大多数任务的动态模拟是复杂且不可知的，限制了这些方法的应用。

DeepMind 提出了一种基于树搜索和可学习模型的算法 MuZero[1]，能够对 AlphaZero 实现更通用扩展，不再需要依赖领域规则知识，只需要在与环境交互时给出合法的动作空间

以及恰当的结束条件。以下对 MuZero 的调研结果进行阐述，与上文的 AlphaZero 方法进行对比，由于时间安排没能训练进行实验的比较，作为未来的进一步工作等待后续完成。

Muzero 的主要思想是这样的：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）。

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）。在 MuZero 中，这些都被一个隐式的神经网络模型来替代，进一步减少了对人类知识的需求。

# 5 总结、展望和分工

本次课程设计我们对棋盘类智能博弈技术进行了调研，主要对极大极小搜索和基于 alpha zero 的技术在五子棋上进行了实验与分析，并提出了改进的观点与方案。但由于时间以及相关知识的不足，还存在许多问题需要解决，对未来工作的展望上，我们希望继续解决目前在 alpha zero 实现上观察到的存在问题，以及考虑进一步完成 MuZero 在五子棋上进行实现与实验分析。

在分工上，前期的调研工作由我们两名同学一起完成，在后续的实验和报告撰写方面，赵云龙同学负责极大极小搜索部分的原理实现以及相关对比实验和报告撰写，以及代码与文档的整理完善和 UI 界面的修改。李曦云同学主要负责 alphazero 部分的实验分析与报告撰写，模型结构修改训练与 baseline 模型的训练和不同 setting 下 baseline 模型的自博弈，UI 界面的完成。

# 参考文献

[1] 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.

[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, 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.

[4] 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.

[5] 兴军亮. 计算博弈原理与应用第十一讲 *AlphaGo* 技术演化史. 2020.