中国科学技术大学“科学与社会”新生研讨课研究报告

报告题目：基于神经网络的棋类博弈研究

小组组长：周翟恩和

小组成员：方致远

导师姓名：蔡一夫

项目及源代码地址：https://git.lug.ustc.edu.cn/liuhanhuangdou/reversi

2023 年 7 月 4 日

1 研究小组成员及其承担的主要工作

姓名	学号	所在学院	在研究和报告撰写中承担的主要工作
周翟恩和	PB22000008	少年班学院	负责神经网络库，基于神经网络的估价函数编写以及搜索算法剪枝分析。
方致远	PB22000001	少年班学院	负责搜索算法编写以及调研报告的撰写。

2 进度安排

2022 年 11 月开始进行神经网络的编写。

2023 年 2 月完成神经网络的编写，开始进行棋类估价函数的训练。

2023 年 5 月完成黑白棋估价函数，开始进行搜索算法的编写与分析。

2023 年 7 月，完成该项目。

3 摘要、关键词

【摘要】我们组在编写黑白棋类博弈程序的过程中，对神经网络的实现、alpha-beta 剪枝的效率进行了研究。我们利用神经网络对棋盘的状态进行分析，训练出一个较为准确的估价函数。在得到估价函数后，我们利用 alpha-beta 剪枝的算法对每一步棋的下法进行搜索，最终得到一个基于神经网络的棋类博弈程序。在实现过程中，我们模拟了数种棋盘状态，利用 alpha-beta 算法进行搜索，记录下在搜索过程中，剪枝的数量。并与 MIN-MAX 搜索进行比较，进而对 alpha-beta 剪枝搜索算法的性能进行分析。

【关键词】卷积网络　alpha-beta 剪枝　 MIN_MAX 搜索

4 研究报告

4.1 背景和目标

黑白棋是 19 世纪末英国人发明，直到 20 世纪 70 年代日本人长谷川五郎将其进行发展和推广。棋盘为 8×8 的方格布局，开局时在棋盘正中有摆好的四枚棋子，黑白各 2 枚，交叉放置，由执黑棋的一方先落子，双方交替下子，棋子落在方格内，一局游戏结束后双方更换执子颜色。合法的棋步包括：在一个空格新落下一个棋子，并且翻转对手一个或多个棋子。下子方式：把自己颜色的棋子放在棋盘的空格上，而当自己放下的棋子在横、竖、斜八个方向内有一个自己的棋子，则被夹在中间的对方棋子全部翻转会成为自己的棋子。夹住的位置上必须全部是对手的棋子，不能有空格。并且，只有在可以翻转棋子的地方才可以下子。一步棋可以在数个方向上翻棋，任何被夹住的棋子都必须被翻转过来，棋手无权选择不去翻某个棋子。当一方无法下子时，将由另一方继续下子，直至可以该方可以下子，当双方都无法下子时，棋局结束，在棋盘上棋子更多的一方获胜。上世纪末，对黑白棋的算法研究进一步发展。本次研究，基于前人对

黑白棋的研究，编写了卷积网络函数库，并利用其进行估价函数的训练，最后用 alpha-beta 剪枝算法对每一步的下法进行搜索，实现了基于神经网络的。

4.2 研究报告正文

4.2.1 神经网络部分

4.2.1.1 估价函数

定义一个局面的估价函数 $f(S)$ = 从局面 S 开始，双方都按最优策略走棋，终局时的棋子数量差。目前广泛采用的估价函数基于黑白棋高手自身对于棋盘的理解，评估指标包括：

- 地势估值表。黑白棋和围棋一样，也遵守着“金角银边烂肚皮”的定律，四个角的地势值非常大，其次是四条边。因此我们再给 8*8 地图点分配地势值时，大体满足角边重，中腹轻的模式。
- 行动力。黑白棋中，大体来说，下一步的选择越多，则越占优势。
- 稳定子。指这些棋子之后无论如何都不可能再被对手翻转
- 基于模板的估价 [3]。在更专业的黑白棋程序中（如知名的 ZEBRA），会对棋盘上出现的一些局部结构（也称“模板”）打分。

上述方法大多基于人们在下棋中发现的规律。那么，能否训练一个神经网络，从零学习这些规律?

4.2.1.2 模型结构及实现

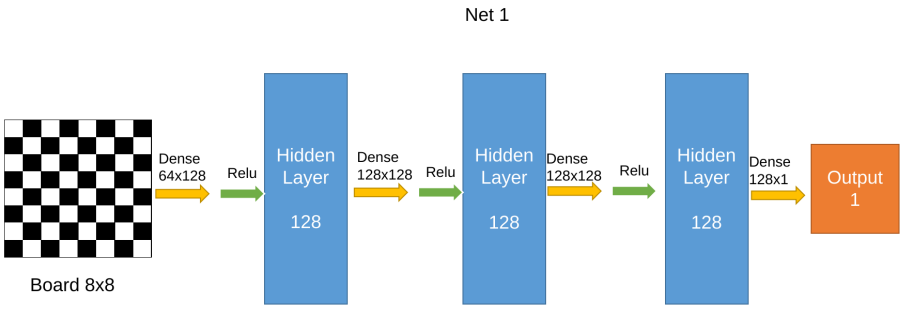


图 1: 网络结构

网络结构如上图所示，网络输入一个 8*8 的棋盘数据，经过三层隐藏层，最后输出一个对当前局面的估价。代码实现基于作者在项目前期开发的 MicroDNN 库, 项目主页：链接。

4.2.1.3 训练方法

使用法国黑白棋联合会提供的 WThor 棋谱库作为训练数据。但是由于我们只有棋局过程和棋局的结果，如何训练出一个模型，来估计整个下棋过程中，每时每刻棋局的局面的函数。

因为棋谱由人类高手下出，在假定人类高手聪明绝顶的前提下，可以得到整个棋局过程中，棋盘估价都应该等于最终的结果。但这显然是不可能的，因为这样就意味着初始局面会对应多个不同估价。如果以棋谱中每个局面作为输入，一盘棋的结果作为标签来训练网络，则训练出的网络在实际走棋时，相邻两步估价市场会会波动较大，不利于用来 minmax 搜索。

同时，虽然人类高手也无法做到每步都下最优策略，但是可以保证每一步都以较大概率是最优策略。因此，另一种可行的训练方式如下图所示：

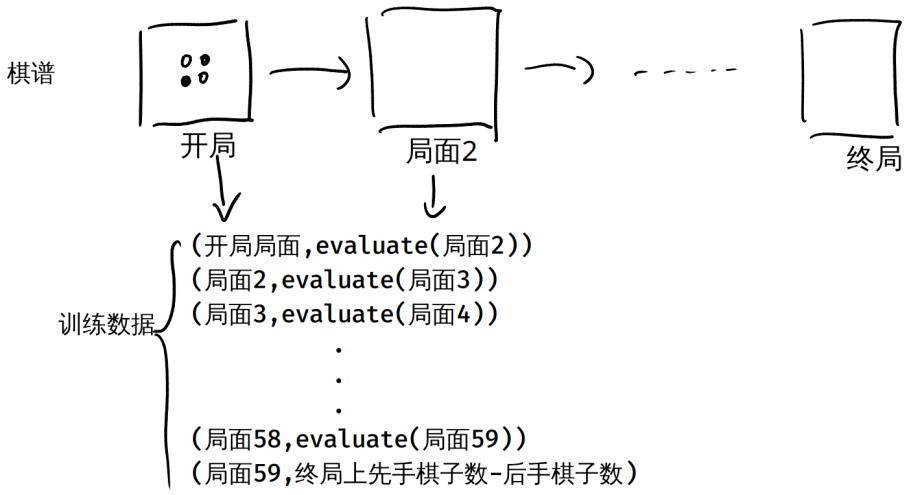


图 2: 另一种训练方式

对于棋谱中的一场对局，每一个输入对应的 label 为下一个局面的估价。然而这样的数据训练出的网络容易退化成如下情形：

$$f(S) = \begin{cases} 0 & \text{S 离终局还远} \\ sth & \text{S 靠近终局} \end{cases}$$

从而导致在棋局的前半程，估价函数差别都太小，没有作用。为了平衡这两方面：相邻两步，估价函数的连续性; 棋局前半段，估价函数的有效性，我们最终采用以下训练方式，每组训练数据为：

$$(\text{局面 i}, \frac{\text{evaluate(局面 i+1)} + \text{终局先手棋子数-终局后手棋子数}}{2})$$

实际训练结果表示，这种确定训练数据的方式训练出的模型确实优与前面两种。

训练进行了 20 轮迭代，每轮优化 10000 个 epoch, 前 10 轮使用 adam 优化，后是 10 轮使用 sgd 微调。

4.2.1.4 训练指标

模型训练完成后，在测试集上误差标准差为 8.36，进行单层 minmax 搜索后预测下一步的准确率为 39.4%。那么这一神经网络实际的棋力如何?

作者此前曾开发过一个基于传统估价函数的黑白棋 AI, 项目地址：https://github.com/bdfzoier/Reversi。这一 AI 的估价函数基于对地势，稳定子，行动力的评判，将这几者的线性组合作为估价函数，线性组合的系数通过梯度下降法训练得到。这一程序在北大的程序对战平台的 Botzone 中 Rating 为 1114, 属于中上排名。

为了排除估价函数计算速度，其余代码执行效率的影响，将两者代码中 minmax 的搜索层数都限制为 1 层。对战结果显示，神经网络先手以 16:48 不敌传统估价函数，而后手以 37:27 战胜传统估价程序。

这一结果表明，虽然理论上三层隐藏层的神经网络可以拟合任意函数，但是在黑白棋估价这一任务中，在同等量级的计算量下，较浅的神经网络很难学习到深层的规律，并不显著优于传统的估价方法。这也解释了为什么目前黑白棋主流的估价函数依然是基于传统方法的。

4.2.1.5 地势评分建议

许多基于传统算法的黑白棋程序中，地势评分都由手工决定，但我们实际可以把上述多层的神经网络换成单层的线性层，同样可以训练得到最优参数。并且这一参数的意义与前文所属的“地势估价”相同。

5.03	1.61	0.01	0.13	0.13	-0.02	1.54	5.
1.56	1.81	-1.24	-0.4	-0.36	-1.18	1.85	1.58
-0.06	-1.26	-0.9	0.01	-0.02	-0.93	-1.22	-0.01
0.14	-0.46	-0.02	-0.07	-0.17	-0.01	-0.47	0.11
0.09	-0.4	-0.02	-0.1	-0.09	-0.05	-0.41	0.11
-0.04	-1.27	-0.91	-0.01	0.02	-0.91	-1.2	-0.04
1.58	1.82	-1.22	-0.42	-0.47	-1.22	1.86	1.57
5.1	1.55	-0.02	0.16	0.06	-0.05	1.52	5.01

图 3: 地势得分

可视化后如下图所示，可以明显看到靠边元素权重极高

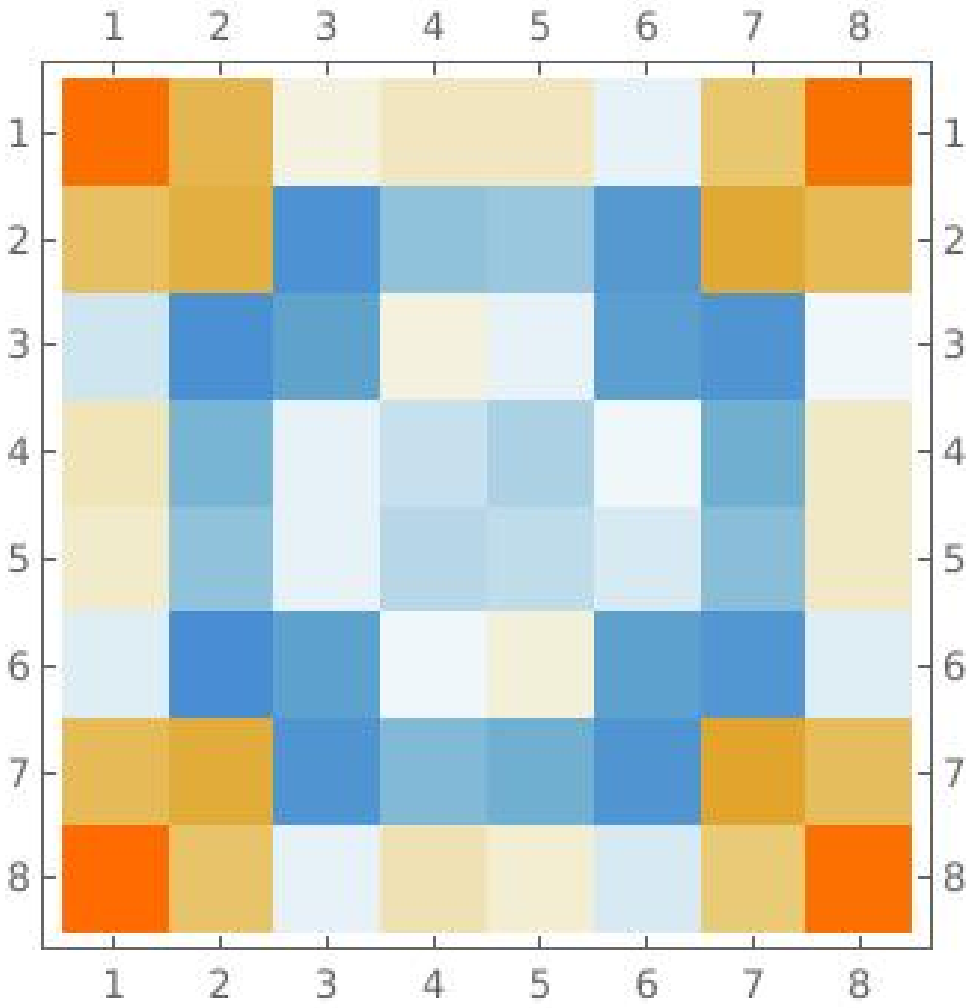


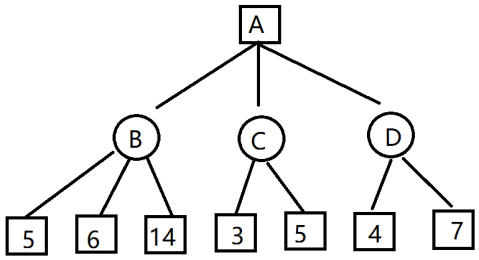
图 4: 地势得分（可视化）

4.2.2 搜索部分

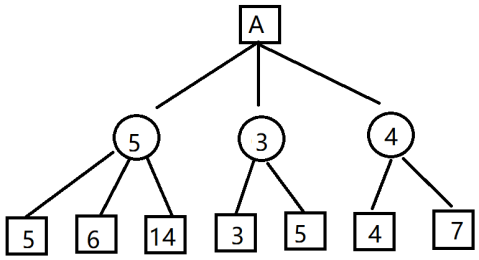
4.2.2.1 MinMax 搜索

MinMax 搜索又名极小化极大算法，是一种找出失败的最大可能性中最小值的算法。MinMax 搜索方法通常用于棋类等两方较量的游戏和程序。一方的选择要使其优势最大化，而另一方的选择要使得优势最小化。在该算法中，假设对手总能找到令本方优势最小化的方法。基于该假设，我们就能设计出一个算法，使得本方的选择使得失败的最大可能性最小。假设我们由如下图的博弈树，则可根据 MinMax 算法的原理，得到每一步选择的胜率。博弈树中，圆圈为对方行动，方框为我方行动。

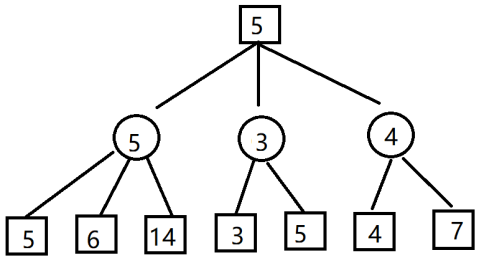
在第二层的树中，为对方行动轮，根据假设：对手总能找到令本方优势最小化的方法，则在 B 这个节点，会选择下一层中对手胜率最小的走法，则 B 节点的胜率改为 5。根据此原理，将 C、D 节点的胜率分



别改为 3，4。



在第一层 A 节点时，为我方行动轮，则会选择我方胜率最大的下法。则 A 节点的胜率被修改为 5。



根据以上的分析，可按照此方法编写出搜索函数。
函数定义：

```
1 float MIN_MAX(int depth, board &b)
```

将棋盘遍历一遍，利用递归调用，可获得在每一处落子的胜率。在递归调用中，设置一个层数，当到达最底层时，即 depth=0，为博弈树叶子节点，直接调用卷积网络所得的估价函数，获得叶子节点的胜率。当为我方行动轮时，需要选择我方胜率最大的走法，当为对方行动轮时，选择使得我方胜率最小的走法。
函数核心部分如下：

```
if (-b.turn == 1) {
    best_value = -INT_MAX;
    for (int i = 1; i <= SIZE; i++) {
       for (int j = 1; j <= SIZE; j++){
           if (b.putchess({i, j}, -b.turn, 1)){
               float value = MIN_MAX(depth - 1, b);
                      if (best_value < value) {
                          best_value = value;
                      }
                      b = save;
            }
       }
    }
} else {
    best_value = +INT_MAX;
    for (int i = 1; i <= SIZE; i++) {
       for (int j = 1; j <= SIZE; j++){
           if (b.putchess({i, j}, -b.turn, 1)){
               float value = MIN_MAX(depth - 1, b);
                      if (best_value > value) {
                          best_value = value;
                      }
                      b = save;
            }
       }
    }
}
```

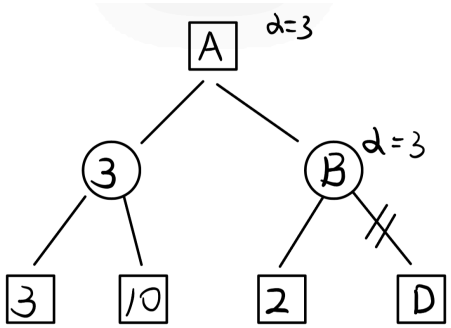
b.turn 表示当前轮为我方轮还敌方轮，及为 Min 节点，还是 Max 节点。其中 best_value 记录下最优解，save 记录下初始的棋盘状态，用于搜索完一个子节点后回溯。

4.2.2.2 alpha-beta 剪枝

由上述分析可知，MinMax 搜索算法有一个极大的缺点。在搜索过程中，会构建出一个极大的博弈树。当每一步有很多步选择时，假设为 N, 则当搜索层数为 D 时，需要遍历到的节点数为 N_D。遍历到的节点数会随 D 的增长指数爆炸。

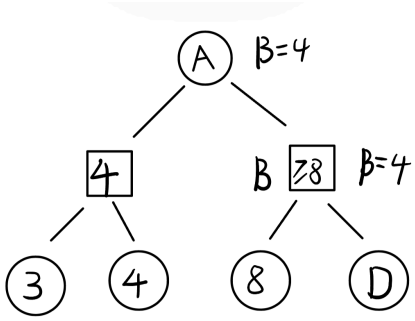
在此种情况下，我们需要对 MinMax 搜索算法进行优秀。我们选择了经典的 alpha—beta 剪枝算法。算法的基本原理如下：

我们在每个节点都记录下该节点可能获得的最大胜率值与最小胜率值。对于 Max 节点，记录下其可能获得的胜率最小值 alpha。在搜索其 Min 的子节点时，若发现 Min 节点可能获得比 alpha 更小的胜率 q 时，将不再继续搜索此子树，因为 Min 节点会选择胜率更小的走法，其胜率一定小于等于 q，而 q 小于等于 alpha, 此节点不可能获得比 alpha 更大的胜率。



如上图所示，当搜索到 B 节点时，发现一子节点的胜率为 2，因为 B 节点为 Min 节点，其得到的胜率值一定小于 2。因为其 alpha 值为 3，A 得到的胜率值目前最小为 3。2<3，B 节点不可能改变 A 的胜率，所以接下来就不需要继续搜索 B 的子节点，对 B 进行剪枝，即 alpha 剪枝。

对于 Min 节点，我们记录下其可获得胜率的最大值 beta。在搜索其 Max 子节点时，若发现 Max 节点可能获得比 beta 更大的胜率 q 时，将不会再搜索此子树。此 Max 子节点会选择胜率大于等于 q 的走法，而 q 大于等于 beta，此节点为 Min 节点，要选择胜率更小的走法，可对此节点进行剪枝。



如上图所示，当搜索到 B 节点时，发现一子节点的胜率为 8，因为 B 节点为 Max 节点，其得到的胜率值一定大于等于 4。因为其 beta 值为 4，A 得到的胜率值目前最大为 4。8>4，B 节点不可能改变 A 的

胜率，所以接下来就不需要继续搜索 B 的子节点，对 B 进行剪枝，即 beta 剪枝。

根据以上原理的推导，可以得到 alpha—beta 剪枝算法的具体过程。每个节点都储存 α 与 β 变量。其 α 表示目前所有可能解中的最大下界，β 表示目前所有可能解中的最小上界。因此，如果搜索树上的一个节点被考虑作为最优解的路上的节点（或者说是这个节点被认为是有必要进行搜索的节点），那么它一定满足以下条件（N 是当前节点的估价值）：$\alpha \leq N \leq \beta$ 在我们进行求解的过程中，α 和 β 会逐渐逼近。如果对于某一个节点，出现了 $\alpha > \beta$ 的情况，那么，说明这个点一定不会产生最优解了，所以，我们就不再对其进行扩展（也就是不再生成子节点）。

其代码实现如下：

```
if (-b.turn == 1) {
        best_value = -INT_MAX;
        for (int i = 1; i <= SIZE; i++) {
           for (int j = 1; j <= SIZE; j++) {
               if (b.putchess({i, j}, -b.turn, 1)) {
                   float value = alpha_beta(alpha, beta, depth - 1, b);
                   alpha = max(alpha, value);
                   best_value = max(best_value, value);
                   if (alpha >= beta)
                            return alpha;
                   b = save;
               }
           }
        }
} else {
        best_value = +INT_MAX;
        for (int i = 1; i <= SIZE; i++) {
           for (int j = 1; j <= SIZE; j++) {
               if (b.putchess({i, j}, -b.turn, 1)) {
                   float value = alpha_beta(alpha, beta, depth - 1, b);
                   beta = min(beta, value);
                   best_value = min(best_value, value);
                   if (beta <= alpha)
                            return beta;
                   b = save;
               }
           }
        }
```

```
29 }
```

4.2.2.3 剪枝效率分析

根据测量，在一局典型的黑白棋棋局上（从棋谱库中随机选取 10 局取平均），朴素 MINMAX 搜索,AlphaBeta 剪枝算法搜索的平均叶子节点数量增长速度如下所示：

表 1: 搜索节点数统计

搜索深度/d	MinMax 平均叶子数	AlphaBeta 剪枝平均叶子数	预测值	启发式算法平均叶子数
1	8.0×10^0	8.0×10^0	4.3×10^0	8.0×10^0
2	7.9×10^1	3.4×10^1	1.9×10^1	1.7×10^1
3	8.2×10^2	2.1×10^2	8.0×10^1	8.7×10^1
4	9.3×10^3	8.3×10^2	3.5×10^2	2.0×10^2
5	1.0×10^5	3.9×10^3	1.5×10^3	9.1×10^2
6	1.3×10^6	1.6×10^4	6.4×10^3	2.1×10^3

根据参考文献 [1][2] 的结果，假如搜索树为完全 n 叉树时，且估价值在叶子上均匀随机分布，则 alphabeta 剪枝的 branching factor 的下界为：

$$R = \frac{\epsilon}{1-\epsilon}$$

其中 ϵ 是方程 $x^n + x + 1 = 0$ 的唯一正根，根据上方数据，得到黑白棋的搜索树近似于一个完全 8 叉树），并且估价函数近似随机分布，可以代入上述结论得：

$$R = 4.31$$

因此，alphabeta 剪枝期望搜索到的叶子数为

$$N = 4.31^d$$

与测量结果对比，发现近似效果良好。

4.2.2.4 剪枝次数进一步优化

而上述下界是基于“叶子上估价函数均匀分布”这一结论得到的。但在事实上，黑白棋的局势优劣在叶子上并不均匀分布。评分较高的叶子节点都会聚集在少数子树上。凭借这一点，我们可以用启发式的搜

索增加 alphabeta 剪枝的次数，减少搜索到的叶子数。们在 alphabeta 搜索中枚举所有可能的下一步棋时，可以先用下一步的估价函数大致估算一下这一子树的优劣，再优先搜索较优的分支。

在 Max 层，这种启发式可以让 alpha 提前增大，使 beta 提前减小，从而增加剪枝次数。在 n=6 时，这一优化使搜索到的叶子数在 alphabeta 的基础上进一步减少了 87%！详见上面图表

核心代码如下：

```
float best_value;
board save = b;
std::vector<std::pair<float, coor>> nx;
if (-b.turn == 1) {
    best_value = -INF;

    for (int i = 1; i <= SIZE; i++) {
        for (int j = 1; j <= SIZE; j++) {
            if (b.putchess({i, j}, -b.turn, 1)) {
                nx.push_back(std::make_pair(-assess(b), coor{i, j}));
                b = save;
            }
        }
    }
    sort(nx.begin(), nx.end());
    for (auto &i : nx)
        if (b.putchess(i.second, -b.turn, 1)) {
            float value = alpha_beta(alpha, beta, depth - 1, b, cnt);
            alpha = max(alpha, value);
            best_value = max(best_value, value);
            if (alpha >= beta) return best_value;
            b = save;
        }
} else {
    best_value = +INF;

    for (int i = 1; i <= SIZE; i++) {
        for (int j = 1; j <= SIZE; j++) {
            if (b.putchess({i, j}, -b.turn, 1)) {
                nx.push_back(std::make_pair(assess(b), coor{i, j}));
                b = save;
```

```
            }
        }
    }

    sort(nx.begin(), nx.end());
    for (auto &i : nx)
        if (b.putchess(i.second, -b.turn, 1)) {
            float value = alpha_beta(alpha, beta, depth - 1, b, cnt);
            beta = min(beta, value);
            best_value = min(best_value, value);
            if (beta <= alpha) return best_value;
            b = save;
        }
}
```

在使用上述启发式算法时，可以发现估价函数越准确，越能预测子树的优劣，则剪枝数量越多。因此上述启发式算法搜索的叶子节点数也可以作为衡量估价函数好坏的一个标准。

4.3 结论/总结

论文中，我们首先实现了一个微型的神经网络库，并且给出了一种在可以推广搭配无监督情况（没有给出每个局面评分）下，训练博弈游戏估价函数的方法。在黑白棋情境下对比了采用这一方法训练的神经网络与传统方法的效果。同时，对传统方法中的地势权重给出了建议。

在搜索算法方面，我们实现了 MinMax 搜索，alphabeta 剪枝，及其的一个启发性改进，对比了他们搜索的叶子数。并根据已有的研究给出了黑白棋中 alphabeta 剪枝搜索到的叶子数量的估计公式。

4.4 致谢

感谢蔡一夫老师及罗文涛老师在学术写作和论文阅读上的教学指导。

感谢 LUG 提供的代码托管平台 git.lug.ustc.edu.cn，以及学校网络信息管理部提供的 latex.ustc.edu.cn，大大方便了论文写作。

4.5 参考文献

参考文献

[1] The Solution for the Branching Factor of the Alpha-Beta Pruning Algorithm and its Optimality,Judea Pearl

[2] ANALYSIS OF THE ALPHA-BETA PRUNING ALGORITHM,S. H. Fuller, J. G. Gaschnig and J. J. Gillogly

[3] Experiments with Multi-ProbCut and a New High-Quality Evaluation Function for Othello, Michael Buro

5 导师评审意见

导师根据报告的整体质量、平时讨论时的表现、答辩的情况及在报告中各学生的贡献程度，着重考察学生的科学探索精神、自主学习能力、独立思考能力、逻辑推理能力、实际动手能力、团队合作精神等要素，给小组成员做出评价。

导师签字: 日期: