**AlphaGo原理：**用深度神经网络和树搜索征服围棋

AlphaGo 总体上由两个神经网络（大脑）构成，即一个可在当前局面下判断下一步可以往哪里走的**决策函数**（除了简单地观察数据训练还可以通过自己和自己下棋的强化学习模式来实现机器学习），一个**评估函数**（通过海量自我对局训练，学习评估整体盘面的优劣）。

第一个大脑的简单模式会判断出在当前局面下有哪些走法值得考虑。第一个大脑的复杂模式通过蒙特卡洛树来展开各种走法，即所谓的“算棋”，以判断每种走法的优劣。在这个计算过程中，第二个大脑会协助第一个大脑通过判断局面来砍掉大量不值得深入考虑的分岔树，从而大大提高计算效率。与此同时，第二个大脑本身通过下下一步棋导致的新局面的优劣本身也能给出关于下一步棋的建议。最终，两个大脑的建议被平均加权，做出最终的决定。

两个网络的输入都是整个棋盘的状态，不存在一个注重局部一个注重整体。区别在于功能，policy network用于直接给出落子策略，value network用于快速估计当前局面导致最终获胜的概率。MCTS在一个简化版的policy network的指导下多次搜索到终局来实时（但比较慢）估计当前局面最终获胜的概率。value network估算的概率和MCTS估算的概率直接加权平均，最终选择落子位置的时候是依据这个加权平均和完整版policy network给出的对各落子位置的收益的一个加权平均，但是一个位置被MCTS搜索（类似于计算）的次数越多，policy network的结果（类似于经验和直觉）给的权值也越小。所以这一步并不是一个简单的加权平均，也不是局部判断和全局判断的加权平均。两个网络考虑了多少全局因素和局部因素只跟训练数据和当前棋盘状态有关，跟用的哪个网络没有关系。

在论文中一个有趣的结论是：**两个网络取平均的结果比依赖两者各自得出的结果都要好很多**。这应当是让 AlphaGo 表现出和人类相似性的关键所在。

**AlphaGo主要部分的纯Python实现** https://github.com/tensorflow/minigo

AlphaGo 项目主页：http://www.deepmind.com/alpha-go.html

Nature 论文：http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html

Nature 报道：http://www.nature.com/news/google-ai-algorithm-masters-ancient-game-of-go-1.19234

Dan Maas 对论文的浅显总结：<http://www.dcine.com/2016/01/28/alphago/>

AlphaGo 论文中文翻译

<https://blog.csdn.net/u013390476/article/details/50925347?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522159028724019195162521150%2522%252C%2522scm%2522%253A%252220140713.130102334.pc%255Fblog.%2522%257D&request_id=159028724019195162521150&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~blog~first_rank_v2~rank_blog_default-4-50925347.pc_v2_rank_blog_default&utm_term=%E5%A6%82%E4%BD%95%E7%94%A8%E7%A5%9E%E7%BB%8F%E7%BD%91%E7%BB%9C%E5%86%99%E5%AF%B9%E5%BC%88%E8%AF%84%E4%BC%B0%E5%87%BD%E6%95%B0>

# 评估函数怎么写？

**通用评估函数构成**：

(1) f(局面) = 子力(我的子数 - 对手的子数)；

(2) g(局面) = 角(我控制的 - 对手控制的)；

(3) h(局面) = 机动性(我可以走的)。

你必须组合这些函数(可能还有其他项)来构成一个评价函数：eval = a·f + b·g + c·h。例如，你可以尝试：eval = -1·f + 10·g + 1·h。但是这些数值从哪里来？哪种数值的组合可以得到最好的效果？下面是手工找到这些数值的一些方法：　

**(1) 规格化(Normalize)。**如果你只关心评价的顺序，而通常不怎么关心评价值，那么你就可以把每一项都乘以同样的常数。这就意味着你对某个特定的项目(比如说兵的价值)可以硬性设一个值，其他值就表示成它们相当于多少个兵。这个做法可以让你减少一个需要设定的参数。　

**(2) 约束法(Deduce Constraints)。**你希望让电脑作出什么样的判断，考虑这些问题就可以确定一些参数了。例如在国际象棋中，即使你赚到一个兵，用车换象或马通常还是坏的，但是如果你赚到两个兵那还是好的，因此子力价值要满足R>B+P(防止换单兵)和R<B+2P(鼓励换双兵)。这样的不等式你给得越多，合适的权重组合就越少。在一开始设定权重值的时候，这个方法通常可以得到合适的值，但是后面你仍然需要做一些调整。　

**(3) 交手法(Hand Tweaking)。**这是很常用的方法，仅仅是让你的程序对弈足够多的次数，来找到它的优势和弱点，猜测哪些参数会让程序更好，然后挑选新的参数。这个方法可以很快得到合理的结果，但是你需要对这种棋类有足够的了解，这样就可以根据程序的对局来做分析，知道程序的问题在哪里。(也就是说，当程序很笨但是你很聪明时，这个方法最有用。)　　　不需要人工干预的方法有：　

**(4) 爬山法(Hill-Climbing)。**类似于交手法，每次对权重作很小的改变，测试改变后的表现，仅当成绩提高时才采纳这个改变，需要重复很多次。这个方法看上去很慢，并且只能找到“局部最优”的组合(即评价可能很差，但是任何很小的改变都会使评价更差)。　

**(5) 模拟退火法(Simulated Annealing)。**类似于爬山法，也是对权重做改变来提高成绩的。但是如果改变没有提高成绩，有时候(随机地，给定一个几率)也采纳改变，试图跳出全局最优。这个方法需要给定一些几率，从几率高、梯度大的条件开始，然后逐渐减小。模拟退火法比爬山法更慢，但是最终可能得到比较好的值。　

**(6) 遗传算法(Genetic Algorithms)。**爬山法和模拟退火法可以得到一组好的权重，它们是逐渐变化的。相反，遗传算法可以得到几组不同的好的权重，不断增加新的组合跟原来的做比较(取用某组中的某个权重，另一组中的另一个权重，互相交换得到新的)，通过淘汰坏的组合来控制种群的数量。　

**(7) 神经网络(Neural Networks)。**实际上这更多地是一种评价函数的类型，而不是用来选择权重的：神经元是阈值(输入权重的和)的函数，第一层神经元输入的关于局面的性质(例如位棋盘表示中的某几个位)就可以构造网络，然后前一层的结果输入到后一层。因此单输入神经元的单层网络就等同于我们上次讨论过的一阶评价函数，但是接下来就可以构造更复杂的神经网络了，而且用这种方法作为评价函数是不难的(只要根据输入的改变来重新计算神经元的输出就可以了)。问题仍然像前面所说的，如何设置权重？除了前面的方法外，针对神经网络还发展出一些方法，例如“暂时差别学习”(Temporal Difference Learning)。其基本思想是确定网络何时会作出坏的评价，并且让每个权重增加或减小看是否会评价得更好，这很类似于爬山法。跟其他自动学习的方法相比，神经网络的好处就在于它不需要很多人类的智慧：你不需要懂得太多的棋类知识，就可以让程序有个比较好的评价函数。**但是根据目前我们掌握的情况，根据自己的智慧来做评价函数，要比机器学习做得好，并且做得快**。

**Minimax算法的步骤：**

1. 首先确定最大搜索深度D，D可能达到终局，也可能是一个中间搜索。
2. 在最大深度为D的搜索树叶子节点上，使用预定义的价值评价函数对叶子节点价值进行评价。
3. 自底向上为非叶子节点赋值。其中max节点取子节点最大值，min节点取子节点最小值。
4. 每次轮到我方时（此时必处在搜索树的某个max节点），选择价值等于此max节点价值的那个子节点路径。

**Alpha-beta剪枝是对Minimax的补充和改进**

Alpha-beta基于这样一种朴素的思想：时时刻刻记得当前已经知道的最好选择，如果从当前格局搜索下去，不可能找到比已知最优解更好的解，则停止这个格局分支的搜索（剪枝），回溯到父节点继续搜索。基本原理是从根节点开始采用深度优先的方式构造格局树，在构造每个节点时，都会读取此节点的alpha和beta两个值，其中alpha表示搜索到当前节点时已知的最好选择的下界，而beta表示从这个节点往下搜索最坏结局的上界。由于我们假设对手会将局势引入最坏结局之一，因此当beta小于alpha时，表示从此处开始不论最终结局是哪一个，其上限价值也要低于已知的最优解，也就是说已经不可能此处向下找到更好的解，所以就会剪枝。

**使用蒙特卡洛树搜索的意义：**

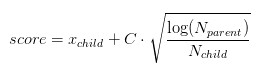
**1、它可以给出一个局面评估，虽然不准，但比没有强**。

**2、根据它的设计，搜索树会较好地自动集中到“更值得搜索的变化”（注意，也不一定准）**如果发现一个不错的着法，蒙特卡洛树搜索会较快地把它看到很深，可以说它结合了广度优先搜索和深度优先搜索，类似于启发式搜索。随着搜索树的自动生长，蒙特卡洛树搜索可以保证在足够长的时间后收敛到完美解（但可能需要极长的时间）

步骤：

1. **选择**（Selection）。从根节点往下走，每次都选一个“最值得看的子节点”，直到来到一个“存在未扩展的子节点”的节点（这个局面存在未走过的后续着法）。

最简单有效的选择公式是这样的：



其中 x 是节点的当前胜率估计（注意，如前所述，要考虑当前是谁先手），N 是节点的访问次数。C 是一个常数。C 越大就越偏向于广度搜索，C 越小就越偏向于深度搜索。有时需要根据实际情况调参。

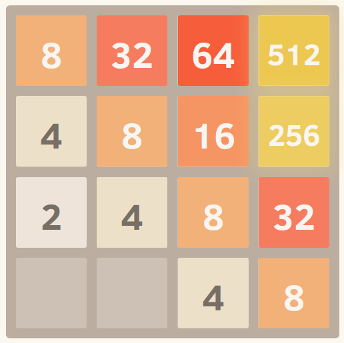
1. **扩展**（Expansion），我们给这个节点加上一个 0/0 子节点，对应之前所说的“未扩展的子节点”，就是还没有试过的一个着法。
2. **模拟**（Simluation）。从上面这个没有试过的着法开始，用快速走子策略（Rollout policy）走到底，得到一个胜负结果。按照普遍的观点，快速走子策略适合选择一个棋力很弱但走子很快的策略。因为如果这个策略走得慢（比如用 AlphaGo 的策略网络走棋），虽然棋力会更强，结果会更准确，但由于耗时多了，在单位时间内的模拟次数就少了，所以不一定会棋力更强，有可能会更弱。这也是为什么我们一般只模拟一次，因为如果模拟多次，虽然更准确，但更慢。
3. **回溯**（Backpropagation）。把模拟的结果加到它的所有父节点上。

蒙特卡洛树搜索（MCTS）代码详解【python】

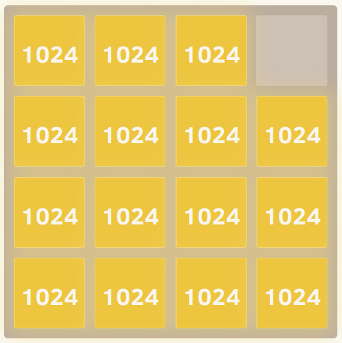
<https://blog.csdn.net/windowsyun/article/details/88770799>

# 2048评估函数构建可参考的几点启发式指标：

### 单调性 单调性指方块从左到右、从上到下均遵从递增或递减。一般来说，越单调的格局越好。下面是一个具有良好单调格局的例子：



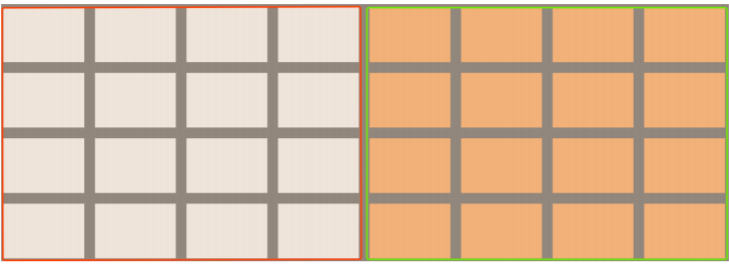
### 平滑性 平滑性是指每个方块与其直接相邻方块数值的差，其中差越小越平滑。例如2旁边是4就比2旁边是128平滑。一般认为越平滑的格局越好。下面是一个具有极端平滑性的例子：



### 空位数 这个很好理解，因为一般来说，空位越少对玩家越不利。所以我们认为空位越多的格局越好。

### 孤立空格数 这个指标评价空格被分开的程度，空格越分散则格局越差。

### 敌方空位中我方棋子的个数、大小、位置 一般来说，该指标的个数越多，敌方可操作空间越小，我方格局越有利；棋子数值越大，对方越难以将其合并以破局，我方格局越有利。棋子在对方中央的2\*2区域内最有利，敌方无法在“合并”阶段移动“身处曹营”的我方棋子，当棋子位于中央区域，会严重干扰对方的合并操作，此外，位于图示的白色区域的我方棋子对格局更有利，因为其既有上述优点且容易实现己方对其的合并加分。对于四边，我做了一个优先级排序（做得着急，欢迎大家修改）



2

4

4

3

2

1