【AlphaGo】AlphaGo背后的力量：蒙特卡洛树搜索入门指南 - 产业智能官

选自int8 Blog

机器之心编译

> 我们都知道 DeepMind 的围棋程序 AlphaGo，以及它超越人类的强大能力，也经常会听到「蒙特卡洛树搜索」这个概念。事实上，蒙特卡洛树搜索是在完美信息博弈场景中进行决策的一种通用技术，除游戏之外，它还在很多现实世界的应用中有着广阔前景。本文中，我们会以 AlphaGo 为例子，对这一方法进行详细介绍。

长久以来，学术世界一直认为计算机在围棋这个复杂游戏上达到超越人类的水平是几乎无法实现的。它被视为人工智能的「圣杯」——一个我们原本希望在未来十年挑战的遥远里程碑。在国际象棋上，「深蓝」曾在 20 多年前实现了自己的目标，而其后数年，没有一个围棋引擎能够打败人类顶尖棋手。围棋及其引发的「数字混沌」是如此令人着迷，以至于人们一度将其想象为人类「对抗」计算机的最后壁垒。

然而正如我们所知，2016 年 DeepMind 推出的人工智能围棋程序 AlphaGo 结束了这一局面，它在当年 3 月份的系列比赛中以 4：1 的比分击败了来自韩国的前世界冠军李世石。AlphaGo 证明了世人对于虚拟和现实世界的怀疑是错误的。而在短短一年之后，新一代围棋程序 AlphaGo Zero 在测试中就能够以 100：0 的成绩击败旧程序，这无疑宣告了人类在围棋上和计算机的差距已经越来越远。

作为今天最被人们所熟知的人工智能系统（没有之一），AlphaGo/Zero 是一个多种计算方法的集合体，人类工程学的杰作，其核心组件包含：

- 蒙特卡洛树搜索——内含用于树遍历的 PUCT 函数的某些变体
- 残差卷积神经网络——其中的策略和价值网络被用于评估棋局，以进行下一步落子位置的先验概率估算。
- 强化学习——通过自我对弈进行神经网络训练

在本文中，我们只着重于介绍蒙特卡洛树搜索（MCTS/Monte Carlo Tree Search）。这个算法很容易理解，而且也在游戏人工智能领域外有很多应用。

目录

介绍

蒙特卡洛树搜索是由前里尔第三大学助理教授 Rémi Coulom 在围棋程序 Crazy Stone 中首先引入的方法——后者是第一个在围棋上达到职业五段水平的计算机程序。

从最直观的角度来看，蒙特卡洛树搜索有一个主要目的：给出一个「游戏状态」并选择「胜率最高的下一步」。在本文中，我会试图解释蒙特卡洛树搜索的大多数细节，其中我们也会不时回顾 AlphaGo/Zero，并试图解释那些在 DeepMind AI 程序系列中使用的 MCTS 变体。

有限两人零和回合制游戏

蒙特卡洛树搜索运行的框架/环境是「游戏」，其本身是一个非常抽象的广义术语，所以在这里我们只针对于一种游戏类型：有限两人零和回合制游戏——这听起来或许有点复杂，不过其实很简单，让我们来分析一下：

- 「游戏」意味着处理「互动情况」。互动意味着有玩家会参与进来（一个或多个）
- 「有限」表示在任何时间点上，玩家之间都有有限的互动
- 「两人」有限游戏，顾名思义
- 「回合制」表示玩家按照一定顺序进行游戏——轮流出招
- 最后「零和游戏」——这意味着游戏双方有着相反的目标，换句话说：在游戏的任何终结状态下，所有玩家获得的总和等于零。有时这样的游戏也被称为严格竞争博弈

我们可以轻易验证围棋、国际象棋或井字棋是有限两人零和回合制游戏。的确，它们都是两个玩家，游戏可选的下一步也是有限的，且游戏是严格竞争的——两名玩家会进行对抗（游戏的所有输出之和为零）。

Notes：请注意，为了简化本教程，我们只专注于可能场景的某系子集，蒙特卡洛树搜索是一个应用广泛的工具，适用于两人有限零和游戏以外。更为全面的概述请参阅：http://mcts.ai/pubs/mcts-survey-master.pdf

如何表征一个博弈

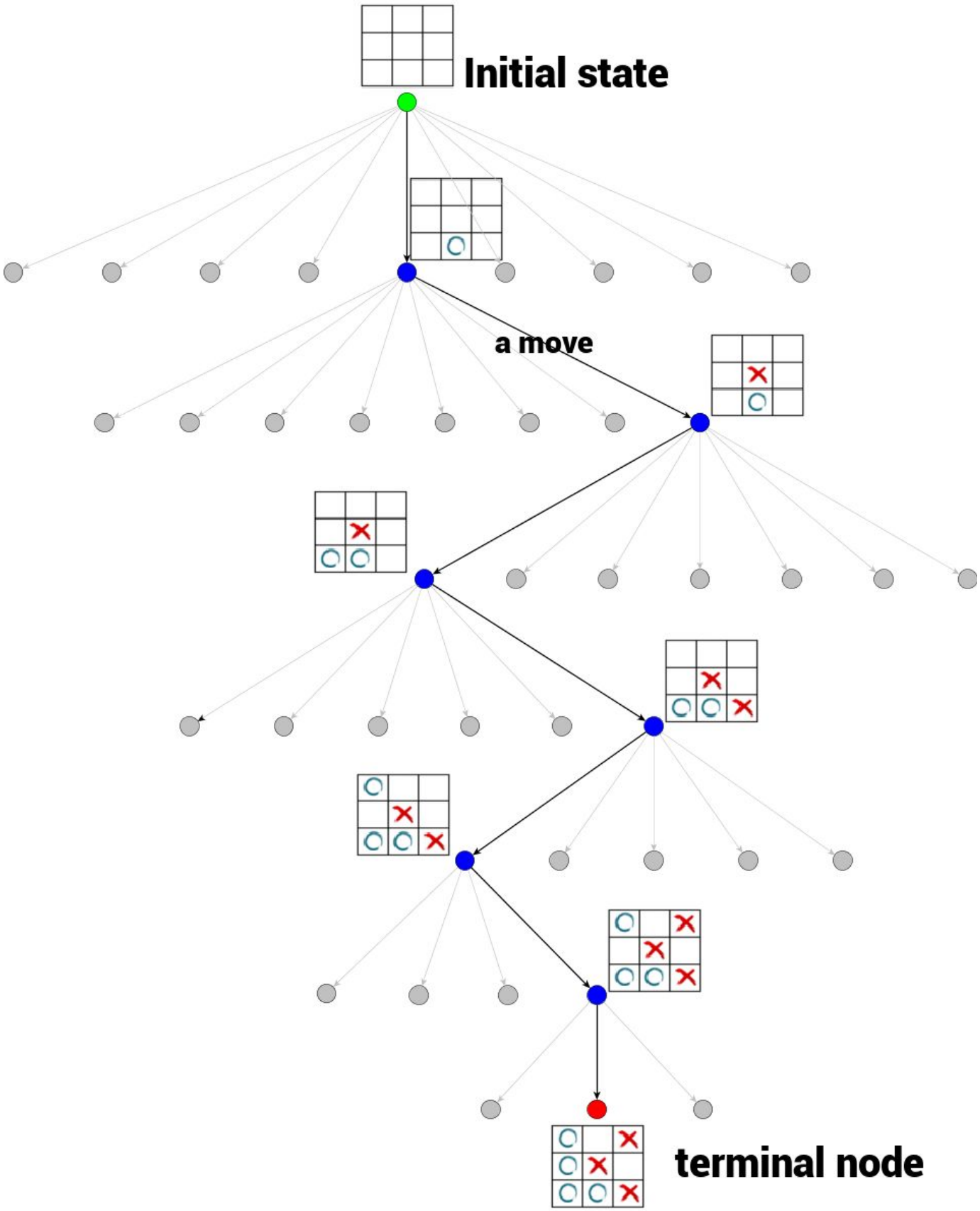
形式上，一个博弈由一系列的基本数学实体表征。在一本 PhD 级别的博弈论书中你可以找到这样的定义：

定义 1. 一个博弈的扩展式可以用一个多元组来定义：

$$\Gamma_E = \{\mathcal{X}, \mathcal{A}, I, p, \alpha, \mathcal{H}, H, \iota, \rho, u\}$$

从计算机编程的角度来看形式化的定义可能难以理解，但幸运的是，我们可以使用一种著名的数据结构以简单的形式来表征一个博弈：博弈树。

博弈树是一种树结构，其中每一个节点表征博弈的确定状态。从一个节点向其子节点的转换被称为一个行动（move）。节点的子节点数目被称为分支因子（branching factor）。树的根节点表征博弈的初始状态。我们还区分了博弈树的端节点（terminal nodes），即没有子节点的节点，表示博弈无法再继续进行。端节点的状态可以被评估，并总结博弈的结果。



为了限制博弈树的大小，仅访问被展开的状态，未被展开的状态被标记为灰色。

在上图的井字棋博弈树（部分展示）的例子中：

- 在顶部，你可以看到树的根节点，其表征了井字棋博弈的初始状态，即空白棋盘（标记为绿色）；
- 任何从一个节点向另一个节点的转换被称为一个行动；
- 井字棋的分支因子是变化的，它依赖于树的深度；

- 从一个根节点到一个端节点的树遍历表征了单个博弈过程。

博弈树是一种递归的数据结构，因此当你选择了一个最佳行动并到达一个子节点的时候，这个子节点其实就是其子树的根节点。因此，你可以在每一次（以不同的根节点开始），将博弈看成由博弈树表征的「寻找最有潜力的下一步行动」问题的序列。在实践中很常见的是，你不需要记住到达当前状态的路径，因为它在当前的博弈状态中并不重要。

什么是最有潜力的下一步行动？简要介绍极小极大（minimax）策略和 alpha-beta 剪枝算法

再次提醒，我们的最终目标是在给定博弈状态的前提下，利用博弈树寻找最有潜力的下一步行动。但这究竟是什么意思呢？

这个问题并没有直接的答案。首先你不能提前知道对手的策略，对手可能是个高手，也可能是个菜鸟。假定在国际象棋中，你知道对手是个业余爱好者（数学家会说，你的对手使用的是混合策略），你可以使用简单的策略来尝试欺骗对手并快速获得胜利。但很明显，同样的策略在面对强大的对手时将适得其反。

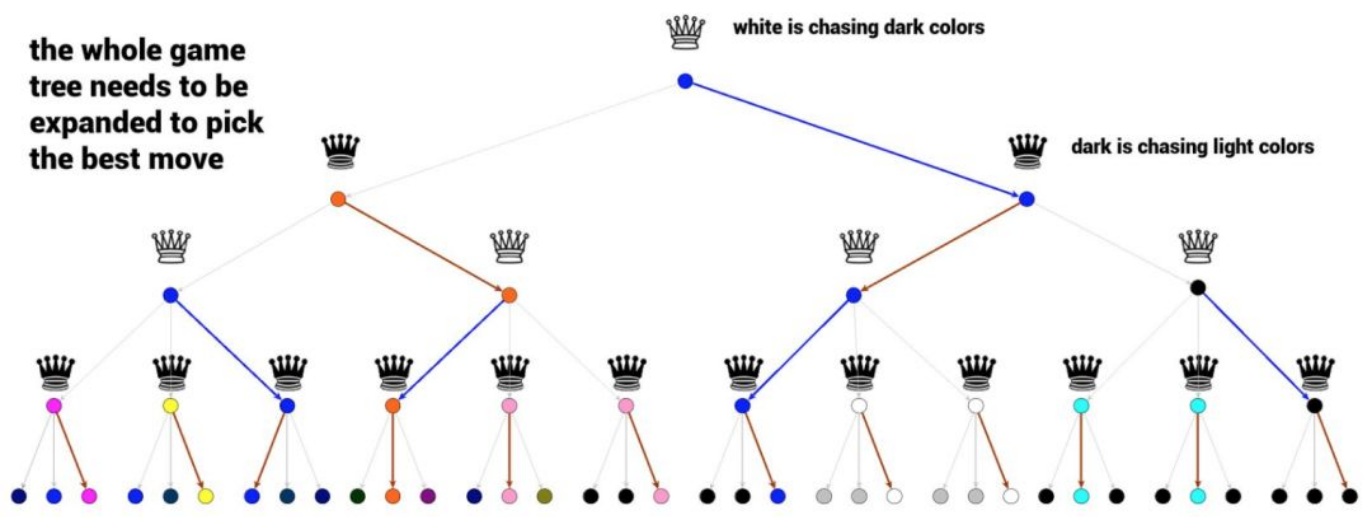
如果你完全不了解对手，那么你可以使用一种非常保守的策略即极小极大算法，在假定你的对手执行最佳行动的前提下，最大化你的收益，也可以说在各种获得最小收益的策略中选择有最大收益的策略。这种方法以放弃最优策略为代价，从而最小化了风险，因此它是一种非常保守的方法。在 A 和 B 的两人有限零和序列博弈中（其中 A 尝试最大化其收益，而 B 尝试最小化 A 的收益），极小极大算法可以用以下的递归形式来描述：

$$v_A(s_i) = \max_{a_i} v_B(\text{move}(s_i, a_i)) \qquad v_A(\ddot{s}) = \text{eval}(\ddot{s})$$
$$v_B(s_i) = \min_{a_i} v_A(\text{move}(s_i, a_i)) \qquad v_B(\hat{s}) = -\text{eval}(\hat{s})$$

其中：

- v_A 和 v_B 分别是玩家 A 和玩家 B 的效用函数（效用=收益）；
- move 是一个函数，它在给定当前状态 s_i 和在该状态的动作 a_i 下，生成下一个博弈状态（当前节点的子节点之一）；
- eval 是一个评估最终博弈状态（在端节点处）的函数；
- s hat 是任意的最终博弈状态（即一个端节点）；
- 右下方式子的负号表示该博弈是一个零和博弈。

简单来说，给定状态 s，并假定对手尝试最小化你的收益，你希望找到能最大化收益的动作 a_i。这正是该算法被称为极小极大的原因。我们需要做的就是展开整个博弈树，并反向传播由递归形式的规则得到的值。



上图中的博弈树展示了极小极大算法中的最佳行动选择过程。白皇后希望博弈的结果尽可能的黑暗（冷色，奖励值=像素强度），而黑皇后希望博弈的结果尽可能的明亮（暖色）。每一个层级的选择都是极小极大判断的最优结果。我们可以从底部的终端节点开始，其中的选择是很明显的。黑皇后将总是选择最明亮的颜色，然后白皇后将寻找最大的奖励并选择到达最暗颜色的路径，等等。这正是基础的极小极大算法的执行过程。

极小极大算法的最大弱点是它需要展开整个博弈树。对于有高分支因子的博弈（例如围棋或国际象棋），该算法将导致巨大的博弈树，使得计算无法进行。

那么有什么解救的办法吗？其中一个方法是仅在确定的阈值深度 d 内展开博弈树，但是我们无法保证在阈值深度 d 处的任何节点是否端节点。因此我们一个函数来评估非终端博弈状态。这对于人类来说很自然：即使博弈仍在进行，你也可能通过观察围棋或国际象棋的棋盘预测胜者。例如，对以下棋局可以很容易知道结束棋局的走法。

另一种克服博弈树规模过大问题的方法是通过 alpha-beta 剪枝算法来修剪博弈树。alpha-beta 剪枝是提升版的极小极大算法，它以极小极大算法的形式遍历博弈树，并避免某些树分支的展开，其得到的结果在最好的情况下等于极小极大算法的结果。alpha-beta 剪枝通过压缩搜索空间提高搜索效率。

极小极大算法和 alpha-beta 修剪算法已经是相当成熟的解决方案，目前已被用于多个成功的博弈引擎例如 Stockfish——AlphaZero 的主要对手之一。

蒙特卡洛树搜索的基本概念

在蒙特卡洛树搜索算法中，最优行动会通过一种新颖的方式计算出来。顾名思义，蒙特卡洛树搜索会多次模拟博弈，并尝试根据模拟结果预测最优的移动方案。

蒙特卡洛树搜索的主要概念是搜索，即沿着博弈树向下的一组遍历过程。单次遍历的路径会从根节点（当前博弈状态）延伸到没有完全展开的节点，未完全展开的节点表示其子节点至少有一个未访问到。遇到未完全展开的节点时，它的一个未访问子节点将会作为单次模拟的根节点，随后模拟的结果将会反向传播回当前树的根节点并更新博弈树的节点统计数据。一旦搜索受限于时间或计算力而终止，下一步行动将基于收集到的统计数据进行决策。

下面有一些关于上述蒙特卡洛树搜索过程的关键问题，它们有助于我们的理解：

- 什么是展开或未完全展开的博弈树？
- 在搜索过程中，向下遍历是什么意思？如何选择访问的下一个子节点？
- 什么是模拟？
- 什么是反向传播？
- 反向传播回的统计数据是什么，在展开博弈树结点更新的是什么？
- 最后的行动策略到底是如何选择的？

下面，我们将依次解决这些问题，因而能对蒙特卡洛树搜索有一个清晰的理解。

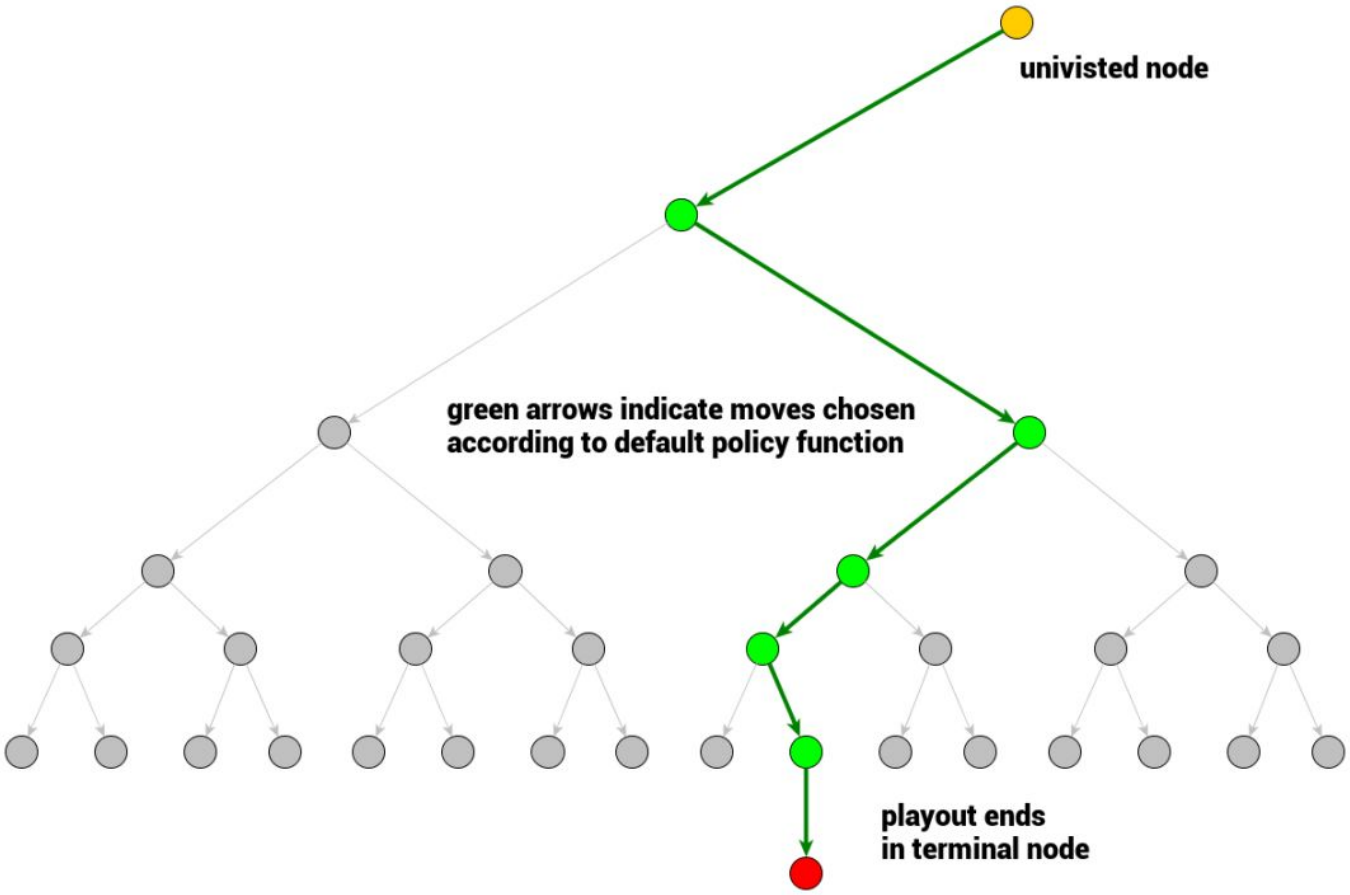
模拟

首先我们会关注于模拟，它并不会过多依赖于其它术语的定义。模拟即单次博弈策略，它是一系列从当前节点（表示博弈状态）开始，并在计算出博弈结果后结束于端节点。模拟是一个在随机博弈初始点开始评估近似计算的博弈树节点。那在模拟中如何选择行动呢？

在模拟中，行动可以通过 rollout 策略函数选择：

$$\mathrm{RolloutPolicy}: s_i \rightarrow a_i$$

该函数将输入一个博弈状态，并产生下一次行动的选择。在实践中，该函数会设计为允许很多次模拟快速进行，一般默认的 rollout 策略函数可以是服从均匀分布的随机采样。



Alpha Go 和 Alpha Zero 中的模拟

在 Alpha Go Lee 叶 S_L 的评估中，它会采用以下两个分量的加权和：

- 带有自定义快速 rollout 策略的标准 rollout 评估 z_L，它是一个带有人工特征的浅层 softmax 神经网络。
- 称之为价值网络的 13 层卷积网络 v_0 从 Alpha Go 自我对抗中抽取 30mln 不同位置进行训练，并最后预测评估位置。

$$V(S_L) = (1-\alpha)v_0(S_L) + \alpha z_L$$

Deepmind 的工程师在 Alpha Zero 中更进一步，他们根本不会执行模拟，他们会使用 19 层 CNN 残差网络直接评估当前节点。

$$V(S_L) = f_0(S_L)$$

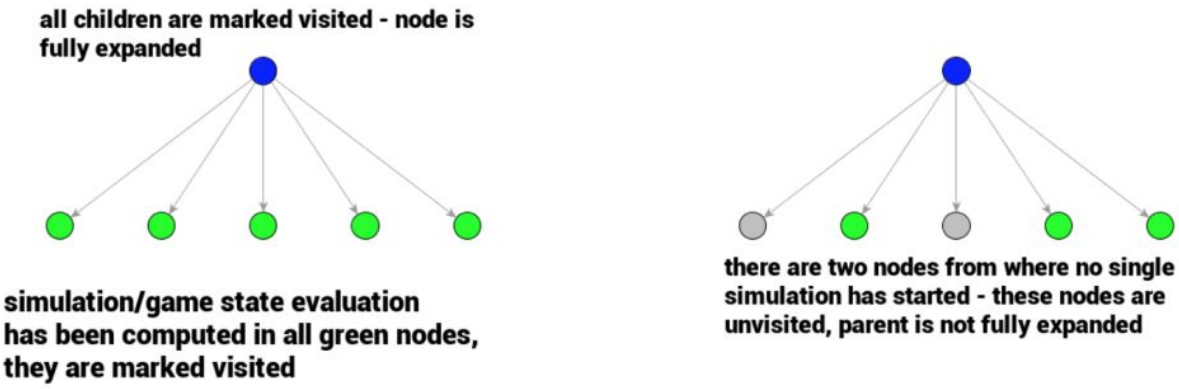
最简单的模拟形式只是在给定博弈状态下的随机行动序列。模拟总会产生一个评估，对于博弈来说，该评估就是胜利、失败或平局等结果，但通常任何值都可以是模拟的合理结果。

在蒙特卡洛树搜索模拟中，我们始终会从一个前面没访问的节点开始，因此下面会介绍关于访问节点的意义。

博弈树的展开节点、完全展开节点和访问节点

现在我们需要思考人类是如何考虑围棋或象棋等博弈的。给定一个根节点并加上博弈的规则，那么博弈树的其余部分就已经隐含表示出来了。我们可以遍历它而不需要将整棵树储存在内存中。在最初的根节点中，它是完全未展开的，我们处于博弈的初始状态，其余所有节点都没有被访问。一旦我们需要执行一个行动，我们就会思考采用该行动后会产生怎样的结果，因此访问一个节点后，需要分析该节点位置与带来的效用。

蒙特卡洛树搜索也是采用相同的特性构建博弈树。所有节点可以分为访问或未访问，那么一个节点的访问到底指的是什么？一般而言，如果模拟将该节点作为初始节点，这就意味着它至少评估了一次，那么它就可以视为已访问节点。如果某节点的所有子节点都是已访问节点，那么它就可视为完全展开的节点，相对而言也就存在未完全展开的节点。

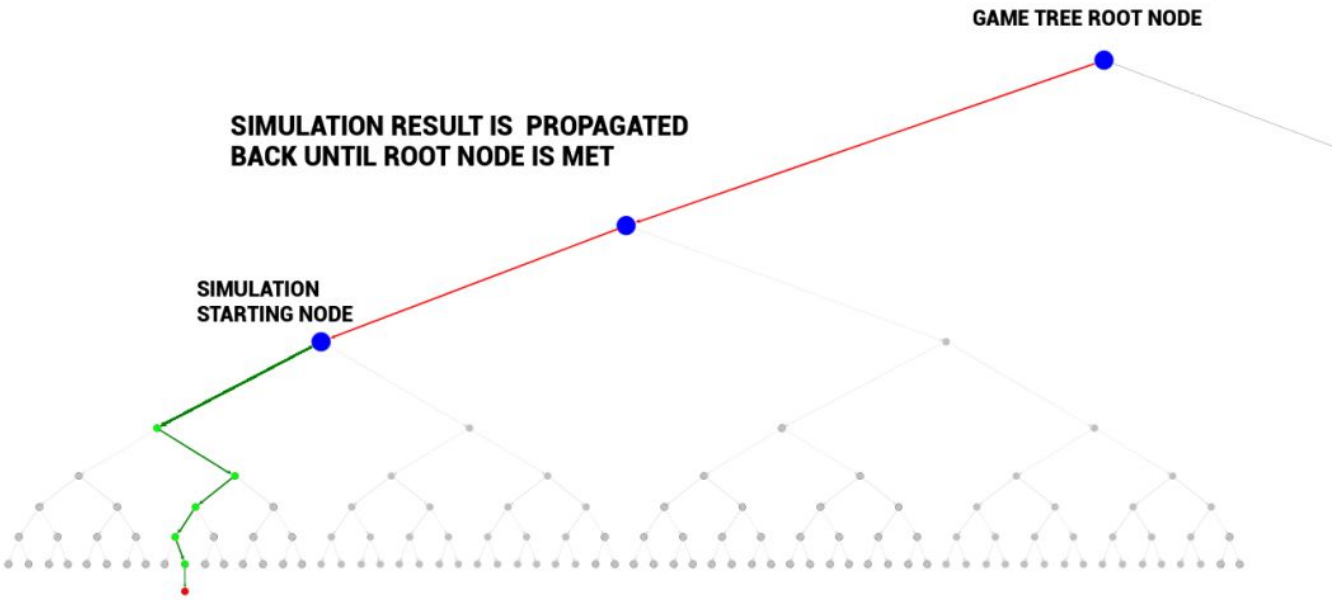


在实践中，搜索开始时，根节点的所有子节点都未被访问。然后一个节点被选中，第一个模拟（评估）就开始了。

请注意：模拟过程中 rollout 策略函数选择的节点并未被访问。它们仍然是未被访问状态，即使 rollout 经过它们，只有模拟开始的那个节点是被访问的状态。

反向传播：将模拟结果传播回去

当初次访问节点的模拟结束后，其结果会反向传播至当前博弈树的根节点。模拟开始的节点被标注为已访问。



反向传播是从子节点（模拟开始的地方）遍历回根节点。模拟结果被传输至根节点，反向传播路径上的每个节点的统计数据都被计算 / 更新。反向传播保证每个节点的数据都会反映开始于其所有子节点的模拟结果（因为模拟结果被传输回博弈树的根节点）。

节点的统计数据

反向传播模拟结果的目的是更新反向传播路径（包括模拟起始的节点）上所有节点 v 的总模拟奖励 Q(v) 以及总访问次数 N(v)。

- Q(v) 即总模拟奖励是节点 v 的一项属性，在最简单的形式中是通过考虑的节点得到的模拟结果的总和。
- N(v) 即总访问次数是节点 v 的另一项属性，表示节点 v 位于反向传播路径上的次数（即它对总模拟奖励做出了多少次贡献）。

每个被访问节点都会保存这两个值，一旦完成了确定次数的模拟之后，被访问的节点就保存了它们被利用/探索（expolited/explored）的信息。

换句话说，当你查看任意节点的统计数据时，这两个值将反映该节点的潜在价值（总模拟奖励）和它被探索的程度（总访问次数）。高奖励的节点是很好的可利用候选，而那些访问次数少的节点也可能是有价值的（因为它们尚未得到很好的探索）。

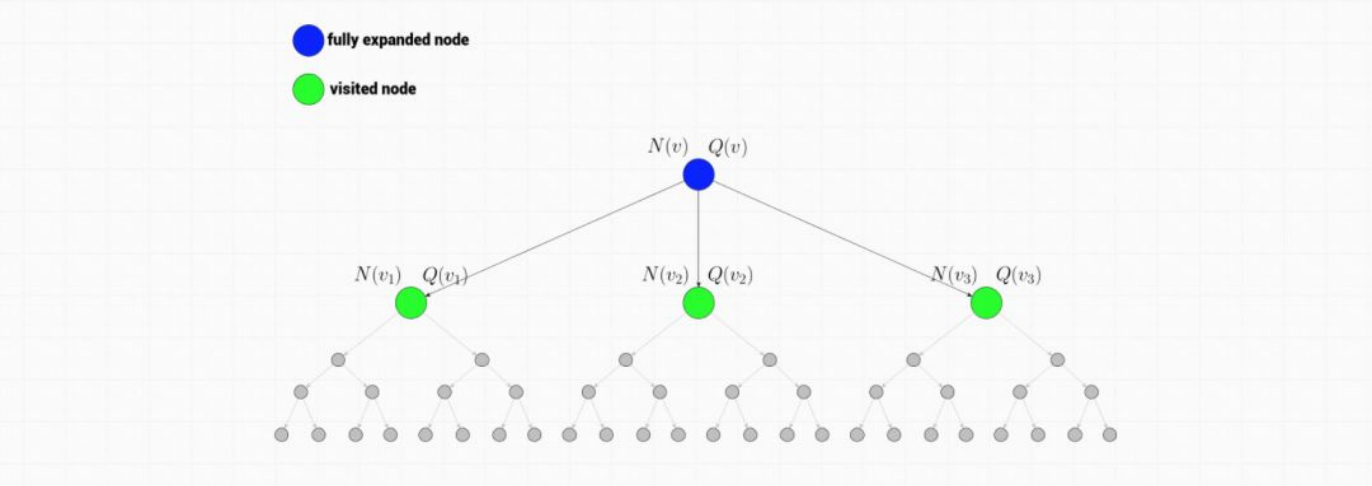
我们还缺少一块拼图。如何从一个根节点到达一个未访问节点，来启动一次模拟呢？

博弈树遍历

在搜索最开始的时候，由于我们还没有进行任何模拟，所以先选择未被访问的节点。在每个未被访问的节点上进行单次模拟，结果被反向传播至根节点，然后根节点即被认为经过了完全展开。

但是接下来怎么做呢？现在我们如何从完全展开的节点导向未被访问的节点呢？我们必须遍历被访问节点的层，目前没有很好的继续进行的方式。

为了在路径中找到下一个节点，以通过完全展开的节点 v 开始下一次模拟，我们需要考虑 v 所有子节点 v_1, v_2, ..., v_k 的信息，以及节点 v 本身的信息。现在我们来思考一下可用信息有哪些：



当前节点（蓝色）是完全展开的，因此它必须经过访问，以存储节点数据：它及其子节点的总模拟奖励和总访问次数。这些值是为了最后一部分：树的置信上限（UCT）做准备。

树的置信上限

UCT 是一个函数，使我们在被访问节点中选择下一个要遍历的节点，这也是蒙特卡洛树搜索的核心函数：

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

UCT 最大的节点就是蒙特卡洛树搜索遍历过程中选择的节点。让我们来看看 UCT 函数如何运行：

首先，该函数为节点 v 的子节点 v_i 而定义，它包括两个组件：第一个组件是

$$\frac{Q(v_i)}{N(v_i)}$$

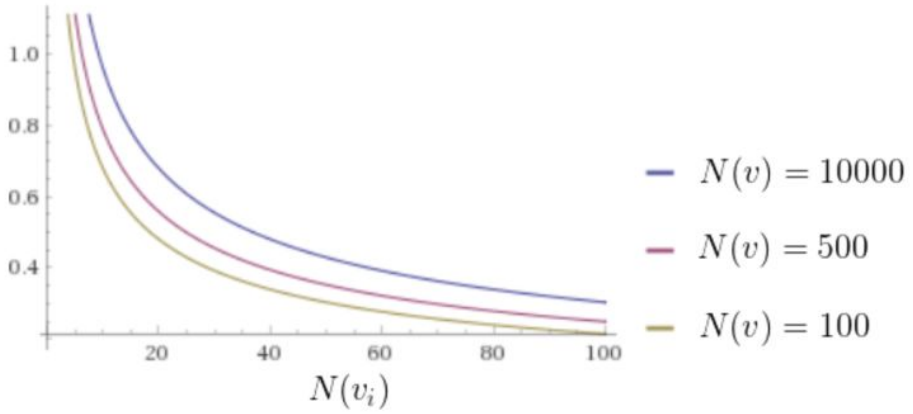
，又叫做 exploitation 组件，可以理解为赢 / 输率，总模拟奖励（simulation reward）除以总访问次数，即节点 v_i 的胜率评估结果。我们当然更想遍历具备高赢率的节点。

为什么不仅仅使用 exploitation 组件呢？因为我们会在搜索开始时很快结束对取得单次获胜的节点的贪婪探索。

简单示例：

假设我们仅使用 exploitation UCT 组件开始蒙特卡洛树搜索。从根节点开始，我们对所有子节点进行一次模拟，然后下一步仅访问那些模拟结果至少有一次是赢的节点。第一次模拟结果不幸失败的节点会立刻被舍弃。

因此我们还要有第二个 UCT 组件 exploration。exploration 组件支持未被探索的节点，这些节点相对来说更少被访问（N(v_i) 较低）。我们来看一下 UCT 函数 exploration 组件的形状：随着节点访问量的增加而递减，给访问量少的节点提供更高的被选中几率，以指引 exploration 探索。



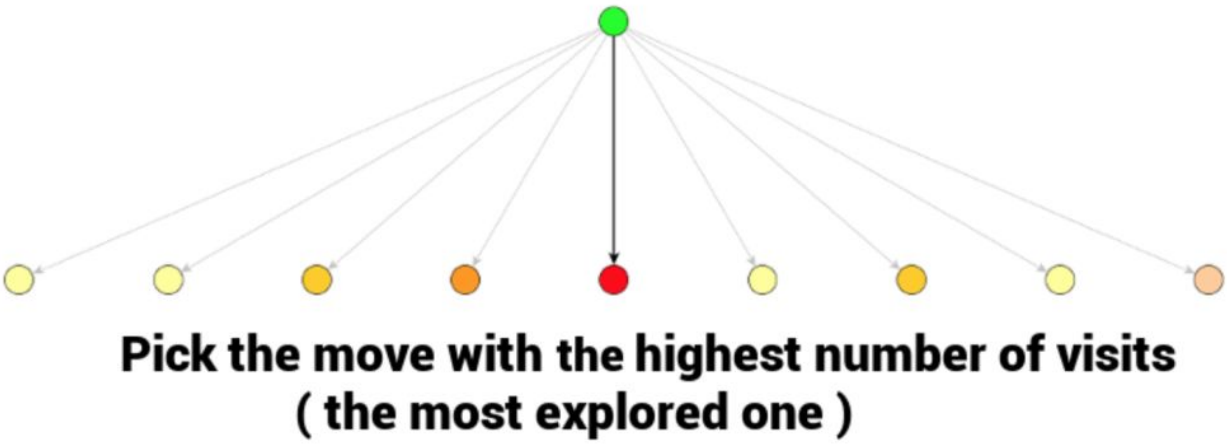
最终，UCT 公式中的参数 c 控制蒙特卡洛树搜索中 expolitation 和 exploration 组件之间的权衡。

UCT 函数中的一个重要标志是：在竞争性游戏中，其 exploitaion 组件 Q_i 的计算通常与在节点 i 处行动的玩家有关，这意味着在遍历博弈树时，玩家视角根据被遍历的节点而变化：对于任意连续节点，玩家视角都是相反的。

终止蒙特卡洛树搜索

现在我们了解了实现蒙特卡洛树搜索所需要的所有因素，但还有一些问题需要回答。首先，我们什么时候可以终止 MCTS？答案是：看情况。如果你构建一个游戏引擎，那么你的「思考时间」有限，计算能力也有限。因此最安全的选择是只要资源允许，就可以一直运行 MCTS。

一旦 MCTS 过程结束，最好的一步通常是具备最高访问量 N(v_i) 的一步，因为它的奖励值评估结果最好（评估的值必须很高，因为它被探索的频率也最高）。



在使用蒙特卡洛树搜索走了一步之后，你的选择节点就变成了对手下一步的起始游戏状态。一旦他走了一步，你就可以执行蒙特卡洛树搜索，从表示对手选择游戏状态的节点开始。之前的 MCTS round 数据可能仍然在你现在考虑的新分支以内。这就可以重新使用数据而不是从头构建新的树，事实上这就是 Alpha Go / Alpha Zero 创造者所做的。

总结

现在，我们来回顾一下蒙特卡洛树搜索的简单定义，并将其封装进伪代码：

```
def monte_carlo_tree_search(root):
    while resources_left(time, computational power):
        leaf = traverse(root) # leaf = unvisited node
        simulation_result = rollout(leaf)
        backpropagate(leaf, simulation_result)
    return best_child(root)

def traverse(node):
    while fully_expanded(node):
        node = best_uct(node)
    return pick_univisted(node.children) or node # in case no children are present / node is terminal

def rollout(node):
    while non_terminal(node):
        node = rollout_policy(node)
    return result(node)

def rollout_policy(node):
    return pick_random(node.children)

def backpropagate(node, result):
   if is_root(node) return
   node.stats = update_stats(node, result)
   backpropagate(node.parent)

def best_child(node):
    pick child with highest number of visits
```

你可以看到它缩减至非常少的函数，这些函数对任何游戏都有效，不只是围棋或象棋。你可以在这里找到蒙特卡洛树搜索用于井字棋（Tic-Tac-Toe）的实现示例：https://github.com/int8/monte-carlo-tree-search。

希望本文对大家有所帮助。

SYNCED

原文链接：https://int8.io/monte-carlo-tree-search-beginners-guide/

本文为机器之心编

人工智能赛博物理操作系统

AI-CPS OS

“人工智能赛博物理操作系统”（新一代技术+商业操作系统“AI-CPS OS”：云计算+大数据+物联网+区块链+人工智能）分支用来的今天，企业领导者必须了解如何将“技术”全面渗入整个公司、产品等“商业”场景中，利用AI-CPS OS形成数字化+智能化力量，实现行业的重新布局、企业的重新构建和自我的焕然新生。

AI-CPS OS的真正价值并不来自构成技术或功能，而是要以一种传递独特竞争优势的方式将自动化+信息化、智造+产品+服务和数据+分析一体化，这种整合方式能够释放新的业务和运营模式。如果不能实现跨功能的更大规模融合，没有颠覆现状的意愿，这些将不可能实现。

领导者无法依靠某种单一战略方法来应对多维度的数字化变革。面对新一代技术+商业操作系统AI-CPS OS颠覆性的数字化+智能化力量，领导者必须在行业、企业与个人这三个层面都保持领先地位：

1. **重新行业布局**：你的世界观要怎样改变才算足够？你必须对行业典范进行怎样的反思？
2. **重新构建企业**：你的企业需要做出什么样的变化？你准备如何重新定义你的公司？
3. **重新打造自己**：你需要成为怎样的人？要重塑自己并在数字化+智能化时代保有领先地位，你必须如何去做？

AI-CPS OS是数字化智能化创新平台，设计思路是将大数据、物联网、区块链和人工智能等无缝整合在云端，可以帮助企业将创新成果融入自身业务体系，实现各个前沿技术在云端的优势协同。AI-CPS OS形成的数字化+智能化力量与行业、企业及个人三个层面的交叉，形成了领导力模式，使数字化融入到领导者所在企业与领导方式的核心位置：

1. **精细**：这种力量能够使人在更加真实、细致的层面观察与感知现实世界和数字化世界正在发生的一切，进而理解和更加精细地进行产品个性化控制、微观业务场景事件和结果控制。
2. **智能**：模型随着时间（数据）的变化而变化，整个系统就具备了智能（自学习）的能力。
3. **高效**：企业需要建立实时或者准实时的数据采集传输、模型预测和响应决策能力，这样智能就从批量性、阶段性的行为变成一个可以实时触达的行为。
4. **不确定性**：数字化变更颠覆和改变了领导者曾经仰仗的思维方式、结构和实践经验，其结果就是形成了复合不确定性这种颠覆性力量。主要的不确定性蕴含于三个领域：技术、文化、制度。
5. **边界模糊**：数字世界与现实世界的不断融合成CPS不仅让人们所知行业的核心产品、经济学定理和可能性都产生了变化，还模糊了不同行业间的界限。这种效应正在向生态系统、企业、客户、产品快速蔓延。

AI-CPS OS形成的数字化+智能化力量通过三个方式激发经济增长：

1. 创造虚拟劳动力，承担需要适应性和敏捷性的复杂任务，即“智能自动化”，以区别于传统的自动化解决方案；
2. 对现有劳动力和实物资产进行有利的补充和提升，提高资本效率；