AlphaGo Zero 原理、实现和讨论

本文回顾了 AlphaGo Zero 的原理，包括蒙特卡罗树搜索、博弈论、极小极大搜索和神经网络。详细介绍了 AlphaGo Zero 的流程和方法，解释了各种细节并有相应的代码。讨论了蒙特卡洛树搜索和蒙特卡洛方法的区别，并说明了蒙特卡罗树搜索在博弈论中的适用范围。

关键词： AlphaGo Zero， 蒙特卡罗树搜索，博弈论

This article reviews the principles of AlphaGo Zero, including Monte Carlo tree search, game theory, minimax search, and neural networks. Detailed description of AlphaGo Zero's process and method, explaining the details and corresponding code. The difference between Monte Carlo tree search and Monte Carlo method is discussed, and the application range of Monte Carlo tree search in game theory is illustrated.

Keywords: AlphaGo Zero, Monte Carlo tree search, game theory

1、AlphaGo Zero 介绍

AlphaGo Zero 用具有参数 θ 的深度神经网络f_θ。该神经网络将棋盘s和前几步的棋盘作为输入，并输出移动概率和价值，$(p, v) = f_\theta(s)$。移动概率p表示选择每个移动的概率，$p_a = \Pr(a|s)$。价值 v 是标量评估，估计当前玩家从棋盘s获胜的概率。该神经网络将策略网络和价值网络合成单个架构。神经网络由许多残差块（residual blocks）组成。

AlphaGo Zero 中的神经网络通过一种新的增强学习算法从自我博弈中进行训练。在每个棋盘s用神经网络f_θ执行 MCTS。MCTS 输出每次移动的概率分布 π。这些产生的概率所带来的移动选择通常比神经网络f_θ执的原始移动概率p强得多。因此，MCTS 是强大的政策改进（policy improvement ）算子。把 MCTS 作为政策改进算子、把最终获胜方z作为价值的自我博弈是一个强大的政策评估（policy evaluation ）算子，就是评估当前神经网络和 MCTS 所产生的选择策略的好坏。我们的强化学习算法的主要思想是在策略迭代过程中重复使用这些算子，更新神经网络的参数以使移动概率和价值$(p, v) = f_\theta(s)$更紧密地匹配改进的搜索概率和赢家（π, z）。这些新参数用于下一次自我博弈迭代，使搜索更加强大。

蒙特卡罗树搜索（MCTS）使用神经网络f_θ来产生棋盘来模拟。搜索树中的每个边（棋盘，下一步位置）(s, a) 存储先验概率$P(s, a)$，访问次数$N(s, a)$ 和动作价值$Q(s, a)$。每盘自我博弈从根状态开始并迭代地选择使上置信区间$Q(s, a) + U(s, a)$最大化的移动（$A(s) = \arg\max_{a \text{ is legal}} Q + U$），其$U(s, a) \propto \frac{P(s,a)}{1+N(s,a)}$，直到遇到叶节点$s'$。该叶节点$s'$被神经网络扩展和评估一次以产生先验概率和价值评估$\left(P(s', *), V(s')\right) = f_\theta(s')$。在模拟中遍历的每个边$(s, a)$更新其访问次数$N(s, a)$，并且更新动作价值$Q(s, a) = \frac{1}{N(s,a)} \sum_{s'|s,a \to s'} V(s')$，其中$s'|s, a \to s'$表示模拟从$s$棋盘下一步走$a$最终移动到棋盘$s'$。

MCTS 可以被视为给定神经网络参数 θ 和初始棋盘s，计算推荐移动概率分布 $\pi = \alpha_\theta(s)$的算法。且 π 与每次移动a的访问次数成正比，设定 $\pi_a \propto N(s,a)^{1/\tau}$，其中 τ 是温度参数。也就是说当 τ 慢慢变小时，各种移动的概率的差距会越来越大，刚开始各种移动都能有所拓展，后来只拓展比较好的移动。

神经网络由自我博弈的强化学习算法训练，该算法使用 MCTS 来进行每次移动。首先，将神经网络初始化为随机权重 θ_0。在每次后续迭代$i \geq 1$时，生成自我博弈的棋局。在每个棋局的第t子，使用神经网络$f_{\theta_{i-1}}$迭代执行 MCTS 得到 $\pi_t = \alpha_{\theta_{i-1}}(s_t)$，并且通过对概率分布 π_t进行采样来选择下一步移动。当两个玩家都pass时(价值低于阈值)或者当游戏超过最大长度时，游戏终止，记为第 T 步。游戏结束时，得到最终棋局价值$r_T \in \{-1,+1\}$。第t步的数据存储为(s_t, π_t, r_t)，其中从第t步的玩家的角度来看，$z_t = \pm r_T$。 新的神经网络参数 θ_i 是从在自我博弈的最后一次迭代的所有数据中均匀采样的数据(s, π, r)训练的。调整神经网络$(p, v) = f_{\theta_i}(s)$使得预测价值v和自我博弈赢家z之间的误差最小化，并使得神经网络移动概率p与 MCTS 下移动概率 π 的相似性最大化。具体而言，参数 θ 通过梯度下降在分别对均方误差和交叉熵损失求和的损失函数l上进行调整，其中$l = (z-v)^2 - \pi^T \log p + c\|\theta\|^2$，c是控制 L2 权重正则化水平的参数（防止过拟合）。

2、自我博弈流程

AlphaGo Zero 的自我博弈分为三个步骤，这三个步骤都是异步并行执行的。神经网络参数 θ_i在最新的自我博弈的棋局下进行最优化；当前产生的 AlphaGo Zero 的决策函数 α_{θ_i} 被评估是否是最优；最优的决策函数 α_{θ^*}被用来生成新的棋局。

在参数的优化过程中，使用 momentum 和 learning rate annealing 的随机梯度下降，损失函数如上。（net_update(X_train, z_train, policy_train) in mian.py）

为了确保我们用当前神经网络生成的数据是良好的，我们将f_{θ_i}与当前存储的最优的神经网络f_{θ^*}进行对抗来评估当前网络的好坏。在文章中，他们对抗 400 局，每一步都用 MCTS 且通过神经网络搜索 1600 次，并且使温度参数 $\tau \to 0$。如果新的f_{θ_i}的胜率$> 55\%$(避免随机噪声影响)，我们用 α_{θ_i} 取代 α_{θ^*}来进行数据的生成。由于时间和其他一些原因，程序无法来得及实现，程序中我们假设当前产生的f_{θ_i}就是最优情况。

当前最优的 α_{θ^*}来进行数据的生成。在每次迭代中，α_{θ^*}每一步都通过 MCTS 和 UCT 算法模拟多次得到最优的下一步。在最开始的三十步中，设定温度参数 $\tau = 1$，这样使得每种下法与他们的概率分布和访问次数成正比，且能够保证遇到的下一步情况足够多。在剩下的几步中，使用很小的温度参数 $\tau \to 0$。但也为了确保各种情况的可能性，我们需要加入 Dirichlet 噪声到节点s的先验概率，$P(s,a) = (1-\epsilon)p_a + \epsilon\eta_a$，

$\eta \sim \mathrm{Dir}(0.03), \epsilon = 0.25$，不过这样搜索也可能会将不好的下法舍弃。为了方便，在main.py，函数 expand(node)中我们取$\eta \sim \mathrm{Uniform}(0,1)$。

3、AlphaGo Zero 的搜索算法

AlphaGo Zero 使用了异步策略和价值 MCTS（APV-MCTS）算法的一个更简单的变体。每个节点s在搜索树中包含边(s, a)对所有合法的下法$a \in \mathcal{A}(s)$，且每条边有一个统计的集合$\{N(s,a), W(s,a), Q(s,a), P(s,a)\}$，$N(s,a)$是访问次数，$W(s,a)$是下法带来价值的总和，$Q(s,a)$是是下法带来价值的平均值，$P(s,a)$是选择这条边的先验概率。在单独的搜索线程上并行执行多个模拟。 算法通过迭代三个阶段（选择，拓展和评估，备份），然后选择一个下法移动往下一层进行搜索。（class Node in main.py）

3.1 选择

每次模拟从根节点s_0开始，结束于叶子节点s_L。在中间的每一步$t < L$中，每一步的选择由搜索树重的统计量来决定。$a_t = \arg\max_a (Q(s_t, a) + U(s_t, a))$，其中$U(s,a)$是UCT(Upper Confidence bounds for Trees)的一个变体。$U(s,a) = c\,P(s,a)\frac{\sqrt{\sum N(s,b)}}{1+N(s,a)}$。C 是一个常量参数（可以控制 exploitation 和 exploration 权重）。这个公式的意思时，对每一个节点求一个值用于后面的选择，这个值有两部分组成，左边是这个节点的平均收益值（越高表示这个节点期望收益好，越值得选择，用于 exploitation），右边的变量是这个父节点的总访问次数除以子节点的访问次数（如果子节点访问次数越少则值越大，越值得选择，用于 exploration），这种搜索策略最初优先考虑具有高先验概率和低访问次数的行为，但是在时间上更喜欢具有高动作值的行为。因此使用这个公式是可以兼顾探索和利用的。(tree_policy(node) and best_child(node, is_exploration) in main.py)

3.2 拓展和评估

将叶子节点s_L加入神经网络的训练集，使得$(d_i(p), v) = f_\theta(d_i(s_L))$（根据l的定义拟合）其中$d_i$表示旋转和翻转(rotate(a, b, c) and reflect(a, b, c) in main.py). 在这之后拓展节点s_L而且每条边(s_L, a)初始化$\{N(s_L,a) = 0, W(s_L,a) = 0, Q(s_L,a) = 0, P(s_L,a) = p_a\}$，然后备份当前的价值到之前的父节点。(default_policy(node) in main.py)

3.3 备份

对于叶子节点s_L之前的父节点$s_t, t < L$，更新相应边的统计量，访问次数$N(s_t,a_t) = N(s_t,a_t) + 1$，价值总和$W(s_t,a_t) = W(s_t,a_t) + v$，价值的平均值$Q(s_t,a_t) = \frac{W(s_t,a_t)}{N(s_t,a_t)}$。（backup(node) in main.py）

总的来说，MCTS 的算法分为四步，第一步是 Select，就是在树中找到一个最好的值得探索的节点，一般策略是先选择未被探索的子节点，如果都探索过就选择 UCB 值最大的子节点。第二步是 Expand，就是在前面选中的子节点中走一步创建一个新的子节点，一般策略是随机自行一个操作并且这个操作不能与前面的子节点重复。第三步是 Simulate，就是在前面新 Expand 出来的节点开始模拟游戏，直到到达游戏结束状

态，这样可以收到到这个 expand 出来的节点的 reward 是多少。第四步是 Backup，就是把前面 expand 出来的节点 reward 反馈到前面所有父节点中，更新这些节点的价值和访问次数，方便后面计算 UCB 值。

3.4 下棋

在 MCTS 搜索结束之后，AlphaGo Zero 从根节点s_0选取一个下法，每个下法的概率与访问次数的指数成正比$\pi(a|s_0) = N(s_0,a)^{1/\tau} / \sum_b N(s_0,b)^{1/\tau}$，$\tau$是温度参数，用来控制拓展的程度。在随后的步骤中多次使用 MCTS：对应于选取下法的子节点成为新的根节点，此子项下面的子树及其所有统计信息都会保留，而树的其余部分将被丢弃。如果 AlphaGo Zero 的根值和最佳子值低于阈值v_{resign}，则 AlphaGo Zero 将停止。而在自己的程序中忽略了这一点(play(node, complete_or_fast) in main.py)

4、极小极大（Minimax）搜索

在搜索树中，每次轮到黑棋走时，走对黑棋最有利的；轮到白棋走时，走对黑棋最不利的。由于围棋是零和游戏，这就可以达到最优解。这是一个由底往上的过程：先把搜索树画到我们可以承受的深度，然后逐层往上取最大值或最小值回溯，就可以看到双方的正解（如果胜率评估是准确的）。而实际编程的时候，是往下不断生长节点，然后动态更新每个父节点的胜率值。

值得注意的是，在实际对局中，胜率评估会有不准确的地方，这就会导致“地平线效应”，即由于电脑思考的深度不够，且胜率评估不够准确，因此没有看见正解。

Minimax 搜索还有许多后续发展，如 Alpha-beta 剪枝，以及更进一步的 Null Window /NegaScout / MTD(f) 等等。可惜这些方法更适合象棋等棋类，对于围棋的意义不大（除非已经接近终局）。

蒙特卡洛树搜索和蒙特卡洛方法的区别在于：蒙特卡洛方法存在偏差（Bias），当然，也有方差（Variance）；蒙特卡洛树搜索在一段时间模拟后，会给出正确的答案，它的结果不存在偏差（Bias），只存在方差（Variance）。但是，对于复杂的局面，它仍然有可能长期陷入陷阱，直到很久之后才开始收敛到正确答案。

如果想把 Minimax 搜索运用到围棋上，立刻会遇到两个大问题：

1、搜索树太广。棋盘太大了，每一方在每一步都有很多着法可选。

2、很难评估胜率。除非把搜索树走到终局，这意味着要走够三百多步（因为对于电脑来说，甚至很难判断何时才是双方都同意的终局，所以只能傻傻地填子，一直到双方都真的没地方可以走为止）。简单地说，搜索树也需要特别深。

蒙特卡洛树搜索的意义在于部分解决了上述两个问题：

1、它可以给出一个局面评估，虽然不准，但比没有强。这就部分解决了第二个问题。

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。这就部分解决了第一个问题。

最后，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）。在 main.py 程序中我们可以看到 backup(node, reward)和 best_child(node, is_exploration)中使用了极小极大（Minimax）搜索。

5、浅谈博弈论

蒙特卡罗树搜索是一种基于树数据结构、能权衡探索与利用、在搜索空间巨大仍然比较有效的的搜索算法。MCTS 的使用场景需要是搜索空间巨大，因为搜索空间如果在计算能力以内，其实是没必要用 MCTS 的，而真正要应用上还需要有其他的假设。涉及到博弈论的概念，我们要求的条件是 zero-sum、fully information、determinism、sequential、discrete，也就是说这个场景必须是能分出输赢（不能同时赢）、游戏的信息是完全公开的（不像打牌可以隐藏自己的手牌）、确定性的（每一个操作结果没有随机因素）、顺序的（操作都是按顺序执行的）、离散的（没有操作是一种连续值），除了博弈论我们要求场景是类似多臂老虎机(multi-armed bandits)的黑盒环境，不能通过求导或者凸优化方法找到最优解，否则使用 MCTS 也是没有意义。

MCTS 是一种找到逼近纳什均衡的搜索策略。那么逼近纳什均衡点的策略还有很多，例如前面提到的 Minmax 算法，还有适用于非信息对称游戏的 CFR（Counterfactual Regret）算法。像德州扑克我们就会考虑用 CFR 而不是 MCTS，因为 MCTS 要求是一种 Combinatorial Game 而不能是信息不对称的。

在博弈论里，信息对称（Perfect information / Fully information）是指游戏的所有信息和状态都是所有玩家都可以观察到的，因此双方的游戏策略只需要关注共同的状态即可，而信息不对称（Imperfect information / Partial information）就是玩家拥有只有自己可以观察到的状态，这样游戏决策时也需要考虑更多的因素。

还有一个概念就是零和（zero-sum），就是所有玩家的收益之和为零，如果我赢就是你输，没有双赢或者双输，因此游戏决策时不需要考虑合作等因素。那么博弈论中，我们把 zero-sum、perfect information、deterministic、discrete、sequential 的游戏统称为 Combinatorial Game，例如围棋、象棋等，而 MCTS 只能解决 Combinatorial Game 的问题。

6、神经网络结构

神经网络的输入是包含 17 个二进制特征平面的 19×19×17 向量。 8 个特征平面 X_t 由表示当前玩家棋子存在的二进制值组成（如果位置i在时间步t包含玩家颜色的棋子，则$X_t^i = 1$；如果位置i含对手棋子，或者$t < 0$，$X_t^i = 0$）。另外 8 个特征平面Y_t代表对手棋子的相应特征。最后一个特征平面C表示要下的棋子的颜色，如果要下黑色，则其常量值为 1 或者如果要下白色则为 0。这些平面连接在一起以给出输入向量$s_t = [X_t, Y_t, X_{t-1}, Y_{t-1}, \ldots, X_{t-7}, Y_{t-7}, C]$。历史棋盘$X_t, Y_t$是必要的，因为围棋不能完全从当前的棋盘状态中观察到，因为有禁止重复的规则；类似地，颜色特征C是必要的，因为贴目是不可观察的。

输入向量s_t由残差塔处理，该残余塔由单个卷积块后跟 19 或 39 个残差块组成。

卷积块应用以下模块：

1.内核大小为 3×3 的 256 个过滤器的卷积，步长为 1
2.批量归一化
3.整流器非线性

每个残余块按顺序将以下模块应用于其输入：

1.内核大小为 3×3 的 256 个过滤器的卷积，步长为 1
2.批量归一化
3.整流器非线性

4.内核大小为 3×3 的 256 个过滤器的卷积，步长为 1
5.批量归一化
6.将输入添加到向量块中
7.整流器非线性

残差塔的输出被传递到两个单独的“头”中，分别用于计算策略函数和价值函数。策略函数应用以下模块：
1.两个内核大小为 1×1 的过滤器的卷积，步长为 1
2.批量标准化
3.整流器非线性
4.完全连接的线性层，输出大小为$19^2 + 1 = 362$的向量，对应于所有位置的概率和 pass

价值函数应用以下模块：
1. 1 个内核大小为 1×1 的过滤器与步长 1 的卷积
2.批量归一化
3.整流器非线性
4.完全连接的线性层到大小为 256 的隐藏层
5.整流器非线性
6.完全连接的线性层到标量
7. tanh 非线性输出[-1,1]范围内的标量

(network(X, _weights, _biases, is_training) and residual_block(f0, _weights, _biases, is_training, number, X_residual) in main.py)但在程序中仍有不正确的地方需要进一步改进。

7、PRP 工作总结

前期我们以 CS231n 的课程为前期内容，了解了深度学习在图像分类和图像处理的原理，掌握 CNN、RNN、LSTM、GANs，并且完成了代码的实现，积累了对网络结构调整、应用的经验。对于一个网络中的优化，可以通过 pool 将低维度，normalization 解决梯度爆炸或消失，dropout 解决非线性和过拟合，改变梯度下降方法，卷积核很小的值变为 0，小数适当调整成整数降低复杂度，调整 learning rate、mini batch、epochs、hidden layers 层数和数量……不过现在成熟的网络很多了，如 U-Net、ResNet 等，所以我们一般很少去修改网络的结构。个人对深度学习的理解，简单来说就是用一些线性和非线性连续函数去拟合训练集，我们可以得到我们认为“最好”（全局最优）的函数。

随后是强化学习的内容，学习了 Reinforcement Learning 一书和 PPT，对强化学习有一定的了解。强化学习主要通过对各个状态进行估计，得到最优的决策，但由于状态空间维数过大，我们从已经得到的一些 episodes 去计算各个状态的最优决策，这就有 exploration 和 exploitation 的权衡。当然也有通过对价值函数或决策的估计（拟合）得到结果。在估计价值函数或决策的过程中，我们可以用类似神经网络的方法，这样得到的结果可以从我们探索到的状态空间延拓到全部的状态空间，这对状态空间巨大的问题非常有用。个人对强化学习的理解，就是我们找到一些 episodes，对于他们价值的高低，我们使我们的决策避免导致坏的，去趋近好的结果。

后期我阅读了 Deepmind 的 Mastering the Game of Go without Human Knowledge。根据文章中提到的内容，再现了其中的算法和框架。由于时间和不熟练等问题对于其中的一些细节并没有还原的很好，但总体来说大体的框架成功还原且具有一定的鲁棒性。且 AlphaGo Zero 中的算法和思想可以延拓到很多方面，比如各种棋类、自动驾驶、动态定价、推荐算法等等。

8、参考文献

[1] David Silver, *Mastering the game of Go without human knowledge*, Nature volume550, pages354–359 (19 October 2017)**, DOI:** https://doi.org/10.1038/nature24270

[2] Cameron B. Browne, *A Survey of Monte Carlo Tree Search Methods*, IEEE Transactions on Computational Intelligence and AI in Games (Volume: 4 , Issue: 1 , March 2012), DOI: 10.1109/TCIAIG.2012.2186810

[3] Christopher Clark, *Training deep convolutional neural networks to play go*, ICML'15 Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37 Pages 1766-1774