AlphaGo Zero

Chen Gong

07 February 2020

目录

1 解决围棋问题的主要思想

人工智能的定义为：在环境中，做出行为 (Action)，以达到目的的个体 (Agent)。在围棋中这个问题中，棋盘，落子，吃子，围棋规则就组成了这个围棋视角的平行世界，这就是 AlphaGo Zero 的环境。而每一步下什么棋，落子在哪里就是它的动作。而它的目的很简单，那就是赢棋。

如果，我们可以遍历所有的情况，那么我们肯定可以战胜一切对手。而在围棋中，有两个方面问题很不好解决，那就是广度和深度的问题。一局棋可能要下 150 步 (深度)，每一步可能有 250 个可能的落子点 (广度)。所以，遍历是不可能的。

而主体思想就是，解决这两个问题。

深度问题，就是说我下了这一步棋，到了最后，我究竟是赢还是输，也就是胜率。如果靠蛮力来算，我需要推演到这个棋局的最后一步，才知道，我究竟是输还是赢。而 AlphaGo 中，通过训练神经网络得到了一个胜率估算的网络，输入棋局输出就直接得出胜率，从而砍掉了后面的深度。

广度问题，在最开始的 AlphaGo 中依靠人类棋手的经验来模拟落子点，使用了 MCTS 算法来对可能落子的地方进行深入分析，大大缩小了 250 个可能落子点的广度。但是，一局棋的落点大约有 150 个是类似的，一步之间只有一个子的位置不一样，其实差距很小的，最开始的 AlphaGo 是背下来所有的过程，从而造成了过拟合的现象。这也是 AlphaGo 在第四局输给李世石的主要原因。而解决方法是在所有的棋局中只抽取一个场面，但是问题又来了，并没有这么多棋局，而 AlphaGo Zero 通过自我博弈的方法产生了很多棋局，从而解决了这个问题。

而上述的胜率策略，下棋策略和 MCTS 搜索三个部分结合起来，就诞生了超越人类智能的 AlphaGo 系列的产品。实际上胜率策略，下棋策略和 MCTS 搜索，以及辅助性简化版下棋策略，强化学习策略每一个看都很简单，组合起来却发挥了强大的威力，共同构建了 AlphaGo 超人工智能，

这节我们主要介绍 AlphaGo 的第二代产品，AlphaGo Zero。AlphaGo Zero 的核心特点可以表述为：

1. 单个神经网络收集棋局特征，在末端分支输出策略和棋局终止时的奖励；
2. 自我对弈的强化学习的蒙特卡罗树搜索 (MCTS)；
3. 探索与利用的结合 (Q+U 置信区间上限)。

2 AlphaGo Zero 概述 (AlphaGo Zero Introduction)

针对描述当前棋盘的一个状态 (位置)s，执行一个由神经网络 f_θ 指导的 MCTS 搜索，MCTS 搜索输出每一步行为 (在某个位置落子) 的概率。MCTS 搜索给出的概率通常会选择那些比神由经网络 $f_\theta(s)$ 给出的更强大的动作。从这个角度看，MCTS 搜索可以被认为是一个强大的策略优化工具。通过搜索来进行自我博弈——使用改善后的基于 MCTS 的策略来指导行为选择。然后使用棋局结果 (哪一方获胜，用-1 和 1 分别表示白方和黑方获胜) 来作为标签数据——可以被认为是一个强大的策略评估工具。

2.1 AlphaGo Zero 主体思想

Alpha Zero 算法的主体思想就是在策略迭代过程中反复使用上述的两个工具：神经网络的参数得以更新，这样就可以使得神经网络的输出 $(p, v) = f_\theta(s)$：移动概率和获胜奖励更接近，经过了改善的搜索得到的概率和通过自我博弈得到的棋局结果，后者用 (π, z) 表示。得到的新参数可以在下一次自我对弈的迭代过程中让搜索变得更加强大。如下图所示：

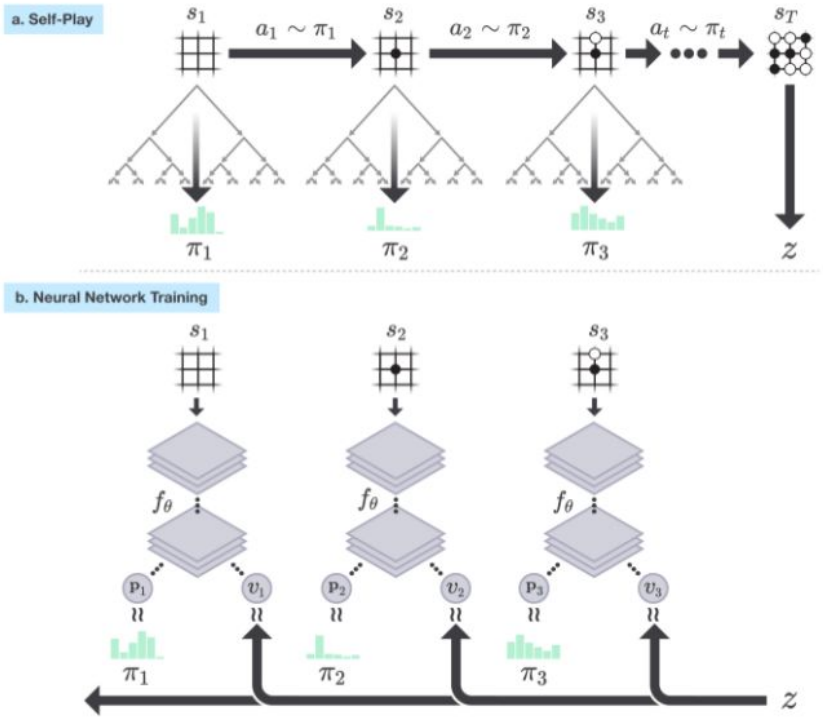

图 1: AlphaGo Zero 主体思想流程图

2.2 Alpha Zero 主要流程

a. 程序自我对弈完成一个棋局产生一个状态序列 $s_1, s_2, \cdots, s_T$，在 T 时刻棋局结束，产生了获胜方，用 z 来表示。在其中的每一个时刻 t，棋局状态用 s_t 来表示，会在神经网络 f_θ 的引导下执行一次 MCTS 搜索 α_θ，通过 MCTS 搜索计算得到的概率 $a_t \sim \pi_t$ 确定如何动作 (在何处落子)。

b. 展示了 Alpha Zero 里神经网络的训练过程。神经网络的输入是某时刻 t 的棋局状态 s_t 外加一些历史和额外信息，输出的是一个行为概率向量 p_t 和一个标量 v_t，前者表示的在当前棋局状态下采取每种可能落子方式的概率，后者则表示当前棋局状态下当前棋手 (还是黑方) 最终是获胜还是落败 (分别用 1 和-1 来表示)。

神经网络的训练目的就是要尽可能的缩小两方面的差距：1. 搜索得到的概率向量 π_t 和网络输出的概率向量 p_t 的差距；2. 网络预测的当前棋手的最终结果 v_t 和最终游戏赢家 (1,-1 表示) 的差距。网络训练得到的新参数会被用来知道下一轮迭代中自我对弈时的 MCTS 搜索。

2.3 Alpha Zero 中的蒙特卡洛树搜索

Alpha Zero 中的蒙特卡洛树搜索如下图所示：

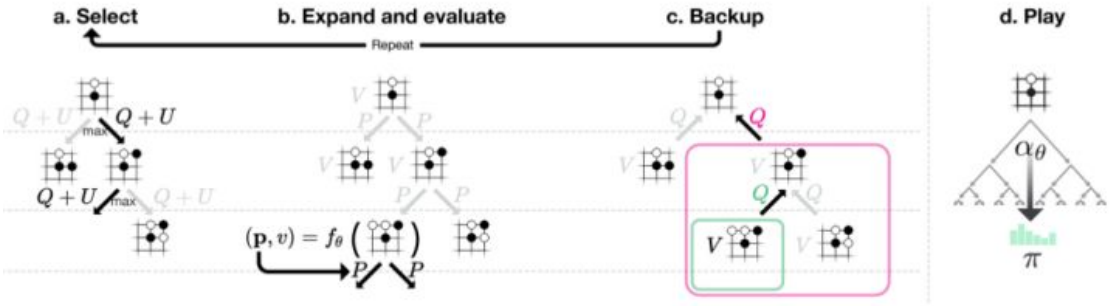

图 2: Alpha Zero 中的蒙特卡洛树搜索

a. 每一次模拟过程会通过选择那些拥有最大 $Q+U$ 值的边 (表示基于某状态下的某一行为) 来遍历树结构。Q 指的是行为价值，U 指的是一个置信区间的上限，可以根据先验概率 P 和该边 (行为) 被遍历的次数计算得出。

b. 当遍历到树中的叶节点时，将使用神经网络来设定该叶节点的 Q 值以及该叶节点下各种可能行为的概率，即：$(P(s,\cdot),V(s)) = f_\theta(s)$。这些 P 值存储在节点 s 连接到子节点的边上。

c. 行为价值 Q 值会得到更新，以便可以追踪到当前行为下方的子树里的所有节点 V 值的平均。

d. 当搜索过程结束时，会产生一个搜索概率结果 π，该结果与 $N^{1/\tau}$ 成正比，其中 N 表示的是从根状态开始的每一个行为被访问 (模拟) 的次数，τ 是一个控制火候 (温度) 的一个参数。

神经网络通过使用 MCTS 搜索的自我对弈强化学习来进行训练。一开始神经网络的参数被随机设置成 θ_0，在之后的每一次迭代中 $i \geq 1$，会通过自我对弈产生许多完整的棋局，在其中每一个完整棋局的一个时间步 T 时，都会用上一个神经网络参数来产生搜索策略 $\pi_t = \alpha_{i-1}(s_t)$，并且利用这个策略的采样产生自我对弈时的行为。

2.4 游戏的终止条件

发生下列任意情况之一，游戏终止于时间 T：

1. 双方都 Pass；
2. 搜索 value 降低至一个被 resignation 的阈值；
3. 游戏对弈达到设定的最大步数。

游戏结束后，会给出一个最终奖励 $r_T \in \{-1,+1\}$，每一个时间步 T 的数据以 (s_t,π_t,z_t) 的形式保存，其中 $z_t = \pm r_T$ 是从 T 时刻玩家的立场得到的胜利者的奖励。自我博弈中的最后几次迭代过程中产生的数据 (s,π,z) 将会被等概率的被选中来训练神经网络。神经网络的损失函数有下式确定：

$$l = (z-v)^2 - \pi^T \log P + c||\theta||^2 \tag{1}$$

其中，c 是控制参数，$L2$ 正则项的一个系数。

3 方法细节

1. 训练神经网络使用了围棋的特点：旋转和镜像棋局时结果不变，这是一个训练神经网络的数据扩增的办法。同样棋子的黑白色也是可以互换进行数据扩增的，实现这个效果的方式是从当前时刻玩家的立场来表示棋局。

2. 神经网络使用的是目前最强大的残差网络，具体的参数设置也是相应的。

3. MCTS 搜索参数通过高斯处理优化来选择。

4. 自我对弈训练路线分成三个组件：神经网络的参数 θ_i 持续得到优化；玩家 α_{θ_i} 持续得到评估；当前最强玩家 α_{θ_*} 被用来生成新的自我对弈数据。

5. 学习率是逐渐衰减的 annealed，动量参数 0.9，交叉熵和均方差损失的权重相等，L_2 正则化系数为 $c = 10^{-4}$。

6. 优化过程每 1000 步会产生一个 checkpoint，将有一个评估算子 (evaluator) 来评估，这个 checkpoint 也可能会被用来产生下一个 batch 的自我对弈数据。

7. Evalutor：checkpoint 将用来和目前为止最好的神经网络 f_{θ_*}，来进行比较。使用 400 个完整游戏来进行评估，每一次移动要进行 1600 次 MCTS 模拟，使用的 infinitesimal temperature $\tau \longrightarrow 0$。如果 checkpoint 对比之前的最好网络能赢得 55% 的胜率，checkpoint 将称为当前的最佳玩家 α_θ，后续的自我对弈将使用这个新的 Best Player。

8. 自我对弈：在每一次迭代过程中，当前最佳玩家 α_{θ_*}。将进行 25000 次完整的自我对弈，每一次移动将使用 1600 次 MCTS 搜索模拟。每一次完整的自我对弈的前 30 步，参数 temperature 被设定为 $1(\tau = 1)$，这是鼓励探索的设置。游戏剩下的步数，该参数会逐渐降低到 $0(\tau \longrightarrow 0)$。在搜索树的根节点 s_0，会通过加入 Dirichlet Noise 来鼓励额外的探索：

$$P(s,a) = (1-\epsilon)p_a + \epsilon\eta_a \tag{2}$$

式中 $\eta \sim Dir(0.03)$，$\epsilon = 0.25$。这种噪声设置基本可以保证所有的移动的可能性都被尝试到。

9. 自我对弈：为了节省计算资源，那些明星会输的游戏会被裁剪 (割弃) 掉，这个阈值 v_{resign} 是通过保存一定 (5%) 的假阳性率 (指的是如果 Alpha Go 没有割弃的话，该局游戏就有可能会赢)。为了测量这个假阳性率，会在 10% 的自我对弈棋局中禁用 “裁剪”，这些棋局将被一直对弈到棋局终止。

10. 监督学习：为了比较。单独通过监督学习一个与 Alpha Zero 网络架构相同的神经网络 θ_{SL}，训练数据 (s,π,z) 来源于 KGS 数据集，同时设定针对人类专家的移动 a 的概率为 1：$\pi_a = 1$。训练参数也与之前的网络架构相同，但是均方差那个损失部分被乘了一个 0.01 的系数。

11. 搜索算法：搜索数中的每一个节点 s 会为每一个合理的行为 $a \in A(s)$ 设置一条边 (Edge)，在这条边中，会存储下面几个统计数据：$\{N(s,a), W(s,a), Q(s,a), P(s,a)\}$。式中，$N(s,a)$ 是该边被遍历的次数；$W(s,a)$ 是总的行为价值；$Q(s,a)$ 是平均行为价值；$P(s,a)$ 是选择该边代表的行为的先验概率。算法通过不停的迭代三个阶段 (图二中的 a-c) 来选择实际要采取的行为 (图二中的 d)。

4 搜索阶段算法

4.1 搜索算法阶段 1(图二——a)——选择 Select

每一次模拟的第一个阶段起自搜索树的根节点 s_0，在第 L 个时间步结束与搜索树的叶节点 s_L。对于其中的任意时间 $t < L$，根据搜索树内的统计数据来决定选择哪一个模拟行为：

$$a_t = \arg\max_a (Q(s_t, a) + U(s_t, a)) \tag{3}$$

其中：

$$U(s, a) = c_{puct} P(s, a) \frac{\sqrt{\sum_b N(s, b)}}{1 + N(s, a)} \tag{4}$$

c_{puct} 是决定探索程度的一个系数。

4.2 搜索算法阶段 2(图二——b)——扩展和评估蒙特卡罗搜索树 Expand & Evaluate

叶节点 s_L 将会等待来自神经网络的评估 $(d_i(p), v) = f_\theta(d_i(s_L))$，其中 d_i 是一个 dihedral reflection 或者 rotation，$i \in [1:8]$。这些等待评估状态会被送入到一个队列中，在神经网络评估队列里的状态时 (mini_batch_size=8)，搜索将被暂时锁定。当神经网络得到结果后，该叶节点会被展开，同时每一条可能的边 (s_L, a) 会以下面的数据进行初始化：

$$\{N(s_L, a) = 0, W(s_L, a) = 0, Q(s_L, a) = 0, P(s_L, a) = P_a\} \tag{5}$$

同时来自神经网络的对该节点的价值估计也会影响路径中每一个节点的统计数据 W，随后进行回溯过程。

4.3 搜索算法阶段 3(图二——c)——回溯 Backup

对于 $t \leq L$，每一个边的统计结果都将被更新。首先：$N(s_t, a_t) = N(s_t, a_t) + 1$；其次：

$$W(s_t, a_t) = W(s_t, a_t) + v, \qquad Q(s_t, a_t) = \frac{W(s_t, a_t)}{N(s_t, a_t)} \tag{6}$$

4.4 搜索算法阶段 4(图二——d)——产生实际行为 Play

路径中所有节点统计得到更新后，在根节点 s_0 处，Alpha Zero 将要根据最新的统计数据来产生一个实际的行为：

$$\pi(a|s_0) = \frac{N(s_0, a)^{1/\tau}}{\sum_b N(s_0, b)^{1/\tau}} \tag{7}$$

式中的 τ 是一个控制搜索程度的参数。在随后的时间步 (time-steps) 中，这个搜索树会继续使用，对应于实际所采取的行为的子节点将变成根节点，该子节点下的子树的统计数据将会被保留，而这颗树的其余部分将会丢弃 Discarded。Alpha Zero 将放弃搜索某子树，如果该子树的根节点和最佳价值子节点的价值低于某一个阈值 v_{resign}。

5 神经网络架构

由残差模块构成的 CNN，输入为 19*19*17。输入是 17 个特征，这 17 个特征是既往 8 个回合的 16 个特征以及一个当前玩家信息特征:

$$s_t = [X_t, Y_t, X_{t-1}, Y_{t-1}, \cdots, X_{t-7}, Y_{t-7}, C] \tag{8}$$

其中 X_t 内包含的是当前棋手的数据：加入当前棋手执黑棋，那么此时棋盘上所有黑棋对应的位置取值 1，白棋和空白位置取值 0。类似的 Y_t 反映的就是白棋信息，当前棋盘上所有白棋的位置取值 1，黑棋和空白处取值 0. C 内的所有 (19x19) 个数据要么都是 1，要么都是 0。如果此时是黑棋要落子则取 1，白棋落子则取 0. 网络的其余架构参考论文原文。

网络的共同部分多数是用的 3x3 的卷积核 (Stride=1)，256 个特征数，后接 Batch Normalization 和 Relu 单元。

对于策略端：输出特征数为 19x19+1，分别代表在棋盘上所有可能位置落子的可能性以及 Pass 的可能性。

对于价值端：全连接一个 256 个特征的隐藏层，最后以 tanh 的激活方式输出 [-1,1] 之间的值。

6 小结 (Conclusion)

AlphaGo Zero 完全不依赖人类棋手的经验，经过 3 天的训练，AlphaGo Zero 就击败了 AlphaGo Master。AlphaGo Zero 最重要的价值在于，它不仅仅可以解决围棋问题，它可以在不需要知识预设的情况下，解决一切棋类的问题，经过几个小时的训练就击败了最强的国际象棋程序 Stockfish。应用场景十分广泛。

在 AlphaGo Lee 版本中，有两个神经网络，一个是策略网络，这是一个有监督学习，它利用了大量的人类高手的对弈棋局来评估下一步的可能性；另一个是价值网络，用来评价当前局面的评分。AlphaGo 的精髓在于 MCTS 算法，因为 Agent 选择的不是现在看上去最好的过程，而是它考虑得最多的过程。它考虑到的次数越多，说明下这一步最靠谱，最顺，这就完美的诠释了“整体远大于部分”的思想。

而在 AlphaGo Zero 版本中，除了围棋规则以外没有任何背景知识，并且只使用了一个神经网络。这个神经网络以 19x19 棋盘为输入，以下一步各下法的概率和胜率作为输出。

AlphaGo Zero 的核心思想是：MCTS 算法生成的对弈可以作为神经网络的训练数据。深度学习中最重要的三个部分是什么？输入，输出，损失！随着 MCTS 的不断执行，下法概率和胜率会趋于稳定，而深度学习网络最重要的也就是下法概率和胜率，而与真实结果的差就是损失。随着训练的不断进行，网络对于胜率的下法概率的估算将会越来越准确。这意味着，即使某一个下法 Agent 没有模拟过，但是通过神经网络依然可以达到蒙特卡罗的模拟效果！通俗的讲就是，这手棋我没下过，但是凭着我在神经网络中训练出的“棋感”，我可以估算出这么走的胜率是多少！

所以，AlphaGo Zero 只要应用深度网络计算出的下法概率，胜率和 MCTS 的置信区间就可以进行选点。