第十八章 AlphaGo 与蒙特卡洛树搜索

之前章节介绍的强化学习方法都是无模型的强化学习 (Model-Free)，包括价值学习 (Value-Based) 和策略学习 (Policy-Based)。本章介绍的蒙特卡洛树搜索 (Monte Carlo Tree Search，缩写 MCTS) 是一种基于模型的强化学习方法 (Model-Based)。MCTS 比价值学习和策略学习更难理解，所以本章结合 AlphaGo 讲解 MCTS。

AlphaGo 的字面意思是"围棋王"，俗称"阿尔法狗"，它是世界上第一个打败人类围棋冠军的 AI。在 2015 年 10 月，AlphaGo 以 5 : 0 战胜欧洲围棋冠军、职业二段选手樊麾。在 2016 年 3 月，AlphaGo 以 4 : 1 战胜世界冠军李世石。2017 年新版的 AlphaGo Zero 更胜一筹，以 100 : 0 战胜 AlphaGo。

AlphaGo 依靠 MCTS 做决策，而决策的过程中需要策略网络和价值网络的辅助。第 18.1 节用强化学习的语言描述围棋的状态和动作，并且构造策略网络和价值网络。第 18.2 节详细讲解 MCTS 的决策过程。第 18.3 节讲解 AlphaGo 2016 版与 AlphaGo Zero 是如何训练策略网络和价值网络的。

18.1 动作、状态、策略网络、价值网络

围棋的棋盘是 19×19 的网格，可以在两条线交叉的地方放置棋子，一共有 361 个可以放置棋子的位置。两个的玩家一方用黑色棋子，另一方用白色棋子，两方交替往棋盘上放置棋子。棋盘上有 361 个可以放置棋子的位置，因此动作空间是 $\mathcal{A} = \{1, \cdots, 361\}$。比如动作 $a = 123$ 的意思是在第 123 号位置上放棋子。

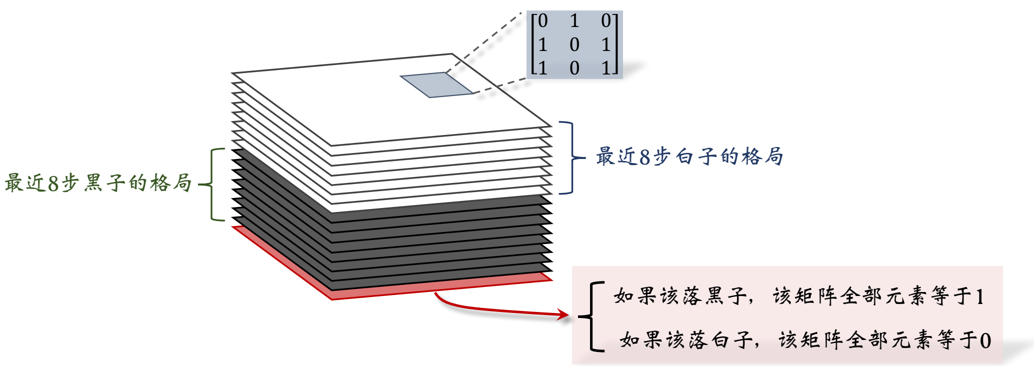

图 18.1: 状态可以表示为 $19 \times 19 \times 17$ 的张量。

AlphaGo 2016 版本使用 $19 \times 19 \times 48$ 的张量 (Tensor) 表示一个状态。AlphaGo Zero 使用 $19 \times 19 \times 17$ 的张量表示一个状态。本书只解释后者；见图 18.1。下面解释 $19 \times 19 \times 17$ 的状态张量的意义。

- 张量每个切片 (Slice) 是 19×19 的矩阵，对应 19×19 的棋盘。一个 19×19 的矩阵可以表示棋盘上所有黑子的位置。如果一个位置上有黑子，矩阵对应的元素就是

1，否则就是 0。同样的道理，用一个 19×19 的矩阵来表示当前棋盘上所有白子的位置。

- 张量一共有 17 个这样的矩阵；17 是这样得来的。记录现在和之前 7 步棋盘上黑子的位置，需要 8 个矩阵。同理，还需要 8 个矩阵记录白子的位置。还需要一个矩阵表示该哪一方下棋；如果该下黑子，那么该矩阵元素全部等于 1；如果该下白子，那么该矩阵的元素全都等于 0。

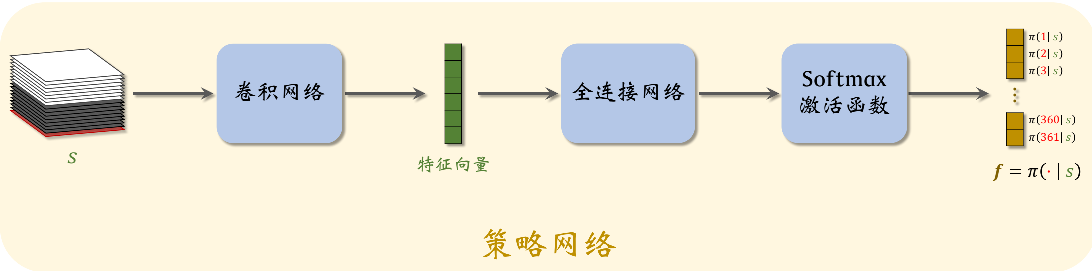

图 18.2: 策略网络的示意图。

策略网络 $\pi(a|s;\boldsymbol{\theta})$ 的结构如图 18.2 所示。策略网络的输入是 $19 \times 19 \times 17$ 的状态 s。策略网络的输出是 361 维的向量 $\boldsymbol{f}$，它的每个元素对应一个动作（即在棋盘上一个位置放棋子）。向量 $\boldsymbol{f}$ 所有元素都是正数，而且相加等于 1。

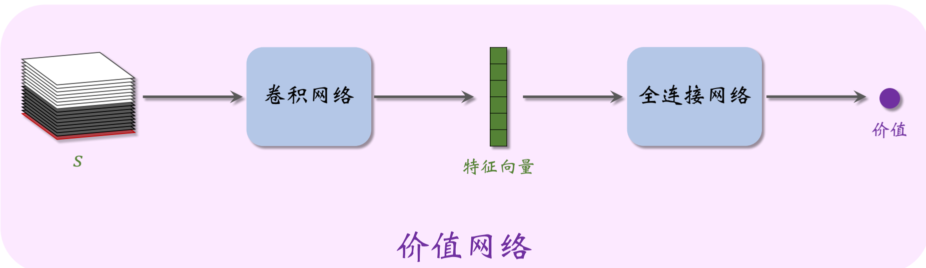

图 18.3: 价值网络的示意图。

AlphaGo 还有一个价值网络 $v(s;\boldsymbol{w})$，它是对状态价值函数 $V_\pi(s)$ 的近似。价值网络的结构如图 18.3 所示。价值网络的输入是 $19 \times 19 \times 17$ 的状态 s。价值网络的输出是一个实数，它的大小评价当前状态 s 的好坏

策略网络和价值的输入相同，都是状态 s。它们都用多个卷积层把 s 映射为特征向量。因此可以让策略网络和价值网络共用卷积层。训练策略网络和价值网络的方法在第 18.3 节解释。

18.2 蒙特卡洛树搜索 (MCTS)

假设此时已经训练好了策略网络 $\pi(a|s;\boldsymbol{\theta})$ 和价值网络 $v(s;\boldsymbol{w})$。AlphaGo 真正跟人下棋的时候，做决策的不是策略网络或者价值网络，而是用蒙特卡洛树搜索 (Monte Carlo Tree Search)，缩写 MCTS。MCTS 不需要训练，可以直接做决策。训练策略网络和价值网络的目的是辅助 MCTS。本节中假设策略网络和价值网络已经训练好，可以直接用；下一节再具体讲解策略网络和价值网络的训练。

18.2.1 MCTS 的基本思想

思考一个问题：人类玩家是怎么下围棋、象棋、五子棋的？人类玩家通常都会向前看几步；越是高手，看得越远。假如现在该我放棋子了，我应该思考这样的问题：当前有几个貌似可行的走法，假如我的动作是 $a_t = 234$，对手会怎么走呢？假如接下来对手把棋子放在 $a_t' = 30$ 的位置上，那我下一步的动作 a_{t+1} 应该是什么呢？做当前决策之前，我需要在大脑里做这样的预判，确保几步以后我很可能会占优势。如果我只根据当前格局做判断，不往前看，我肯定赢不了高手。同理，AI 下棋也应该向前看，应该枚举未来可能发生的情况，从而判断当前执行什么动作的胜算最大；这样做远好于用策略网络计算一个动作。

MCTS 的基本原理就是向前看，模拟未来可能发生的情况，从而找出当前最优的动作。AlphaGo 每走一步棋，都要用 MCTS 做成千上万次模拟，从而判断出哪个动作的胜算最大。做模拟的基本思想如下。假设当前有三种看起来很好的动作。每次模拟的时候从三种动作中选出一种，然后将一局游戏进行到底，从而知晓胜负。(只是计算机做模拟而已，不是真的跟对手下完一局。) 重复成千上万次模拟，统计一下每种动作的胜负频率，发现三种动作胜率分别是 48%、56%、52%。那么 AlphaGo 应当执行第二种动作，因为它的胜算最大。以上只是 MCTS 的基本想法，实际做起来有很多难点需要解决。

18.2.2 MCTS 的四个步骤

MCTC 的每一次模拟选出一个动作 a，执行这个动作，然后把一局游戏进行到底，用胜负来评价这个动作的好坏。MCTC 的每一次模拟分为四个步骤：选择 (Selection)、扩展 (Expansion)、求值 (Evaluation)、回溯 (Backup)。

第一步——选择 (Selection)： 观测棋盘上当前的格局，找出所有空位，然后判断其中哪些位置符合围棋规则；每个符合规则的位置对应一个可行的动作。每一步至少有几十、甚至上百个可行的动作；假如挨个搜索和评估所有可行动作，计算量会大到无法承受。虽然有几十、上百个可行动作，好在只有少数几个动作有较高的胜算。第一步——选择——的目的就是找出胜算较高的动作，只搜索这些好的动作，忽略掉其他的动作。

如何判断动作 a 的好坏呢？有两个指标：第一，动作 a 的胜率；第二，策略网络给

动作 a 的评分（概率值）。用下面这个分值评价 a 的好坏：

$$\text{score}(a) \;\triangleq\; Q(a) + \frac{\eta}{1+N(a)} \cdot \pi(a|s;\boldsymbol{\theta}). \tag{18.1}$$

此处的 η 是个需要调的超参数。公式中 $N(a)$、$Q(a)$ 的定义如下：

- $N(a)$ 是动作 a 已经被访问过的次数。初始的时候，对于所有的 a，令 $N(a) \leftarrow 0$。动作 a 每被选中一次，$N(a) \leftarrow N(a)+1$。
- $Q(a)$ 是之前 $N(a)$ 次模拟算出来的动作价值，主要由胜率和价值函数决定。$Q(a)$ 的初始值是 0；动作 a 每被选中一次，就会更新一次 $Q(a)$；后面会详解。

可以这样理解公式 (18.1)：

- 如果动作 a 还没被选中过，那么 $Q(a)$ 和 $N(a)$ 都等于零，那么

  $$\text{score}(a) \;=\; \eta \cdot \pi(a|s;\boldsymbol{\theta}),$$

  也就是说完全由策略网络评价动作 a 的好坏。
- 如果动作 a 被选中过很多次，那么 $N(a)$ 就很大，导致策略网络在 score(a) 中的权重降低；此时主要基于 $Q(a)$ 判断 a 的好坏，而策略网络已经无关紧要。
- 系数 $\frac{1}{1+N(a)}$ 的另一个作用是鼓励探索，也就是让被选中次数少的动作有更多的机会被选中。假如两个动作有相近的 Q 分数和 π 分数，那么被选中次数少的动作的 score 会更高。

MCTS 根据公式 (18.1) 算出所有动作的分数 score(a)，$\forall a$。MCTS 选择分数最高的动作。图 18.4 的例子中有 3 个可行动作，分数分别为 0.4、0.3、0.5。第三个动作分数最高，会被选中。这一轮模拟会执行这个动作（只是模拟而已，不是 AlphaGo 真的走一步棋）。

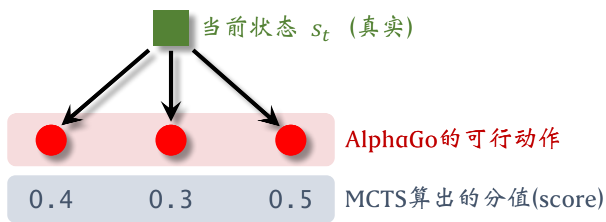

图 18.4: 假设有 3 个可行动作，根据公式 (18.1) 算出它们的分数。

第二步——扩展 (Expansion)： 把第一步选中的动作记作 a_t，它只是个假想的动作，只在“模拟器”中执行，而不是 AlphaGo 真正执行的动作。AlphaGo 需要考虑这样一个问题：假如它执行动作 a_t，那么对手会执行什么动作呢？对手肯定不会把自己的想法告诉 AlphaGo，那么 AlphaGo 只能自己猜测对手的动作。AlphaGo 可以“推己及人”：如果 AlphaGo

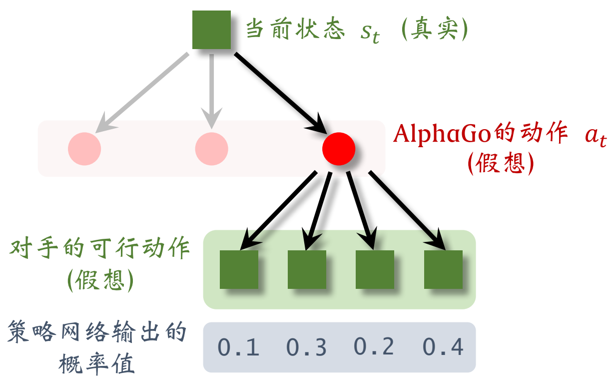

图 18.5: 假设 AlphaGo 有三种可行的动作，AlphaGo 选中第三个，并在模拟中执行。用策略网络模拟对手，策略网络输出对手可行动作的概率值：0.1, 0.3, 0.2, 0.4。

认为几个动作很好，对手也会这么认为。所以 AlphaGo 用策略网络模拟对手，根据策略网络随机抽样一个动作：

$$a_t' \sim \pi\big(\cdot \,\big|\, s_t'; \boldsymbol{\theta}\big)$$

此处的状态 s' 是站在对手的角度观测到的棋盘上的格局，动作 a_t' 是（假想）对手选择的动作。图 18.5 的例子中对手有四种可行动作，AlphaGo 用策略网络算出每个动作的概率值，然后根据概率值随机抽样一个对手的动作，记作 a_t'。假设根据概率值 0.1, 0.3, 0.2, 0.4 做随机抽样，选中第二种动作；见图 18.6。从 AlphaGo 的角度来看，对手的动作就是 AlphaGo 新的状态。

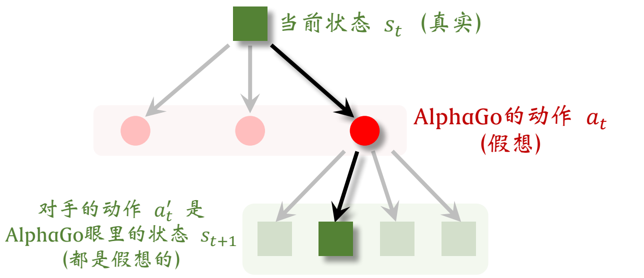

图 18.6: 假设对手有四种可行的动作，AlphaGo 根据概率值做随机抽样，替对手选中了第二种动作。对手的动作就是 AlphaGo 眼里的新的状态。

AlphaGo 需要在模拟中跟对手将一局游戏进行下去，所以需要一个模拟器（即环境）。在模拟器中，AlphaGo 每执行一个动作 a_k，模拟器就会返回一个新的状态 s_{k+1}。想要搭建一个好的模拟器，关键在于使用正确的状态转移函数 $p(s_{k+1}|s_k, a_k)$；如果状态转移函数与事实偏离太远，那么用模拟器做 MCTS 是毫无意义的。

AlphaGo 模拟器利用了围棋游戏的对称性：AlphaGo 的策略，在对手看来是状态转移函数；对手的策略，在 AlphaGo 看来是状态转移函数。最理想的情况下，模拟器的状态转移函数是对手的真实策略；然而 AlphaGo 并不知道对手的真实策略。AlphaGo 退而求其次，用 AlphaGo 自己训练出的策略网络 π 代替对手的策略，作为模拟器的状态转移函数。

想要用 MCTS 做决策，必须要有模拟器，而搭建模拟器的关键在于构造正确的状态转移函数 $p(s_{k+1}|s_k, a_k)$。从搭建模拟器的角度来看，围棋是非常简单的问题：由于围棋的对称性，可以用策略网络作为状态转移函数。但是对于大多数的实际问题，构造状态转移函数是非常困难的。比如机器人、无人车等应用，状态转移的构造需要物理模型，要考虑到力、运动、以及外部世界的干扰。如果物理模型不够准确，导致状态转移函数偏离事实太远，那么 MCTS 的模拟结果就不可靠。

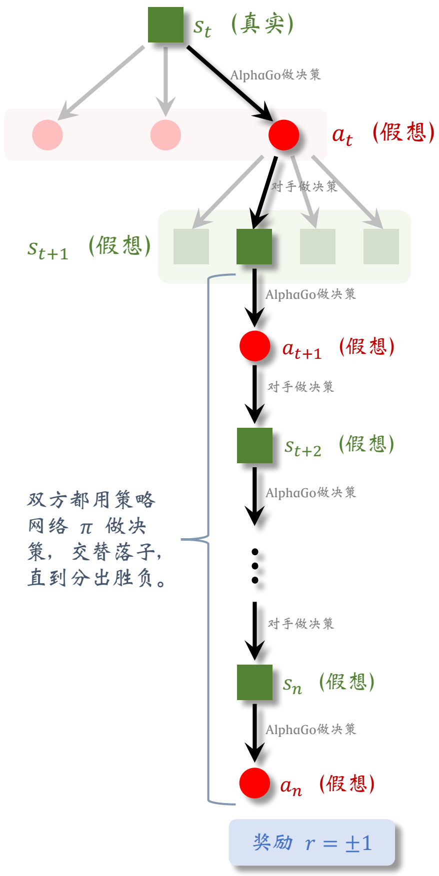

图 18.7: 策略网络自我博弈。

第三步——求值 (Evaluation)： 从状态 s_{t+1} 开始，双方都用策略网络 π 做决策，在模拟器中交替落子，直到分出胜负；见图 18.7。AlphaGo 基于状态 s_k，根据策略网络抽样得到动作

$$a_k \sim \pi\big(\cdot \,\big|\, s_k;\, \boldsymbol{\theta}\big).$$

对手基于状态 s_k'（以对手角度看棋盘上的格局），根据策略网络抽样得到动作

$$a_k' \sim \pi\big(\cdot \,\big|\, s_k';\, \boldsymbol{\theta}\big).$$

当这局游戏结束时，可以观测到奖励 r。如果 AlphaGo 胜利，则 $r=+1$，否则 $r=-1$。

回顾一下，真实棋盘上的状态是 s_t，AlphaGo 在模拟器中执行动作 a_t，然后模拟器中的对手执行动作 a_t'，带来新的状态 s_{t+1}。状态 s_{t+1} 越好，则这局游戏胜算越大。

- 如果这局模拟的游戏赢了 $(r=+1)$，说明 s_{t+1} 可能好；如果输了 $(r=-1)$，则说明 s_{t+1} 可能不好。因此，奖励 r 可以反映出 s_{t+1} 的好坏。
- 此外，还可以用价值网络 v 评价状态 s_{t+1} 的好坏。价值 $v(s_{t+1};\boldsymbol{w})$ 越大，则说明状态 s_{t+1} 越好。

奖励 r 是模拟获得的胜负，是对 s_{t+1} 很可靠的评价，但是随机性太大。价值网络的评估 $v(s_{t+1};\boldsymbol{w})$ 没有 r 可靠，但是价值网络更稳定、随机性小。AlphaGo 的解决方案是把奖励 r 与价值网络的输出 $v(s_{t+1};\boldsymbol{w})$ 取平均，记作：

$$V\big(s_{t+1}\big) \triangleq \frac{r + v(s_{t+1};\boldsymbol{w})}{2},$$

把它记录下来，作为对状态 s_{t+1} 的评价。

实际实现的时候，AlphaGo 还训练了一个更小的神经网络，它做决策更快。MCTS 在第一步和第二步用大的策略网络，第三步用小的策略网络。读者可能好奇，为什么在且仅在第三步用小的策略网络呢？第三步两个策略网络交替落子，通常要走一两百步，导致第三步成为 MCTS 的瓶颈。用小的策略网络代替大的策略网络，可以大幅加速 MCTS。

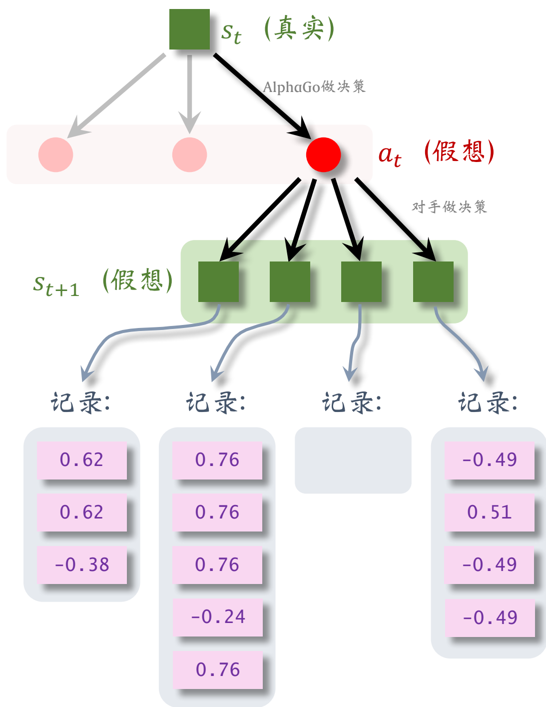

图 18.8: 每一个状态 s_{t+1} 下面都有很多条记录，每一条记录是一个 $V(s_{t+1})$。

第四步——回溯 (Backup)： 第三步——求值——算出了第 $t+1$ 步某一个状态的价值，记作 $V(s_{t+1})$；每一次模拟都会得出这样一个价值，并且记录下来。模拟会重复很多次，于是第 $t+1$ 步每一个可能的状态下面可能会有很多条记录；如图 18.8 所示。第 t 步的动作 a_t 下面有多个可能的状态（子

节点)，每个状态下面有若干条记录。把 a_t 下面所有的记录取平均，记作价值 $Q(a_t)$，它可以反映出动作 a_t 的好坏。

给定棋盘上的真实状态 s_t，有多个动作 a 可供选择。对于所有的 a，价值 $Q(a)$ 的初始值是零。动作 a 每被选中一次（成为 a_t），它下面就会多一条记录，我们就对 $Q(a)$ 做一次更新。

回顾第一步——选择 (Selection)： 基于棋盘上真实的状态 s_t，MCTS 需要从可行的动作中选出一个，作为 a_t。MCTS 计算每一个动作 a 的分数：

$$\text{score}(a) \;\triangleq\; Q(a) + \frac{\eta}{1+N(a)} \cdot \pi(a|s;\boldsymbol{\theta}), \qquad \forall\, a,$$

然后选择分数最高的 a。MCTS 算出的 $Q(a)$ 的用途就是这里。

18.2.3 MCTS 的决策

上一小节讲解了一次模拟的四个步骤，注意，这只是一次模拟。MCTS 想要真正做出一个决策（往真正的棋盘上落一个棋子），需要做成千上万次模拟。在做了无数次模拟之后，MCTS 的决策是

$$a_t \;=\; \underset{a}{\text{argmax}}\; N(a).$$

此时 AlphaGo 才会真正往棋盘上放一个棋子。

为什么要用 $N(a)$ 来做决策呢？每次模拟找出所有可行的动作 $\{a\}$，算出它们的分数 $\text{score}(a)$，然后选择分数最高的动作，在模拟器里执行。如果某个动作 a 在模拟中胜率很大，那么它的价值 $Q(a)$ 就会很大，它的分数 $\text{score}(a)$ 会很高，于是它被选中的几率就大。也就是说如果某个动作 a 很好，它被选中的次数 $N(a)$ 就会大。

当观测到棋盘上当前状态 s_t，MCTS 做了成千上万次模拟更新每个动作 a 的价值 $Q(a)$ 和次数 $N(a)$，最终做出决策 $a_t = \text{argmax}_a N(a)$。到了下一时刻，状态变成了 s_{t+1}。MCTS 得把所有动作 a 的 $Q(a)$、$N(a)$ 全都初始化为零，从头开始做模拟，而不能利用上一次的结果。

AlphaGo 下棋非常“暴力”：每走一步棋之前，它先在“脑海里”模拟几千、几万局，它可以预知它每一种动作带来的后果，对手最有可能做出的反应都在 AlphaGo 的算计之内。由于计算量差距悬殊，人类面对 AlphaGo 时不太可能有胜算。这样的比赛对人来说是不公平的；假如李世石下每一颗棋子之前，先跟柯洁模拟一千局，或许李世石的胜算会大于 AlphaGo。

18.3 训练策略网络和价值网络

上一节假设策略网络和价值网络已经训练好，并且用策略网络和价值网络辅助 MCTS。本节具体讲解如何训练两个神经网络。AlphaGo 有多个版本，其中最著名的是 2016、2017 年发表在 Nature 期刊的两个版本，本书称之为 2016 版和 AlphaGo Zero 版。AlphaGo Zero 实力更强：DeepMind 做了实验，让两个版本博弈 100 次，比分是 100 : 0。

18.3.1 AlphaGo 2016 版本的训练

AlphaGo 2016 版的训练分为三步：第一，随机初始化策略网络 $\pi(a|s;\boldsymbol{\theta})$ 之后，用行为克隆 (Behavior Cloning) 从人类棋谱中学习策略网络；第二，让两个策略网络自我博弈，用 REINFORCE 算法改进策略网络；第三，基于已经训练好的策略网络，训练价值网络 $v(s;\boldsymbol{w})$。

第一步：行为克隆： 一开始的时候，策略网络的参数都是随机初始化的。假如直接让两个策略网络自我博弈，它们会做出纯随机的动作。它们得随机摸索很多很多次，才能做出合理的动作。假如一上来就用 REINFORCE 学习策略网络，最初随机摸索的过程要花很久。这就是为什么 AlphaGo 2016 版用人的知识学一个初步的策略网络。

有一个叫 KGS 的在线围棋游戏程序，它在 2000 年的时候上线，让玩家在线比赛。KGS 会把每一局游戏都记录下来。KGS 有 16 万局是六段以上的高级玩家的记录。每一局游戏有很多步，每一步棋盘上的格局作为一个状态 s_k，下一个棋子的位置作为动作 a_k，这样得到数据集 $\{(s_k, a_k)\}$。数据集中一共有 $m = 2.94 \times 10^7$ 个 (s_k, a_k) 这样的二元组。

AlphaGo 用行为克隆训练策略网络 $\pi(a|s;\boldsymbol{\theta})$。第 17.1 节详细介绍了行为克隆，这里只是简单概括一下。设 361 维的向量

$$\boldsymbol{f}_k \;=\; \pi(\,\cdot\,|s_k;\boldsymbol{\theta}) \;=\; \Big[\pi\big(1\,\big|\,s_k;\boldsymbol{\theta}\big),\, \pi\big(2\,\big|\,s_k;\boldsymbol{\theta}\big),\, \cdots\, \pi\big(361\,\big|\,s_k;\boldsymbol{\theta}\big)\Big]$$

是策略网络的输出，设 $\bar{\boldsymbol{a}}_k$ 是对动作 a_k 的 One-Hot 编码。函数 $H(\bar{\boldsymbol{a}}_k,\, \boldsymbol{f}_k)$ 是交叉熵 (Cross Entropy)，衡量 $\bar{\boldsymbol{a}}_k$ 与 $\boldsymbol{f}_k$ 的差别。行为克隆可以描述成这样一个优化问题：

$$\min_{\boldsymbol{\theta}} \; \frac{1}{m} \sum_{k=1}^{m} H\big(\,\bar{\boldsymbol{a}}_k,\, \boldsymbol{f}_k\,\big).$$

可以用随机梯度下降 (SGD) 求解这个优化问题。每次随机从 $\{1, \cdots, m\}$ 中选出一个序号，记作 j。设当前策略网络参数为 $\boldsymbol{\theta}_{\text{now}}$ 用随机梯度更新 $\boldsymbol{\theta}$：

$$\boldsymbol{\theta}_{\text{new}} \;\leftarrow\; \boldsymbol{\theta}_{\text{now}} \;-\; \eta \cdot \nabla_{\boldsymbol{\theta}} H\Big(\,\bar{\boldsymbol{a}}_j,\, \pi\big(\,\cdot\,\big|s_j;\boldsymbol{\theta}_{\text{now}}\big)\Big).$$

此处的 η 是学习率。这样可以让策略网络的决策 $\pi(\cdot|s_k;\boldsymbol{\theta})$ 更接近人类高手的动作 $\bar{\boldsymbol{a}}_j$。

KGS 中的 16 万局游戏都是六段以上的高手的博弈。行为克隆得到的策略网络模仿高手的动作，可以做出比较合理的决策。它在实战中可以打败业余玩家，但是打不过职业玩家。第 17.1 节详细讨论过行为克隆的缺点。为了克服行为克隆的缺点，还需要继续用强化学习训练行为克隆。在行为克隆之后再做强化学习改进策略网络，可以击败只用行为克隆的策略网络，胜算是 80%。

第二步——用 REINFORCE 训练策略网络： AlphaGo 让策略网络做自我博弈，用胜负作为奖励，更新策略网络。博弈的策略网络一个叫做“玩家”，用最新的参数，记作 $\boldsymbol{\theta}_{\text{now}}$；另一个叫做“对手”，它的参数是从过时的参数中随机选出来的，记作 $\boldsymbol{\theta}_{\text{old}}$。“对手”的作用相当于模拟器（环境）的状态转移函数，只是陪玩。训练的过程中，只更新“玩家”的参数，不更新“对手”的参数。

让“玩家”和“对手”博弈，将一局游戏进行到底，假设走了 n 步。游戏没结束的时候，奖励全都是零：

$$r_1 \;=\; r_2 \;=\; \cdots \;=\; r_{n-1} \;=\; 0.$$

游戏结束的时候，如果“玩家”赢了，奖励是 $r_n = +1$，于是所有的回报都是正一：[1]

$$u_1 \;=\; u_2 \;=\; \cdots \;=\; u_n \;=\; +1.$$

如果“玩家”输了，奖励是 $r_n = -1$，于是所有的回报都是负一：

$$u_1 \;=\; u_2 \;=\; \cdots \;=\; u_n \;=\; -1.$$

用同样的回报 u 相当于不区分哪一步棋走得好，哪一步走得烂；只要赢了，每一步都算是“好棋”；假如输了，每一步都被看做“臭棋”。

REINFORCE 是一种策略梯度方法，它用观测到的回报 u 近似动作价值 Q_π。REINFORCE 更新策略网络的公式是：

$$\boldsymbol{\theta}_{\text{new}} \;\leftarrow\; \boldsymbol{\theta}_{\text{now}} \;+\; \beta \cdot \sum_{t=1}^{n} u_t \cdot \nabla \ln \pi\big(a_t \,\big|\, s_t;\, \boldsymbol{\theta}_{\text{now}}\big).$$

此处的 β 是学习率。

第三步——训练价值网络： 价值网络 $v(s;\boldsymbol{w})$ 是对状态价值函数 $V_\pi(s)$ 的近似，用于评估状态 s 的好坏。在完成第二步——训练策略网络 π——之后，用 π 辅助训练 v。虽然此处有一个策略网络 π 和一个价值网络 v，但这不属于 Actor-Critic 方法。此处先训练 π，再训练 v，用 π 辅助训练 v；而 Actor-Critic 则是同时训练 π 和 v，用 v 辅助训练 π。

让训练好的策略网络做自我博弈，记录状态—回报二元组 (s_k, u_k)，存到一个数组里。自我博弈需要重复非常多次，把最终得到的数据集记作 $\big\{(s_k, u_k)\big\}_{k=1}^{m}$。根据定义，状态价值 $V_\pi(s_k)$ 是回报 U_k 的期望：

$$V_\pi\big(s_k\big) \;=\; \mathbb{E}\big[U_k \,\big|\, S_k = s_k\big].$$

我们希望价值网络 $v(s_k;\boldsymbol{w})$ 接近 V_π，也就是上面的期望，于是让 $v(s_k;\boldsymbol{w})$ 去拟合回报 u_k。定义回归问题 (Regression)：

$$\min_{\boldsymbol{w}} \; \frac{1}{2m} \sum_{k=1}^{m} \Big[v\big(s_k;\boldsymbol{w}\big) \;-\; u_k \Big]^2.$$

可以用随机梯度下降 (SGD) 求解这个优化问题。设当前价值网络参数为 $\boldsymbol{w}_{\text{now}}$ 每次随机从 $\{1,\cdots,m\}$ 中选出一个序号，记作 j。用价值网络做预测：$\widehat{v}_j = v(s_j;\boldsymbol{w}_{\text{now}})$。用随机梯度更新 $\boldsymbol{w}$：

$$\boldsymbol{w}_{\text{new}} \;\leftarrow\; \boldsymbol{w}_{\text{now}} \;-\; \alpha \cdot \big(\widehat{v}_j - u_j\big) \cdot \nabla_{\boldsymbol{w}} v\big(s_j;\boldsymbol{w}_{\text{now}}\big).$$

[1] 回报的定义是 $u_t = r_t + r_{t+1} + \cdots + r_n$，折扣率是 $\gamma = 1$。

此处的 α 是学习率。

18.3.2 AlphaGo Zero 版本的训练

AlphaGo Zero 与 2016 版本的最大区别在于训练策略网络 $\pi(a|s;\boldsymbol{\theta})$ 方式。训练 π 的时候，不再从人类棋谱学习，也不用 REINFORCE 方法，而是向 MCTS 学习。其实可以把 AlphaGo Zero 训练 π 的方法看做是模仿学习，模仿对象是 MCTS。

自我博弈： 用 MCTS 控制两个玩家对弈。每走一步棋，MCTS 需要做成千上万次模拟，并记录下每个动作被选中的次数 $N(a)$，$\forall\, a \in \{1,2,\cdots,361\}$。设当前是 t 时刻，当前状态是 s_t，MCTS 完成成千上万次模拟，得到 361 个整数（动作被选中的次数）：

$$N(1),\ N(2),\ \cdots,\ N(361).$$

对这些 N 做归一化，得到的 361 个数，它们相加等于 1；把这 361 个数记作 361 维的向量：

$$\boldsymbol{p}_t \ = \ \text{normalize}\Big(\Big[N(1),\ N(2),\ \cdots,\ N(361)\Big]\Big).$$

设这局游戏走了 n 步之后游戏分出胜负；奖励 r_n 要么等于正一，要么等于负一，取决于胜负。于是得到回报 $u_1 = \cdots = u_n = r_n$。记录下这些数据：

$$\big(s_1,\ \boldsymbol{p}_1,\ u_1\big),\quad \cdots,\quad \big(s_n,\ \boldsymbol{p}_n,\ u_n\big).$$

用这些数据更新策略网络和价值网络。

更新策略网络： 上一节讨论过，MCTS 做出的决策优于策略网络 π 的决策，这就是为什么 AlphaGo 用 MCTS 做决策，而 π 只是用来辅助 MCTS。既然 MCTS 比 π 更好，那么可以把 MCTS 的决策作为目标，让 π 去模仿。这其实是行为克隆，模仿的对象是 MCTS，希望 π 做出的决策

$$\boldsymbol{f}_t \ = \ \pi\big(\,\cdot\,\big|\,s_t;\,\boldsymbol{\theta}\big) \ \in \ \mathbb{R}^{361}$$

尽量接近 $\boldsymbol{p}_t \in \mathbb{R}^{361}$，也就是让交叉熵 $H(\boldsymbol{p}_t,\,\boldsymbol{f}_t)$ 尽量小。定义优化问题：

$$\min_{\boldsymbol{\theta}} \ \bigg\{\frac{1}{n}\sum_{t=1}^{n} H\Big(\boldsymbol{p}_t,\ \pi\big(\,\cdot\,\big|\,s_t\ \boldsymbol{\theta}\big)\Big)\bigg\}.$$

设 π 当前参数是 $\boldsymbol{\theta}_{\text{now}}$。做一次梯度下降更新参数：

$$\boldsymbol{\theta}_{\text{new}} \ \leftarrow \ \boldsymbol{\theta}_{\text{now}} \ - \ \beta\cdot\frac{1}{n}\sum_{t=1}^{n}\nabla_{\boldsymbol{\theta}}\, H\Big(\boldsymbol{p}_t,\ \pi\big(\,\cdot\,\big|\,s_t\ \boldsymbol{\theta}_{\text{now}}\big)\Big). \tag{18.2}$$

此处的 β 是学习率。

更新价值网络： 训练价值网络的方法与 AlphaGo 2016 版本基本一样，都是让 $v(s_t;\boldsymbol{w})$ 拟合回报 u_t。定义优化问题：

$$\min_{\boldsymbol{w}} \ \frac{1}{2n}\sum_{t=1}^{n}\Big[v\big(s_t;\boldsymbol{w}\big) \ - \ u_t\Big]^2.$$

设 v 当前参数是 $\boldsymbol{w}_{\text{now}}$。用价值网络做预测：$\widehat{v}_t = v(s_t;\boldsymbol{w}_{\text{now}})$，$\forall t = 1,\cdots,n$。做一次梯

度下降更新 $\boldsymbol{w}$：

$$\boldsymbol{w}_{\text{new}} \;\leftarrow\; \boldsymbol{w}_{\text{now}} \;-\; \alpha \cdot \frac{1}{n} \sum_{t=1}^{n} \big(\widehat{v}_t - u_t\big) \cdot \nabla_{\boldsymbol{w}} v\big(s_t; \boldsymbol{w}_{\text{now}}\big). \tag{18.3}$$

训练流程： 随机初始化策略网络参数 $\boldsymbol{\theta}$ 和价值网络参数 $\boldsymbol{w}$。然后让 MCTS 自我博弈，玩很多局游戏；每完成一局游戏，做下面的步骤，更新一次 $\boldsymbol{\theta}$ 和 $\boldsymbol{w}$：

1. 让 MCTS 自我博弈，完成一局游戏，收集到 n 个三元组：$(s_1, \boldsymbol{p}_1, u_1), \cdots, (s_n, \boldsymbol{p}_n, u_n)$。
2. 按照公式 (18.2) 做一次梯度下降，更新策略网络参数 $\boldsymbol{\theta}$。
3. 按照公式 (18.3) 做一次梯度下降，更新价值网络参数 $\boldsymbol{w}$。

相关文献

早在很多年前，AI 就在棋类游戏中战胜了人类，比如国际象棋 (Chess) [24]，西洋跳棋 (Checker) [86, 85]，黑白棋 (Reversi 或 Othello) [22]，双陆棋 (Backgammon) [103]。这些棋类游戏可能存在的状态空间远比围棋的状态空间小，所以做搜索会相对比较容易。

AlphaGo 的论文首先发表在 Nature 2016 [91]。改进版本 AlphaGo Zero 发表在 Nature 2017 [93]。在 AlphaGo 之前一直有对围棋 AI 的探索，尽管 AI 尚无法击败人类围棋冠军。其中最有名的围棋 AI 包括 Pachi [11], Fuego [36], GNU Go (1999 年发布，2009 年停更), Crazy Stone (2006 年发布）。Crazy Stone 虽然不及人类冠军，但是在对手让 4 子的情况下打败过 9 段高手。有兴趣的读者可以参考这些论文：[4, 73, 107, 18, 32, 36, 11]。

蒙特卡洛树搜索 (MCTS) 的名字最早在 2006 年发表的论文 [31] 中提出。另外两篇 2006 年的论文 [26, 55] 提出了类似的想法。2008 年发表的论文 [25] 将 MCTS 概括为今天我们众所周知的四个步骤。本书篇幅有限，不深入介绍 MCTS。有兴趣的读者可以阅读综述 [21] 和书 [27]。