# 第三章 对抗搜索

- 对抗搜索：博弈

- 博弈问题
- 极小极大方法
- α-β剪枝
- 蒙特卡洛博弈方法
- AlphaGo原理
- 围棋中的深度强化学习
- AlphaGo Zero原理

# 棋类人机大战简史

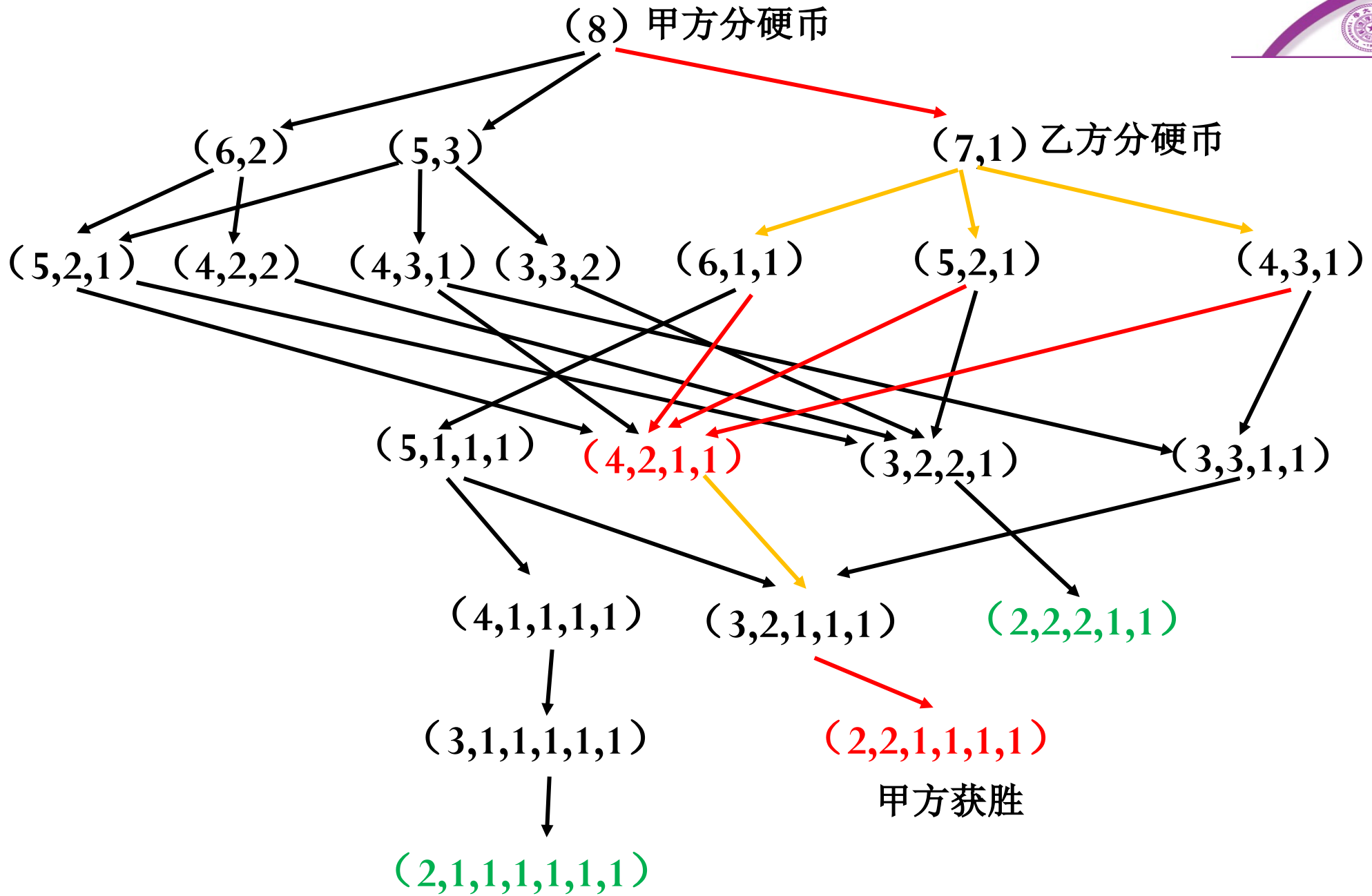
# 棋类人机大战简史

- 50年代麦卡锡提出α-β剪枝算法
- 1962年西洋跳棋程序战胜美国的州冠军
- 1996年深蓝首次挑战卡斯帕罗夫失败
- 1997年深蓝再战克斯帕罗夫获胜
- 2006年中国象棋程序战胜柳大华等5位棋手
- 2016年AlphaGo战胜李世石
- 2017年AlphaGo战胜柯洁

- 博弈问题
  - 双人
  - 一人一步
  - 双方信息完备
  - 零和

# 3.1 可以穷举吗?

- 从分钱币游戏说起


(8) 甲方分硬币
(6,2)
(5,3)
(7,1) 乙方分硬币
(5,2,1)
(4,2,2)
(4,3,1)
(3,3,2)
(6,1,1)
(5,2,1)
(4,3,1)
(5,1,1,1)
(4,2,1,1)
(3,2,2,1)
(3,3,1,1)
(4,1,1,1,1)
(3,2,1,1,1)
(2,2,2,1,1)
(3,1,1,1,1,1)
(2,2,1,1,1,1)
甲方获胜
(2,1,1,1,1,1,1)

# 可以穷举吗?

- 以中国象棋为例
  - 总状态数约为$10^{150}$
  - 假设1毫微秒走一步，约需$10^{134}$年
- 宇宙年龄：$1.38*10^{10}$年
- 如果一个原子存储一个状态，需要$10^{100}$个地球
- 结论：不可能穷举

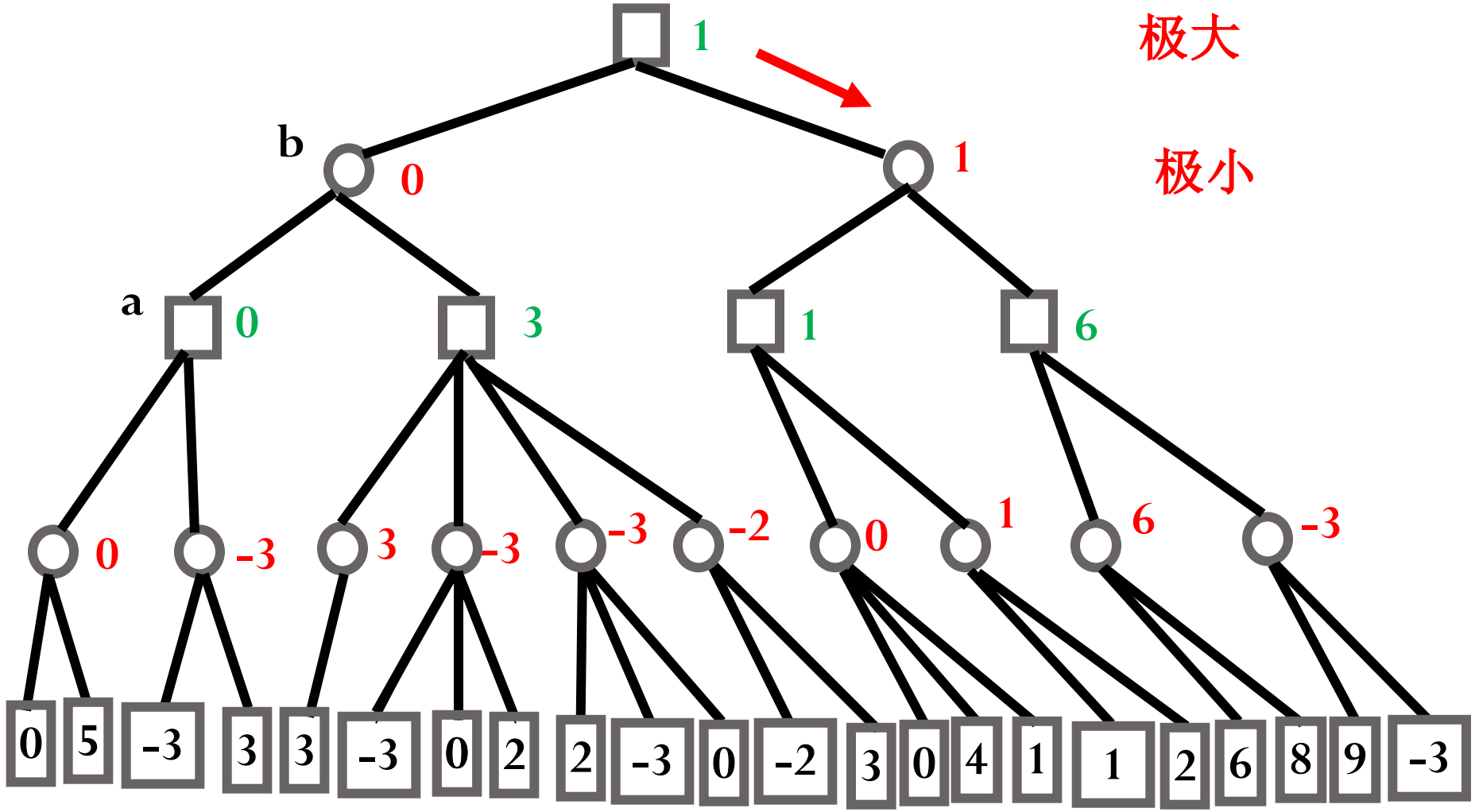
# 小结

- 对于象棋、围棋这类的棋类问题，不可能依靠穷举的办法解决。

# 3.2 极小-极大模型

- 人类下棋的思考过程
  - 向前看若干步
  - 从最不利的情景中选择最有利的走法

# 极小-极大模型



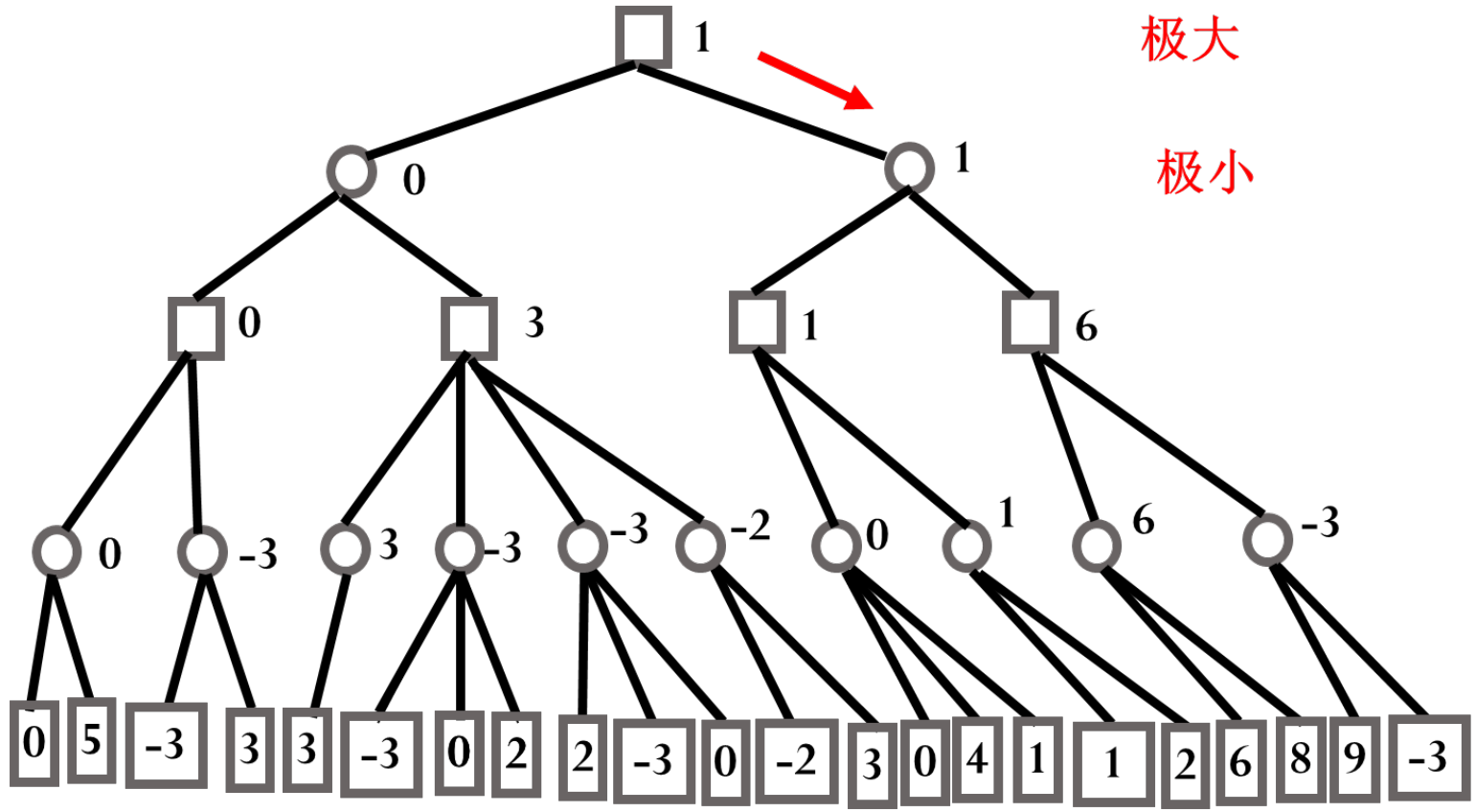
# 限定深度就可以穷举吗?

- 深蓝每下-
- 如果搜索1

# 小结

- 极小-极大模型
  - 模仿了人类下棋的思考过程
- 有限深度内的穷举也行不通

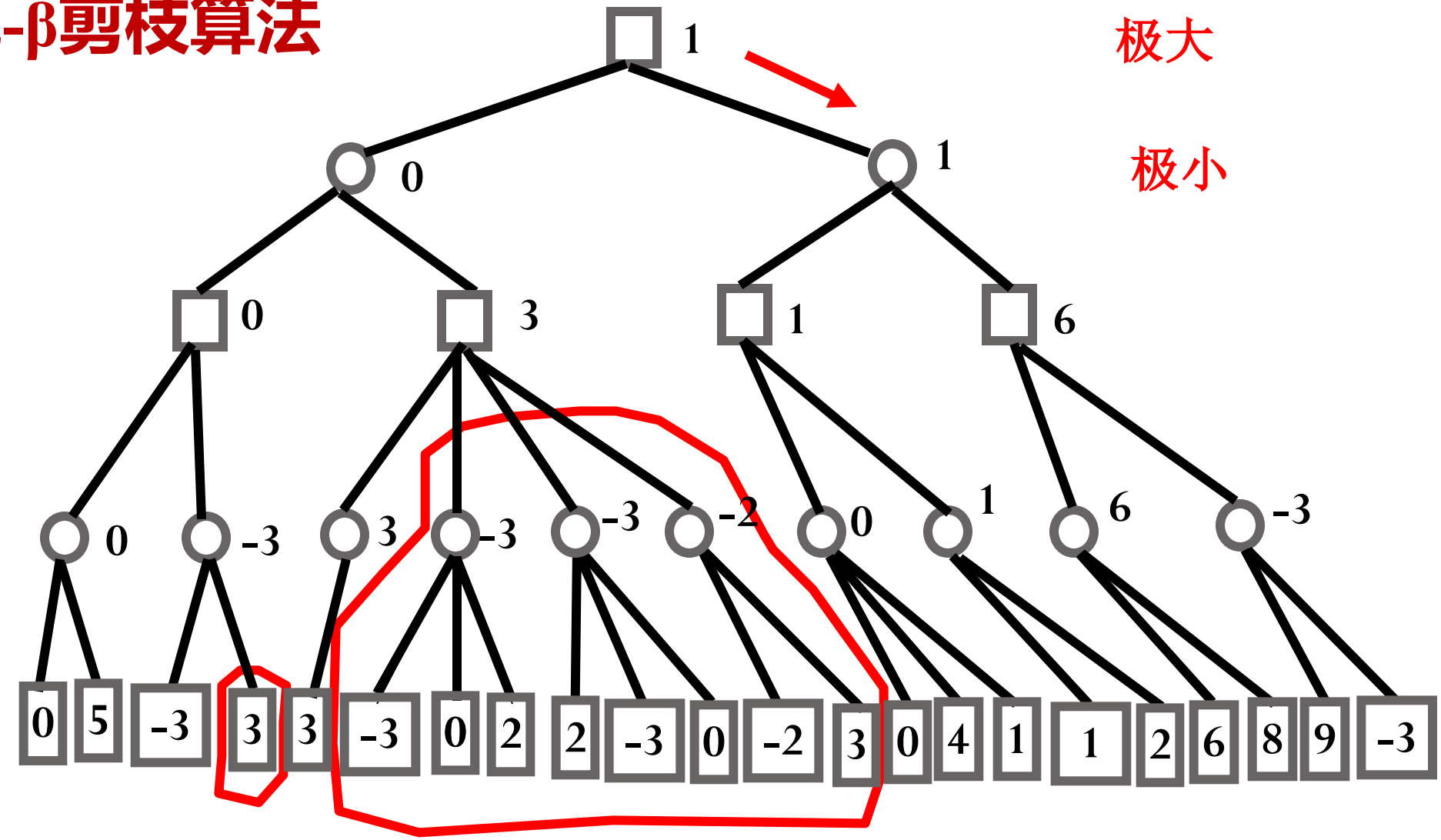
# 极小-极大模型存在的问题



# 人如何处理这个问题呢?

- 只在少数可能的走步范围内考虑

# 3.3 α-β剪枝算法

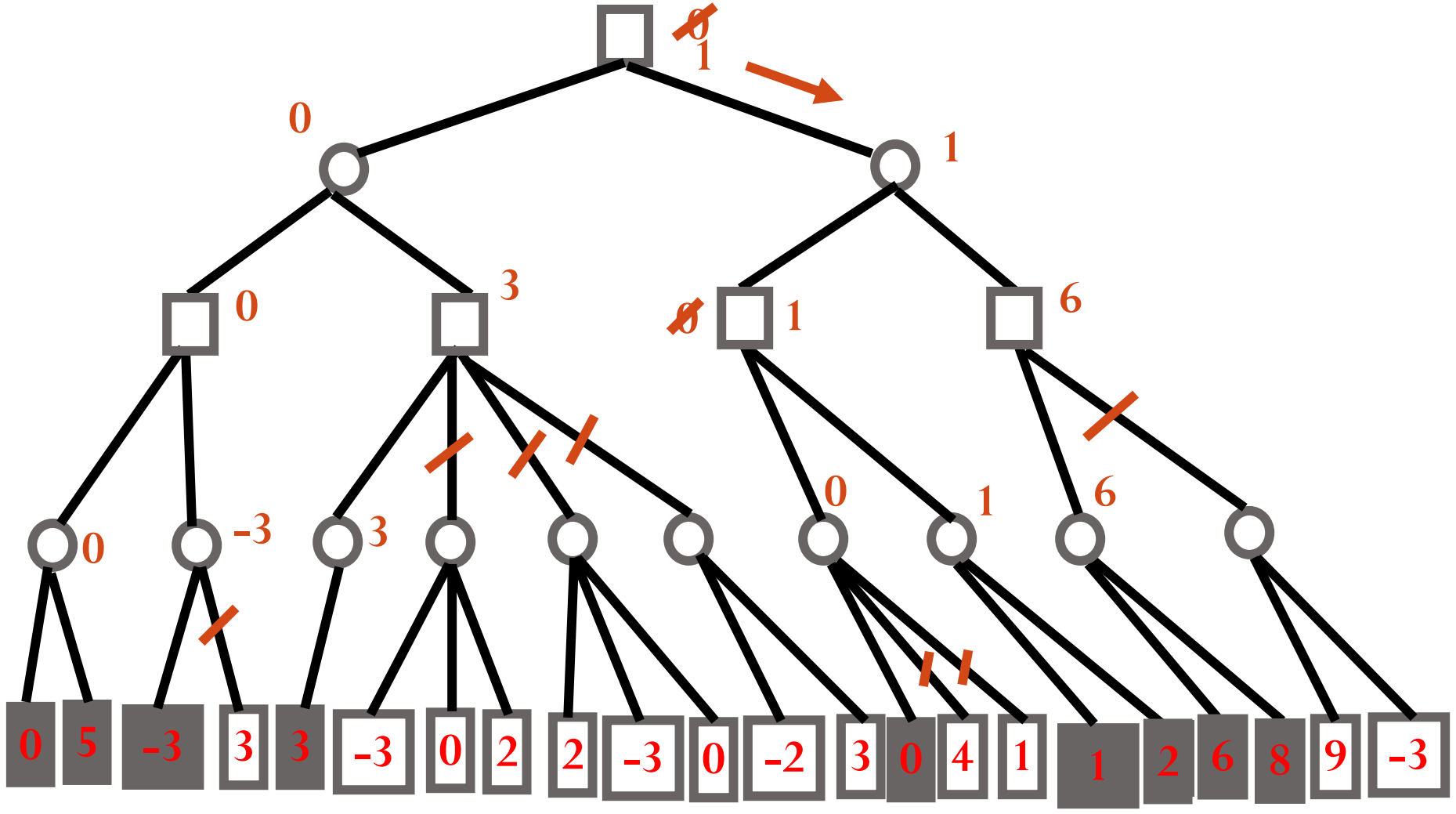



极大

极小

# α-β剪枝算法

- 极大节点的下界为α。
- 极小节点的上界为β。
- 剪枝的条件：
  - 后辈节点的β值≤祖先节点的α值时， α剪枝
  - 后辈节点的α 值≥祖先节点的β值时， β剪枝
- 简记为：
  - 极小≤极大，剪枝
  - 极大≥极小，剪枝



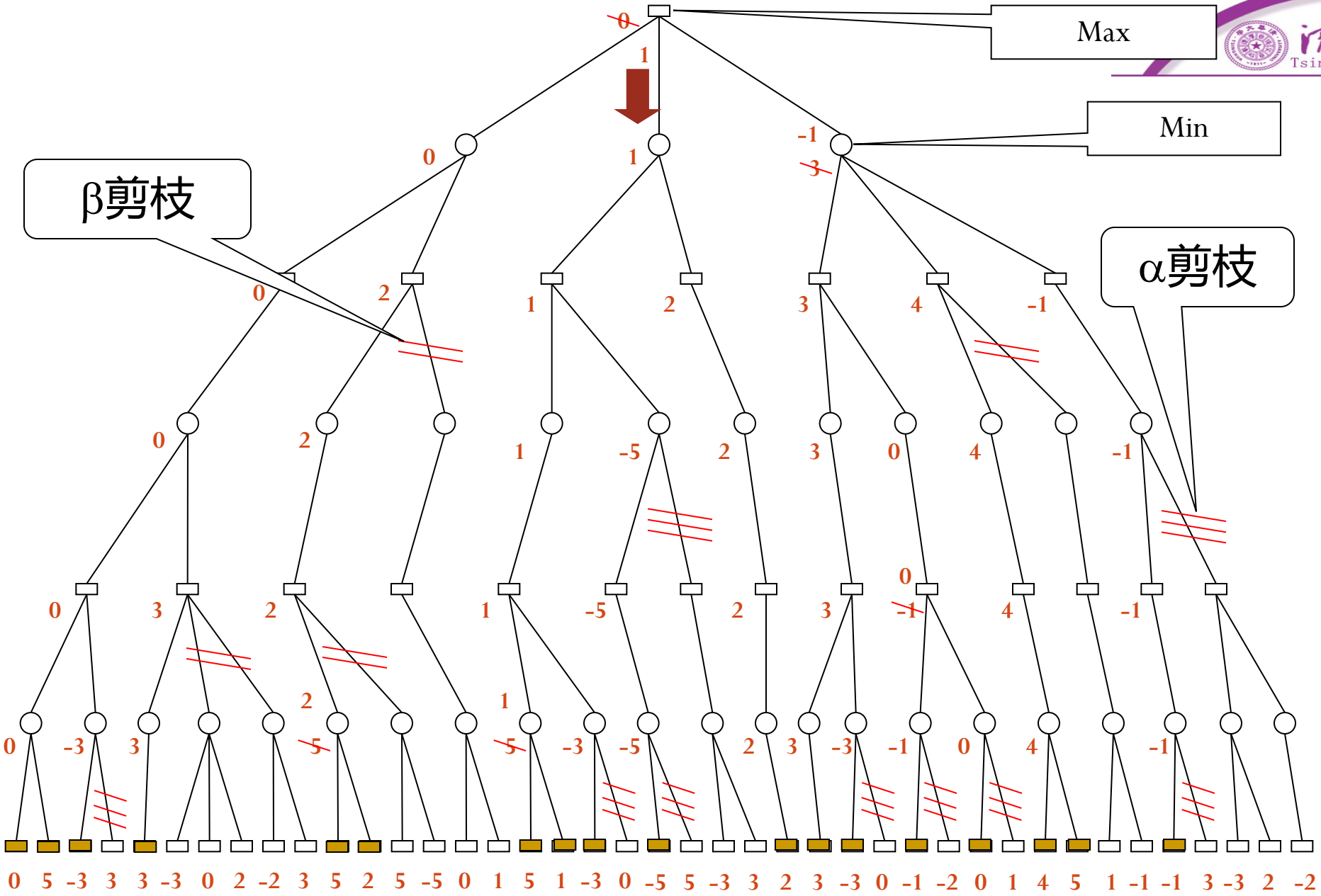

大
小
大
小
大

# 如何估值?

- 总结专家知识进行估值

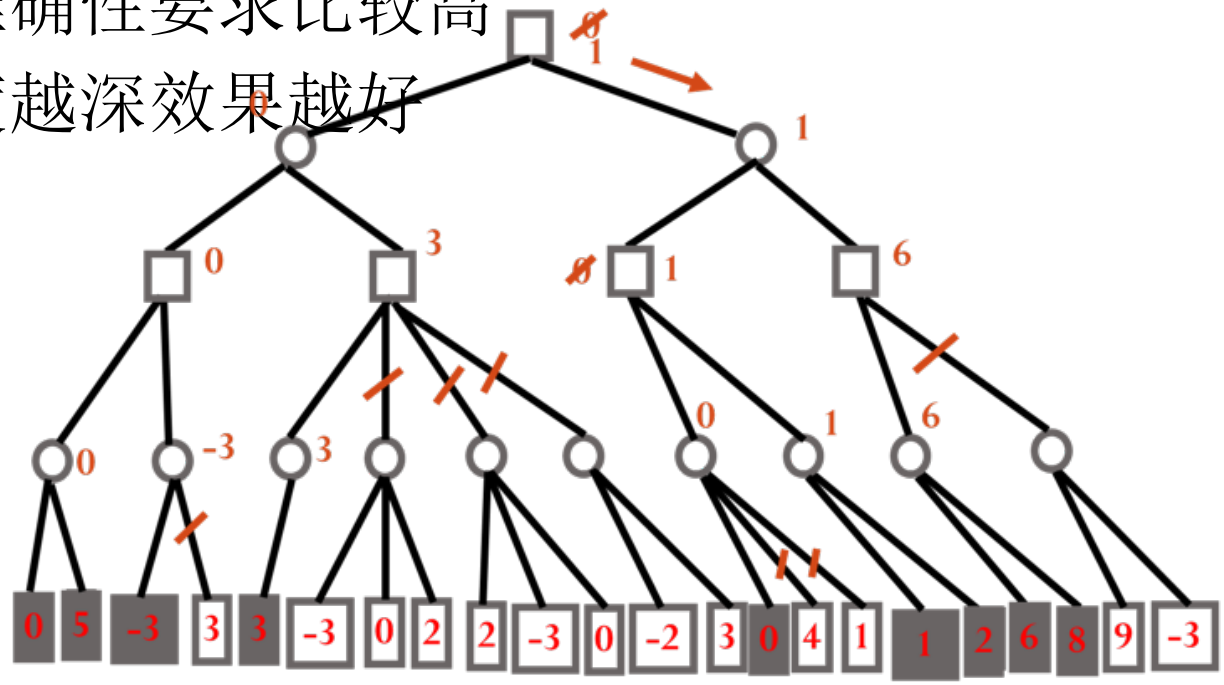
# α-β剪枝效果如何?

- 国际象棋、中国象棋软件的系统框架


Max
Min
β剪枝
α剪枝
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2 3 -3 0 -1 -2 0 1 4 5 1 -1 -1 3 -3 2 -2

# 几点注意

- α-β剪枝只是得到一步结果
- 注意比较的时候要跟祖先比
- 估值的准确性要求比较高
- 搜索深度越深效果越好



# 练习题

- 上述例题自己独立做一遍。

多选题 1分

设置

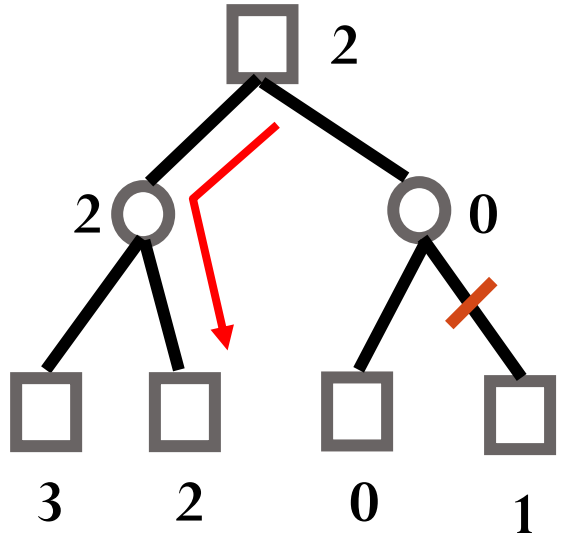
如图所示，剪枝是否正确？所选择的走步标记是否正确？

A 剪枝正确

B 剪枝错误

C 走步标记正确

D 走步标记错误




提交

# 小结

- α-β剪枝算法
  - 利用已有的搜索结果进行剪枝
- 剪枝的条件：
  - 后辈节点的β值≤祖先节点的α值时， α剪枝
  - 后辈节点的α 值≥祖先节点的β值时， β剪枝
- 一次剪枝过程只得到一次走步

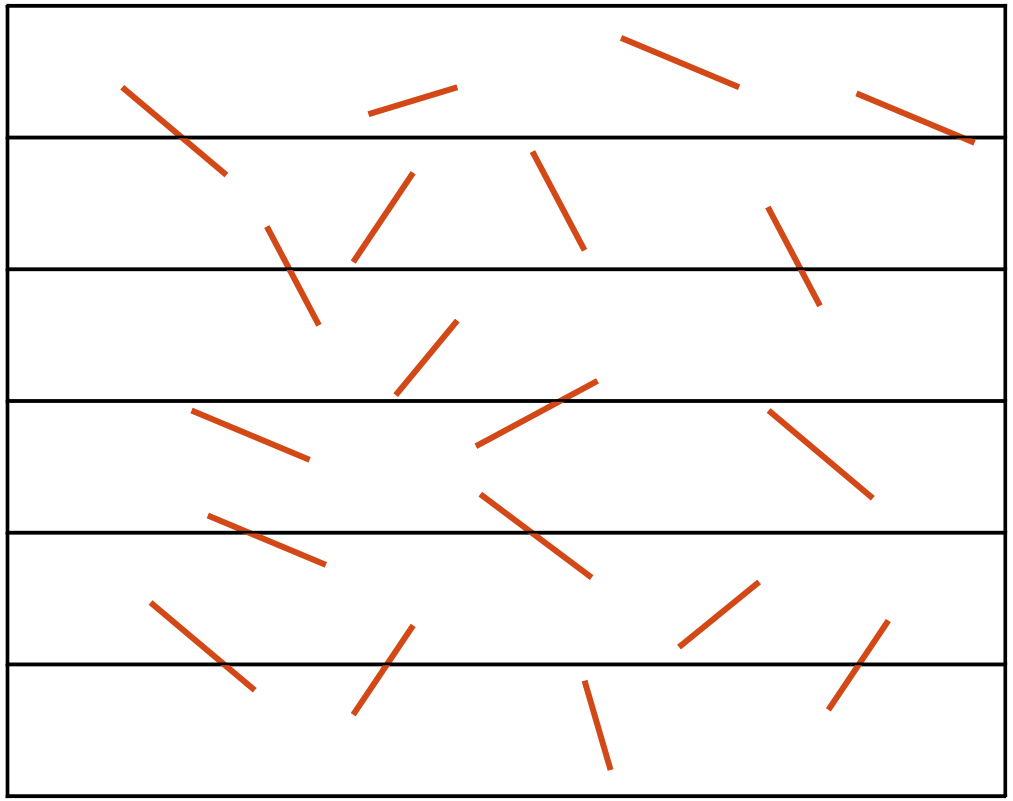
# 为什么α-β剪枝方法在围棋上失效?

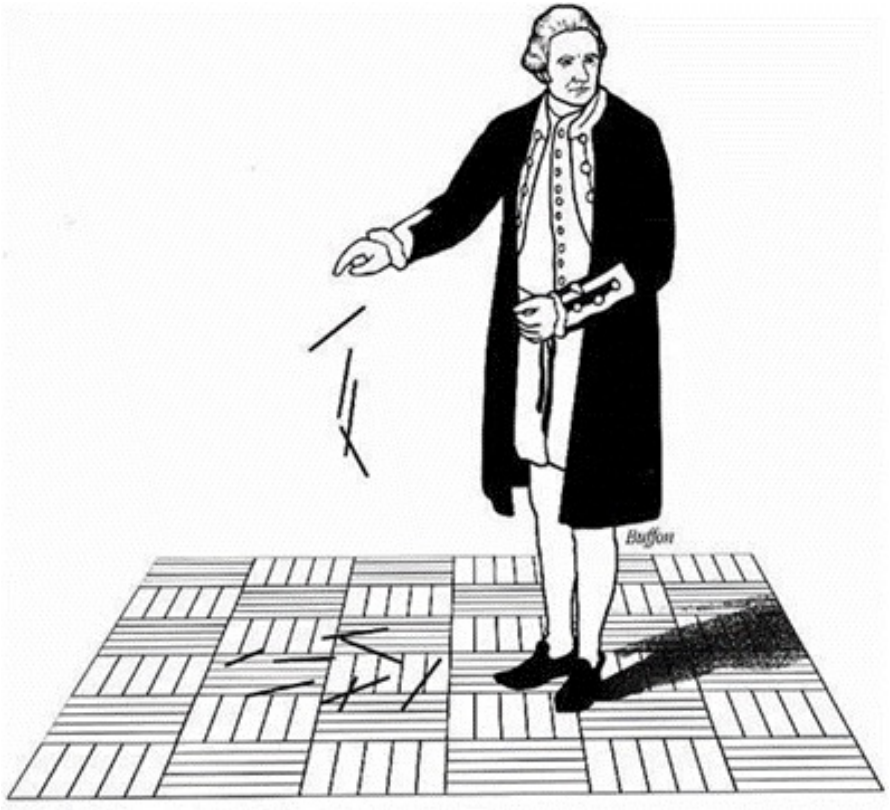
- 是因为状态多吗?
  - 状态多不是本质原因
- α-β剪枝方法存在的问题
  - 依赖于局面评估的准确性
- 局面评估问题
  - 大量专家知识
  - 知识的统一性问题
  - 人工整理
- 逻辑思维与形象思维

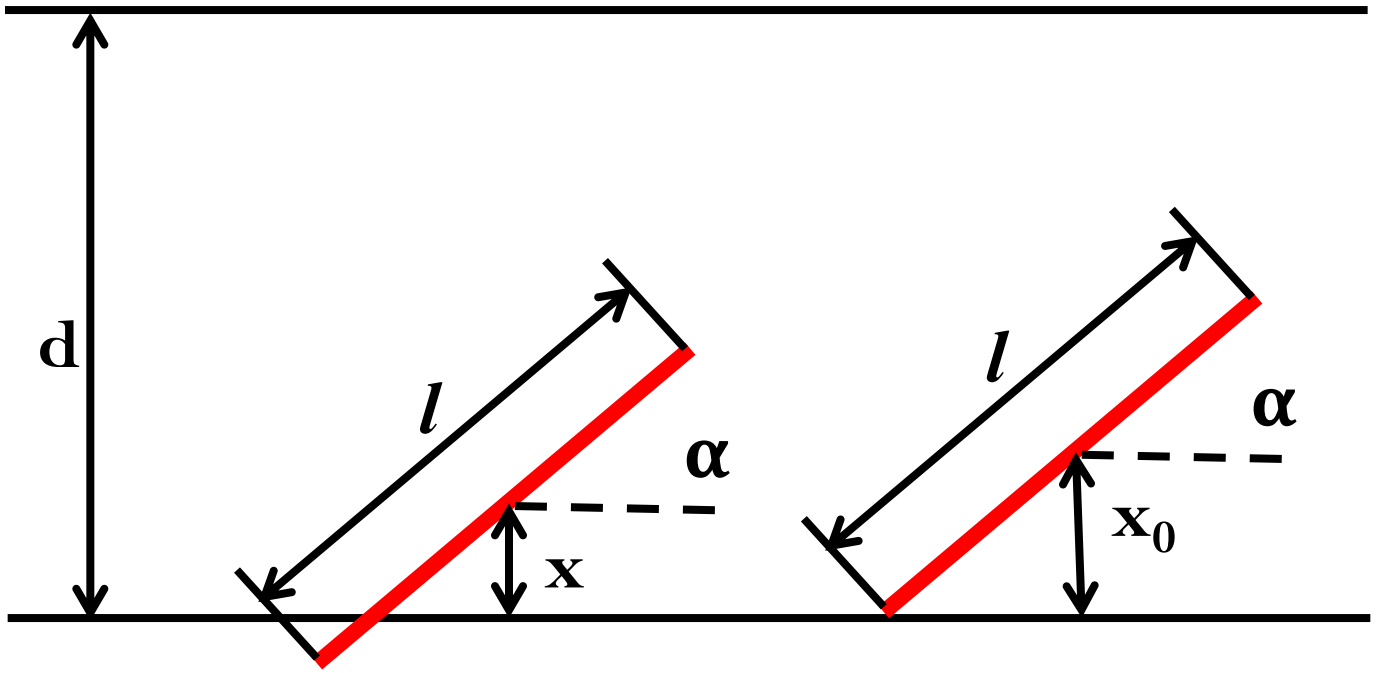
# 3.4 蒙特卡洛方法

- 二十世纪40年代中期S.M.乌拉姆和J.冯·诺伊曼提出的一种随机模拟方法
  - 多重积分
  - 矩阵求逆
  - 线性方程组求解
  - 积分方程求解
  - 偏微分方程求解
  - 随机性问题模拟

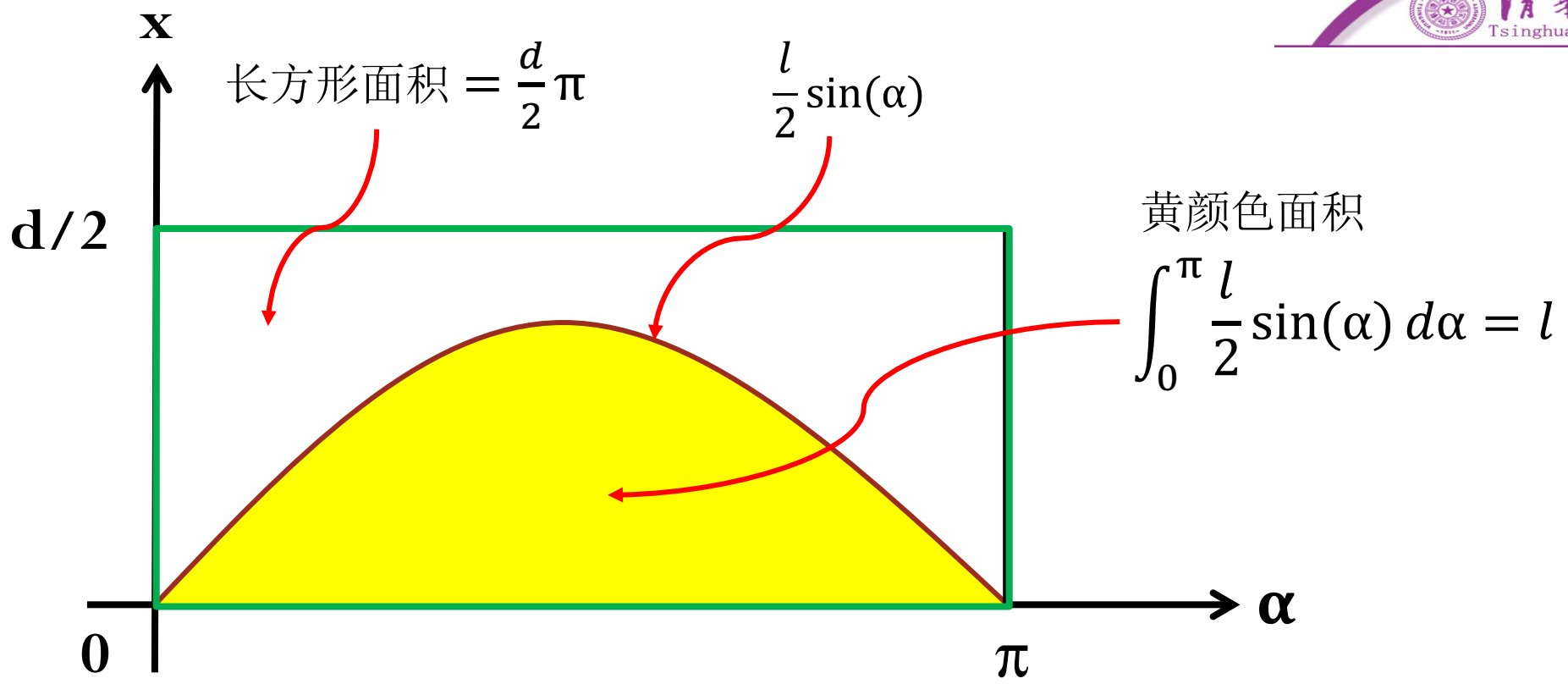
# 蒲丰投针问题








- (x, α)决定了针的位置
- $x_0 = (l/2)\cdot\sin\alpha$
- 针与直线的相交条件：$x \le (l/2)\cdot\sin\alpha$
- 其中：$x \in [0, d/2]$, $\alpha \in [0, \pi]$



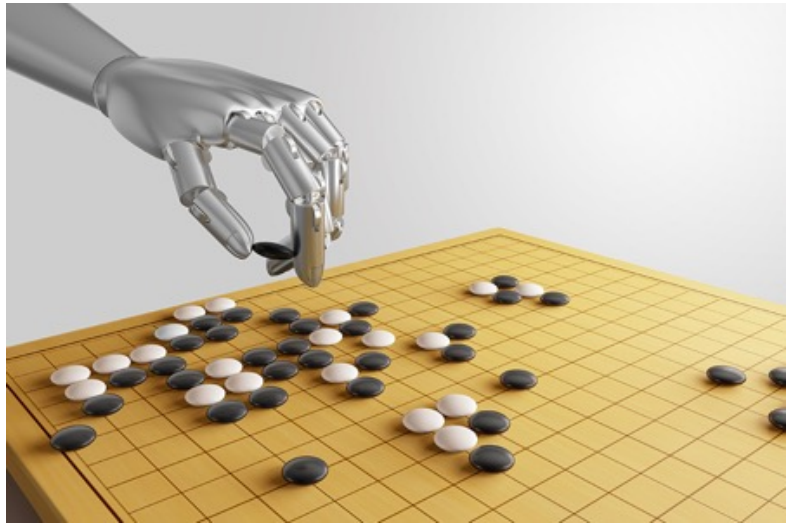

- 黄颜色部分与长方形面积之比即为针与直线相交的概率：

$$p = \frac{l}{\frac{d}{2}\pi} = \frac{2l}{d\pi} \approx \frac{m}{n} \Longrightarrow \pi \approx \frac{2nl}{md}$$

其中：n为掷针次数
m为相交次数

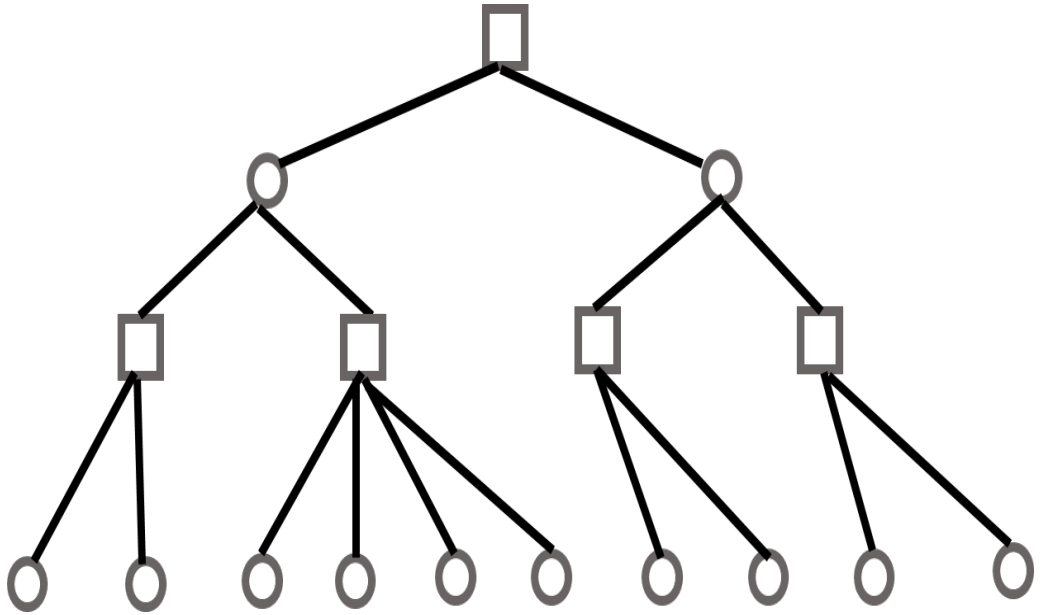
# 棋局的蒙特卡洛评估

- 从当前局面的所有可落子点中随机选择一个点落子
- 重复以上过程
- 直到胜负可判断为止
- 经多次模拟后，选择胜率最大的点落子

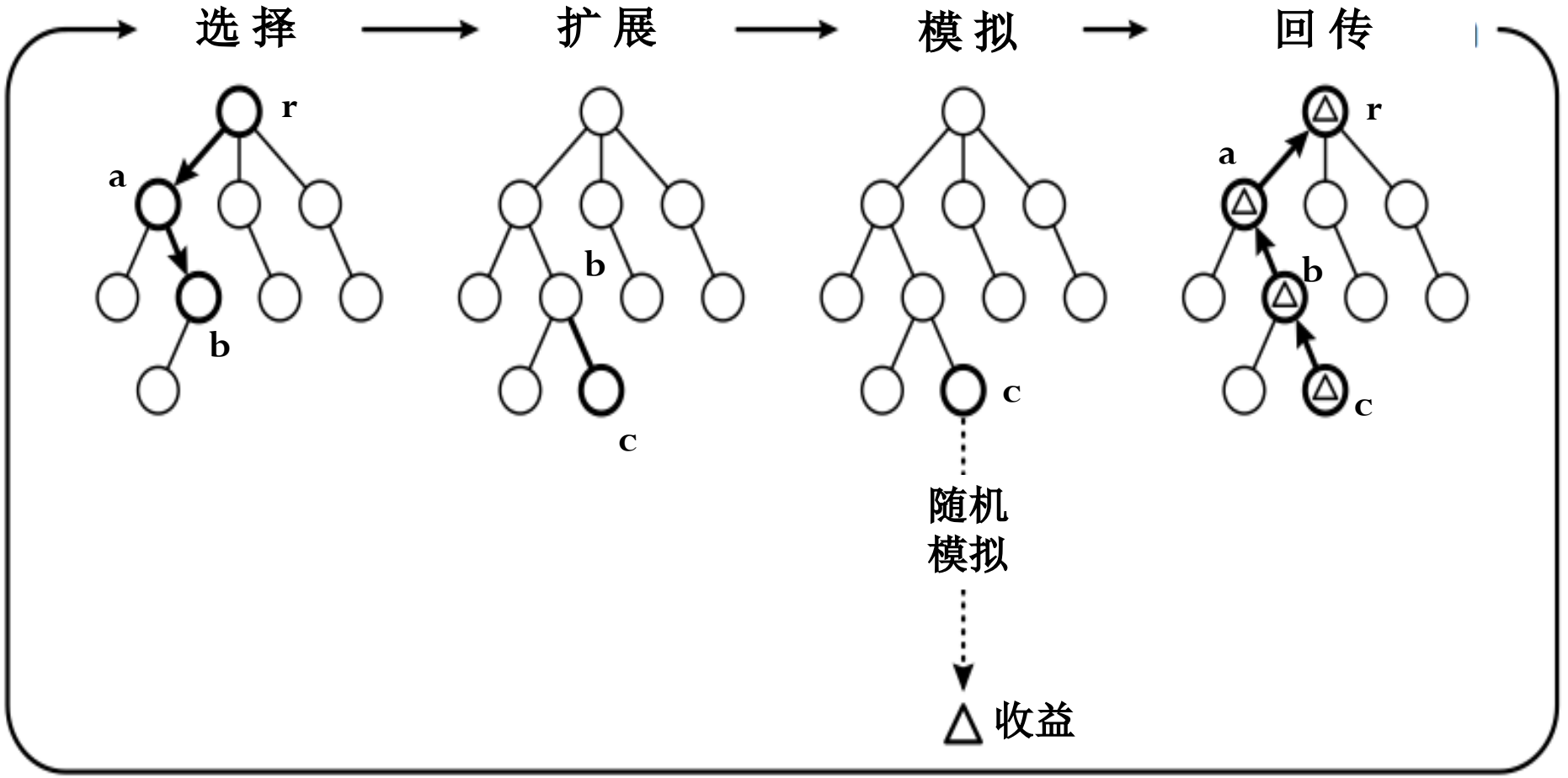


# 蒙特卡洛树搜索（MCTS: Monte Carlo Tree Search）

- 基本思想：
  - 将可能出现的状态转移过程用状态树表示
  - 从初始状态开始重复抽样，逐步扩展树中的节点
  - 父节点可以利用子节点的模拟结果，提高了效率
  - 在搜索过程中可以随时得到行为的评价



# 蒙特卡洛树搜索过程



# 选择策略

- 两方面的因素
  - 对尚未充分了解的节点的探索
  - 对当前具有较大希望节点的利用


# 选择策略：多臂老虎机模型

- 1952年Robbins提出的一个统计决策模型
- 多臂老虎机
  - 多臂老虎机拥有k个拉杆，拉动每个拉杆所获得的收益遵循一定的概率且互不相关，如何找到一个策略，使得拉动拉杆获得的收益最大化。

# 信心上限算法（UCB：Upper Confidence Bound）

function UCB1

for each 拉杆j:

访问该拉杆并记录收益

end for

while 尚未达到访问次数限制 do:

计算每个拉杆的UCB1信心上界$I_j$

访问信心上界最大的拉杆

end while

# 信心上限的计算

$$I_j = \overline{X}_j + \sqrt{\frac{2\ln(n)}{T_j(n)}}$$

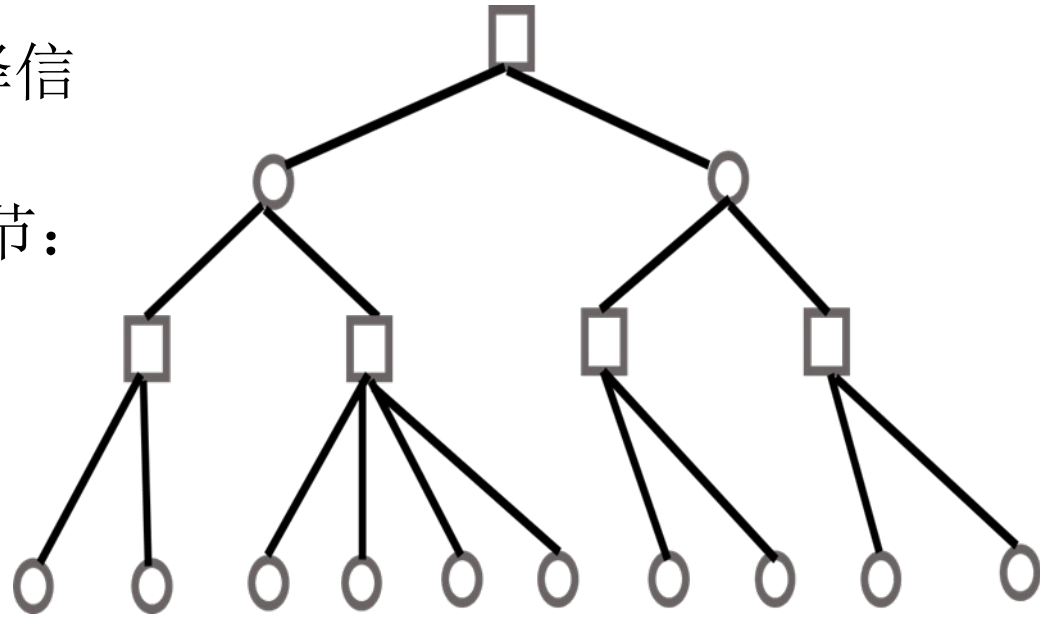
- 其中：
- $\overline{X}_j$ 是拉杆j所获得回报的均值
- n是到当前这一时刻为止所访问的总次数
- $T_j(n)$ 是拉杆j到目前为止所访问的次数

- 上式考虑了“利用”和“探索”间的平衡

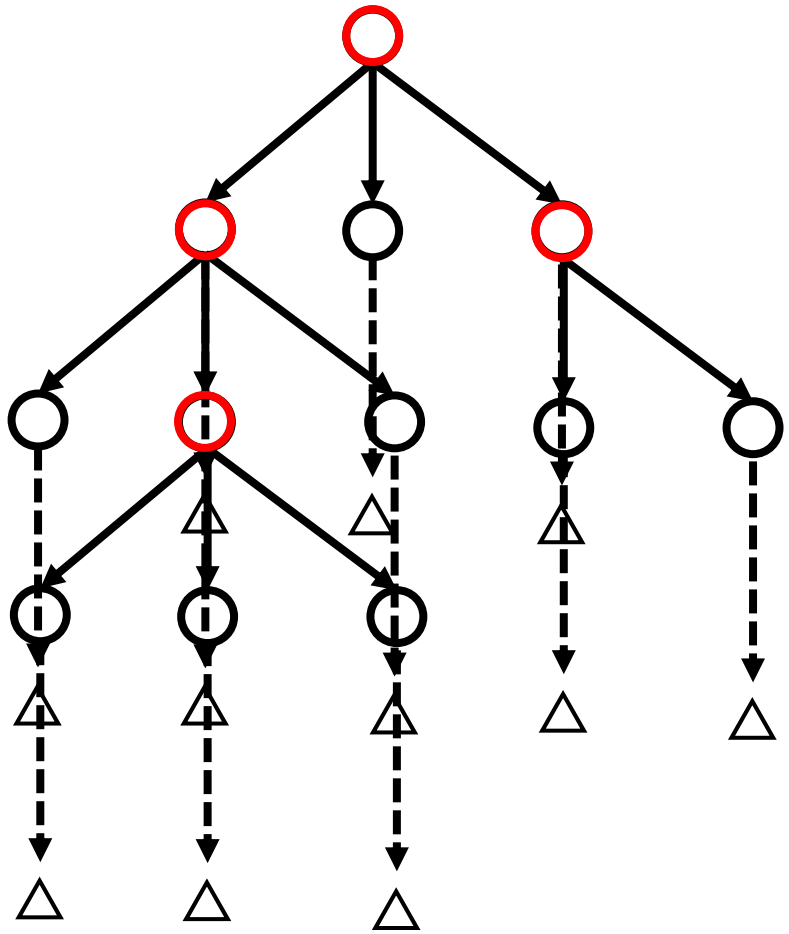
# 信心上限树算法UCT

- 将UCB1算法应用于蒙特卡洛树搜索中，用于选择可落子点
  - 节点不是随机选择，而是根据UCB1选择信心上限值最大的节点
  - 实际计算UCB1时，加一个参数c进行调节：

$$I_j = \overline{X}_j + c\sqrt{\frac{2\ln(n)}{T_j(n)}}$$



# 信心上限树算法示例



# 信心上限树算法

信心上限树算法（UCT）

function UctSearch($s_0$)

以状态$s_0$创建根节点$v_0$;

while 尚未用完计算时长 do:

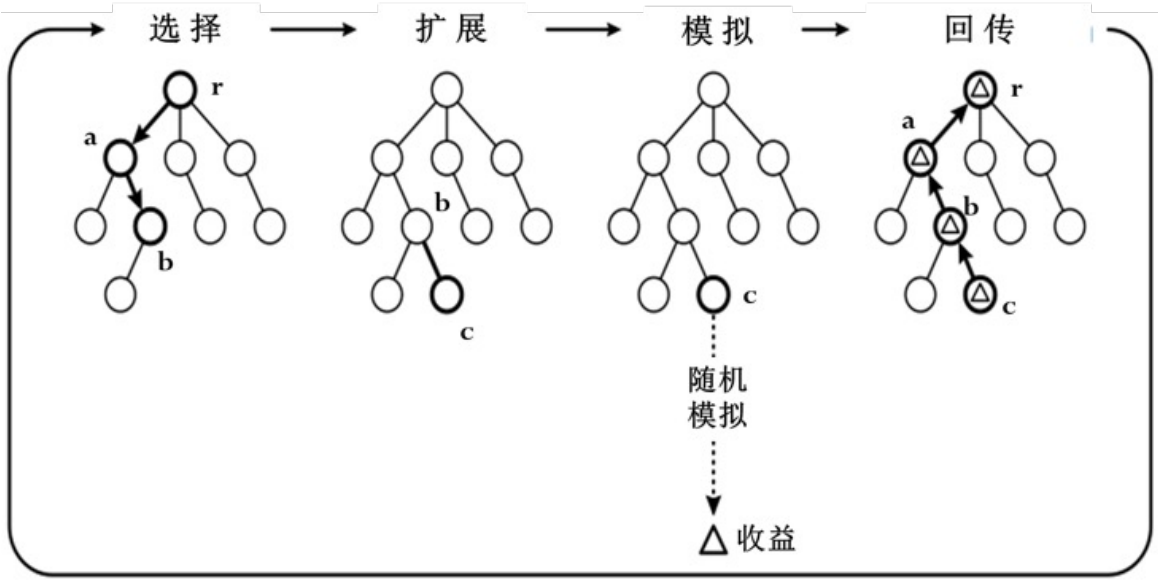
$v_l$ = TreePolicy($v_0$);

△ = DefaultPolicy($s(v_l)$);

Backup($v_l$，△);

end while

return a(BestChild($v_0$，0));

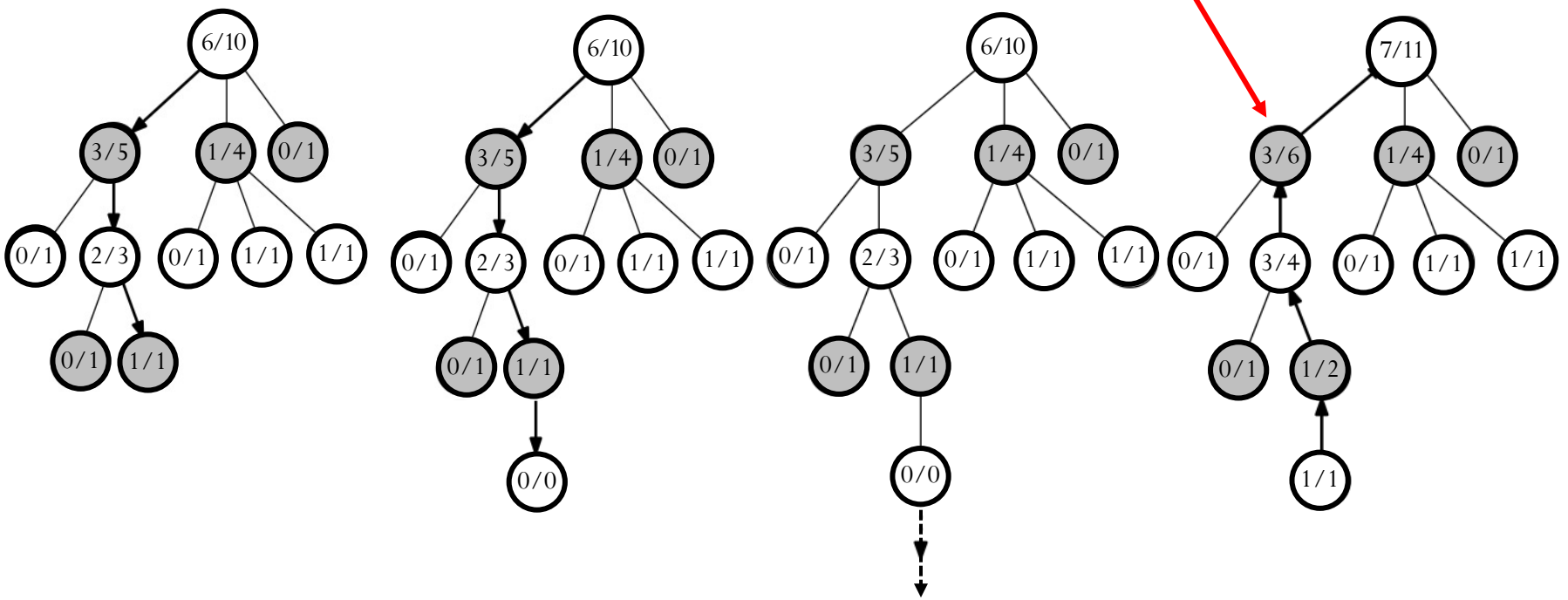



TreePolicy($v_0$)返回最适合且仍可以探索的节点（该节点还有子节点未探索）

DefaultPolicy($s(v_l)$)模拟计算$v_l$扩展后的节点收益

# UCT算法示例

重复多次后
选取胜率最
大者为走步




节点：获胜次数/模拟总次数

获胜次数是从本节点下棋的人的角度说的

模拟总次数是本节点模拟次数+子节点模拟次数

假设c=0，即 $I_j = \overline{X}_j$

$$I_j = \overline{X}_j + c\sqrt{\frac{2\ln(n)}{T_j(n)}}$$

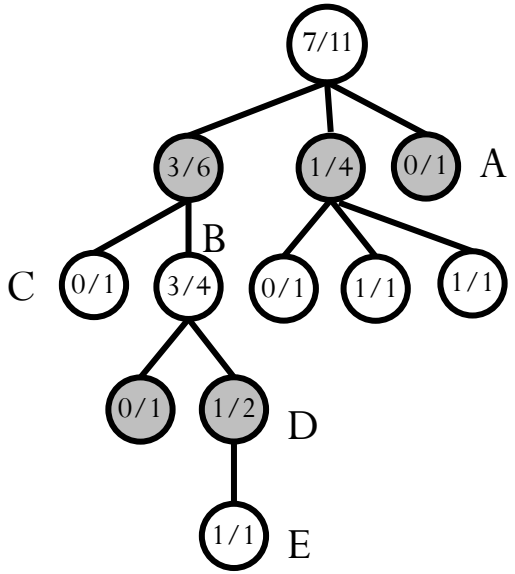
# 蒙特卡洛树搜索的效果

- 2006年将蒙特卡洛树搜索算法首次应用于计算机围棋中
- 可以达到业余4、5段的水平
- 将计算机围棋引向了正确的发展方向

# 多选题 1分

设置

如图为蒙特卡洛树搜索的一个中间状态，假设 $I_j = \overline{X}_j + c\sqrt{\frac{2\ln(n)}{T_j(n)}}$ 且c=0，可能被选择扩展的节点是：




A A节点

B B节点

C C节点

D D节点

E E节点

提交

# 多选题 1分

设置

前题中，假设选择了D扩展，并且模拟结果为负，则更新后，B和D的结果分别为：

A D为1/3

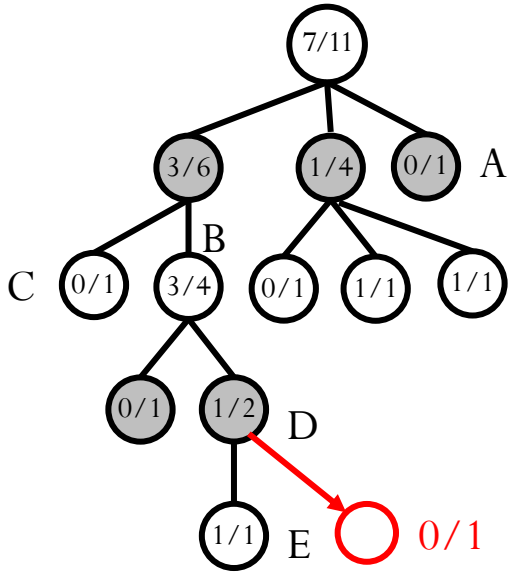
B D为2/3

C D为2/2

D B为3/5

E B为4/5

F B为3/4




提交

# 小结

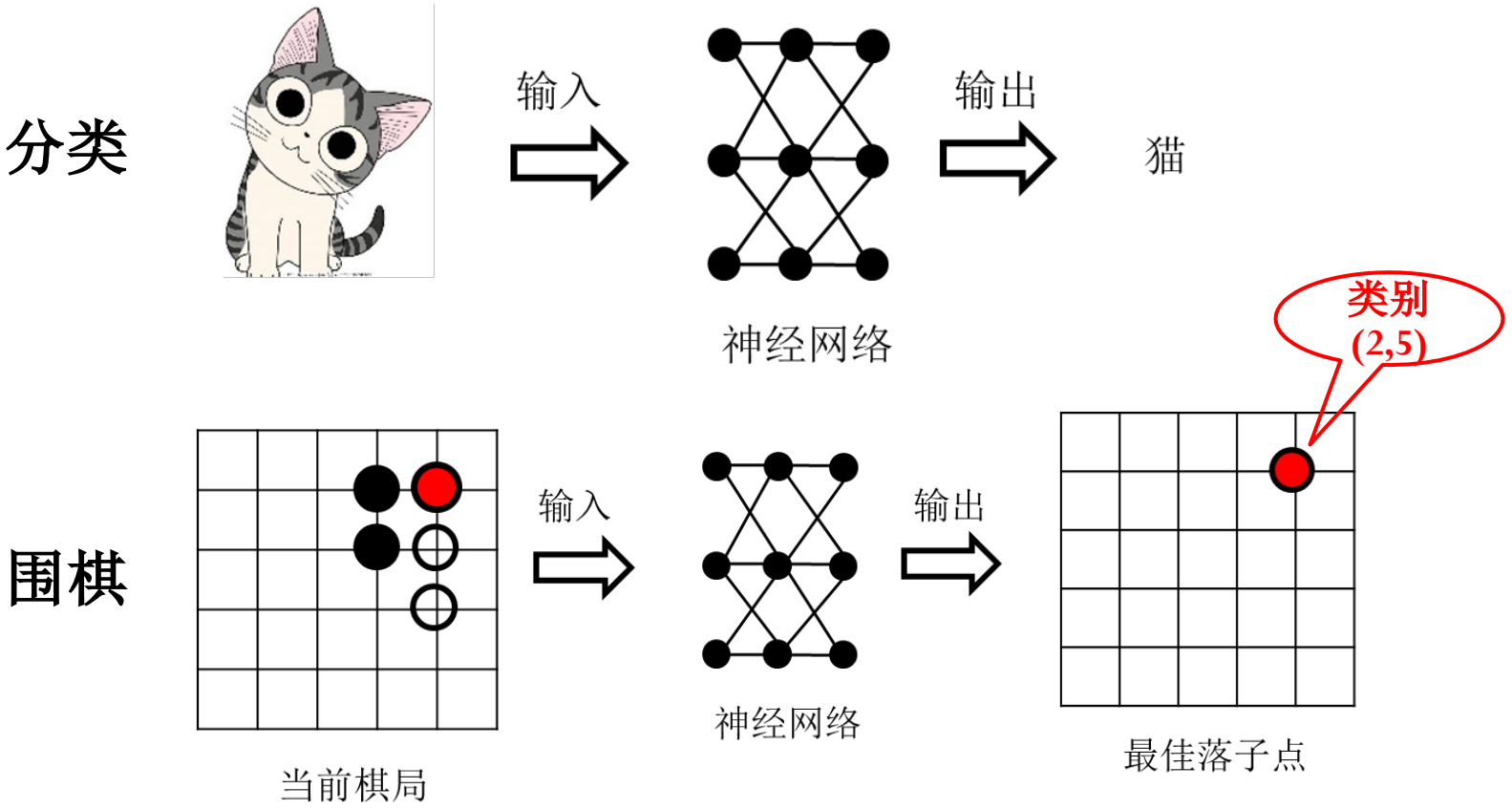
- 蒙特卡洛方法
  - 一类基于概率方法的统称
- 蒙特卡洛树搜索
  - 四个过程：选择，扩展，模拟，回传
- 多臂老虎机模型
  - 每次选择信心上限最大的
- 信心上限树算法
  - 将信心上限算法用于蒙特卡洛树搜索

# 3.5 AlphaGo原理

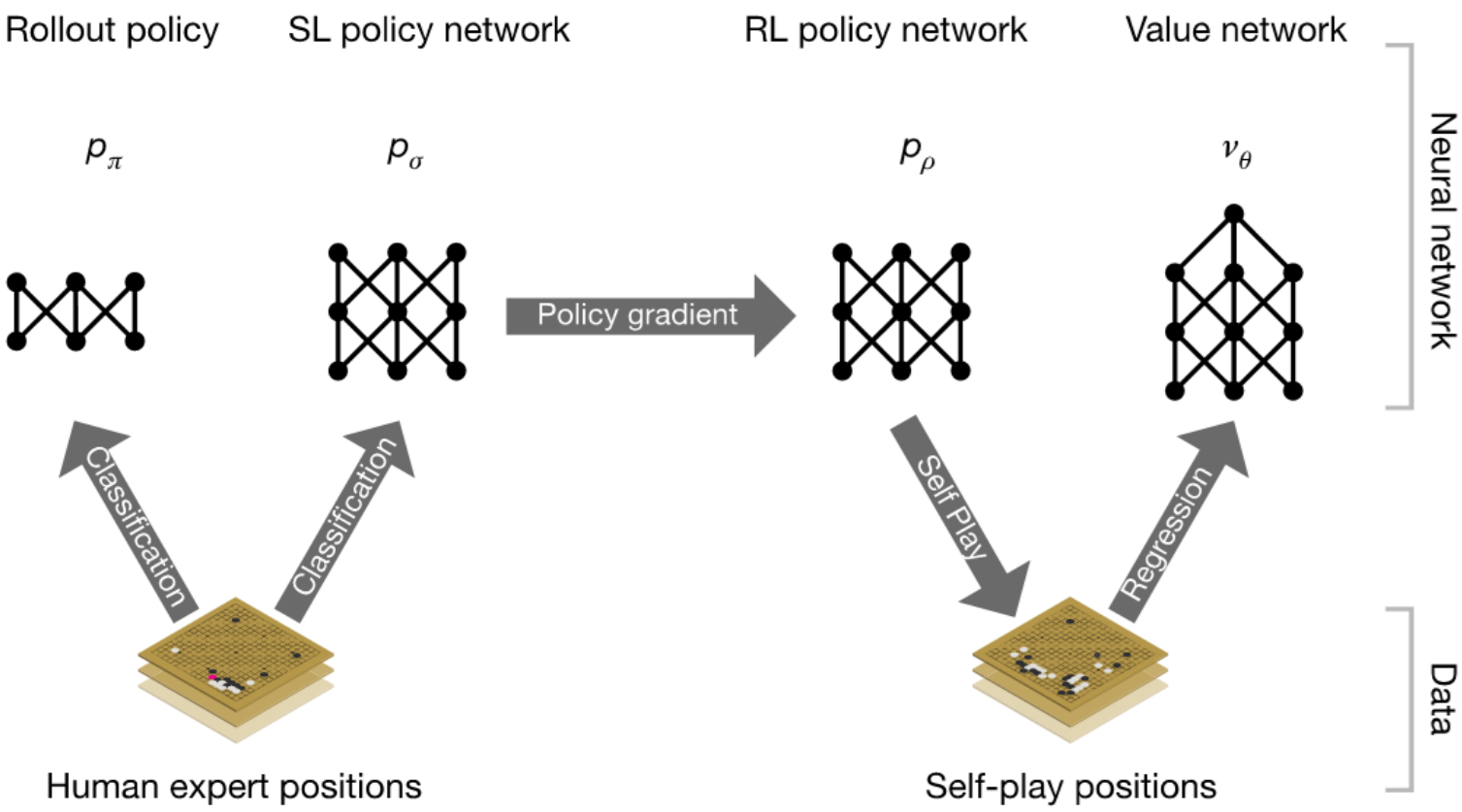
- 蒙特卡洛树搜索存在的问题
  - 生成所有子节点
  - 模拟具有盲目性
- AlphaGo将神经网络与蒙特卡洛树搜索结合在一起
  - 引入围棋的经验知识，减少盲目性
  - 缩小了搜索范围
  - 提高了模拟水平

# 如何引入经验知识?

- 把围棋类比为一个分类任务



# AlphaGo用的两类网络



# 策略网络

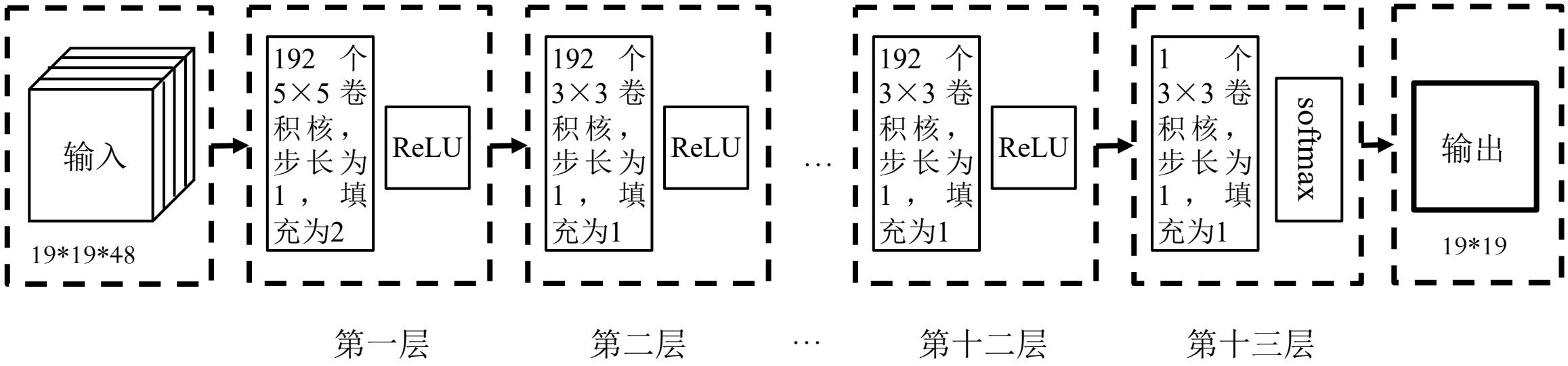
- 策略网络由一个神经网络构成
- 输入：当前棋局
  - 48个通道，每个通道大小为19*19
- 输出：棋盘上每个点的行棋概率
  - 概率越大越是好的行棋点

# 策略网络

- 输入
  - 执子颜色：3个通道，分别为执子方、对手方、空点位置
  - 壹平面：1个通道，全部填入1
  - 零平面：1个通道，全部填入0
  - 明智度：1个通道，合法落子点且不会填补本方眼位，填入1，否则为0
  - 回合数：8个通道，记录一个落子距离现在的回合数
    - 第n个通道记录到当前n个回合的落子
  - 气数：8个通道，当前落子棋链的气数
  - 动作后气数：8个通道，落子之后剩余气数

- 吃子数：8个通道，落子后吃掉对方棋子数
- 自劫争数：8个通道，落子后乙方有多少子会陷入劫争（可能会被提掉）
- 征子提子：1个通道，这个子是否会被征子提掉
- 引征：1个通道，这个子是否起到引征的作用
- 当前执子方：1个通道，当前执子为黑棋全部填1，否则全部填0
  - 该通道只用于估值网络，策略网络不使用

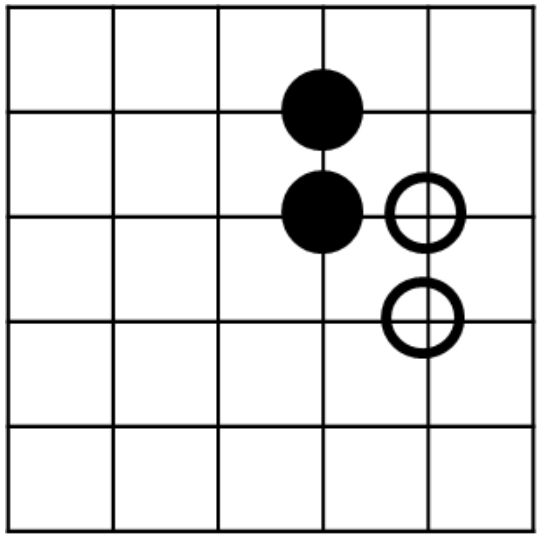
# 策略网络

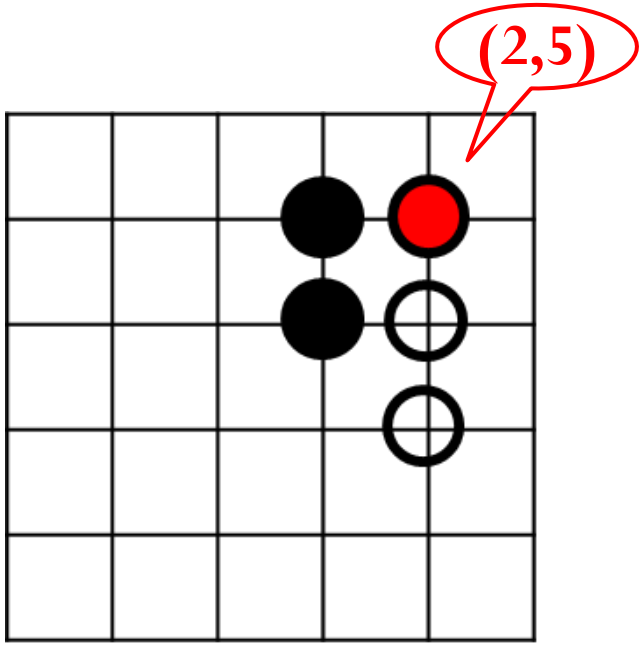

输入
19*19*48
192 个 5×5 卷积核，步长为 1，填充为2
ReLU
192 个 3×3 卷积核，步长为 1，填充为1
ReLU
…
192 个 3×3 卷积核，步长为 1，填充为1
ReLU
1 个 3×3 卷积核，步长为 1，填充为1
softmax
输出
19*19
第一层
第二层
…
第十二层
第十三层

# 策略网络的目标：像人类那样下棋

- 训练数据：
  - 16万盘人类棋手的数据
- 等效为一个分类问题
  - 任意一个棋局分类为361类之一，人类行棋点为标记

# 策略网络的目标：像人类那样下棋





# 策略网络的目标：像人类那样下棋

- 训练数据：
  - 16万盘人类棋手的数据
- 等效为一个分类问题
  - 任意一个棋局分类为361类之一，人类行棋点为标记

- 损失函数：

$$L(w) = -t_a \log(p_a)$$

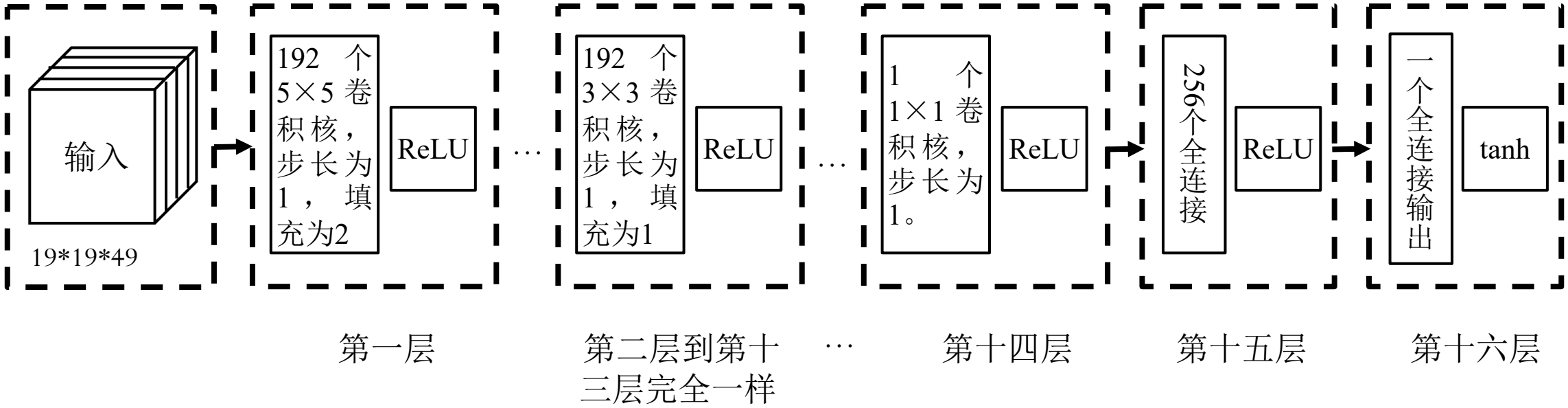
$t_a$：当前棋局下棋手落子在a处时为1，否则为0

$p_a$：策略网络在a出落子的概率

# 估值网络

- 估值网络由一个神经网络构成
- 输入：当前棋局
  - 49个通道，每个通道大小为19*19
  - 比策略网络多一个通道
- 输出：当前棋局的收益
  - 收益的取值范围为[-1, 1]

# 估值网络


输入
19*19*49
192 个5×5 卷积核，步长为1，填充为2
ReLU
192 个3×3 卷积核，步长为1，填充为1
ReLU
1 个1×1 卷积核，步长为1。
ReLU
256个全连接
ReLU
一个全连接输出
tanh
第一层
第二层到第十三层完全一样
第十四层
第十五层
第十六层

# 估值网络

- 训练样本
  - 16万人类棋手的数据
- 等效为一个回归问题
  - 获胜时收益为1，失败时收益为-1

- 损失函数

$$L(w) = (R - V(s))^2$$

R为棋局的胜负，胜为1，负为-1

V(s)为估值网络的输出，即预测的收益。

# 与蒙特卡洛树搜索融合

- MCTS中的选择原则
  - 利用：收益好的节点
  - 探索：模拟次数少的节点
- AlphaGo增加了第三个原则
  - 经验：落子概率高的节点
- 充分利用两个网络

- 节点s第i次模拟的收益

$$v_i(s) = \lambda value(s) + (1-\lambda) rollout(s)$$

其中：$value(s)$是估值网络的输出，$rollout(s)$是一次模拟结果

- 平均收益

$$Q(s_a) = \frac{\sum_{i=1}^{n} v_i(s_a)}{n}$$

其中：$s_a$为s棋局下在a处落子后的棋局

- 探索项

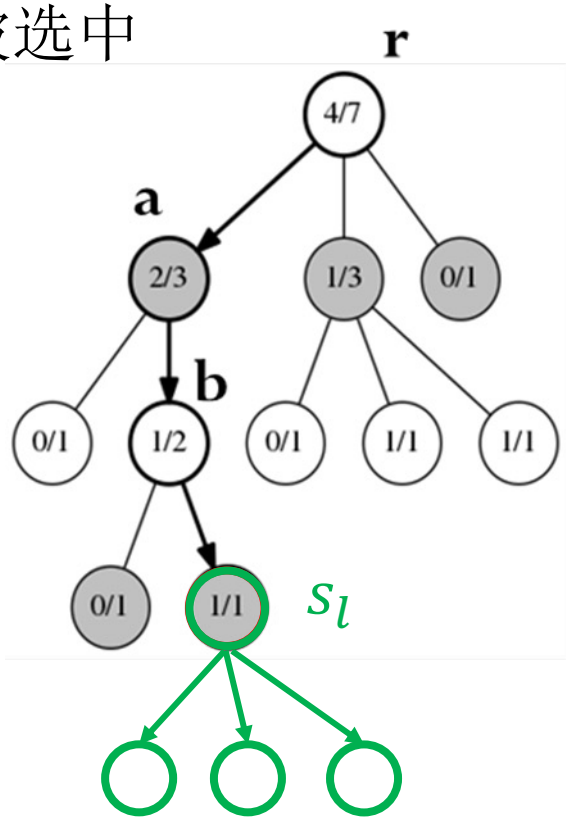
$$u(s_a) = c \cdot p(s_a) \frac{\sqrt{N(s)}}{N(s_a)+1}$$

其中：$N(\cdot)$为模拟次数、$p(s_a)$为策略网络在a出下棋的概率，c为加权系数

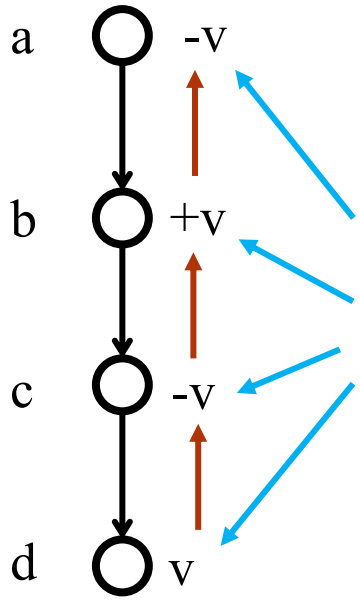
$$I_j = \bar{X}_j + c\sqrt{\frac{2\ln(n)}{T_j(n)}}$$

- 选择过程
  - 用$Q(s_a)+u(s_a)$代替信心上限$I_j$，优先选择$Q(s_a)+u(s_a)$大的子节点
  - 遇到当前书中的叶节点$s_l$结束，该节点被选中
- 生成过程
  - 生成$s_l$的所有子节点
  - 规定了最大的节点深度
- 模拟过程
  - 对$s_l$进行模拟，计算：

    $$v_i(s)=\lambda value(s)+(1-\lambda)rollout(s)$$
  - 模拟过程采用推演策略网络
    - 推演策略网络速度快是策略网络的1000倍
    - 规定了总的模拟次数



- 回传过程

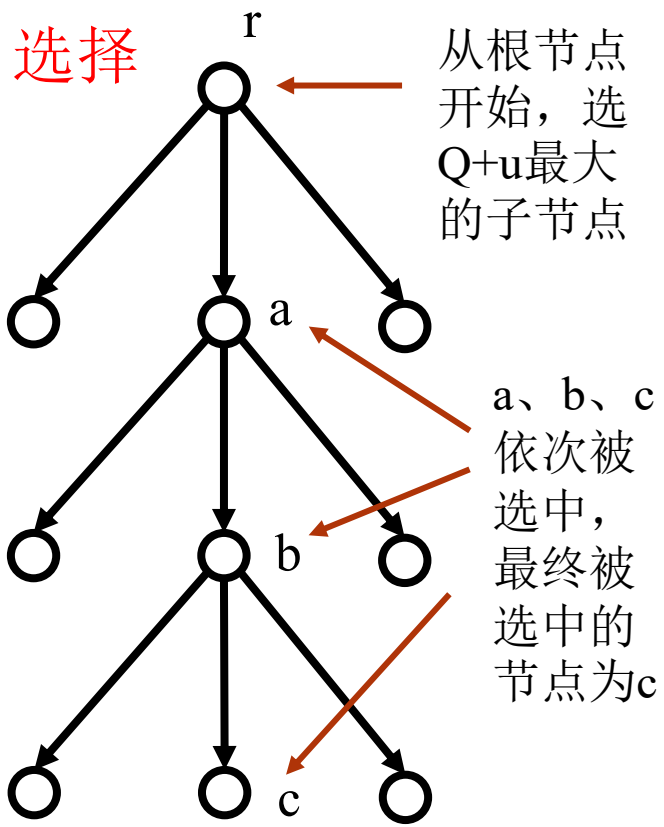



d的收益回传到其祖先节点，回传过程中要注意正负号的变化，一方的正收益对于另一方就是负的。

# AlphaGo的MCTS过程

- 每个节点记录的信息
  - 总收益
  - 行棋到该节点的概率
  - 被选择次数

## 1，选择

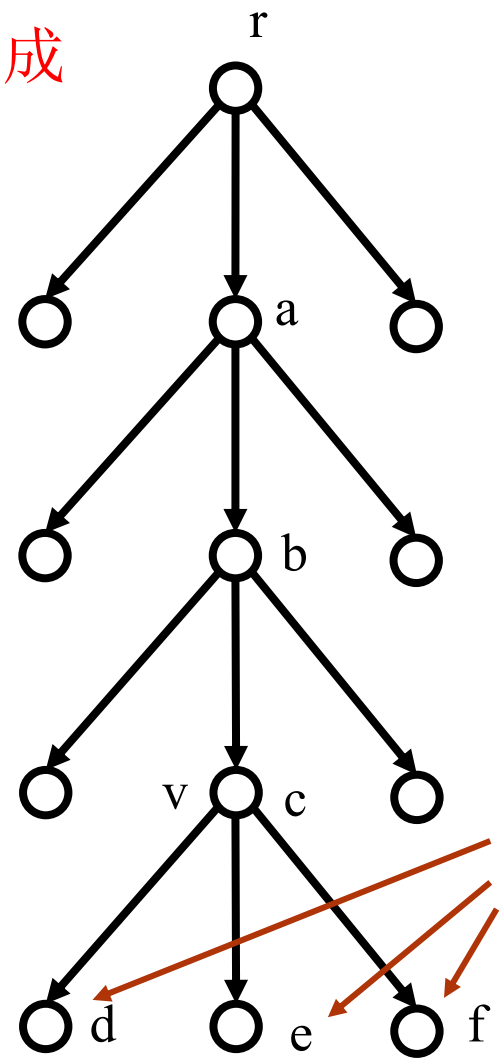



从根节点开始，选Q+u最大的子节点

a、b、c依次被选中，最终被选中的节点为c

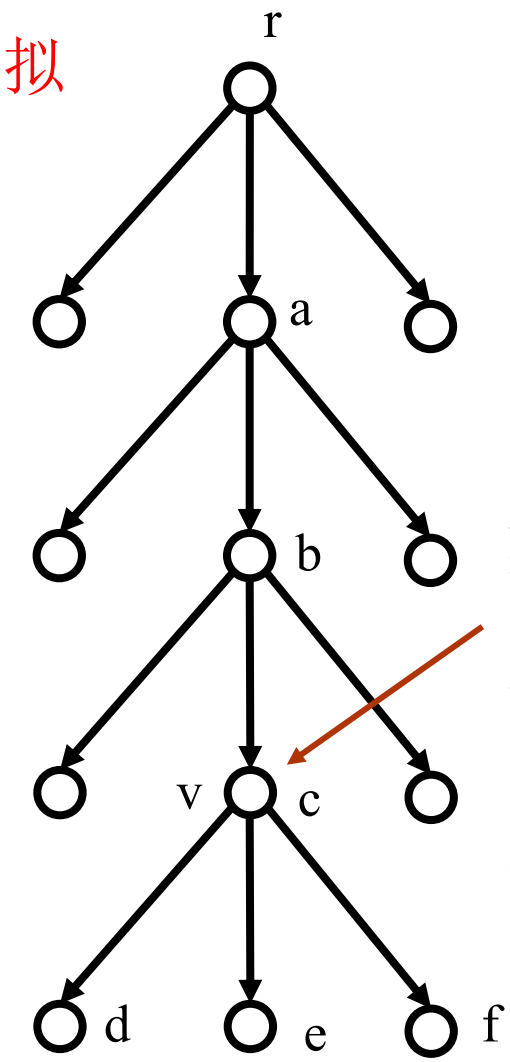
每个节点记录总收益、行棋到该节点的概率和被选择次数。

## 2，生成




生成c的子节点d、e、f，策略网络计算出从c到这些节点的概率，并设置总收益为0，选中次数为0。

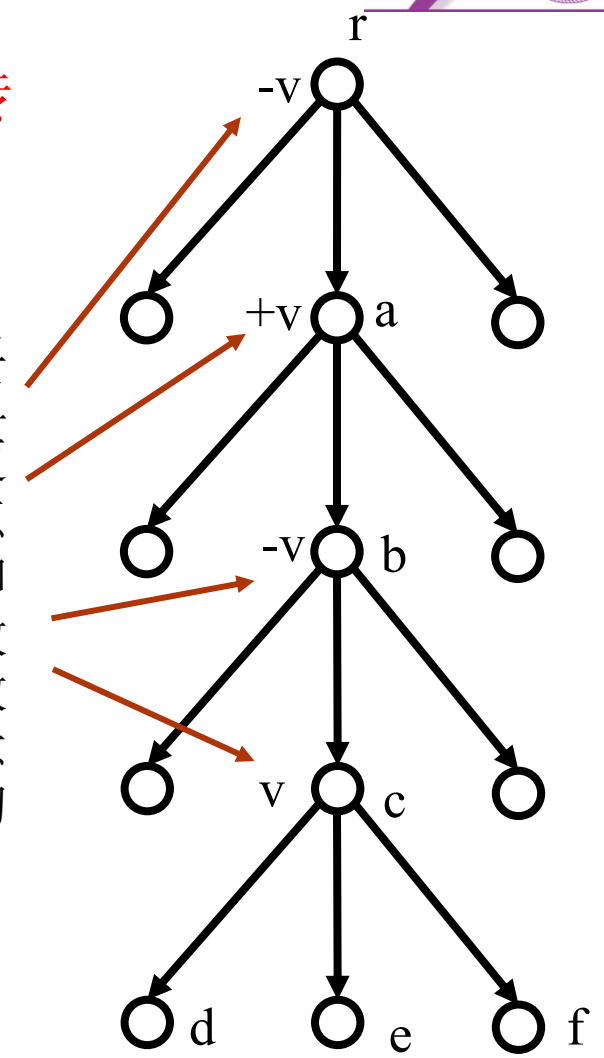
## 3，模拟




对c进行模拟，模拟结果与c的估值（由估值网络计算得到）加权平均作为c的收益v。

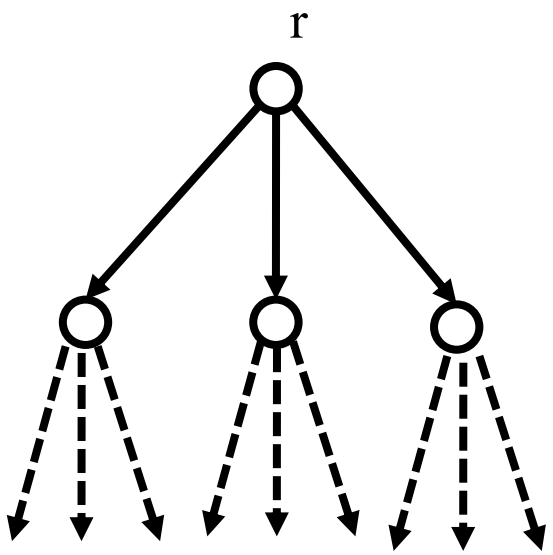
## 4，回传

v值依次上传到c的祖先节点，更新这些节点的总收益和选择次数（选择次数+1），注意正负号的切换。



# AlphaGo如何确定走步?

- 根节点子节点中被选择次数最多的节点作为最终的走步



# 小结

- AlphaGo的基本框架是MCTS
- 在MCTS中引入了策略网络和估值网络
  - 收益计算综合了估值网络的输出和模拟结果
  - 在选择过程中，考虑了行棋概率
- 利用推演策略网络进行模拟
- 限定了MCTS的最大深度
- 限定了MCTS的总模拟次数

# 3.6 围棋中的深度强化学习方法

- 从宠物训练说起

小狗

驯兽师

# 围棋中的深度强化学习方法

- 强化学习
  - 学习“做什么才能使得收益最大化”的方法
  - 学习者不会被告知如何做，必须自己通过尝试发现哪些动作会产生最大的收益
    - 监督学习与强化学习
  - 两个特征：试错和延迟收益
- 深度强化学习
  - 用深度学习（神经网络）方法实现的强化学习

# 深度强化学习的关键问题

- 如何获得指示信号

- 监督学习：情景与标注一一对应
- 强化学习：将收益转化为“标注”
  - 不能获得所有情况下既正确又有代表性的示例

- 手段：将深度强化学习问题转化为神经网络训练问题
  - 不同的转换方法构成了不同的深度学习方法
  - 关键是损失函数的定义

# 围棋中的深度强化学习方法

- 通过自己博弈训练策略网络
- 三种实现方法
  - 基于策略梯度的强化学习
  - 基于价值评估的强化学习
  - 基于演员-评价方法的强化学习

# 1，基于策略梯度的强化学习

- 数据：自我博弈产生

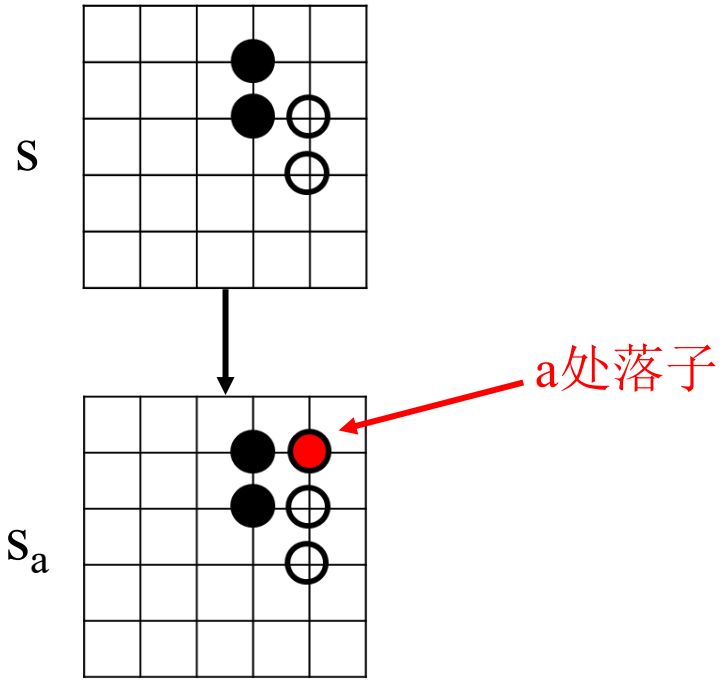
  $(s, a, p_a, t_a)$

  s：当前棋局

  a：s棋局下在a处行棋

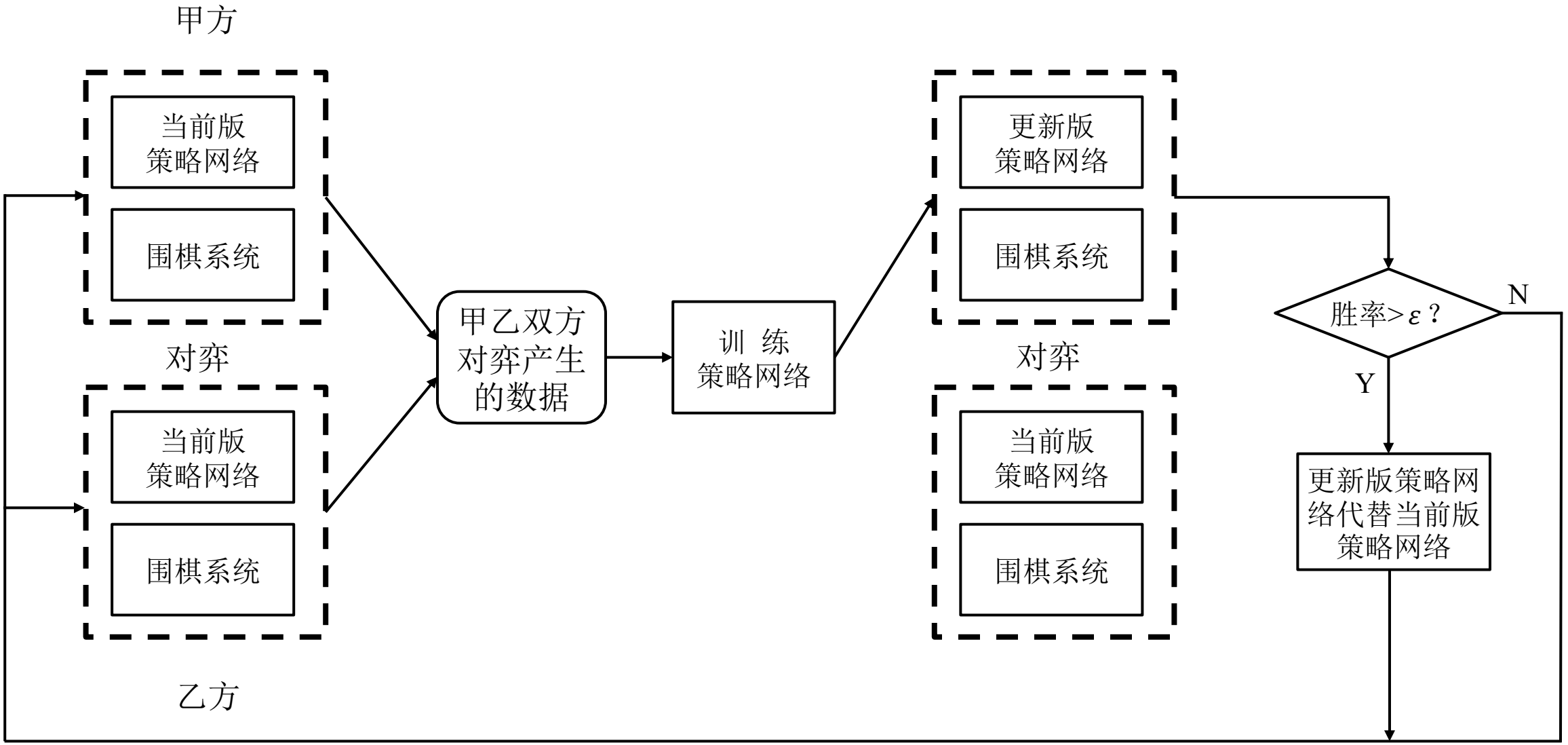
  $p_a$：s棋局下在a处行棋的获胜概率

  $t_a$：胜负值，胜为1，负为-1




- 损失函数：$L(w) = -t_a \log(p_a)$
  - 假设获胜者的行为都是正确的，负者行为都是不正确的
  - 假设获负时对权重的修改量大小与获胜时一样，方向相反

# 基于策略梯度的强化学习流程

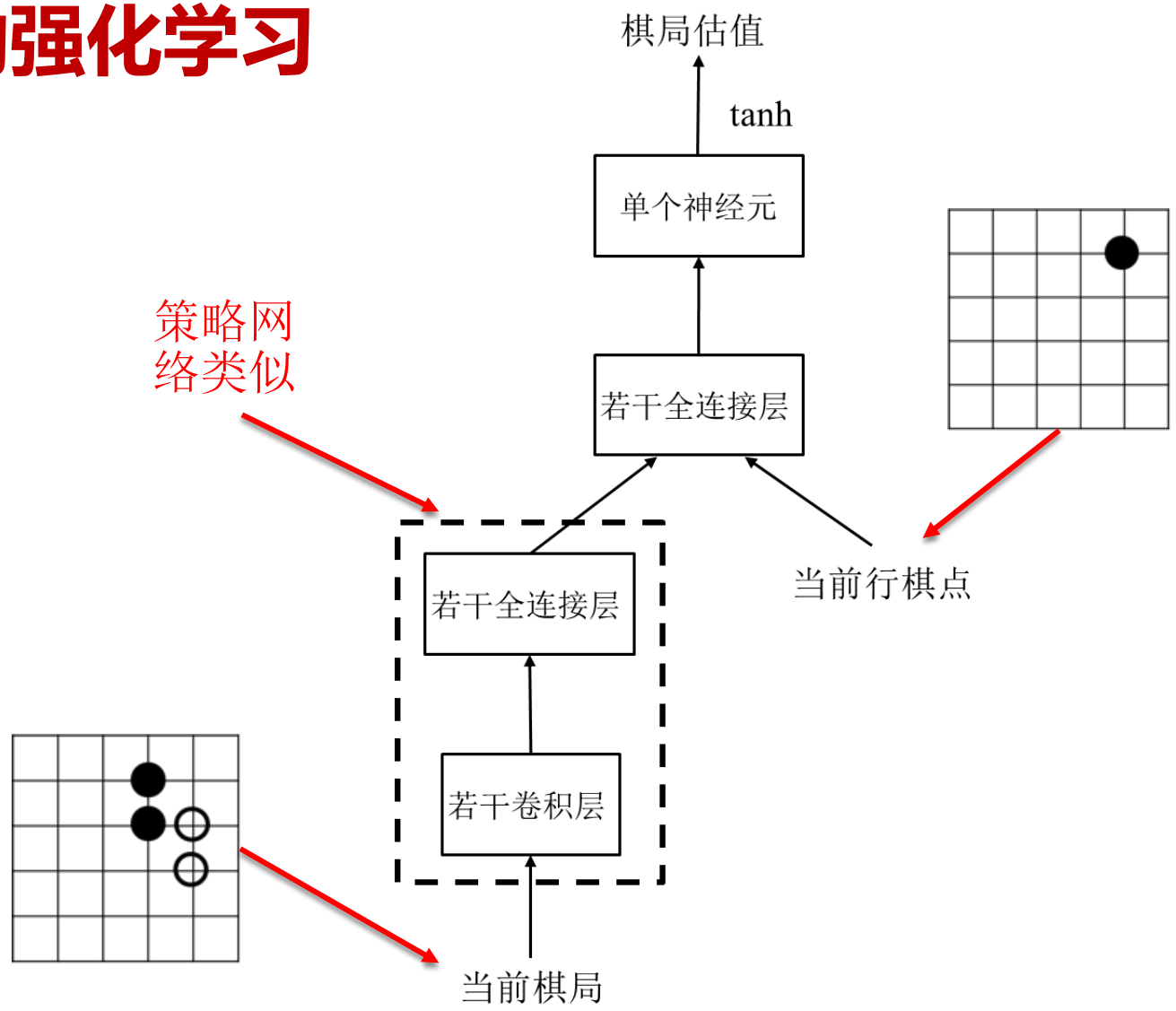


# 基于策略梯度的强化学习

- 注意点
  - 在强化学习过程中，每个样本只使用一次
  - 基于策略梯度的强化学习方法学到的是在每个可落子点行棋的获胜概率
  - （监督学习策略网络学到的是在某个可落子点行棋的概率）

# 2，基于价值评估的强化学习

- 价值评估网络
  - 对一个行棋点的价值，也就是收益进行评估
  - 输入：当前棋局和行棋点
  - 输出：取值在-1、1之间的估值



# 基于价值评估的强化学习

- 数据：自我博弈产生

  (s, a, V(s, a), R)

  s：当前棋局

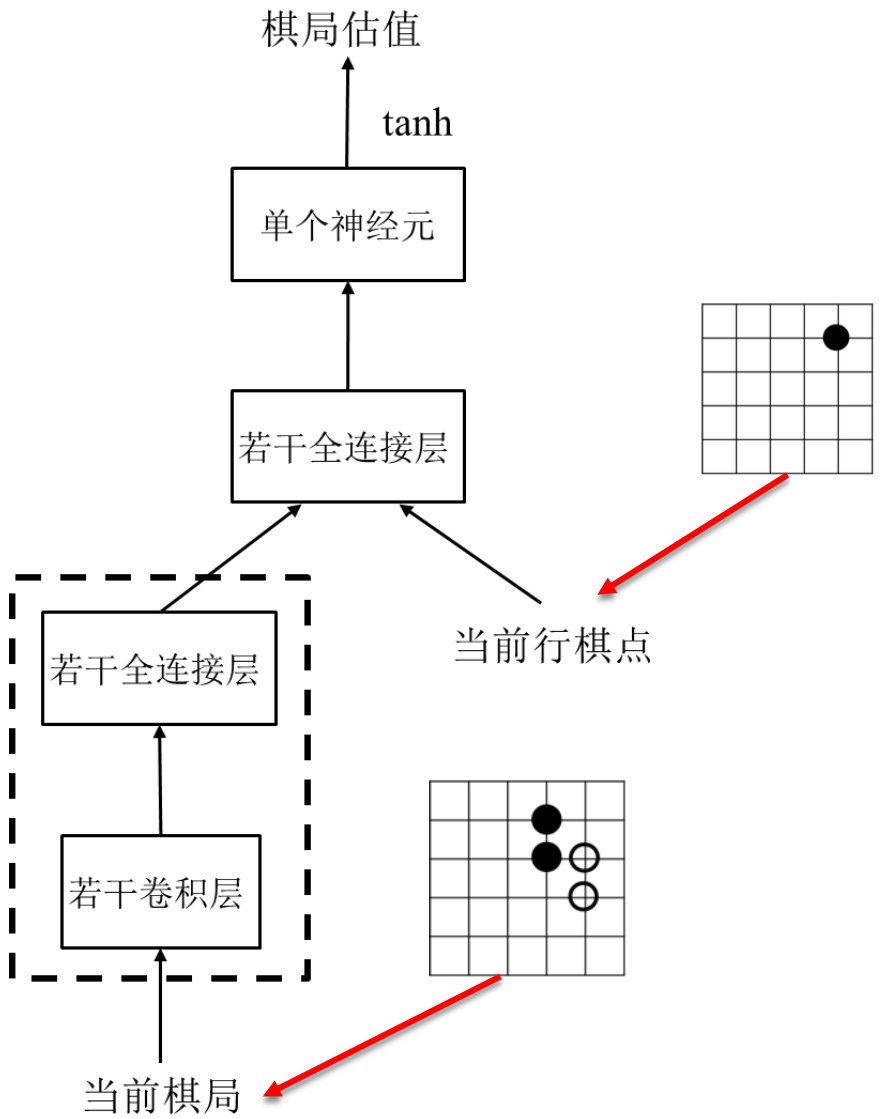
  a：s棋局下在a处行棋

  V(s, a)：棋局s下在a处落子时网络的输出

  R：胜负值，胜为1，负为-1

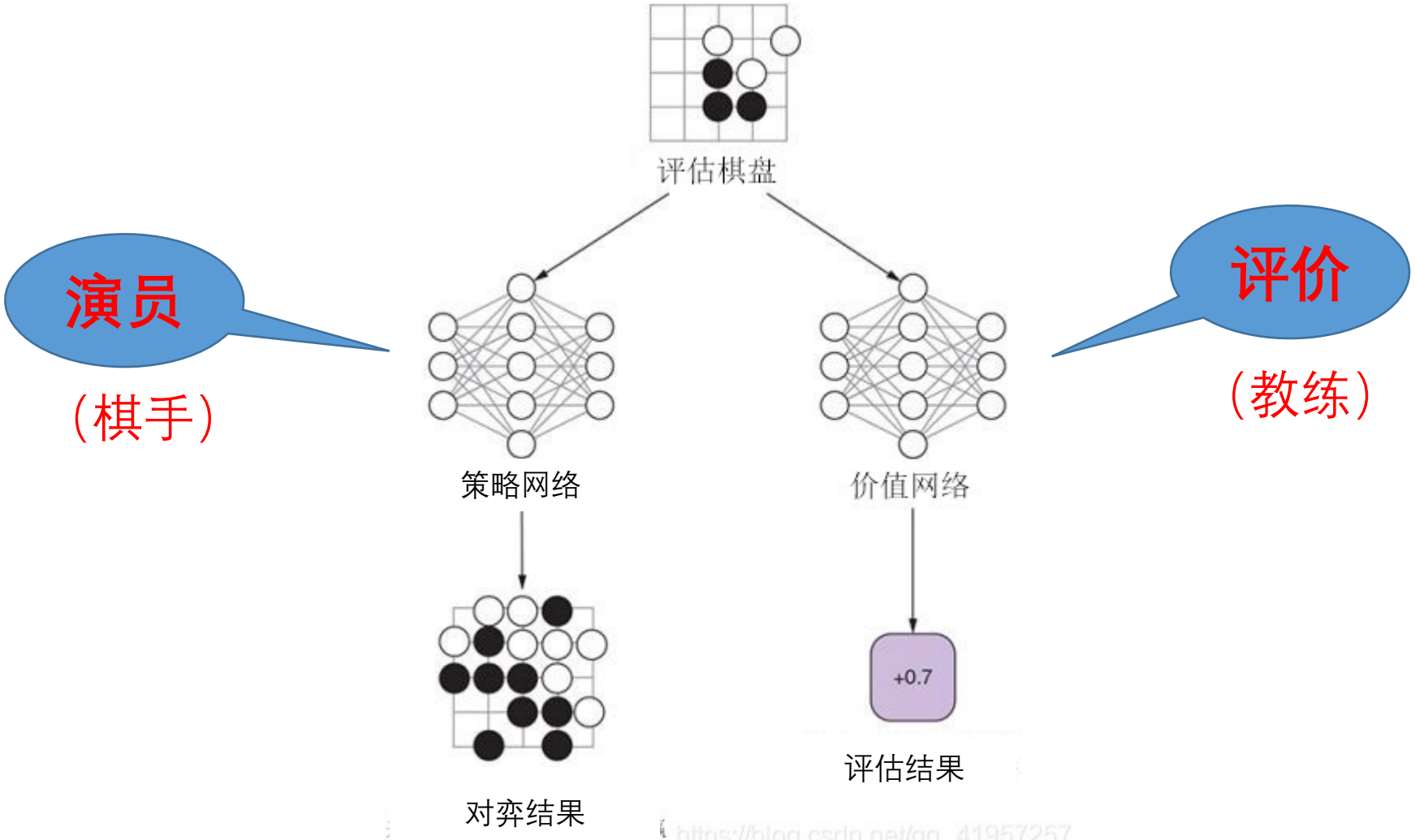
- 损失函数：

$$L(w) = (R - V(s,a))^2$$



# 3，基于演员-评价方法的强化学习

# 基于演员-评价方法的强化学习



# 基于演员-评价方法的强化学习

- 哪些是重要的？
  - 锦上添花与雪里送炭
  - 一着不慎，满盘皆输

# 基于演员-评价方法的强化学习

- 收益增量
  - 评价一步棋的好坏

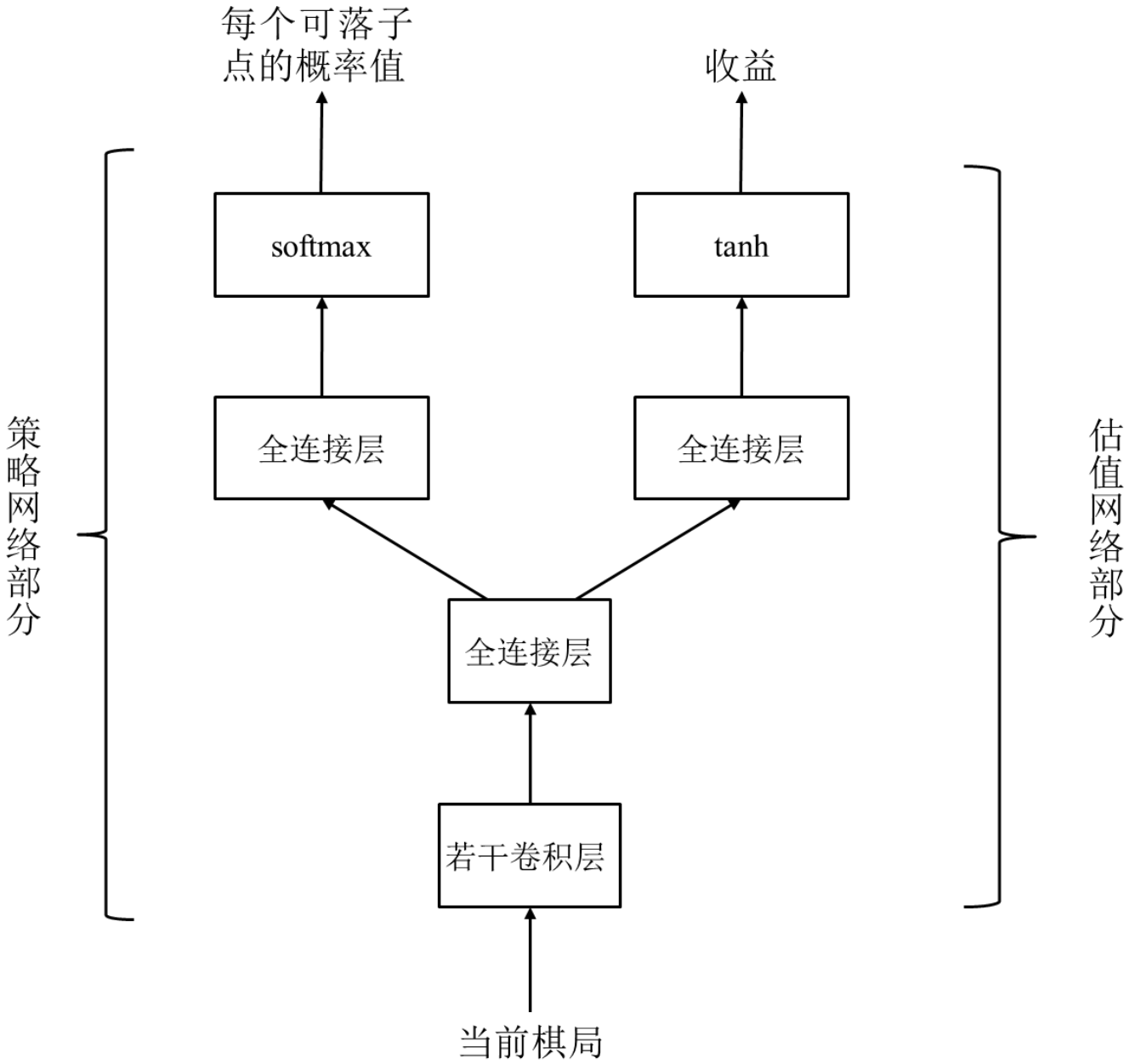
$$A = Q(s,a) - V(s)$$

  - V(s)为棋局s的预期收益，取值范围为[-1, 1]
  - Q(s,a)为在a处行棋后的收益，取值范围为[-1, 1]
  - A为收益增量，取值范围为[-2, 2]
  - A越大越说明走了一步妙招，越小越说明走了一步败招

收益增量的计算

$$A = R - V(s)$$

  - R为胜负值，胜为1，负为-1


每个可落子点的概率值
收益
softmax
tanh
全连接层
全连接层
全连接层
若干卷积层
当前棋局
策略网络部分
估值网络部分

# 基于演员-评价方法的强化学习

- 损失函数

评价部分：$L_1(w) = (R - V(s))^2$

R：为胜负值，胜为1，负为-1

$V(s)$：为棋局s的预期收益，取值范围为[-1, 1]

演员部分：$L_2(w) = -Alog(p_a)$

$p_a$：为策略网络在a处行棋的概率

A：为在a处行棋后的收益增量（$A = R - V(s)$）

综合损失函数：

$$L(w) = L_1(w) + \lambda L_2(w)$$

λ：为调节系数

# 小结

- 深度强化学习，根据任务找到合适的指示信号
- 损失函数体现了学习的内容
  - 基于策略梯度的强化，通过每局棋的胜负指导学习，学习到的是每个落子点获胜的概率
  - 基于价值评估的强化学习，通过每局棋的胜负指导学习，学习到的是每个落子点获取最大收益的概率
  - 基于演员-评价的强化学习，强调的是重要行棋点的学习，通过收益增量对走法的重要性进行评价，学习到的是每个落子点获得最大收益增量的概率

# 3.7 AlphaGo Zero原理

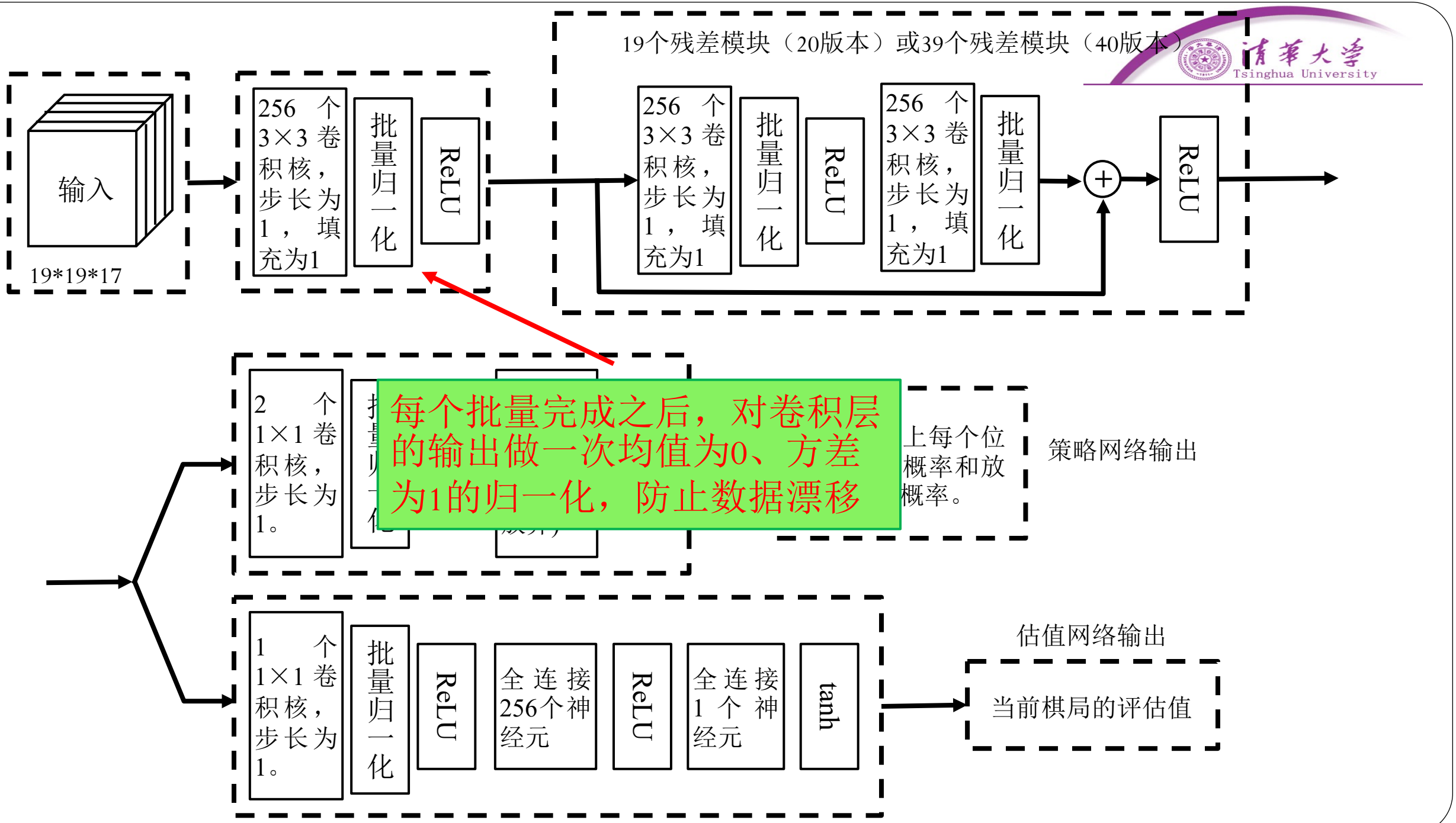
- AlphaGo Zero是AlphaGo的升级版

- 实现了从零学习
  - 不再使用人类棋手的数据
  - 不再使用人工特征作为输入
  - 利用强化学习从零学习

- 训练3天后，战胜AlphaGo Lee
- 训练40天后，战胜AlphaGo Master

# AlphaGo Zero的网络结构

- 将策略网络、估值网络合并为一个“双输出”网络
- 输入：17个通道
  - 16个通道：记录到目前为止的八个棋局，每个棋局两个通道，分别记录黑棋、白棋位置
  - 1个通道：当前行棋方为黑棋时全部填1，白棋时全部填0
- 策略网络的输出为19×19+1
  - 多了一个“放弃”行为
- 估值网络的输出为当前棋局的估值，取值范围为[-1, 1]


19个残差模块（20版本）或39个残差模块（40版本）
输入
19*19*17
256 个 3×3 卷积核，步长为1，填充为1
批量归一化
ReLU
256 个 3×3 卷积核，步长为1，填充为1
批量归一化
ReLU
256 个 3×3 卷积核，步长为1，填充为1
批量归一化
ReLU
每个批量完成之后，对卷积层的输出做一次均值为0、方差为1的归一化，防止数据漂移
2 个 1×1 卷积核，步长为1。
上每个位
概率和放
概率。
策略网络输出
1 个 1×1 卷积核，步长为1。
批量归一化
ReLU
全连接256个神经元
ReLU
全连接1个神经元
tanh
估值网络输出
当前棋局的评估值

# AlphaGo Zero中的MCTS

AlphaGo Zero

- 节点s第i次模拟的收益（AlphaGo）

$$v_i(s) = \lambda value(s) + (1-\lambda) rollout(s) \Longrightarrow v_i(s) = value(s)$$

其中：$value(s)$是估值网络的输出，$rollout(s)$是一次模拟结果

- 平均收益

$$Q(s_a) = \frac{\sum_{i=1}^{n} v_i(s_a)}{n}$$

其中：$s_a$为s棋局下在a处落子后的棋局

- 探索项

$$u(s_a) = c \cdot p(s_a) \frac{\sqrt{N(s)}}{N(s_a)+1}$$

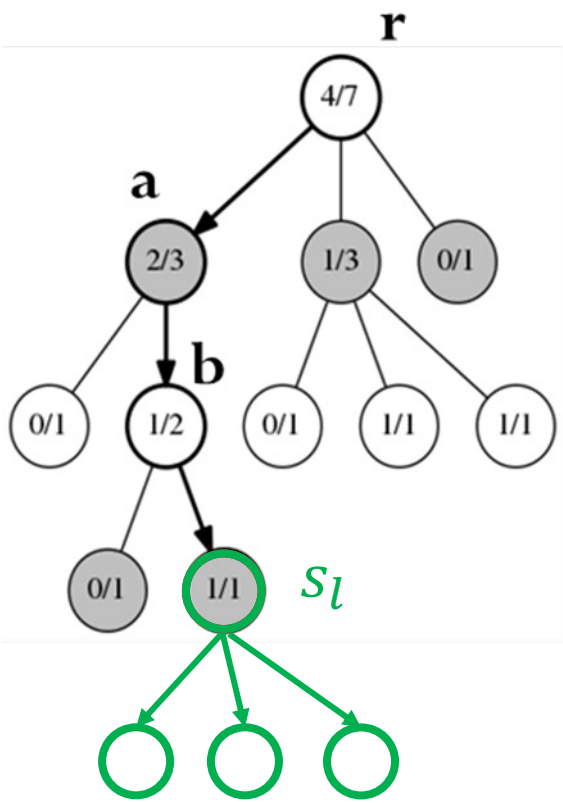
其中：$N(\cdot)$为模拟次数、$p(s_a)$为策略网络在a出下棋的概率，c为加权系数

$$Q(s_a) + u(s_a)$$

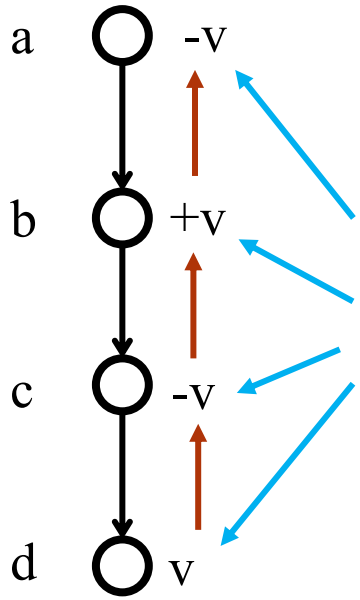
- 选择过程
  - 用$Q(s_a)+u(s_a)$代替信心上限$I_j$，优先选择$Q(s_a)+u(s_a)$大的子节点
  - 遇到叶节点$s_l$结束，该节点被选中
- 生成过程
  - 生成$s_l$的所有子节点
  - 规定了最大的节点深度
- 模拟过程
  - 用估值网络输出取代模拟过程：

$$v_i(s_l)=value(s_l)$$

  - 规定了总的模拟次数

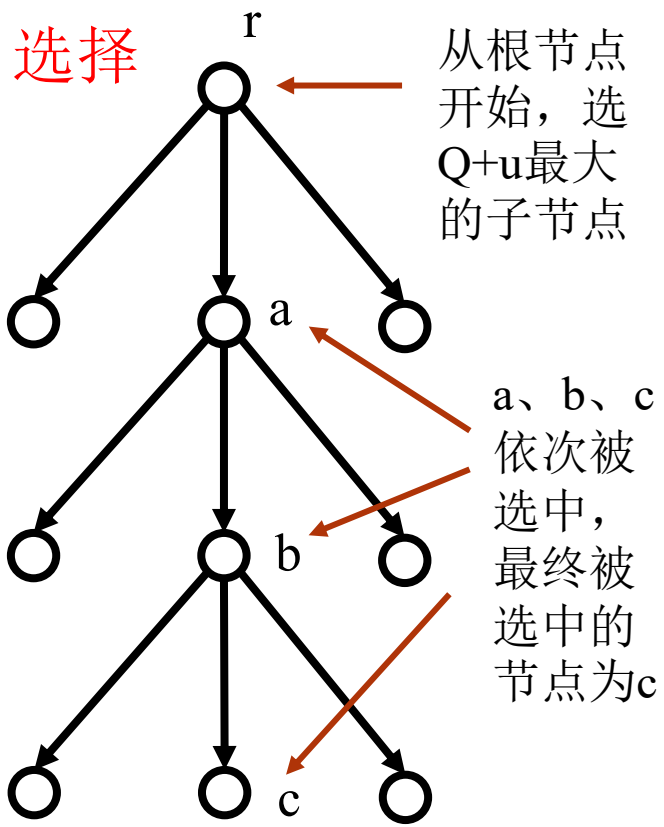


- 回传过程




d的收益回传到其祖先节点，回传过程中要注意正负号的变化，一方的正收益对于另一方就是负的。

## 1，选择

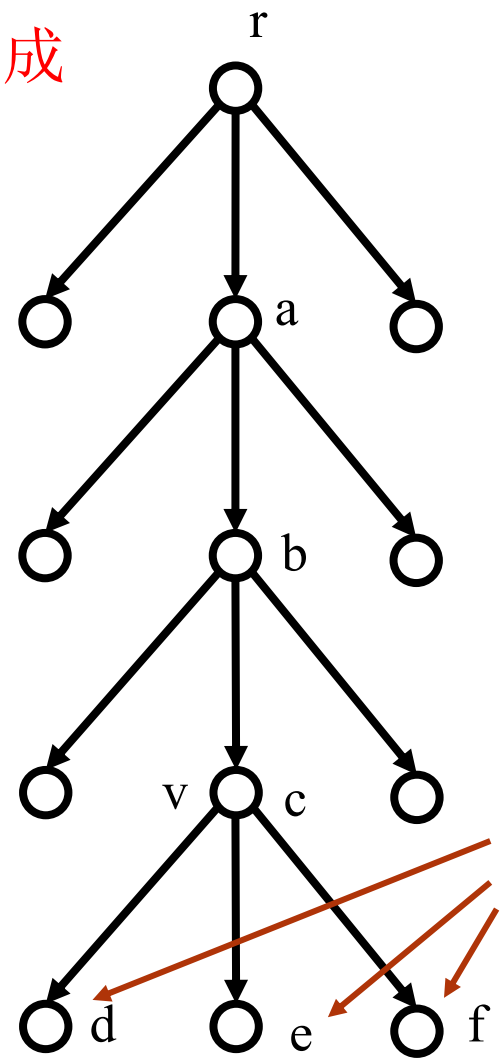



从根节点开始，选Q+u最大的子节点

a、b、c依次被选中，最终被选中的节点为c

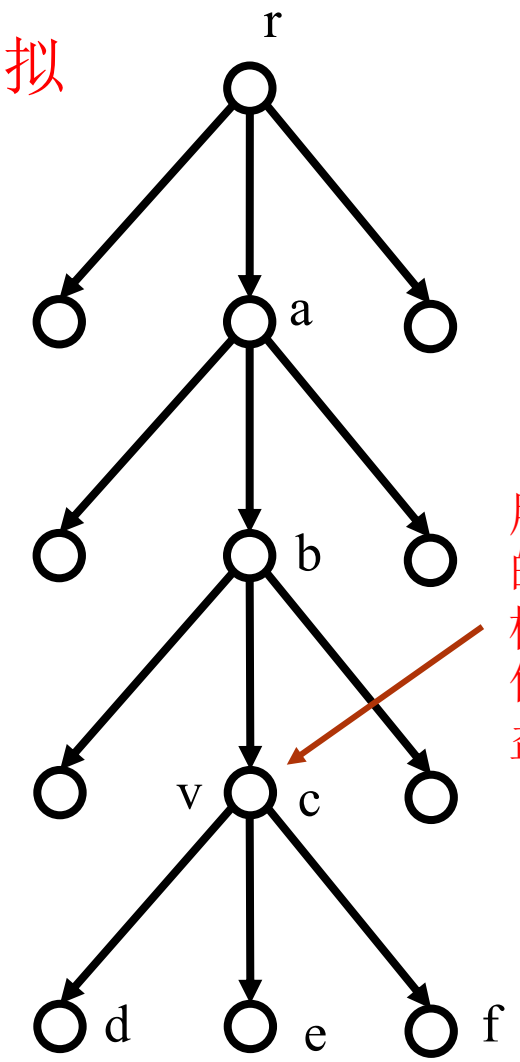
每个节点记录总收益、行棋到该节点的概率和被选择次数。

## 2，生成




生成c的子节点d、e、f，策略网络计算出从c到这些节点的概率，并设置总收益为0，选中次数为0。

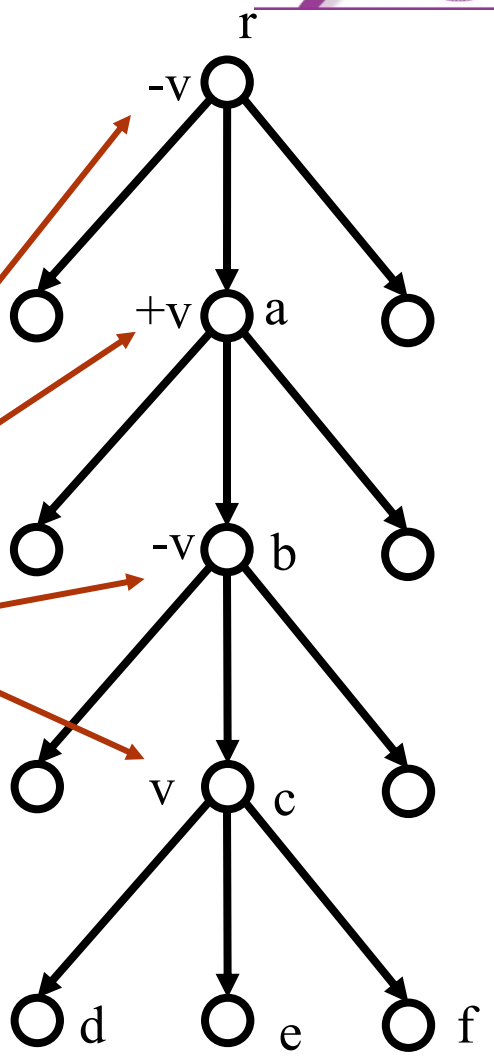
## 3，模拟




用估值网络的输出代替模拟结果，作为c的收益v。

## 4，回传

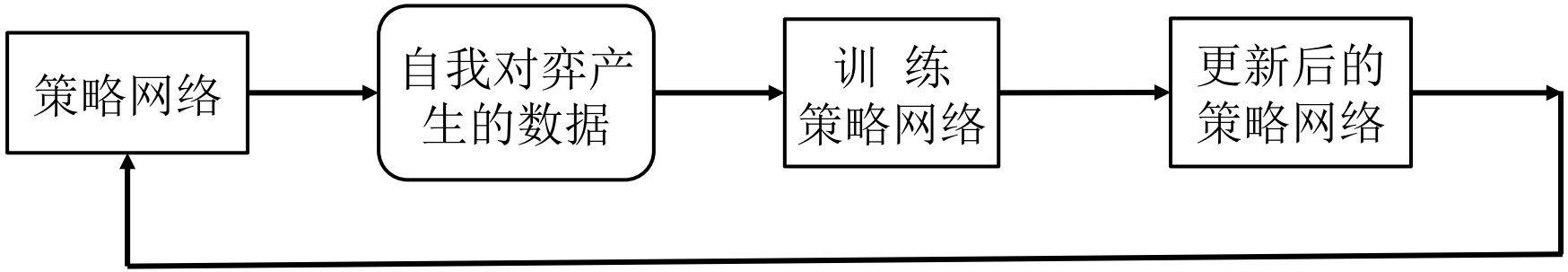
v值依次上传到c的祖先节点，更新这些节点的总收益和选择次数（选择次数+1），注意正负号的切换。



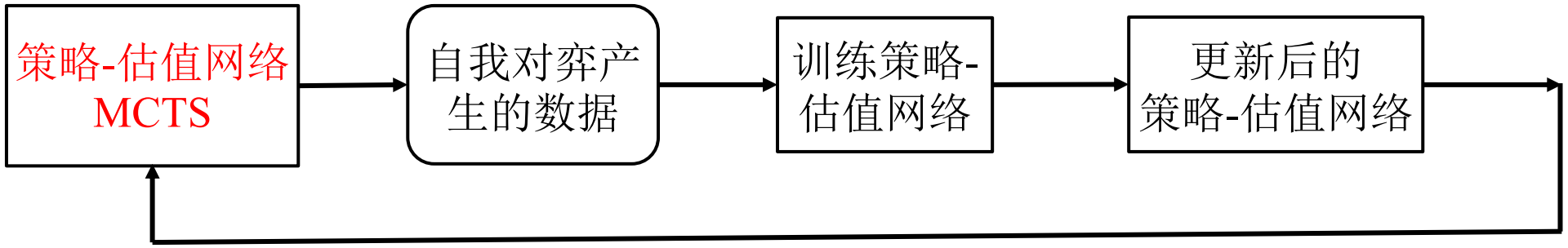
# AlphaGo Zero中的深度强化学习

- 将MCTS结合到深度强化学习中

AlphaGo
的强化学习




AlphaGo Zero
的强化学习



- 损失函数

估值网络： $L_1 = (z - v)^2$

其中：获胜时z为1，失败时为-1，v为估值网络的输出

策略网络： $L_2 = -\pi_1 \log(p_1) - \pi_2 \log(p_2) - \cdots - \pi_{362} \log(p_{362})$

其中：$\pi_i$为MCTS给出的每个落子点的概率（含放弃）

$p_i$为策略网络输出的每个落子点的概率（含放弃）

- 总损失函数： $L = L_1 + L_2 + \|\theta\|_2^2$

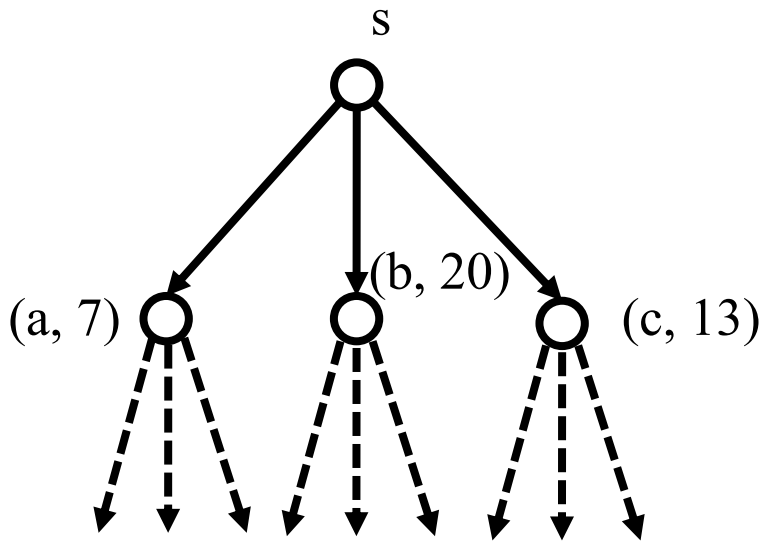
- 如何获得$\pi_i$?
- 选中次数转化为概率

$$\pi(a) = \frac{7}{7+20+13} = 0.175$$

$$\pi(b) = \frac{20}{7+20+13} = 0.5$$

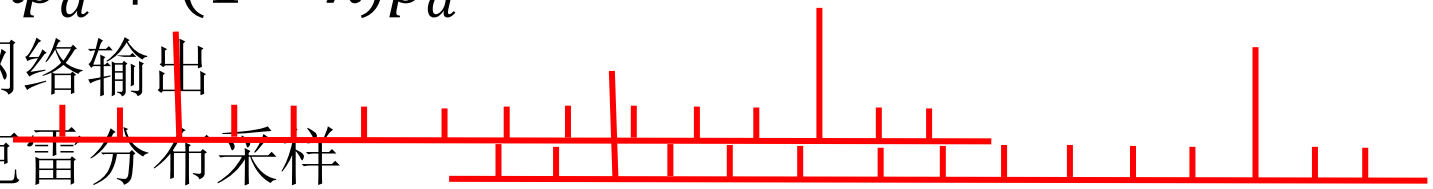
$$\pi(c) = \frac{13}{7+20+13} = 0.325$$

- 对选择次数进行优化



# 引入多样性

- 防止走向错误的方向，人为引入噪声
  - 对策略网络的输出增加噪声
- 狄利克雷分布
  - 通过参数可以产生一些符合一定条件的概率分布
  - 控制参数：
    - n：概率分布向量的长度
    - α：分布浓度
  - 当α比较小时，产生的概率分布多数为0，少数比较大
- 落子概率：$\lambda p_a + (1-\lambda)p_d$
  - $p_a$：策略网络输出
  - $p_d$：狄利克雷分布采样

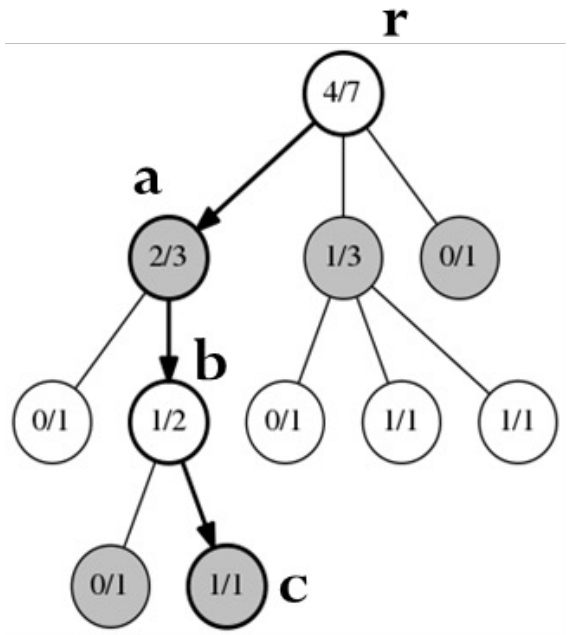


# 探索的合理性

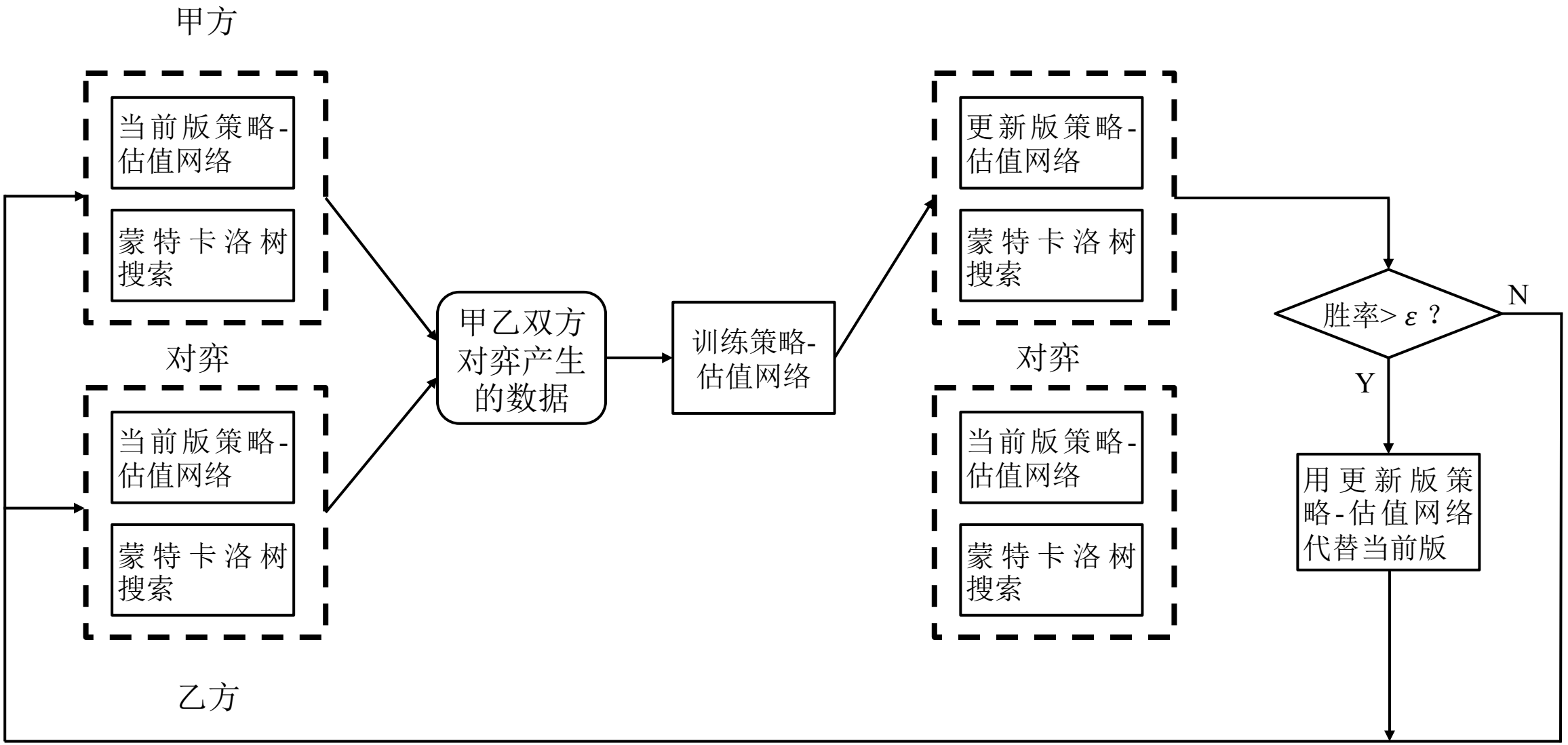
- 引入噪声会引起“不良反应”吗
- MCTS的“纠错”能力

$$Q(s_a) = \frac{\sum_{i=1}^{n} v_i(s_a)}{n}$$

$$u(s_a) = c \cdot p(s_a) \frac{\sqrt{N(s)}}{N(s_a)+1}$$



# AlphaGo Zero的强化学习过程



# 硬件构成

- 早期的AlphaGo： 176个GPU
- AlphaGo Lee： 48个TPU+多台服务器
- AlphaGo Zero： 4个TPU+一台服务器
  - 训练3天后，可以战胜AlphaGo Lee
  - 训练40天后，可以战胜AlphaGo Master

# 小结

- AlphaGo Zero从零学习
  - 不再使用人类棋手的数据
  - 不再使用人类提供的特征
  - “放弃”也是通过学习得到
- 将策略网络和估值网络融合为一个网络
- 在MCTS中舍弃了模拟过程，用估值结果代替
- 将MCTS结合到深度强化学习中

# 3.8 总结

- 通过分钱币问题引出了计算机下棋问题
- 极小-极大模型
- α-β剪枝算法
- 蒙特卡洛树搜索
- AlphaGo原理
- 深度强化学习
- AlphaGo Zero