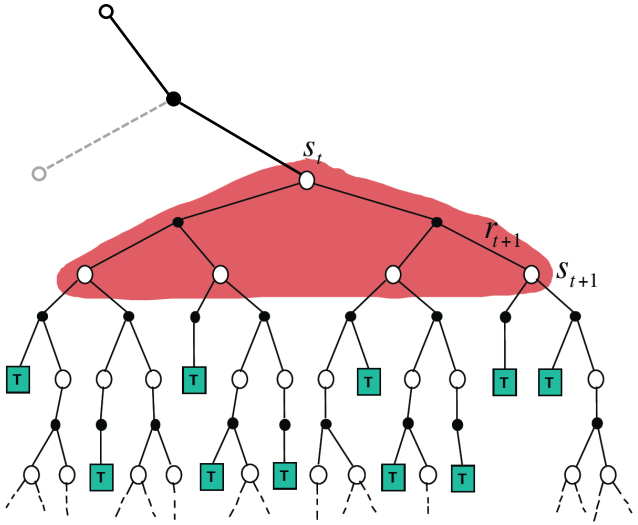
规划与学习: 决策时规划

01

实时动态规划

实时动态规划

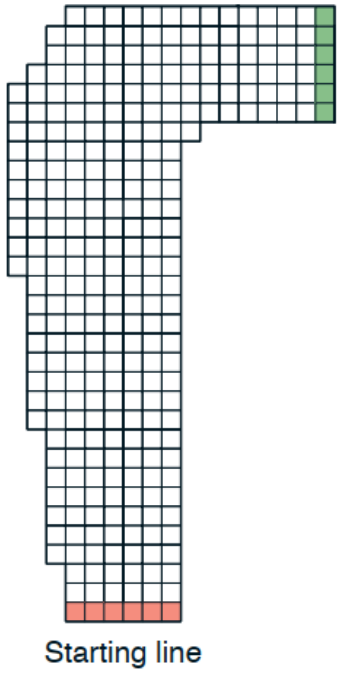
- 和传统动态规划的区别
  - 实时的轨迹采样
  - 只更新轨迹访问的状态值

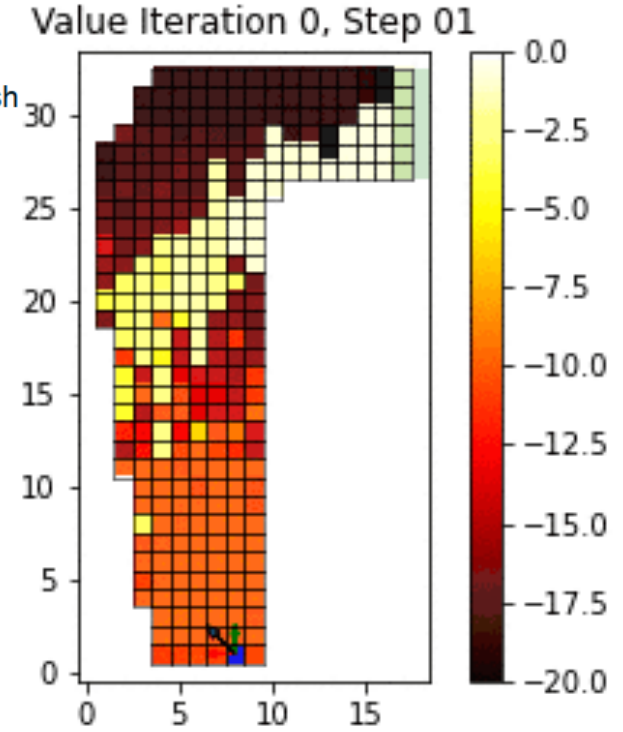


- 优势
  - 能够跳过策略无关的状态
  - 在解决状态集合规模大的问题上具有优势
  - 满足一定条件下可以以概率1收敛到最优策略

实时动态规划(RTDP)

- 跑道问题（Racetrack）
- 环境:
  - 任务: 从起点跑到终点
  - 状态: 二维坐标、二维速度
  - 动作: 每维速度的+1，-1，不变
- 结果:
  - 可到达状态:
    - 随机策略: 9115
    - 最优策略: 599
  - 更新次数少了一半





	DP	RTDP
Average computation to convergence	28 sweeps	4000 episodes
Average number of updates to convergence	252,784	127,600
Average number of updates per episode	—	31.9
% of states updated $\leq$ 100 times	—	98.45
% of states updated $\leq$ 10 times	—	80.51
% of states updated 0 times	—	3.18

https://towardsdatasolving-racetrack-in-reinforcement-learning-using-monte-carlo-control-bdee2aa4f04escience.com/

02

决策时规划

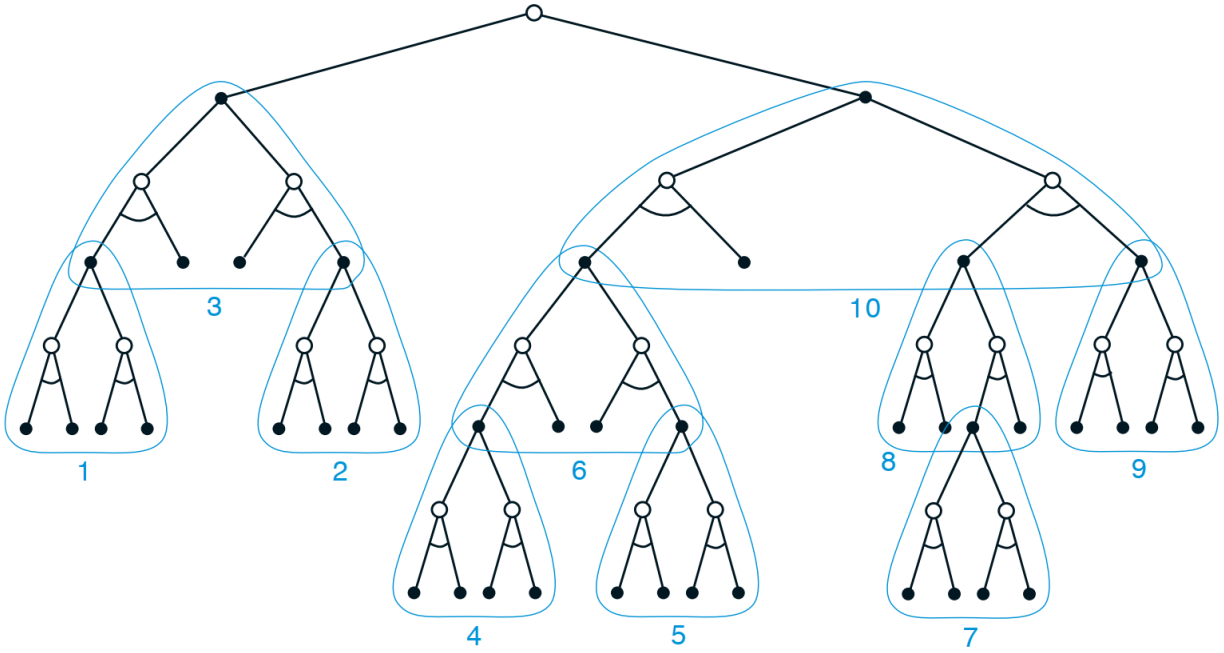
决策时规划

- 背景规划（Background Planning）
  - 规划是为了更新很多状态值供后续动作的选择
  - 如动态规划，$Dyna$

- 决策时规划（Decision-time Planning）
  - 规划只着眼于当前状态的动作选择
  - 在不需要快速反应的应用中很有效，如棋类游戏

启发式搜索

- 访问到当前状态(根节点)，对后续可能的情况进行树结构展开
- 叶节点代表估计的值函数
- 回溯到当前状态(根节点)，方式类似于值函数的更新方式



启发式搜索

- 决策时规划，着重于当前状态
- 贪婪策略在单步情况下的扩展
  - 启发式搜索看多步规划下，当前状态的最优行动
- 搜索越深，计算量越大，得到的动作越接近最优
- 性能提升不是源于多步更新，而是源于专注当前状态的后续可能

Rollout算法

- 从当前状态进行模拟的蒙特卡洛估计
- 选取最高估计值的动作
- 在下一个状态重复上述步骤

特点

- 决策时规划，从当前状态进行$rollout$
- 直接目的类似于策略迭代和改进，寻找更优的策略
- 表现取决于蒙特卡洛方法估值的准确性

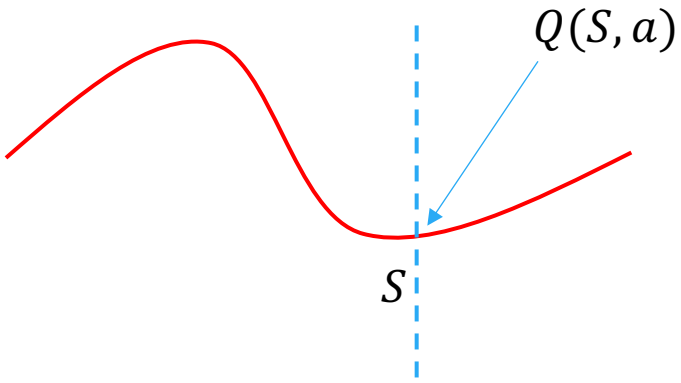
Rollout算法

时间复杂度

- $Time = \sum_{tr=1}^{N} \sum_{st=1}^{K} [\sum_{a=1}^{A} T_{eval}(S(st), a) + T_{choose}(A)]$
  - A: 决策的动作空间
  - K: $rollout$ 一个轨迹的平均步数
  - $T_{eval}(S(st), a)$: 在第 st 步下，估计 (s, a) 值函数的时间
  - $T_{choose}(A)$: $rollout$ 每步做出决策的时间
  - N: $rollout$ 轨迹的次数

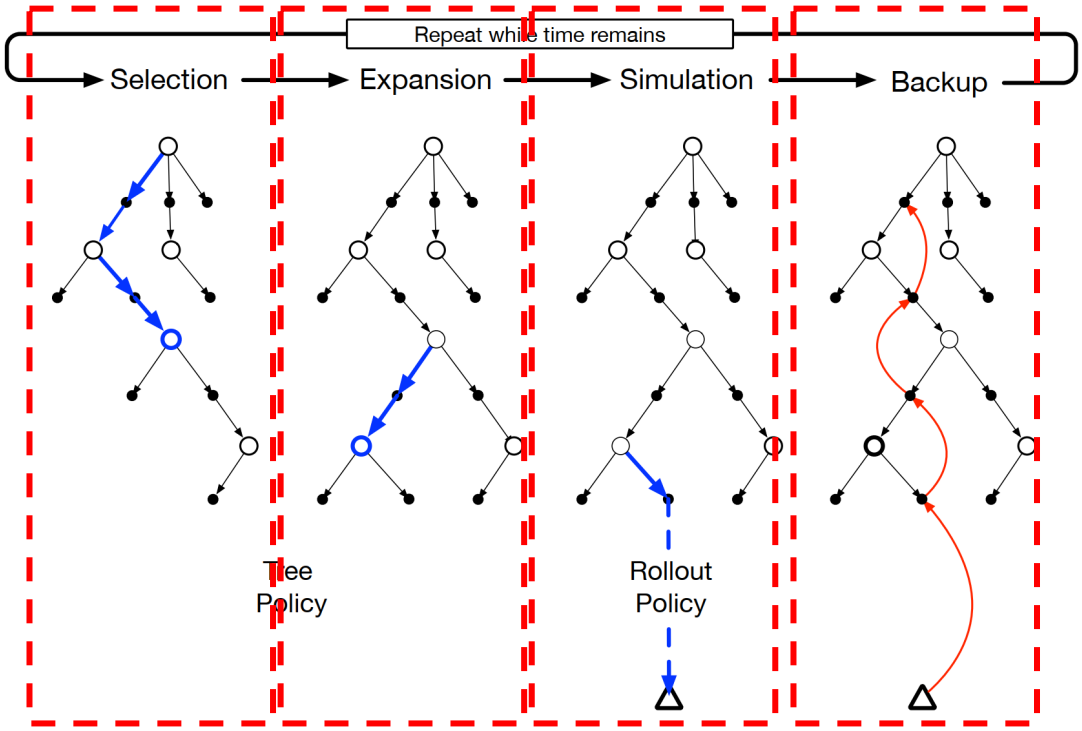
Rollout算法的加速方法

- 多个处理器并行采样
- 轨迹截断，用存储的值估计代替回报
- 剔除不可能成为最佳动作的动作



蒙特卡洛树搜索

1. 选择: 根据树策略(动作值函数)遍历树到一个叶节点
2. 扩展: 从选择的叶节点出发选择未探索过的动作到达新的状态
3. 模拟: 从新的状态出发按照 $rollout$ 策略进行轨迹模拟
4. 回溯: 得到的回报回溯更新树策略, $rollout$ 访问的状态值不会被保存
5. 重复上述步骤直至计算资源耗尽，从根节点选择最优动作
6. 得到新状态, 保留原有树的新状态下的部分节点
7. 重复上述步骤直至游戏结束



Upper Confidence Bounds to Trees

$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln t}{n_i}}$$

- w_i代表第i次移动后取胜的次数；
- n_i代表第i次移动后仿真的次数；
- c为探索参数—理论上等于$\sqrt{2}$；在实际中通常可凭经验选择；
- t代表仿真总次数，等于所有n_i的和。

目前蒙特卡洛树搜索的实现大多是基于UCT的一些变形。

蒙特卡洛树搜索

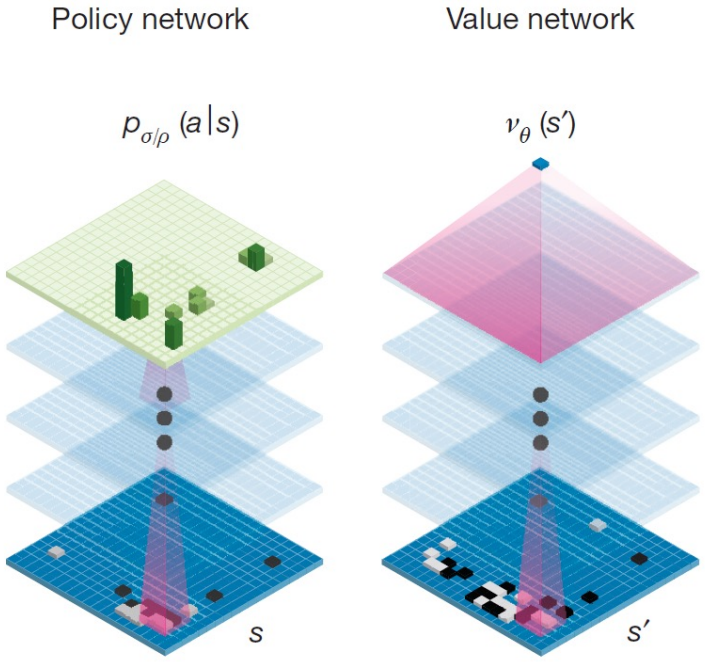
- 蒙特卡洛控制+决策时规划（类似于 $rollout$ ）
- 保留了过去一部分的经验数据。
  - 下一个状态树的初始树是上一个状态树具有高回报的部分

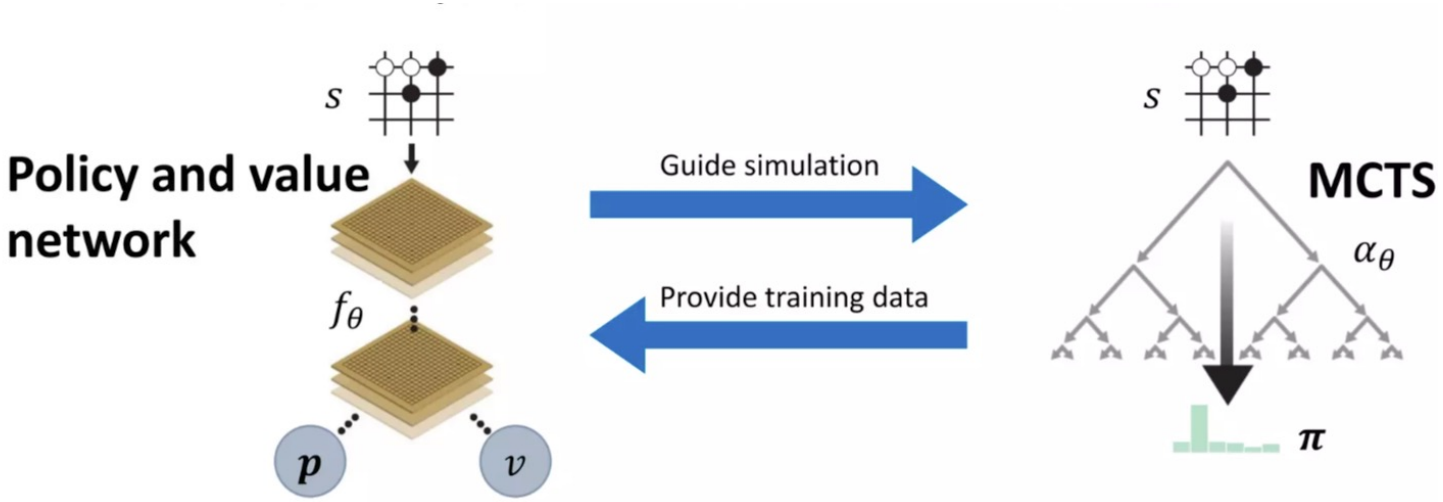
应用

- AlphaGo
  - 16 年五番棋比赛中 4:1 李世石
  - 17 年乌镇围棋峰会中 3:0 柯洁

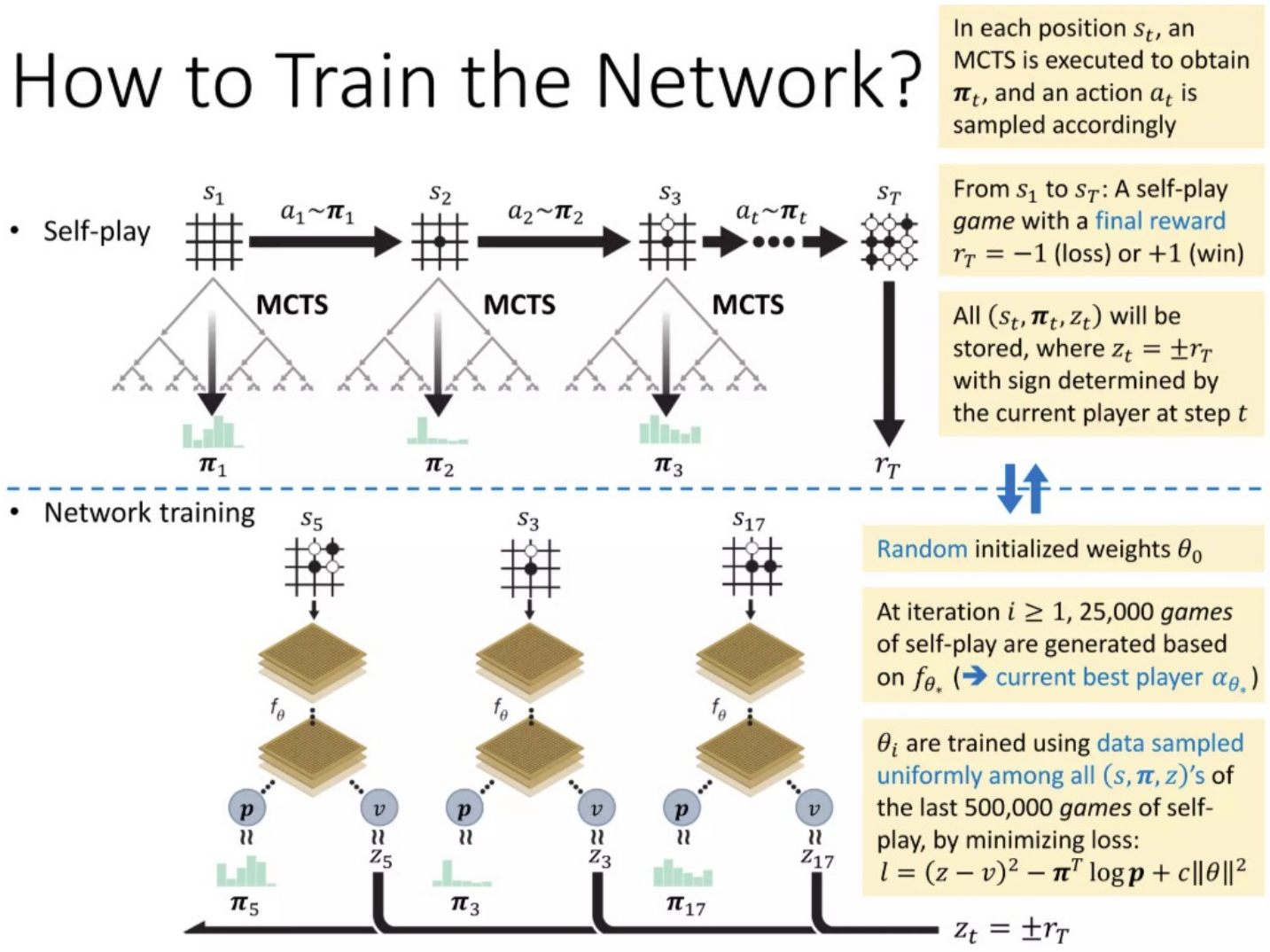
神经网络与蒙特卡洛树搜索

- 策略网络
  - 监督学习
    - 预测下一步移动的最佳结果
  - 强化学习
    - 学习去选择下一步移动去最大化获胜率
- 价值网络
  - 在给定当前状态的情况下获胜的期望
- 通过（深度）神经网络实现

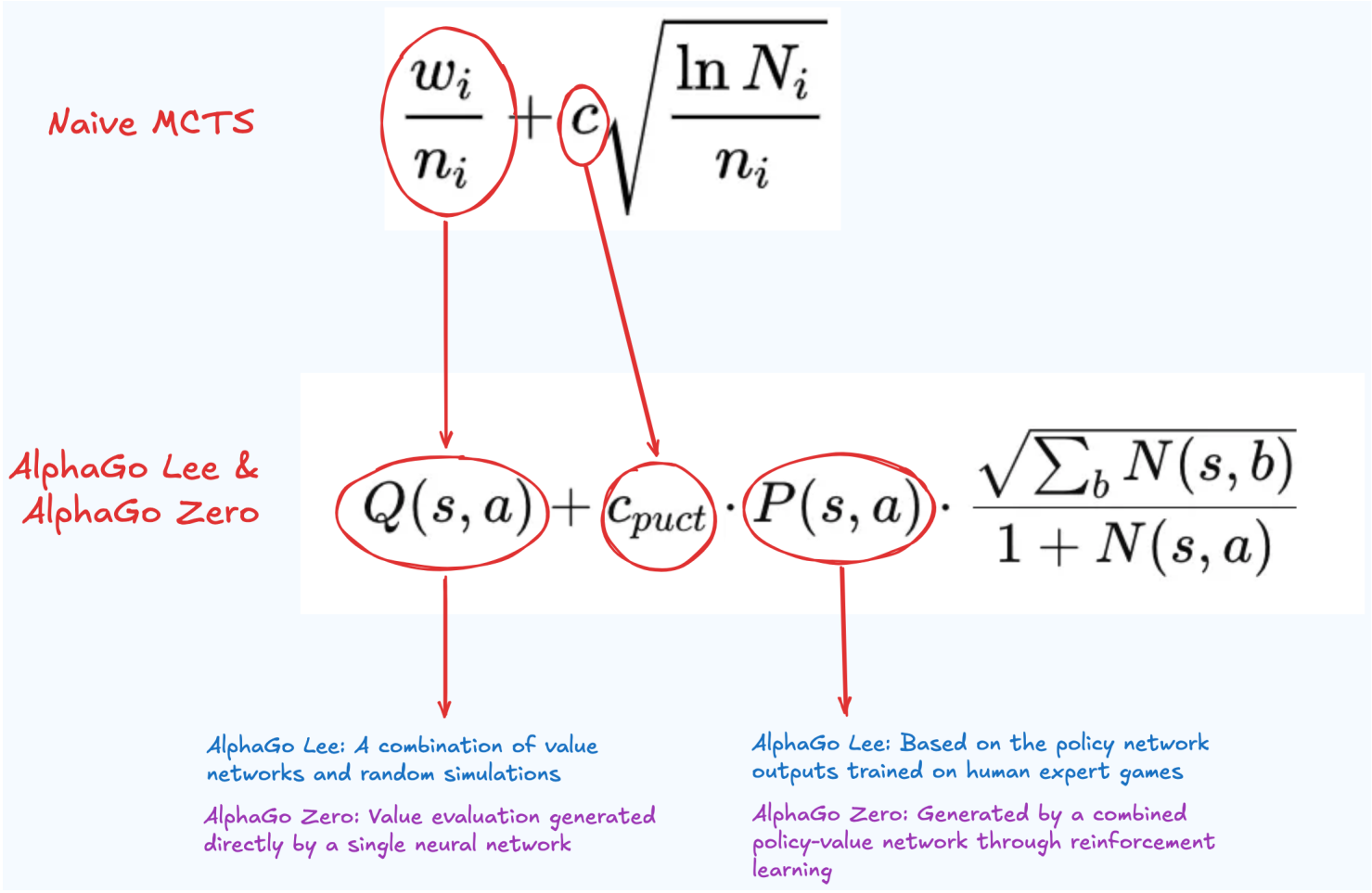




神经网络的训练


How to Train the Network?
In each position s_t, an MCTS is executed to obtain π_t, and an action a_t is sampled accordingly
Self-play
s_1
$a_1 \sim \pi_1$
s_2
$a_2 \sim \pi_2$
s_3
$a_t \sim \pi_t$
s_T
MCTS
MCTS
MCTS
π_1
π_2
π_3
r_T
From s_1 to s_T: A self-play game with a final reward $r_T = -1$ (loss) or $+1$ (win)
All (s_t, π_t, z_t) will be stored, where $z_t = \pm r_T$ with sign determined by the current player at step t
Network training
s_5
s_3
s_{17}
f_θ
p
v
z_5
z_3
z_{17}
π_5
π_3
π_{17}
$z_t = \pm r_T$
Random initialized weights θ_0
At iteration $i \geq 1$, 25,000 games of self-play are generated based on f_{θ_*} (➔ current best player α_{θ_*})
θ_i are trained using data sampled uniformly among all (s, π, z)'s of the last 500,000 games of self-play, by minimizing loss: $l = (z - v)^2 - \pi^T \log p + c\|\theta\|^2$

UCT in AlphaGo Zero



https://aiexpjourney.substack.com/p/unveiling-monte-carlo-tree-search