强化学习及其应用

Reinforcement Learning and Its Applications

第八章 蒙特卡罗树搜索

Monte Carlo Tree Search

授课人：周晓飞
zhouxiaofei@iie.ac.cn
2023-6-15

第八章 蒙特卡罗树搜索

第八章 蒙特卡罗树搜索

Decision Time Planning

- **两种 Planning**（都是基于已有 episodes 不充分的假设，需要生成 episodes）
  - **Background Planning**

    目标：学习背景和环境的 Model，进而采集 Model 的 episodes 来优化策略；

    典型的方法：Dynamic Programming 和 Dyna。
  - **Decision-Time Planning**

    目标：不以学习背景环境为目的，重点要学习状态 S_t 下的最优策略 a_t；结合了策略改善和环境模拟。

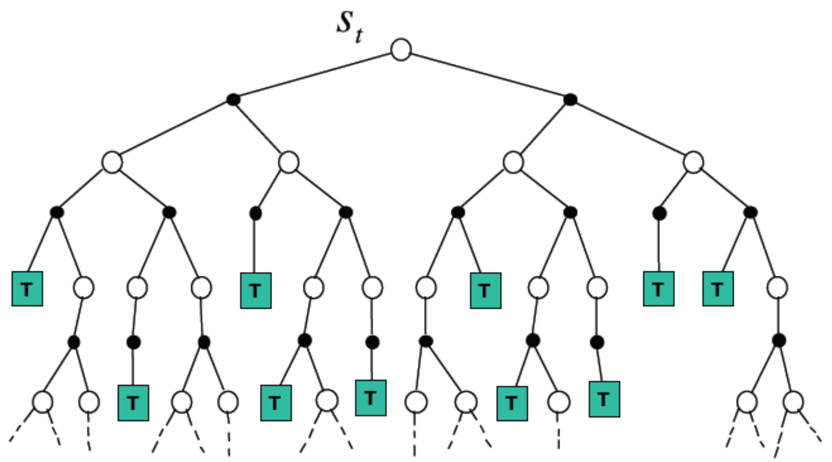
    典型的方法：MCTS

Decision Time Planning

例如 MTCS：MC 估计+启发式搜索

Episodes 本质源于一颗以 S_t 为 root 的树，MC 估计 S_t 的值函数；
通过启发式搜索来引导对树上 episodes 的采集。

值估计的随机逼近：优先采集 Q(s,a)大的行动，使得 V(S)不断改善



No need to solve whole MDP, just sub-MDP starting from now

启发式搜索原则：优先采集 Q(s,a)大的行动分支

Decision Time Planning

问题描述

- Simulate episodes of experience from now with the model

$$\{s_t^k, a_t^k, r_{t+1}^k, \ldots, s_T^k\}_{k=1}^K \sim \mathcal{M}_\nu$$

- Apply model-free RL to simulated episodes
  - Monte-Carlo control $\rightarrow$ Monte-Carlo search
  - Sarsa $\rightarrow$ TD search

第八章 蒙特卡罗树搜索

Simple Monte-Carlo Search

- Given a model $\mathcal{M}_\nu$ and a policy π
- For each action $a \in \mathcal{A}$
  - Simulate K episodes from current (real) state s_t

$$\{s_t, a, r_{t+1}^k, s_{t+1}^k, a_{t+1}^k, \ldots, s_T^k\}_{k=1}^K \sim \mathcal{M}_\nu, \pi$$

  - Evaluate actions by mean return (Monte-Carlo evaluation)

$$Q(s_t, a) = \frac{1}{K}\sum_{k=1}^{K} v_t \xrightarrow{P} Q^\pi(s_t, a)$$

- Select current (real) action with maximum value

$$a_t = \underset{a\in\mathcal{A}}{\operatorname{argmax}}\, Q(s_t, a)$$

蒙特卡罗树搜索

Monte-Carlo Tree Search

- Given a model $\mathcal{M}_\nu$
- Simulate K episodes from current state s_t using current simulation policy π

$$\{s_t, a_t^k, r_{t+1}^k, s_{t+1}^k, \ldots, s_T^k\}_{k=1}^K \sim \mathcal{M}_\nu, \pi$$

- Build a search tree containing visited states and actions
- Evaluate states $Q(s, a)$ by mean return of episodes from s, a

$$Q(s, a) = \frac{1}{N(s, a)} \sum_{k=1}^{K} \sum_{u=t}^{T} \mathbf{1}(s_u, a_u = s, a) v_u \xrightarrow{P} Q^\pi(s, a)$$

- After search is finished, select current (real) action with maximum value in search tree

$$a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q(s_t, a)$$

- In MCTS, the simulation policy π improves
- Each simulation consists of two phases (in-tree, out-of-tree)
  - Tree policy (improves): pick actions to maximise $Q(s, a)$
  - Default policy (fixed): pick actions randomly

- Repeat (each simulation)
  - Evaluate states $Q(s, a)$ by Monte-Carlo evaluation
  - Improve tree policy, e.g. by ϵ − greedy(Q)

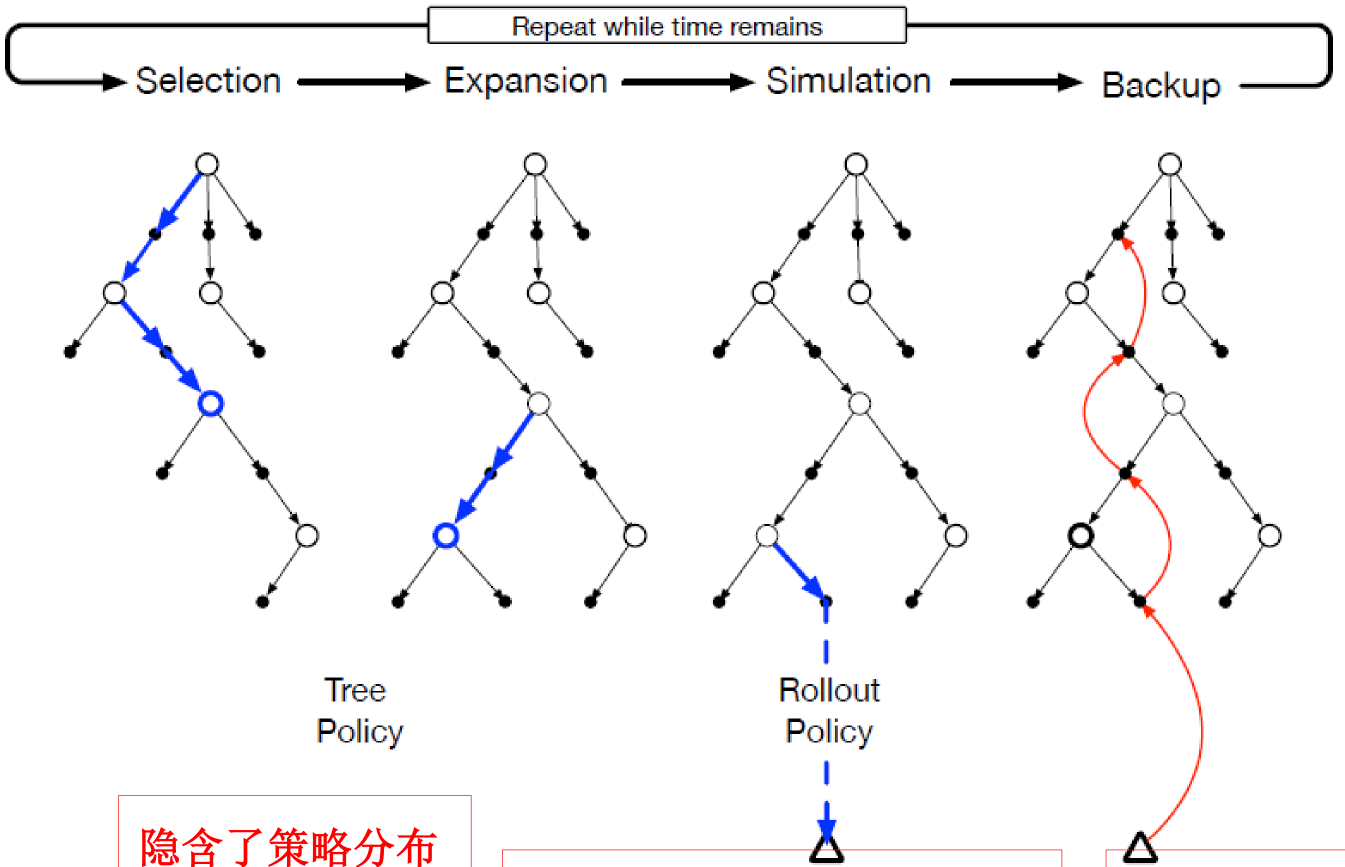
蒙特卡罗树搜索

Monte-Carlo Tree Search

- In MCTS, the simulation policy π improves
- Each simulation consists of two phases (in-tree, out-of-tree)
  - Tree policy (improves): pick actions to maximise $Q(s, a)$
  - Default policy (fixed): pick actions randomly
- Repeat (each simulation)
  - Evaluate states $Q(s, a)$ by Monte-Carlo evaluation
  - Improve tree policy, e.g. by ϵ − greedy(Q)
- Monte-Carlo control applied to simulated experience
- Converges on the optimal search tree, $Q(s, a) \to Q^*(s, a)$

蒙特卡罗树搜索

Basic Version of MCTS



隐含了策略分布

固定了行动，隐含了值估计

Back up，统计 Q(s,a)平均

蒙特卡罗树搜索

Basic Version of MCTS

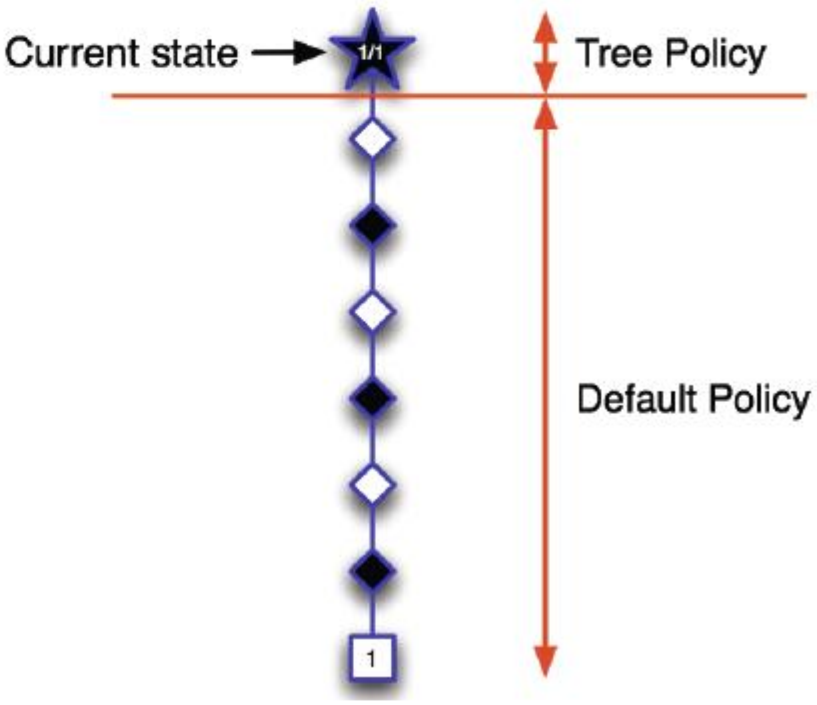
1. **Selection.** Starting at the root node, a *tree policy* based on the action values attached to the edges of the tree traverses the tree to select a leaf node.

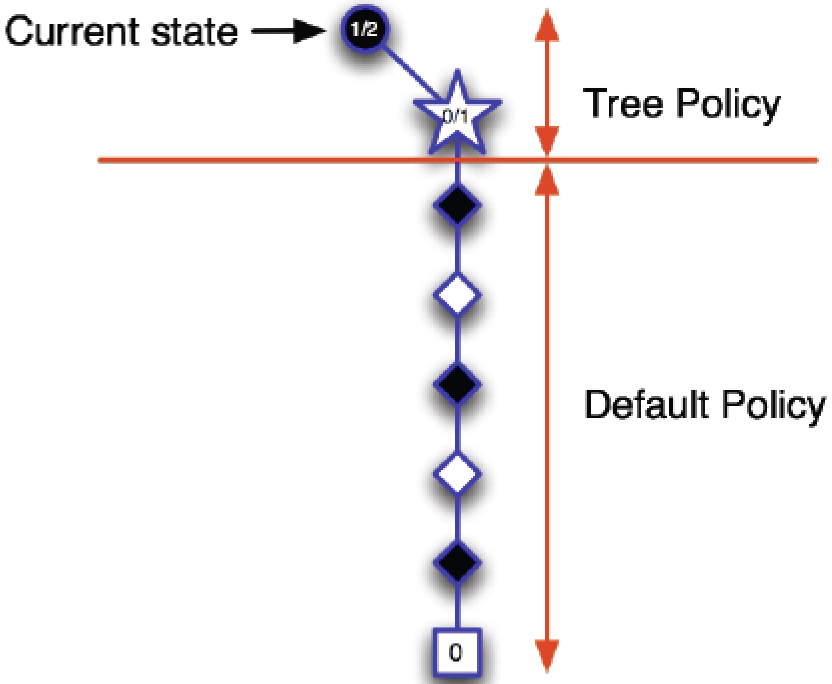
2. **Expansion.** On some iterations (depending on details of the application), the tree is expanded from the selected leaf node by adding one or more child nodes reached from the selected node via unexplored actions.

3. **Simulation.** From the selected node, or from one of its newly-added child nodes (if any), simulation of a complete episode is run with actions selected by the rollout policy. The result is a Monte Carlo trial with actions selected first by the tree policy and beyond the tree by the rollout policy.

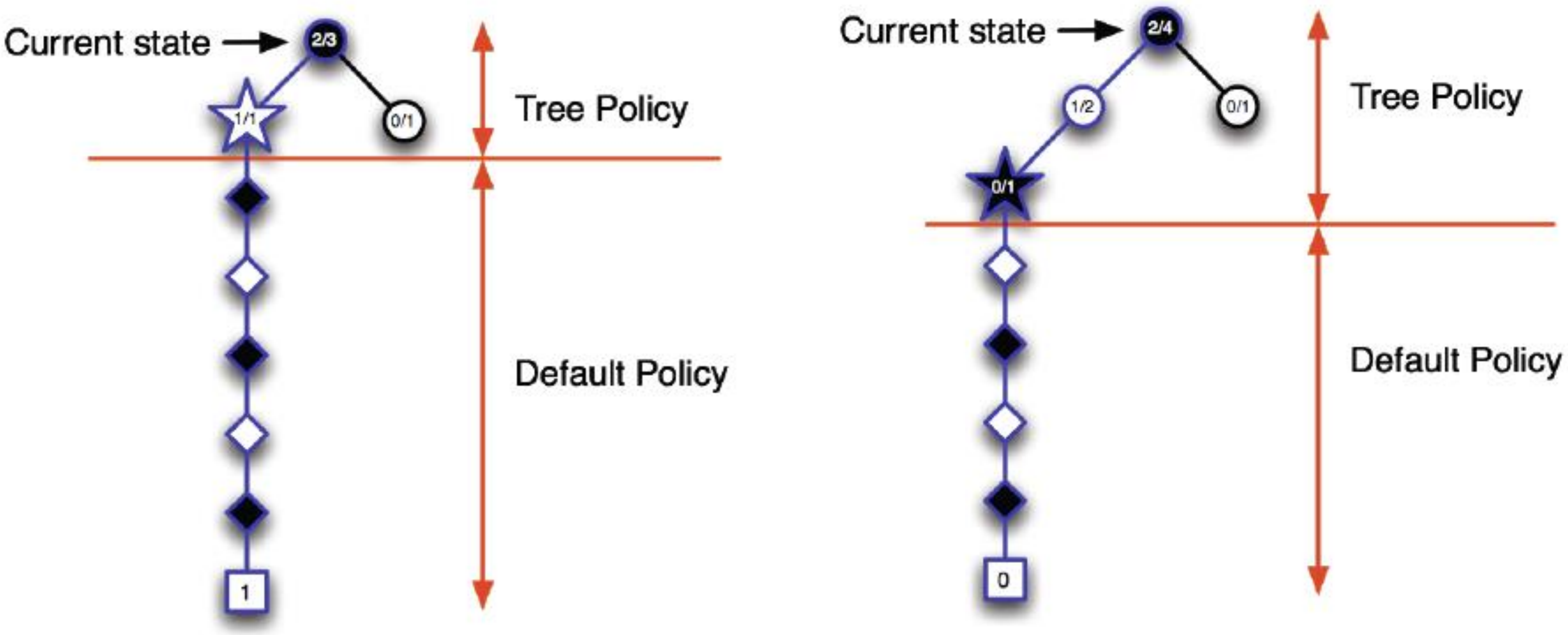
4. **Backup.** The return generated by the simulated episode is backed up to update, or to initialize, the action values attached to the edges of the tree traversed by the tree policy in this iteration of MCTS. No values are saved for the states and actions visited by the rollout policy beyond the tree. Figure 8.13 illustrates this by showing a backup from the terminal state of the simulated trajectory directly to the state–action node in the tree where the rollout policy began (though in general, the entire return over the simulated trajectory is backed up to this state–action node).

蒙特卡罗树搜索


Current state
1/1
Tree Policy
Default Policy
1

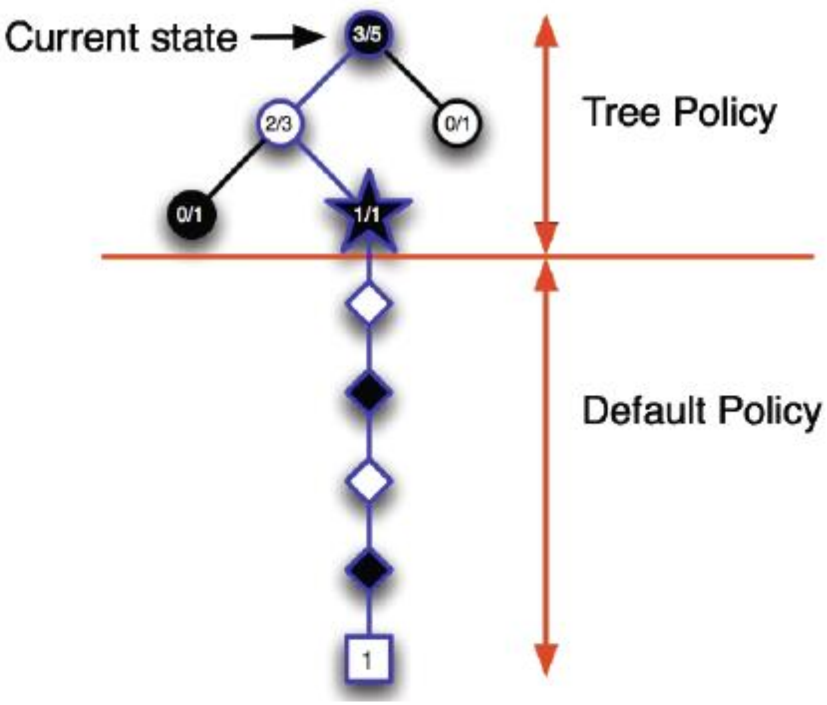

Current state
1/2
0/1
Tree Policy
Default Policy
0

蒙特卡罗树搜索

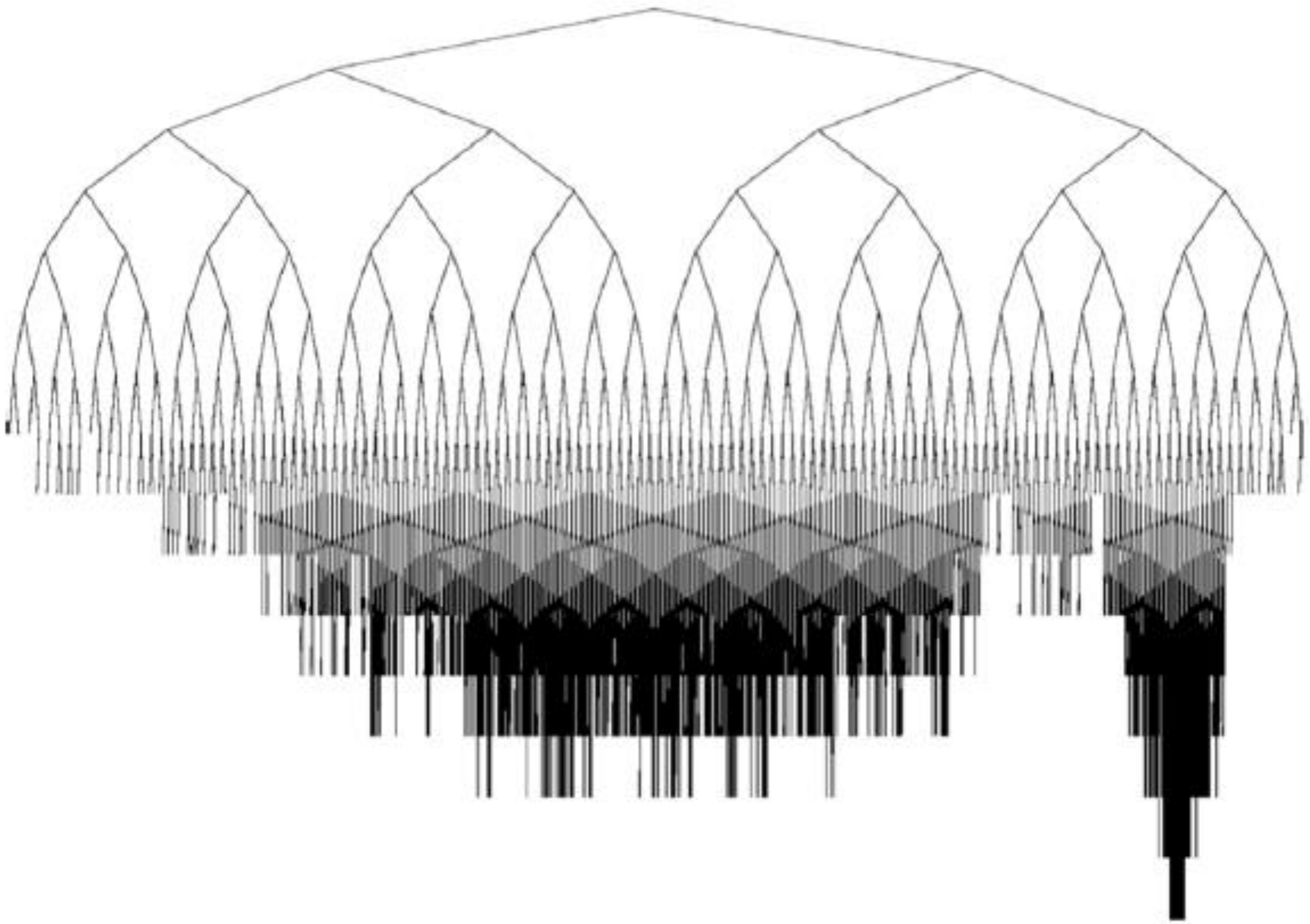


Current state
2/3
1/1
0/1
Tree Policy
Default Policy
1
Current state
2/4
1/2
0/1
0/1
Tree Policy
Default Policy
0

蒙特卡罗树搜索



蒙特卡罗树搜索



蒙特卡罗树搜索

MCTS Algorithm

Algorithm 1: General MCTS approach

function $\text{MCTSSEARCH}(s_0)$
  create root node v_0 with state s_0
  while within computational budget **do**
    $v_l \leftarrow \text{TREEPOLICY}(v_0)$
    $\Delta \leftarrow \text{DEFAULTPOLICY}(s(v_l))$
    $\text{BACKUP}(v_l, \Delta)$
  return $a(\text{BESTCHILD}(v_0))$

蒙特卡罗树搜索

MCTS Algorithm

Algorithm 2: The UCT algorithm

function UCTSEARCH(s_0)
  create root node v_0 with state s_0
  while within computational budget **do**
    $v_l \leftarrow$ TREEPOLICY(v_0)
    $\Delta \leftarrow$ DEFAULTPOLICY($s(v_l)$)
    BACKUP(v_l, Δ)
  return a(BESTCHILD($v_0, 0$))

function TREEPOLICY(v)
  while v is nonterminal **do**
    if v not fully expanded **then**
      return EXPAND(v)
    else
      $v \leftarrow$ BESTCHILD(v, Cp)
  return v

function EXPAND(v)
  choose $a \in$ untried actions from $A(s(v))$
  add a new child v' to v
    with $s(v') = f(s(v), a)$
    and $a(v') = a$
  return v'

function BESTCHILD(v, c)

$$\textbf{return} \quad \mathop{\arg\max}_{v' \in \text{children of } v} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N(v)}{N(v')}}$$

function DEFAULTPOLICY(s)
  while s is non-terminal **do**
    choose $a \in A(s)$ uniformly at random
    $s \leftarrow f(s, a)$
  return reward for state s

function BACKUP(v, Δ)
  while v is not null **do**
    $N(v) \leftarrow N(v) + 1$
    $Q(v) \leftarrow Q(v) + \Delta(v, p)$
    $v \leftarrow$ parent of v

蒙特卡罗树搜索

MCTS 优点

- Highly selective best-first search
- Evaluates states *dynamically* (unlike e.g. DP)
- Uses sampling to break curse of dimensionality
- Works for "black-box" models (only requires samples)
- Computationally efficient, anytime, parallelisable

蒙特卡罗树搜索

分析：

$$\textbf{function } \text{BESTCHILD}(v, c)$$

$$\textbf{return} \quad \underset{v' \in \text{children of } v}{\arg\max} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N(v)}{N(v')}}$$

较大$Q(s_t,a)$值的 a 分支被搜索次数多，对应的$N(s_t,a)$较大，搜索频率逼近树策略分布.

$$V(s) = \sum_a \pi(a|s) Q(s,a)$$

$$= \frac{1}{N(s)} \sum_a N(s,a) Q(s,a)$$

第八章 蒙特卡罗树搜索

8.1 Decision Time Planning

8.2 蒙特卡罗树搜索

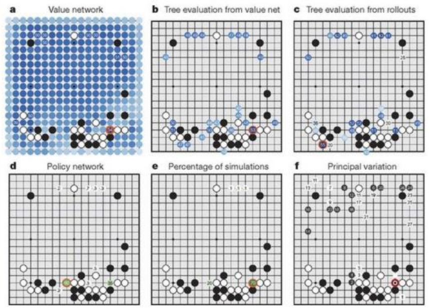
8.3 AlphaGo

8.3.1 Policy Gradient

8.3.2 MCTS 方法

AlphaGo

围棋问题

- **S：棋局；**
- **A：落子动作；**
- **Model：**

状态转移：互搏弈导致的状态转移，S →S’，没有直观的 Pss’ ；

$(S,a_+)\rightarrow S_+$, (正方落子后进入状态 S_+)；$(S_+, a_-)\rightarrow S'$（反方落子后进入状态 S’）

回报值：最终是否获胜，获胜为 1，否则为-1；需要完整的 episode.

问题：在已有棋谱或专家经验（training episodes）作为先验，当前棋局 S_t 的最优行动 a_t ？

AlphaGo

Abstract

- **提出两个方法：**

  1. 不采用前向 lookahead Search，学习策略网络（Policy Network）

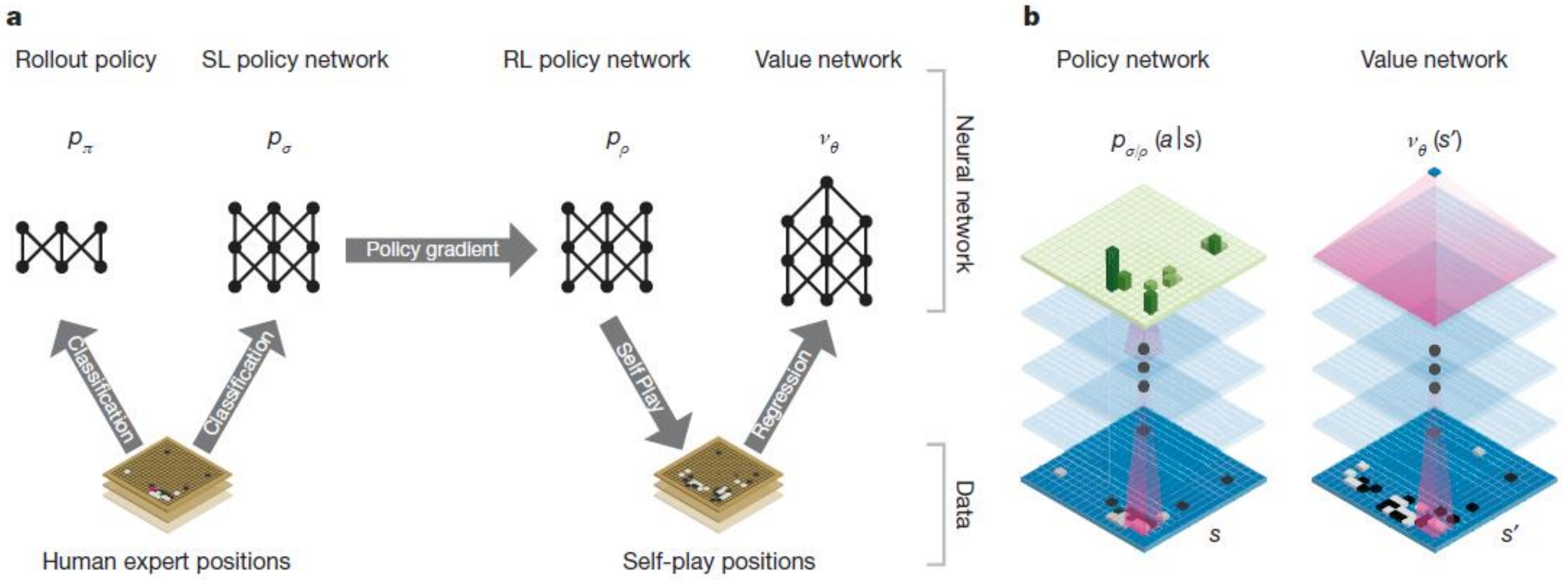
     结合 value networks 和 policy network: Value networks 估计棋局的值函数，Policy networks 选择行动 a_t

  2. 采用前向 lookahead Search 的 MTCS 方法

     结合 value networks，policy network 和 MTCS

两种方法不同：前者学习 Policy probability 决定 a_t，后者是搜最优的 a_t

AlphaGo


a
Rollout policy
SL policy network
RL policy network
Value network
p_π
p_σ
p_ρ
v_θ
Policy gradient
Classification
Classification
Self Play
Regression
Neural network
Data
Human expert positions
Self-play positions
b
Policy network
Value network
$p_{\sigma/\rho}(a|s)$
$v_\theta(s')$
s
s'

方法一： Policy Gradient

- **目标：学习一个策略网络** $p_\rho(a_t|s_t)$
- **参数更新采用策略梯度方法：**

$$\Delta\rho \propto \frac{\partial \log p_\rho(a_t|s_t)}{\partial \rho} z_t$$

初始化网络：

专家经验训练了一个先验策略网络(supervised learning (SL) policy network)作为初始化

$$\Delta\sigma \propto \frac{\partial \log p_\sigma(a|s)}{\partial \sigma}$$

AlphaGo

方法一： Policy Gradient

方法流程

1．学习 SL Policy Network

sampled state-action pairs (s, a), using stochastic gradient ascent to maximize the likelihood of the human move a selected in state s

$$\Delta\sigma \propto \frac{\partial \log p_{\sigma}(a|s)}{\partial \sigma}$$

- **13-layer policy network, which we call the SL policy network, from 30 million positions from the KGS Go Server.**
- **Data: randomly sampled state-action pairs (s,a) on human moves.**

AlphaGo

方法一： Policy Gradient

2．学习 RL Policy Network $p_\rho(a_t|s_t)$

$$\Delta\rho \propto \frac{\partial \log p_\rho(a_t|s_t)}{\partial \rho} z_t$$

- **Its weights ρ are initialized to the same values, $\rho = \sigma$.**
- **Episodes are from playing games between the current policy network p_ρ and a randomly selected previous iteration of the policy network.**
- **Outcomes z_t: +1 for winning and −1 for losing.**

AlphaGo

方法一： Policy Gradient

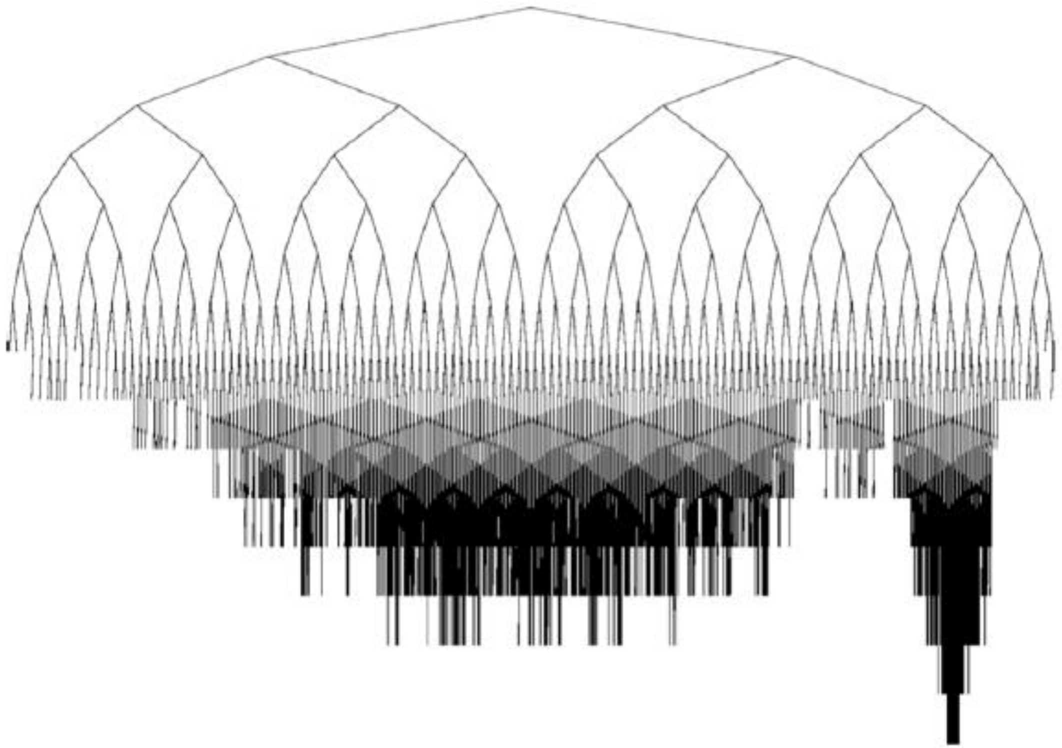
3. 学习 Value Network $V_\theta(S)$

目的：值函数 $V_\theta(S)$替代 z_t

$$\Delta\theta \propto \frac{\partial v_\theta(s)}{\partial\theta}(z - v_\theta(s))$$

- **Train data: we generated a new self-play data set consisting of 30 million distinct positions, each sampled from a separate game. Each game was played between the RL policy network and itself until the game terminated**

方法二：MCTS



以 St 为 root 的子树不断被扩展，

优先搜索和扩展回报值大的节点,

即 Q(s,a)大的节点.

方法二：MCTS

AlphaGo with MCTS

1．Selection

- At each of these time steps, t<L, an action is selected according to the statistics in the search tree:

$$a_t = \operatorname{argmax}_a \left(Q(s_t, a) + u(s_t, a)\right)$$

$$u(s,a) = c_{\mathrm{puct}} P(s,a) \frac{\sqrt{\sum_b N_r(s,b)}}{1 + N_r(s,a)}$$

where c_{puct} is a constant determining the level of exploration; this search control strategy initially prefers actions with high prior probability and low visit count, but asymptotically prefers actions with high action value.

选择的a可能不存在后继结点，是否扩展后继结点依据Expansion。

方法二：MCTS

AlphaGo with MCTS

树结构

Each node s in the search tree contains edges (s, a) for all legal actions $a \in \mathcal{A}(s)$. Each edge stores a set of statistics,

$$\{P(s,a),\ \ N_v(s,a),\ \ N_r(s,a),\ \ W_v(s,a),\ \ W_r(s,a),\ \ Q(s,a)\}$$

where $P(s, a)$ is the prior probability, $W_v(s, a)$ and $W_r(s, a)$ are Monte Carlo estimates of total action value, accumulated over $N_v(s, a)$ and $N_r(s, a)$ leaf evaluations and rollout rewards, respectively, and $Q(s, a)$ is the combined mean action value for that edge. Multiple simulations are executed in parallel on separate search threads.

方法二：MCTS

AlphaGo with MCTS

2．Expansion

- When the visit（selected）count exceeds a threshold, $N_r(s, a) > n_{thr}$, the successor state $s' = f(s, a)$ is added to the search tree. The new node is initialized to:

$$\{N(s', a) = N_r(s', a) = 0,\ W(s', a) = W_r(s', a) = 0,\ P(s', a) = p_\sigma(a|s')\}$$

AlphaGo

方法二：MCTS

AlphaGo with MCTS

3．Evaluation

- These evaluations are combined, using a mixing parameter λ, into a leaf evaluation $V(S_L)$;

$$V(s_L) = (1-\lambda)v_\theta(s_L) + \lambda z_L$$

- The leaf node is evaluated in two very different ways:
  (1) First, by the value network $V_\theta(S_L)$;
  (2) Second, by the outcome Z_L of a random rollout played out until terminal step T using the fast rollout policy p_π;

方法二：MCTS

AlphaGo with MCTS

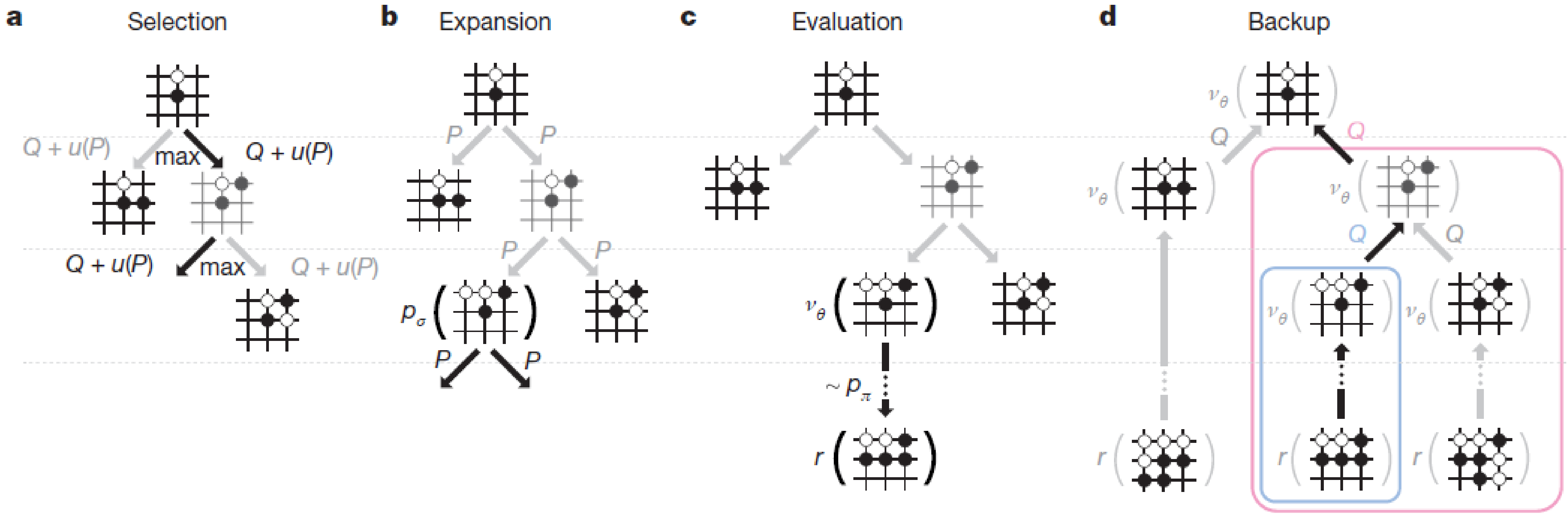
4. Backup

$$N(s,a)=\sum_{i=1}^{n}1(s,a,i)$$

$$Q(s,a)=\frac{1}{N(s,a)}\sum_{i=1}^{n}1(s,a,i)V(s_L^i)$$

where s_L^i is the leaf node from the ith simulation, and $1(s, a, i)$ indicates whether an edge (s, a) was traversed during the ith simulation. Once the search is complete, the algorithm chooses the most visited move from the root position.

AlphaGo



a
Selection
b
Expansion
c
Evaluation
d
Backup
Q + u(P)
max
Q + u(P)
Q + u(P)
max
Q + u(P)
P
P
P
P
P
P
p_σ
v_θ
$\sim p_\pi$
r
v_θ
Q
Q
v_θ
v_θ
Q
Q
v_θ
v_θ
r
r
r

本讲参考文献

1. Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction. (Second edition, in progress，draft).
2. David Silver，Slides@《Reinforcement Learning:An Introduction》,2016.
3. Browne C.B., Daniel W. Cowling P.I, and Samothrakis S. A Survey of Monte Carlo Tree Search Methods, IEEE Transactions on Computational Intelligence and AI In Games, vol. 4, no. 1.
4. David Silver, and Aja Huang et.al. Mastering the game of Go with deep neural networks and tree search. Nature 2016.