上海大学未来技术学院 SCHOOL OF FUTURE TECHNOLOGY, SHANGHAI UNIVERSITY
上海大学人工智能研究院 INSTITUTE OF ARTIFICIAL INTELLIGENCE, SHANGHAI UNIVERSITY

人工智能导论

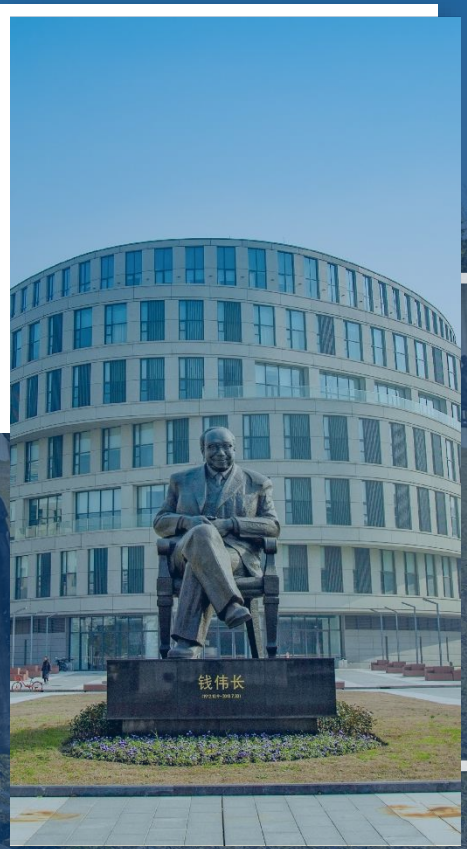
——第3课：人工智能在游戏的应用

(强化学习)

叶林奇

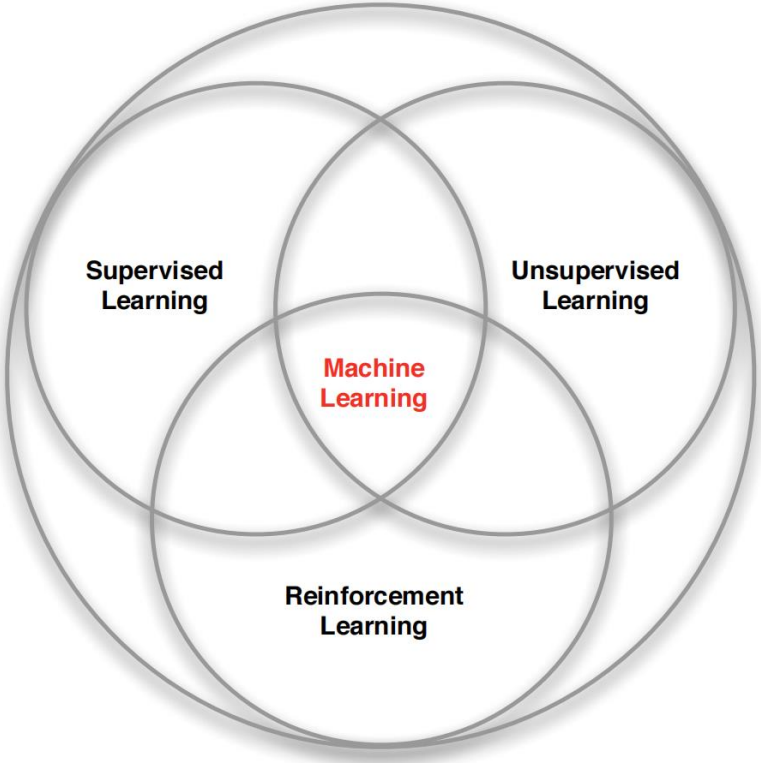
未来技术学院（人工智能研究院）

2023冬季学期



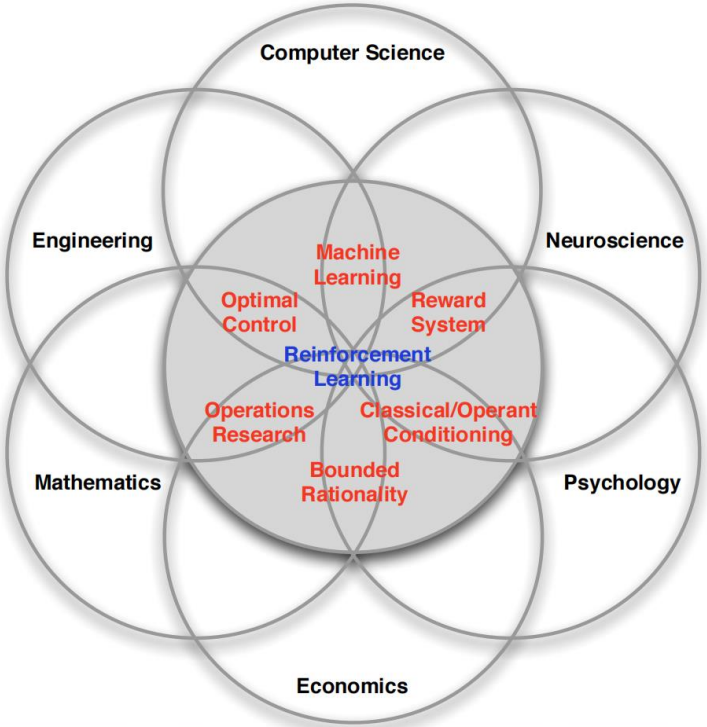
强化学习

Branches of Machine Learning

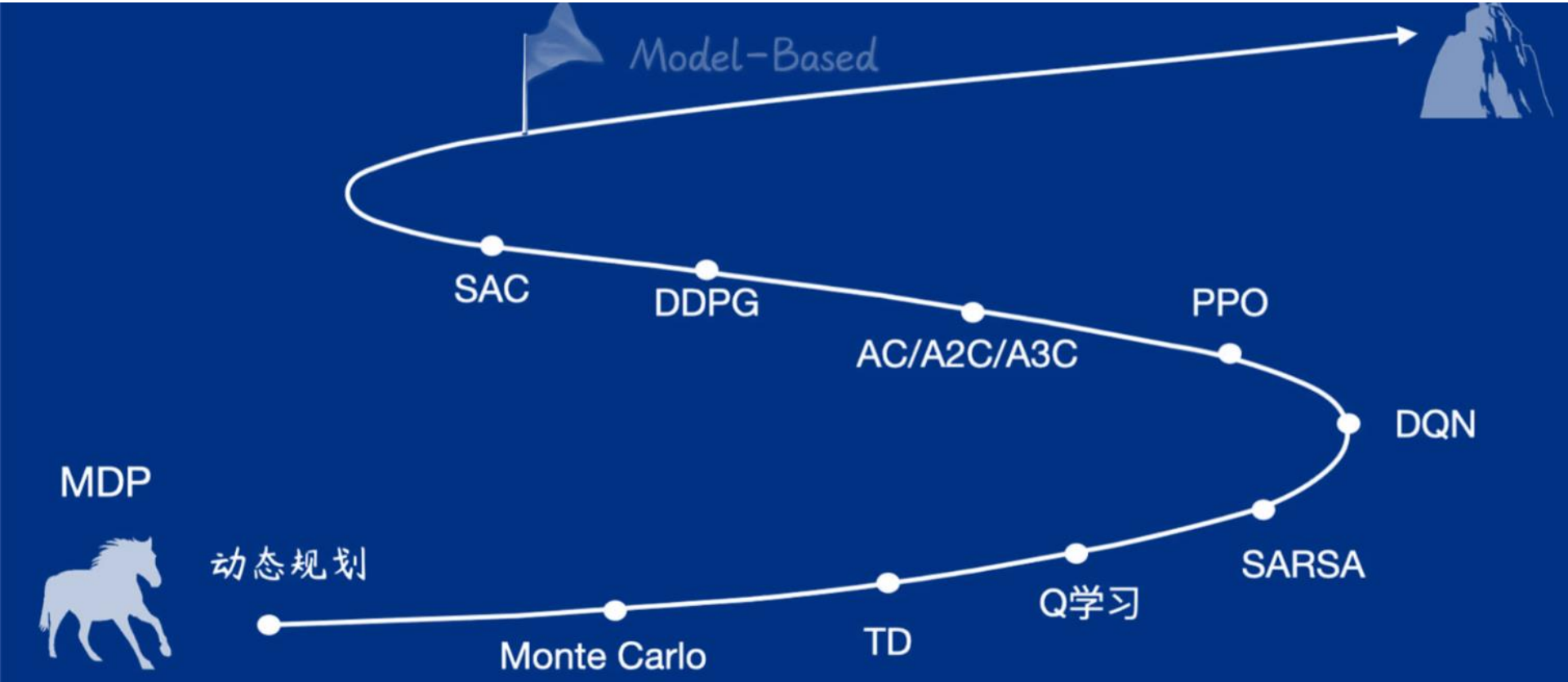


强化学习

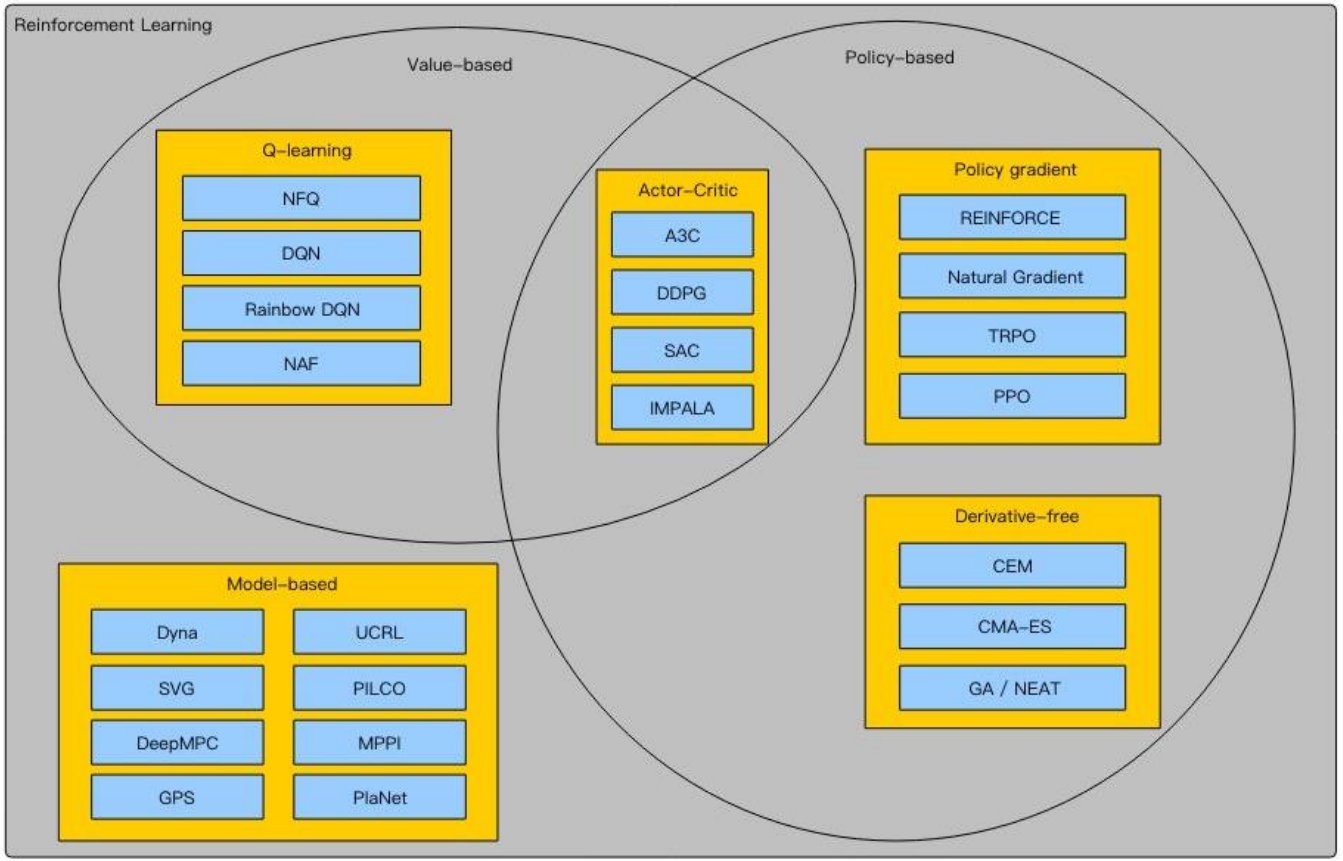
Many Faces of Reinforcement Learning



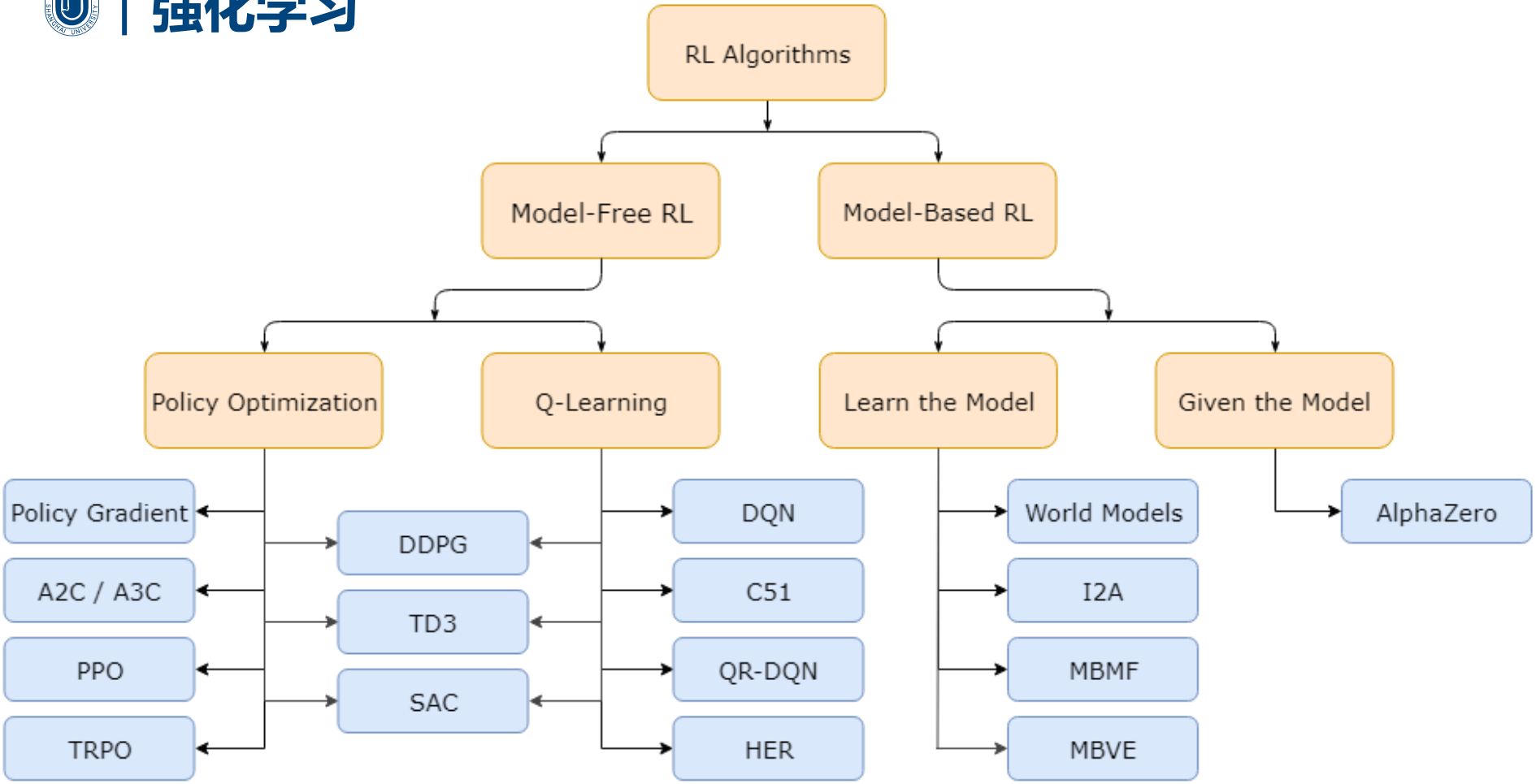
强化学习


Model-Based
SAC
DDPG
AC/A2C/A3C
PPO
DQN
MDP
动态规划
Monte Carlo
TD
Q学习
SARSA

强化学习


Reinforcement Learning
Value-based
Policy-based
Q-learning
NFQ
DQN
Rainbow DQN
NAF
Actor-Critic
A3C
DDPG
SAC
IMPALA
Policy gradient
REINFORCE
Natural Gradient
TRPO
PPO
Derivative-free
CEM
CMA-ES
GA / NEAT
Model-based
Dyna
UCRL
SVG
PILCO
DeepMPC
MPPI
GPS
PlaNet

强化学习


RL Algorithms
Model-Free RL
Model-Based RL
Policy Optimization
Q-Learning
Learn the Model
Given the Model
Policy Gradient
A2C / A3C
PPO
TRPO
DDPG
TD3
SAC
DQN
C51
QR-DQN
HER
World Models
I2A
MBMF
MBVE
AlphaZero

提纲

一、萌芽：从DP到Q Learning

二、大成：从DQN到GATO

三、巅峰：从AlphaGo到MuZero

在与动态环境的交互中学习

有监督、无监督学习

Model ⟵

Fixed Data

强化学习

Agent ⟷

Dynamic Environment

Agent不同，交互出的数据也不同！

和动态环境交互产生的数据分布

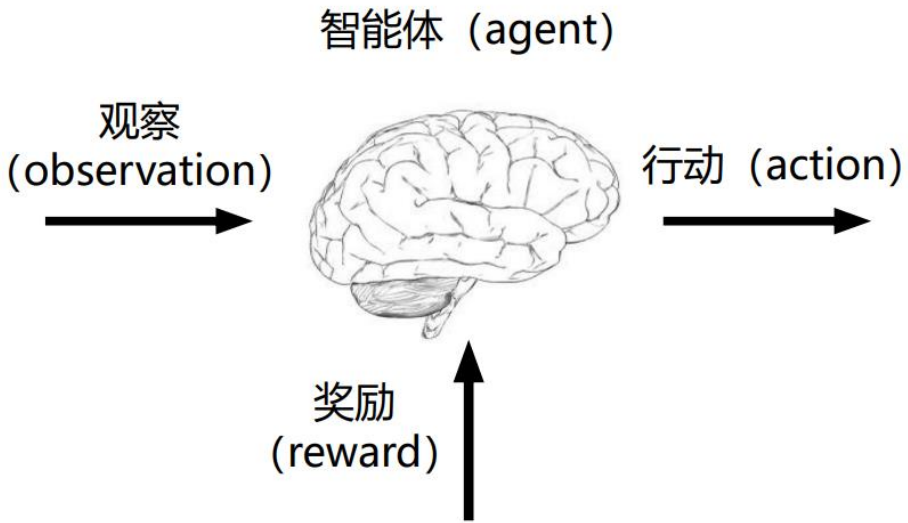
- 给定同一个动态环境（即MDP），不同的策略采样出来的(状态-行动)对的分布是不同的

- 占用度量（Occupancy Measure）

$$\rho^{\pi}(s,a) = \mathbb{E}_{a\sim\pi(s),\, s'\sim p(s,a)}\left[\sum_{t=0}^{T}\gamma^{t}\, p(s_t = s, a_t = a)\right]$$

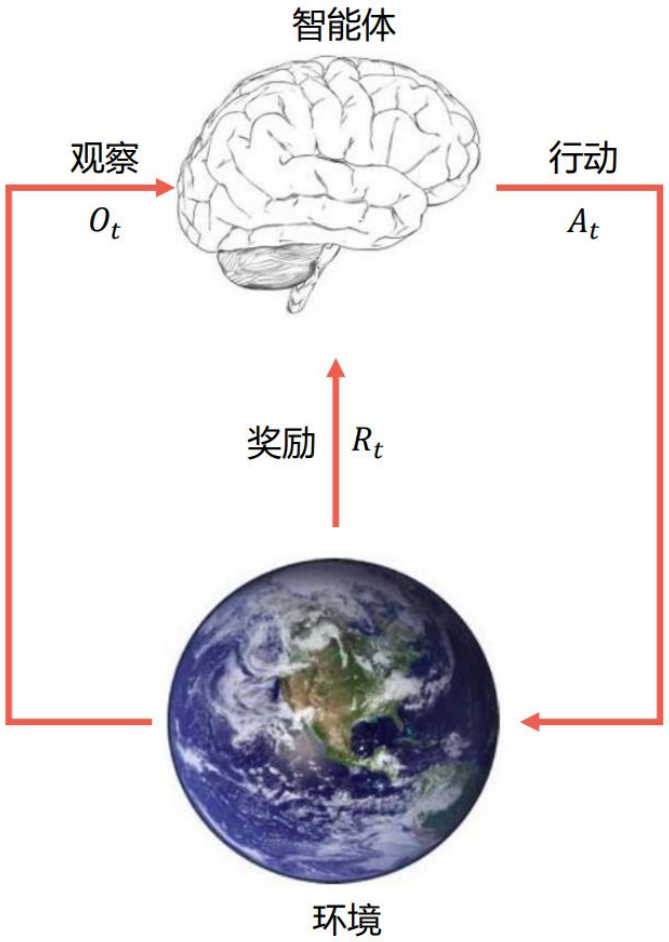
强化学习定义

- 通过从交互中学习来实现目标的计算方法



- 三个方面：
  - 感知：在某种程度上感知环境的状态
  - 行动：可以采取行动来影响状态或者达到目标
  - 目标：随着时间推移最大化累积奖励

强化学习交互过程



- 在每一步 t，智能体：
  - 获得观察 O_t
  - 获得奖励 R_t
  - 执行行动 A_t
- 环境：
  - 获得行动 A_t
  - 给出观察 O_{t+1}
  - 给出奖励 R_{t+1}
- t 在环境这一步增加

强化学习系统要素

- 历史（History）是观察、行动和奖励的序列

$$H_t = O_1, R_1, A_1, O_2, R_2, A_2, \ldots, O_{t-1}, R_{t-1}, A_{t-1}, O_t, R_t$$

  - 即，一直到时间t为止的所有可观测变量
  - 根据这个历史可以决定接下来会发生什么
    - 智能体选择行动
    - 环境选择观察和奖励

- 状态（state）是一种用于确定接下来会发生的事情（行动、观察、奖励）的信息
  - 状态是关于历史的函数

$$S_t = f(H_t)$$

强化学习系统要素

- 策略（Policy）是学习智能体在特定时间的行为方式
  - 是从状态到行动的映射
  - 确定性策略（Deterministic Policy）

$$a = \pi(s)$$

  - 随机策略（Stochastic Policy）

$$\pi(a|s) = P(A_t = a \mid S_t = s)$$

强化学习系统要素

- 奖励（Reward） $R(s,a)$
  - 一个定义强化学习目标的标量
  - 能立即感知到什么是“好”的
- 价值函数（Value Function）
  - 状态价值是一个标量，用于定义对于长期来说什么是“好”的
  - 价值函数是对于未来累积奖励的预测
    - 用于评估在给定的策略下，状态的好坏

$$Q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s, A_t = a]$$
$$= \mathbb{E}_\pi[R_{t+1} + \gamma Q_\pi(s',a') | S_t = s, A_t = a]$$

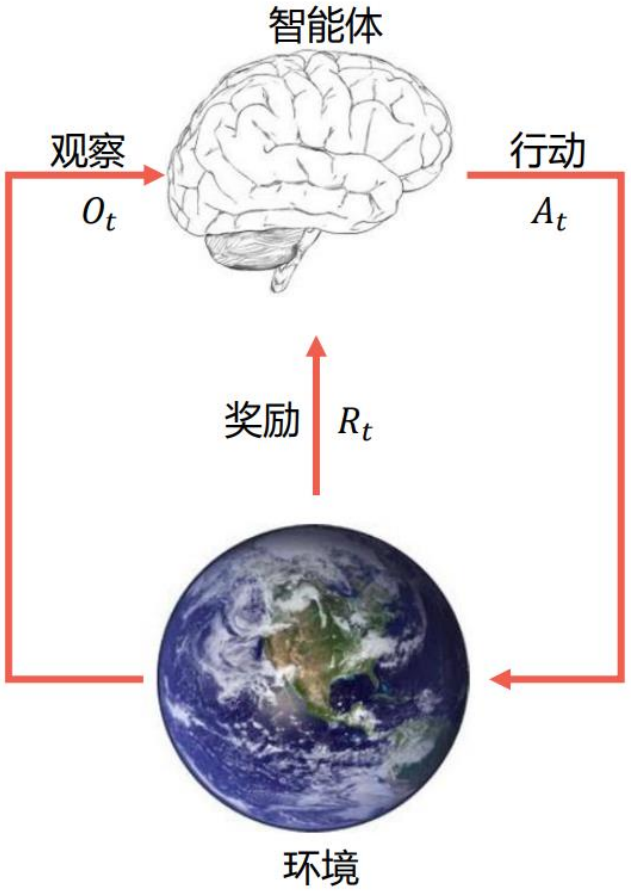
强化学习系统要素

- 环境的模型（Model）用于模拟环境的行为
  - 预测下一个状态

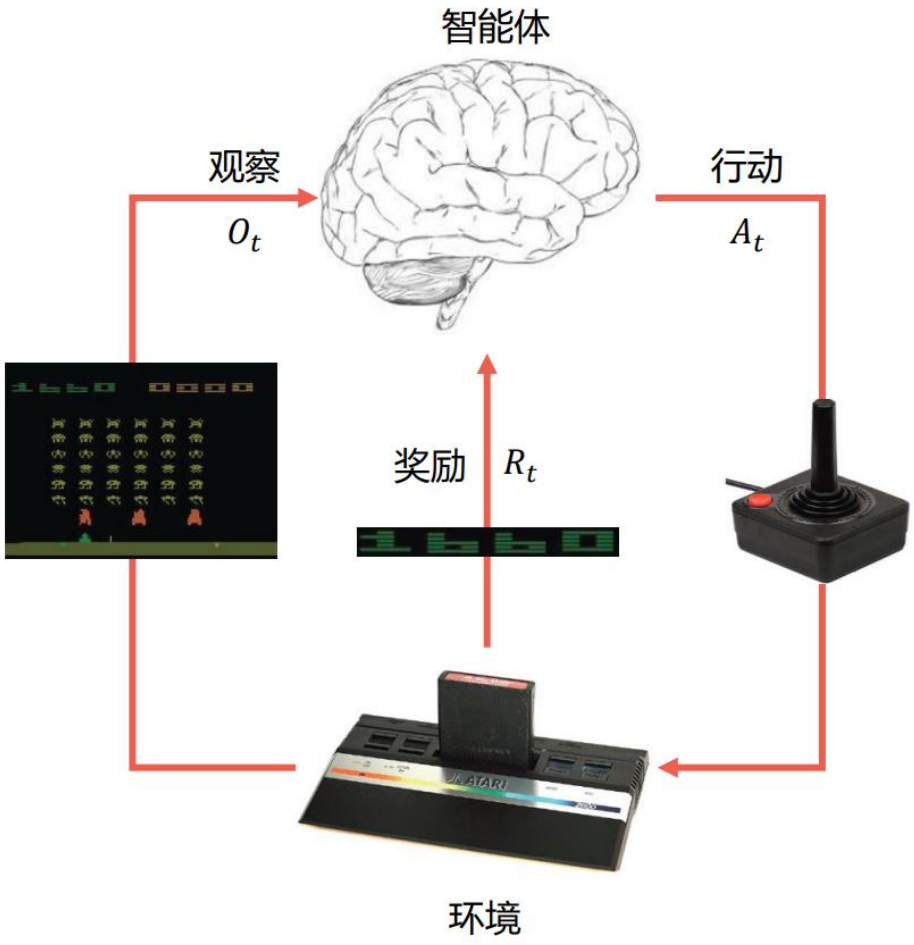
$$\mathcal{P}_{ss'}^{a} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$$

  - 预测下一个（立即）奖励

$$\mathcal{R}_{s}^{a} = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$$



举例：Atari游戏



- 游戏规则未知
- 从交互游戏中进行学习
- 在操纵杆上选择行动并查看分数和像素画面

强化学习的方法分类

- 基于价值：知道什么是好的什么是坏的
  - 没有策略（隐含）
  - 价值函数

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{E}}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)\right) \nabla_{\theta_i} Q(s, a; \theta_i)\right]$$

- 基于策略：知道怎么行动
  - 策略
  - 没有价值函数

- Actor-Critic：学生听老师的
  - 策略
  - 价值函数

价值和策略近似

$Q_\theta(s,a)$

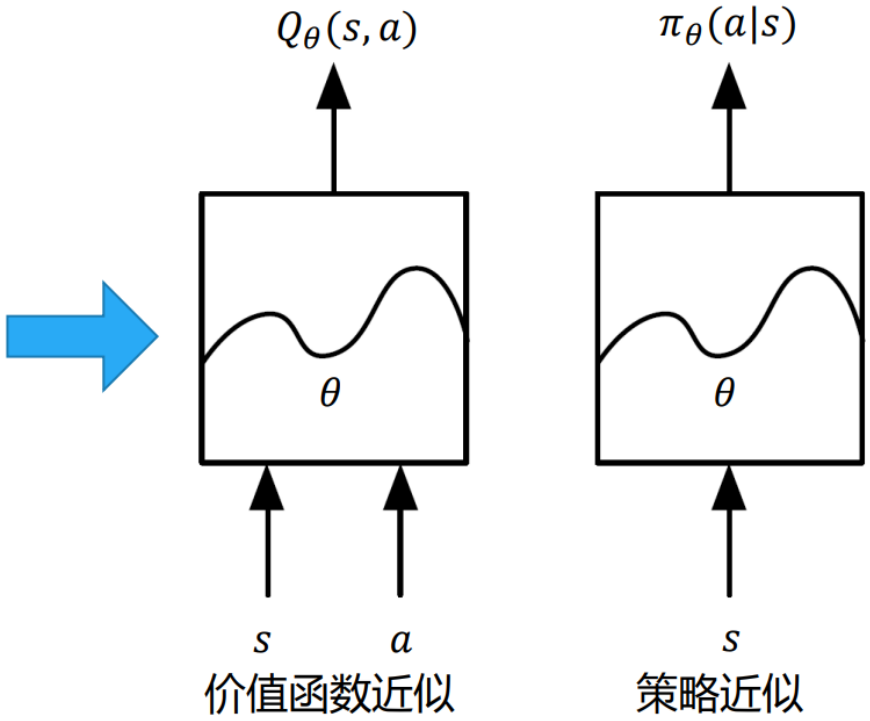
	a_1	a_2
s_1	3.2	4.1
s_2	0.8	1.2
s_3	2.3	-0.3

价值函数表格

$\pi_\theta(a|s)$

	a_1	a_2
s_1	0.2	0.8
s_2	0.4	0.6
s_3	0.9	0.1

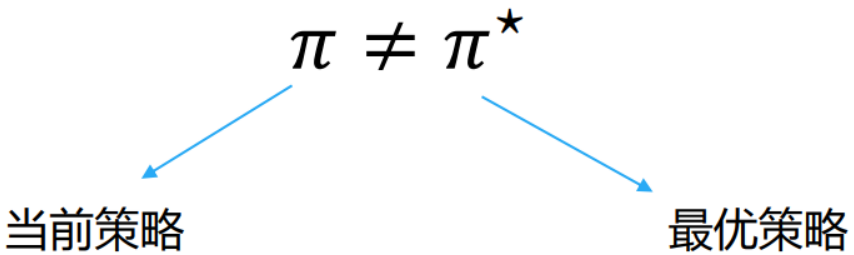
策略表格



- 假如我们直接使用深度神经网络建立这些近似函数呢?
- 深度强化学习!

序列决策任务中的一个基本问题

- 基于目前策略获取已知最优收益还是尝试不同的决策
  - **Exploitation** 执行能够获得已知最优收益的决策
  - **Exploration** 尝试更多可能的决策，不一定会是最优收益



序列决策任务中的一个基本问题

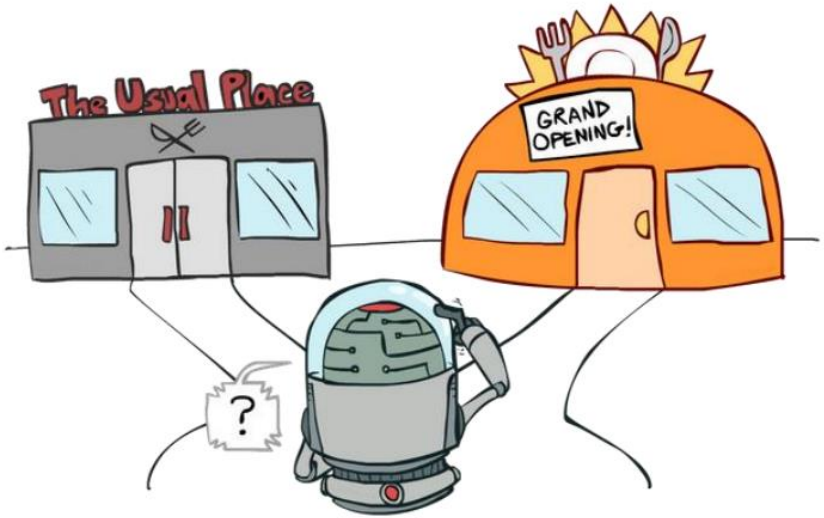
- 基于目前策略获取已知最优收益还是尝试不同的决策
  - **Exploitation** 执行能够获得已知最优收益的决策
  - **Exploration** 尝试更多可能的决策，不一定会是最优收益

$$\mathcal{E}_t = \{\pi_t^i \mid i = 1, \dots, n\} \xrightarrow{\text{探索}} \mathcal{E}_{t+1} = \{\pi_t^i \mid i = 1, \dots, n\} \cup \{\pi_e^j \mid j = 1, \dots, m\}$$

$$\exists\, V^\star(\cdot \mid \pi_t^i \sim \mathcal{E}_t) \le V^\star(\cdot \mid \pi_{t+1}^i \sim \mathcal{E}_{t+1}) \qquad \pi_{t+1}^i \sim \{\pi_e^i \mid i = 1, \dots, m\}$$

探索：可能发现更好的策略

一个例子


The Usual Place
GRAND OPENING!
?


10:20
搜索
美食
吃炸串
地方菜系
全球美食
轻食简餐
甜品饮
综合排序
距离
销量
筛选
守护联盟
津贴优惠
满减优惠
品质联盟
守护联盟
和番丼饭(东川路店)
4.8 月售5143
起送¥15 远距离配送¥5 ¥8
42分钟 4.7km
"不错哦,好待咖喱虾排饭很赞"
20减16 49减22 津贴1元 78减28 120减35
守护联盟
槿巷多料拌饭(闵行旗舰店)
4.6 月售1644
起送¥15 远距离配送¥3.5 ¥7.8
39分钟 4.0km
"第二次吃了,很香的豆腐" 闵行区韩国料理实惠第1名
28减15 45减20 津贴1元 65减25 90减32
蜜哆哆韩式炸鸡(颛桥店)
预订中 10:30配送
4.8 月售1145
起送¥15 远距离配送¥8
41分钟 5.0km
"点的无骨香酥鸡,整体很满意" 元气好店
28减12 48减24 78减33 118减40 6元会员红包
守护联盟
飯豊町·和風精致便当(东川路店)
4.9 月售3842
起送¥20 远距离配送¥3 ¥8.3
46分钟 4.7km
已检测体温 请放心食用
23减14 49减15 80减21 119减38 6元会员红包

策略探索的一些原则

- 朴素方法（Naïve Exploration）
  - 添加策略噪声 ϵ-greedy
- 积极初始化（Optimistic Initialization）
- 基于不确定性的度量 （Uncertainty Measurement）
  - 尝试具有不确定收益的策略，可能带来更高的收益
- 概率匹配（Probability Matching）
  - 基于概率选择最佳策略
- 状态搜索（State Searching）
  - 探索后续状态可能带来更高收益的策略

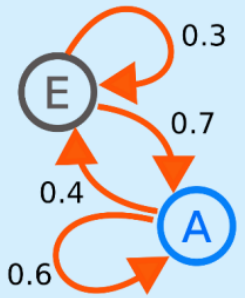
马尔可夫过程

- 马尔可夫过程（Markov Process）是具有马尔可夫性质的随机过程

  “The future is independent of the past given the present”

- 定义：
  - 状态S_t是马尔可夫的，当且仅当

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, \dots, S_t]$$



一个具有两个转换状态的马尔可夫链

- 性质：
  - 状态从历史（history）中捕获了所有相关信息
  - 当状态已知的时候，可以抛开历史不管
  - 也就是说，当前状态是未来的充分统计量

马尔可夫决策过程

- 马尔可夫决策过程（Markov Decision Process，MDP）
  - 提供了一套为在结果部分随机、部分在决策者的控制下的决策过程建模的数学框架

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, \dots, S_t]$$

$$\mathbb{P}[S_{t+1}|S_t, A_t]$$

- MDP形式化地描述了一种强化学习的环境
  - 环境完全可观测
  - 即，当前状态可以完全表征过程（马尔可夫性质）

MDP五元组

- MDP可以由一个五元组表示 $(S, A, \{P_{sa}\}, \gamma, R)$
  - S是状态的集合
    - 比如，迷宫中的位置，Atari游戏中的当前屏幕显示
  - A是动作的集合
    - 比如，向N、E、S、W移动，手柄操纵杆方向和按钮
  - P_{sa}是状态转移概率
    - 对每个状态$s \in S$和动作$a \in A$，P_{sa}是下一个状态在S中的概率分布
  - $\gamma \in [0,1]$是对未来奖励的折扣因子
  - $R: S \times A \mapsto \mathbb{R}$ 是奖励函数
    - 有时奖励只和状态相关

MDP的动态

- MDP的动态如下所示：
  - 从状态s_0开始
  - 智能体选择某个动作$a_0 \in A$
  - 智能体得到奖励$R(s_0, a_0)$
  - MDP随机转移到下一个状态$s_1 \sim P_{s_0 a_0}$
  - 这个过程不断进行

$$s_0 \xrightarrow{a_0, R(s_0,a_0)} s_1 \xrightarrow{a_1, R(s_1,a_1)} s_2 \xrightarrow{a_2, R(s_2,a_2)} s_3 \cdots$$

  - 直到终止状态s_T出现为止，或者无止尽地进行下去
  - 智能体的总回报为

$$R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \cdots$$

MDP的动态性

- 在大部分情况下，奖励只和状态相关
  - 比如，在迷宫游戏中，奖励只和位置相关
  - 在围棋中，奖励只基于最终所围地盘的大小有关
- 这时，奖励函数为 $R(s): S \mapsto \mathbb{R}$
- MDP的过程为

$$s_0 \xrightarrow{a_0, R(s_0)} s_1 \xrightarrow{a_1, R(s_1)} s_2 \xrightarrow{a_2, R(s_2)} s_3 \cdots$$

- 累积奖励为

$$R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$$

MDP目标和策略

- 目标：选择能够最大化累积奖励期望的动作

$$\mathbb{E}[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots]$$

  - $\gamma \in [0,1]$是未来奖励的折扣因子，使得和未来奖励相比起来智能体更重视即时奖励
    - 以金融为例，今天的$1比明天的$1更有价值

- 给定一个特定的策略 $\pi(s): S \to A$
  - 即，在状态 s 下采取动作 $a = \pi(s)$

- 给策略π定义价值函数

$$V^{\pi}(s) = \mathbb{E}[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots | s_0 = s, \pi]$$

  - 即，给定起始状态和根据策略π采取动作时的累积奖励期望

价值函数的Bellman等式

- 给策略π定义价值函数

$$V^{\pi}(s) = \mathbb{E}[R(s_0) + \underbrace{\gamma R(s_1) + \gamma^2 R(s_2) + \cdots}_{\gamma V^{\pi}(s_1)} | s_0 = s, \pi]$$

$$= R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^{\pi}(s') \qquad \text{Bellman等式}$$

立即奖励　　时间折扣　　状态转移　　下一个状态的价值

最优价值函数

- 对状态s来说的最优价值函数是所有策略可获得的最大可能折扣奖励的和

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

- 最优价值函数的Bellman等式

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s')$$

- 最优策略

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s')$$

- 对状态s和策略π

$$V^*(s) = V^{\pi^*}(s) \geq V^{\pi}(s)$$

价值迭代和策略迭代

- 价值函数和策略相关

$$V^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^{\pi}(s')$$

$$\pi(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^{\pi}(s')$$

- 可以对最优价值函数和最优策略执行迭代更新
  - 价值迭代
  - 策略迭代

价值迭代

- 对于一个动作空间和状态空间有限的MDP

$$|S| < \infty, |A| < \infty$$

- 价值迭代过程

1. 对每个状态s，初始化 $V(s) = 0$
2. 重复以下过程直到收敛 {

对每个状态，更新

$$V(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V(s')$$

}

注意：在以上的计算中没有明确的策略

策略迭代

- 对于一个动作空间和状态空间有限的MDP

$$|S| < \infty, |A| < \infty$$

- 策略迭代过程

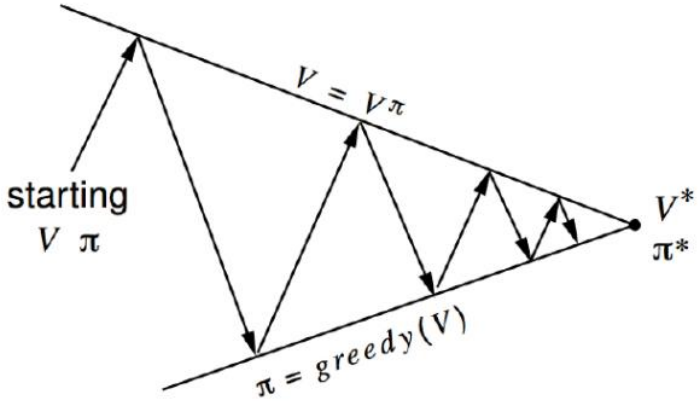
1. 随机初始化策略 π
2. 重复以下过程直到收敛{
   a) 让 $V := V^{\pi}$
   b) 对每个状态，更新

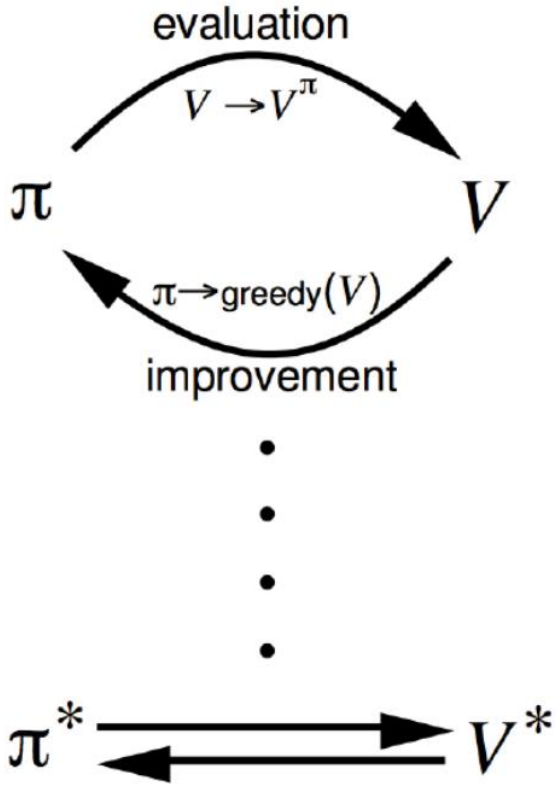
$$\pi(s) = arg \max_{a \in A} \sum_{s' \in S} P_{sa}(s') V(s')$$

}

更新价值函数会很耗时

策略迭代





- 策略评估
  - 估计V^{π}
  - 迭代的评估策略
- 策略改进
  - 生成 $\pi' \geq \pi$
  - 贪心策略改进

- main idea
  - use value functions to structure the search for good policies
  - need a perfect model of the environment

- two main components

  - policy evaluation: compute V^{π} from π
  - policy improvement: improve π based on V^{π}

  - start with an arbitrary policy
  - repeat evaluation/improvement until convergence

价值迭代 vs. 策略迭代

价值迭代

1. 对每个状态s，初始化 $V(s) = 0$
2. 重复以下过程直到收敛 {
   对每个状态，更新

$$V(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V(s')$$

   }

策略迭代

1. 随机初始化策略 π
2. 重复以下过程直到收敛 {
   a) 让 $V := V^{\pi}$
   b) 对每个状态，更新

$$\pi(s) = arg \max_{a \in A} \sum_{s' \in S} P_{sa}(s') V(s')$$

   }

备注：

1. 价值迭代是贪心更新法
2. 策略迭代中，用Bellman等式更新价值函数代价很大
3. 对于空间较小的MDP，策略迭代通常很快收敛
4. 对于空间较大的MDP，价值迭代更实用（效率更高）
5. 如果没有状态转移循环，最好使用价值迭代

时序差分学习（Temporal Difference Learning）

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = R_{t+1} + \gamma V(S_{t+1})$$

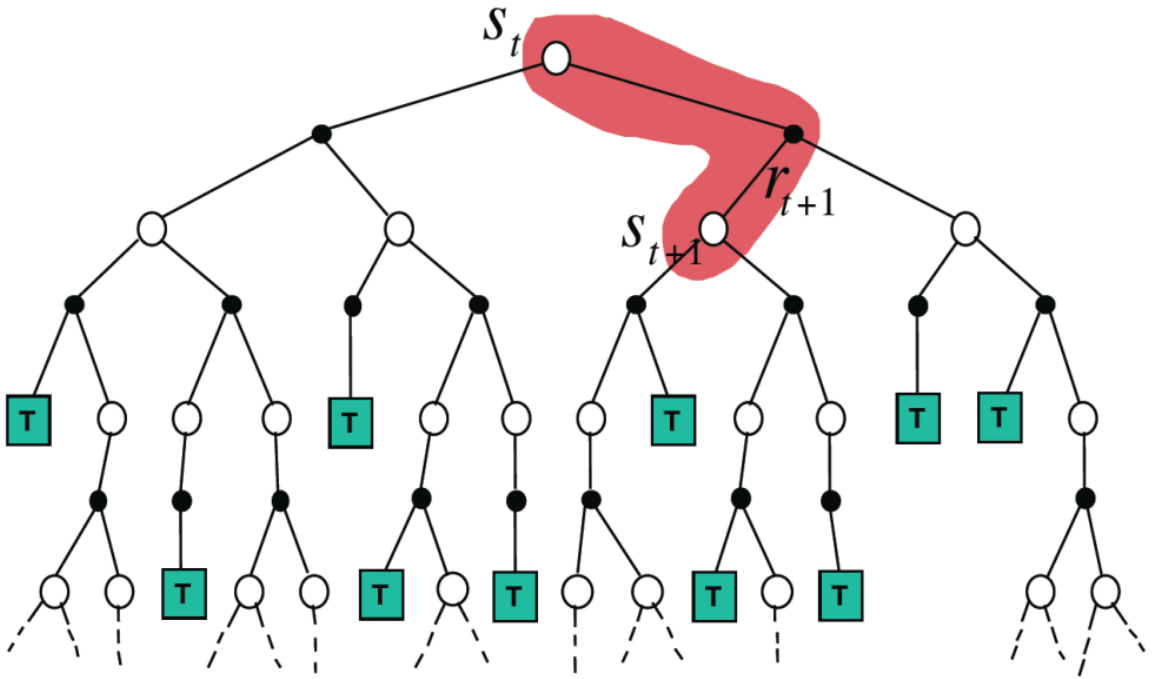
$$V(S_t) \leftarrow V(S_t) + \alpha\big(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\big)$$

观测值　　对未来的猜测

- 时序差分方法直接从经验片段中进行学习
- 时序差分是模型无关的
  - 不需要预先获取马尔可夫决策过程的状态转移/奖励
- 通过bootstrapping，时序差分从不完整的片段中学习
- 时序差分更新当前预测值使之接近估计累计奖励（非真实值）

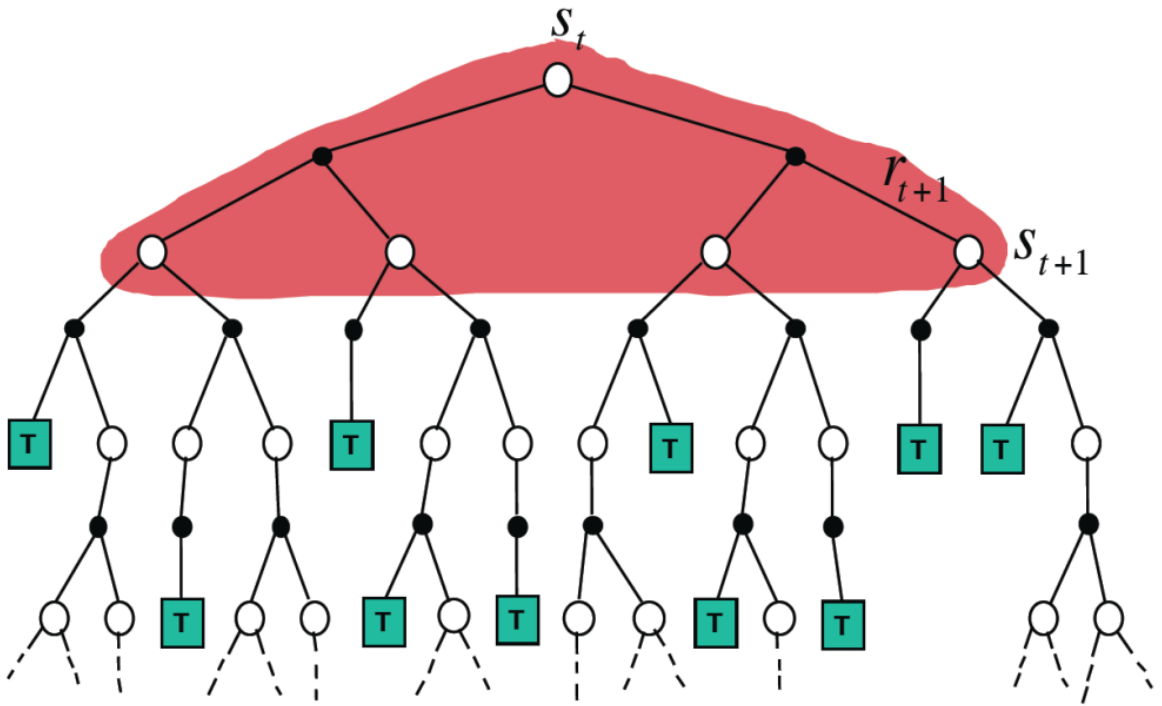
时序差分反向传播（Backup）

$$V(S_t) \leftarrow V(S_t) + \alpha\big(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\big)$$



动态规划反向传播（Backup）

$$V(S_t) \leftarrow \mathbb{E}[R_{t+1} + \gamma V(S_{t+1})]$$



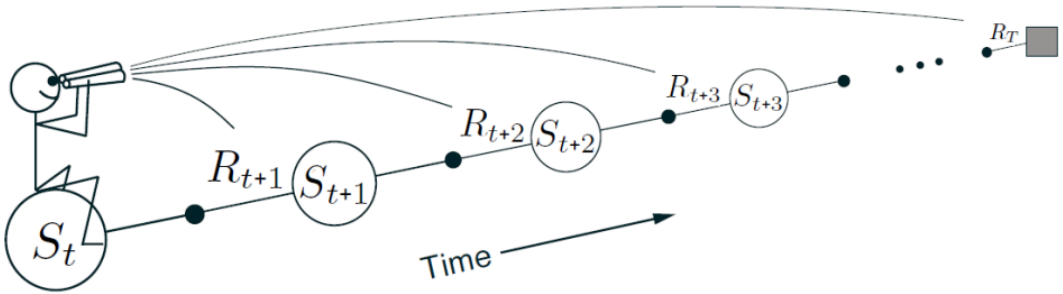
多步时序差分学习

- 对于有时间约束的情况，我们可以跳过n步预测的部分，直接进入模型无关的控制
- 定义n步累计奖励

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

- n步时序差分学习

$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t^{(n)} - V(S_t) \right)$$



从知道什么是好的，到如何做好行动

- 从知道什么是好的：估计$V^{\pi}(S_t)$

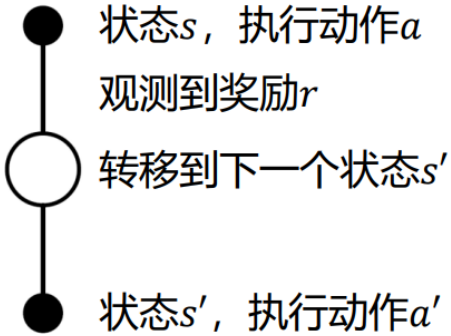
$$V(S_t) \leftarrow V(S_t) + \alpha\big(G_t - V(S_t)\big)$$

$$V(S_t) \leftarrow V(S_t) + \alpha\big(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\big)$$

- 基于V函数，如何选择好的行动？

$$\pi(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s')V(s')$$

需要知道环境模型



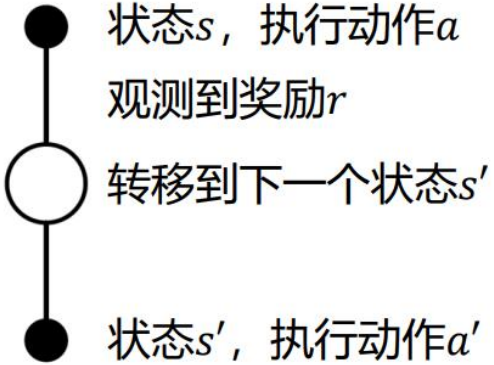
- 基于Q函数，如何选择好的行动？

$$\pi(s) = \arg\max_{a \in A} Q(s, a)$$

  - 因此，估计Q函数对直接做行动（控制）有直接的作用

SARSA

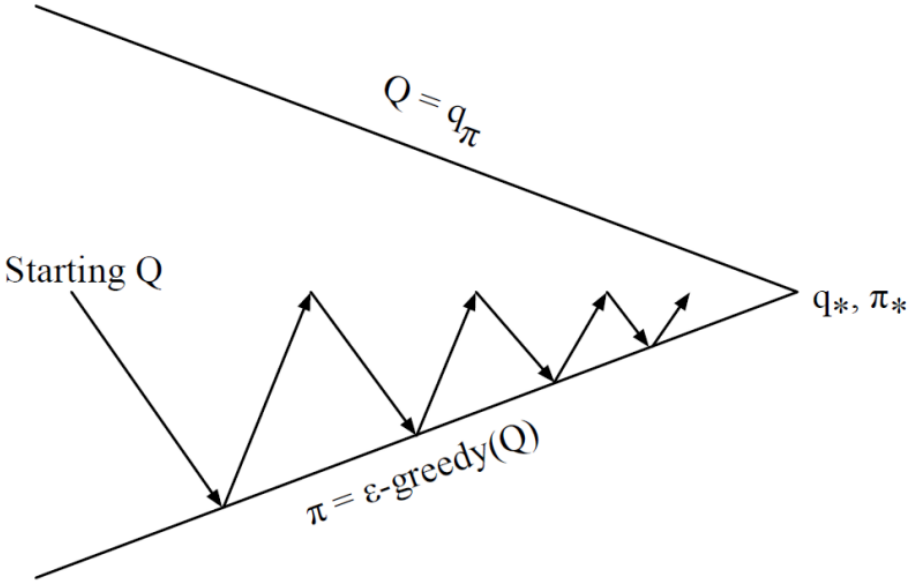
- 对于当前策略执行的每个（状态-动作-奖励-状态-动作）元组



- SARSA更新状态-动作值函数为

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))$$

使用SARSA的在线策略控制



- 每个时间步长：
  - 策略评估：SARSA $Q(s,a) \leftarrow Q(s,a) + \alpha\big(r + \gamma Q(s',a') - Q(s,a)\big)$
  - 策略改进：ε-greedy策略改进

SARSA算法

Sarsa: An on-policy TD control algorithm

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

注：在线策略时序差分控制（on-policy TD control）使用当前策略进行动作采样。即，SARSA算法中的两个“A”都是由当前策略选择的

Q 学习

- 学习状态-动作值函数 $Q(s,a) \in \mathbb{R}$，不直接优化策略
- 一种离线策略（off-policy）学习方法

策略函数，一般是给定的策略，$\mu(\cdot|\, s_t,) \in \mathbb{R}^{|A|}$

动作空间，$a \sim A$

$$Q(s_t,a_t) = \sum_{t=0}^{T} \gamma^t R(s_t,a_t), a_t \sim \mu(s_t)$$

奖励函数，$R(s_t,a_t) \in \mathbb{R}$

迭代式：$Q(s_t,a_t) = R(s_t,a_t) + \gamma Q(s_{t+1},a_{t+1})$

Q-Learning Algorithm for Off-Policy Control

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose A from S using policy derived from Q (e.g., ε-greedy)
        Take action A, observe R, S'
        $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
        $S \leftarrow S'$;

离线策略学习

什么是离线策略学习

- 目标策略 $\pi(a|s)$ 进行值函数评估（$V^{\pi}(s)$或$Q^{\pi}(s,a)$）
- 行为策略 $\mu(a|s)$ 收集数据：$\{s_1, a_1, r_2, s_2, a_2, \ldots, S_T\} \sim \mu$

为什么使用离线策略学习

- 平衡探索（exploration）和利用（exploitation）
- 通过观察人类或其他智能体学习策略
- 重用旧策略所产生的经验
- 遵循探索策略时学习最优策略
- 遵循一个策略时学习多个策略
- 在剑桥MSR研究时的一个例子
  - Collective Noise Contrastive Estimation for Policy Transfer Learning. AAAI 2016

使用Q 学习的离线策略控制

- 允许行为策略和目标策略都进行改进
- 目标策略π是关于$Q(s,a)$的贪心策略

$$\pi(s_{t+1}) = \arg\max_{a'} Q(s_{t+1}, a')$$

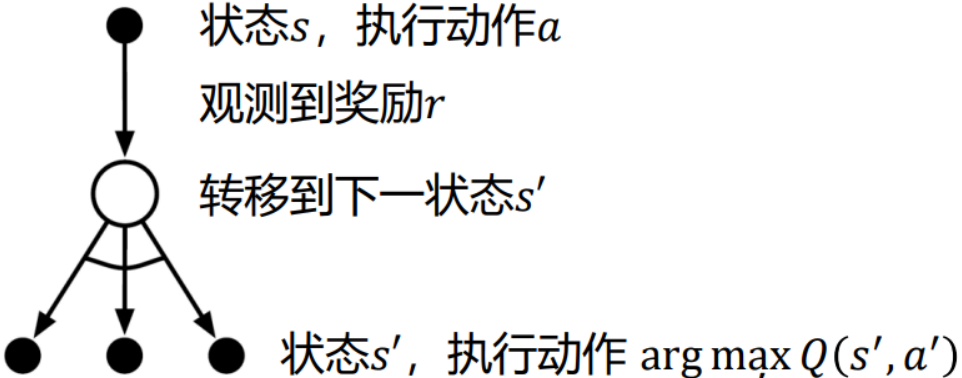
- 行为策略μ是关于$Q(s,a)$的ε-贪心策略
- Q-学习目标函数可以简化为

$$\begin{aligned} r_{t+1} + \gamma Q(s_{t+1}, a'_{t+1}) &= r_{t+1} + \gamma Q(s_{t+1}, \arg\max_{a'_{t+1}} Q(s_{t+1}, a'_{t+1})) \\ &= r_{t+1} + \gamma \max_{a'_{t+1}} Q(s_{t+1}, a'_{t+1}) \end{aligned}$$

- Q-学习更新方式

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a'_{t+1}} Q(s_{t+1}, a'_{t+1}) - Q(s_t, a_t))$$

Q 学习控制算法

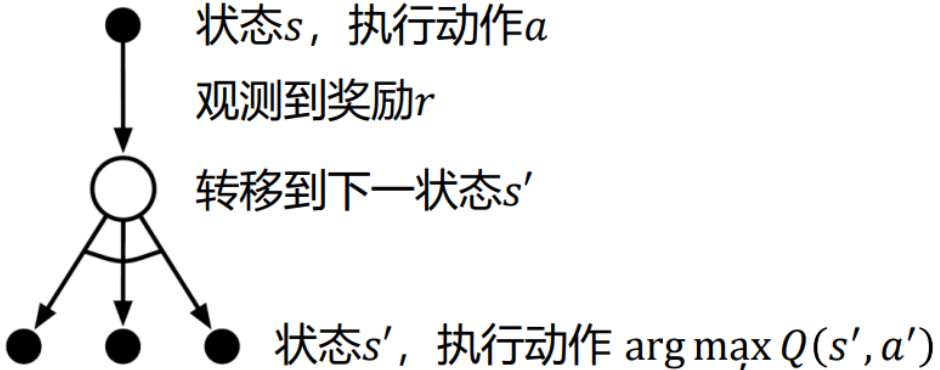


$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a'_{t+1}} Q(s_{t+1}, a'_{t+1}) - Q(s_t, a_t))$$

- 定理：Q-学习控制收敛到最优状态-动作值函数

$$Q(s, a) \rightarrow Q^*(s, a)$$

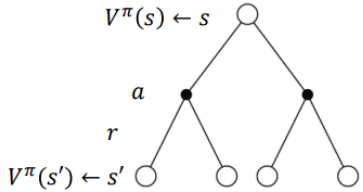
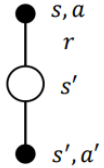
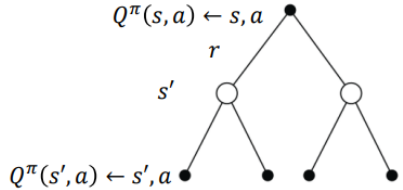
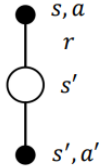
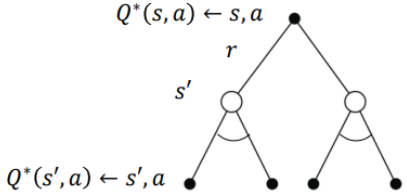
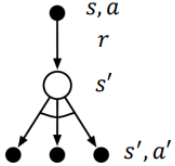
Q 学习控制算法



$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))$$

- 为什么不需要重要性采样?
  - 使用了状态-动作值函数而不是使用状态值函数

回顾：动态规划（DP）和时序差分（TD）的关系

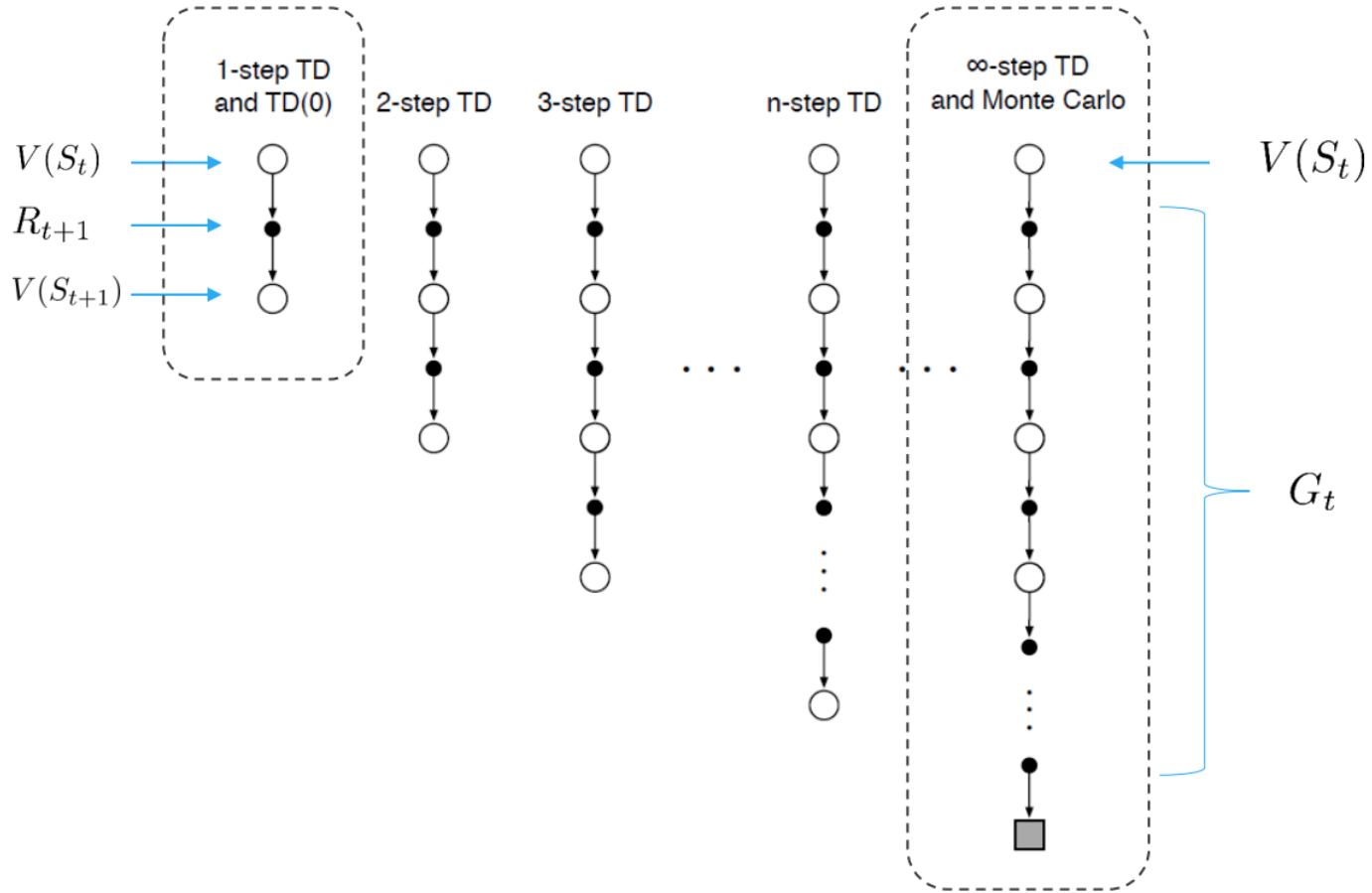
	完全反向传播(DP)	采样反向传播(TD)
状态值函数的贝尔曼期望方程 $V^{\pi}(s)$	 迭代的策略评估	 时序差分学习
状态-动作值函数的贝尔曼期望方程 $Q^{\pi}(s,a)$	 $Q-$策略迭代	 SARSA
状态-动作值函数的贝尔曼最优方程 $Q^{*}(s,a)$	 $Q-$价值迭代	 $Q-$学习

回顾：动态规划（DP）和时序差分（TD）的关系

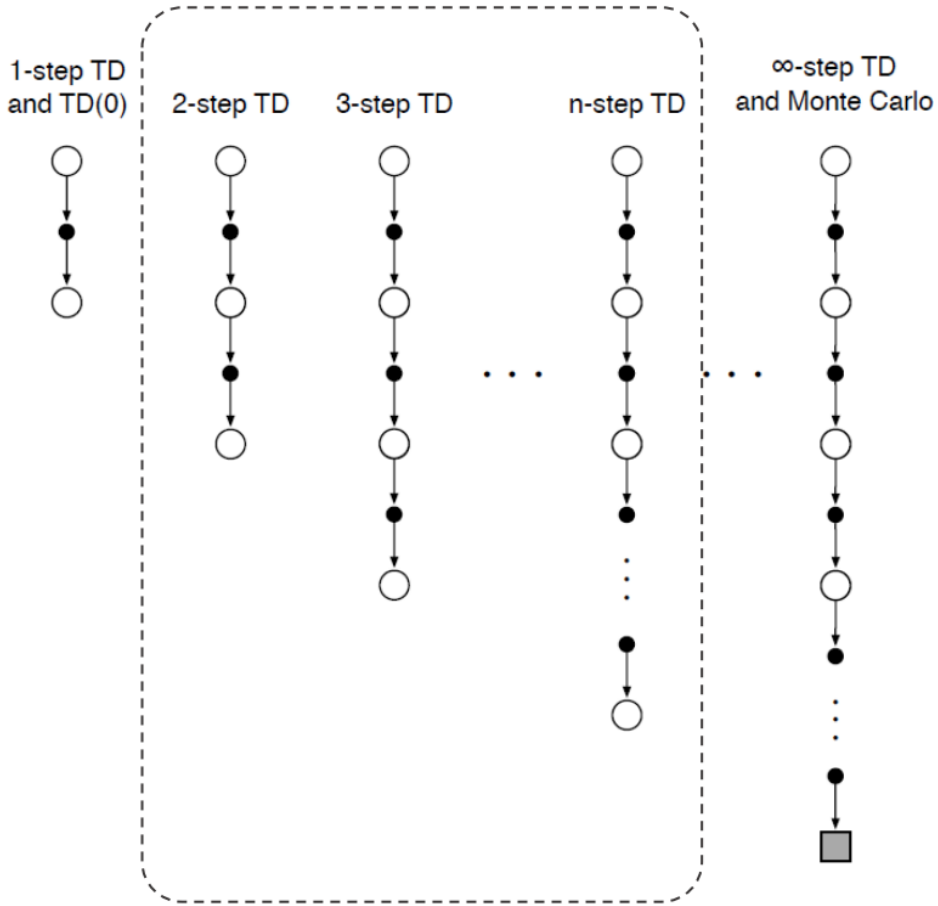
完全反向传播(DP)	采样反向传播(TD)
迭代的策略评估 $V(s) \leftarrow \mathbb{E}[r + \gamma V(s')\|s]$	时序差分学习 $V(s) \overset{\alpha}{\leftarrow} r + \gamma V(s')$
Q – 策略迭代 $Q(s,a) \leftarrow \mathbb{E}[r + \gamma Q(s',a')\|s,a]$	SARSA $Q(s,a) \overset{\alpha}{\leftarrow} r + \gamma Q(s',a')$
Q – 价值迭代 $Q(s,a) \leftarrow \mathbb{E}\left[r + \gamma \max_{a'} Q(s',a') \middle\| s,a\right]$	Q – 学习 $Q(s,a) \overset{\alpha}{\leftarrow} r + \gamma \max_{a'} Q(s',a')$

其中 $x \overset{\alpha}{\leftarrow} y \equiv x \leftarrow x + \alpha(y - x)$

多步时序差分预测


1-step TD
and TD(0)
2-step TD
3-step TD
n-step TD
∞-step TD
and Monte Carlo
$V(S_t)$
R_{t+1}
$V(S_{t+1})$
$V(S_t)$
G_t

多步时序差分预测


1-step TD
and TD(0)
2-step TD
3-step TD
n-step TD
∞-step TD
and Monte Carlo

n 步累计奖励

- 考虑下列 n 步累计奖励，$n = 1,2,\ldots,\infty$

$n = 1$	(TD)	$G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1})$
$n = 2$		$G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2})$
⋮		⋮
$n = \infty$	(MC)	$G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$

- 定义 n 步累计奖励

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

- n 步时序差分学习

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^{(n)} - V(S_t))$$

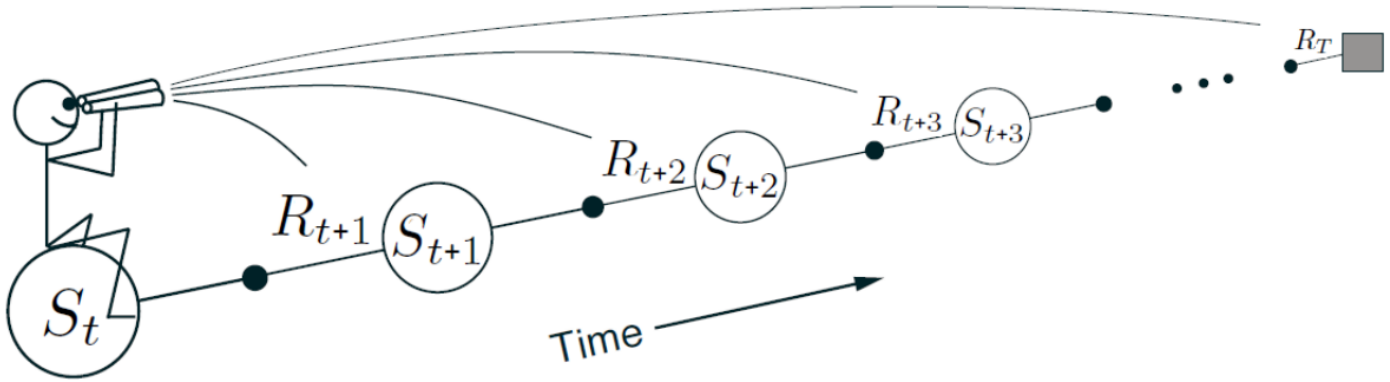
n 步累计奖励

- 定义 n 步累计奖励

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

- n 步时序差分学习

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^{(n)} - V(S_t))$$



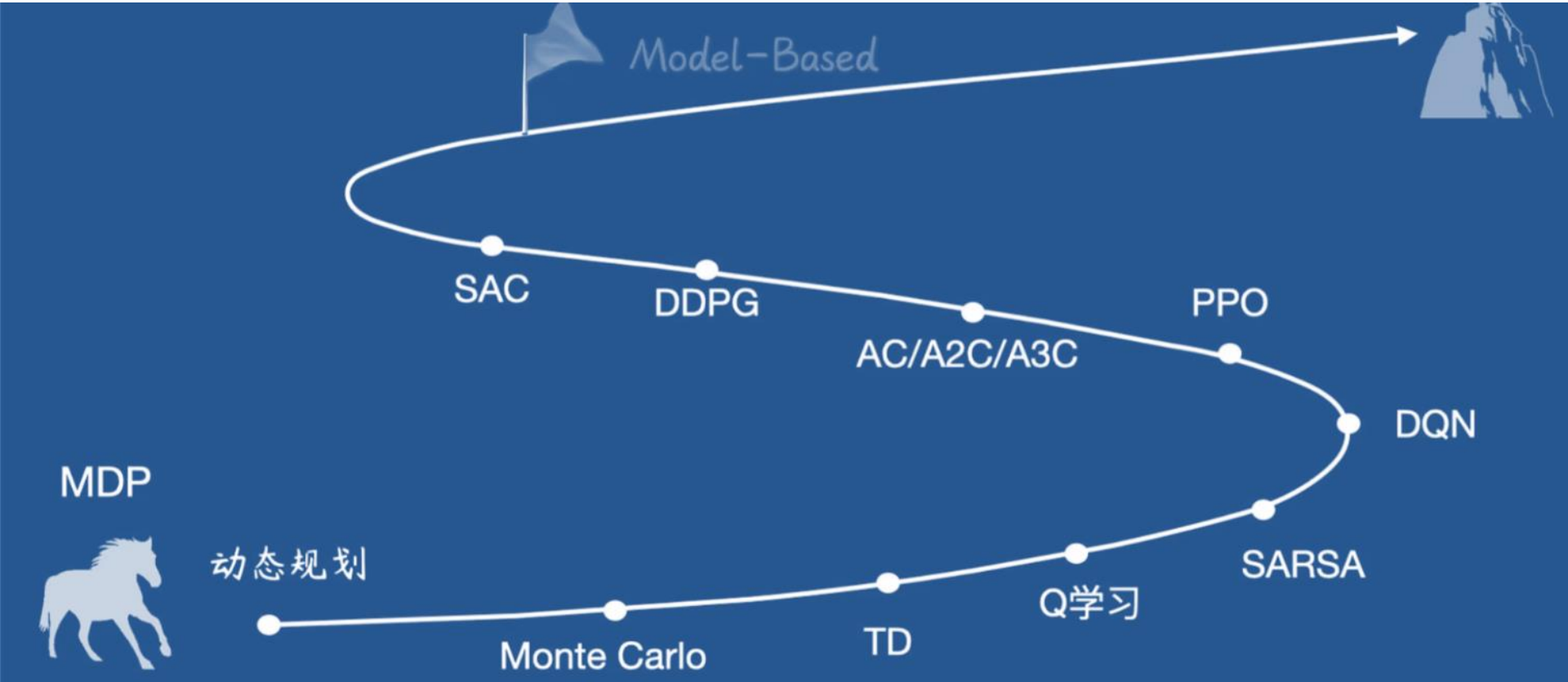
提纲

一、萌芽：从DP到Q Learning

二、大成：从DQN到GATO

三、巅峰：从AlphaGo到MuZero

强化学习


Model-Based
SAC
DDPG
AC/A2C/A3C
PPO
DQN
MDP
动态规划
Monte Carlo
TD
Q学习
SARSA

深度强化学习的崛起

- 2012年AlexNet在ImageNet比赛中大幅度领先对手获得冠军
- 2013年12月，第一篇深度强化学习论文出自NIPS 2013 Reinforcement Learning Workshop

Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih **Koray Kavukcuoglu** **David Silver** **Alex Graves** **Ioannis Antonoglou**

Daan Wierstra **Martin Riedmiller**

DeepMind Technologies

{vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller} @ deepmind.com

DeepMind

“Human-level control through deep reinforcement learning”

letter

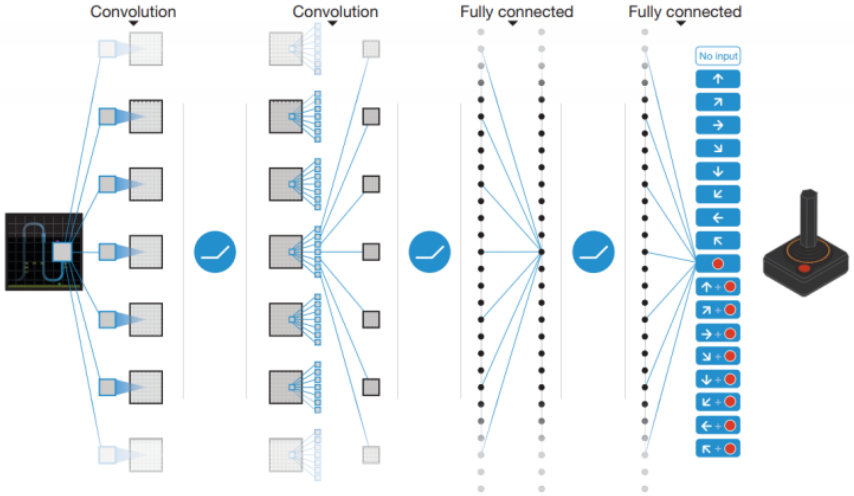
2015

Deep Q-Learning

深度强化学习

深度强化学习

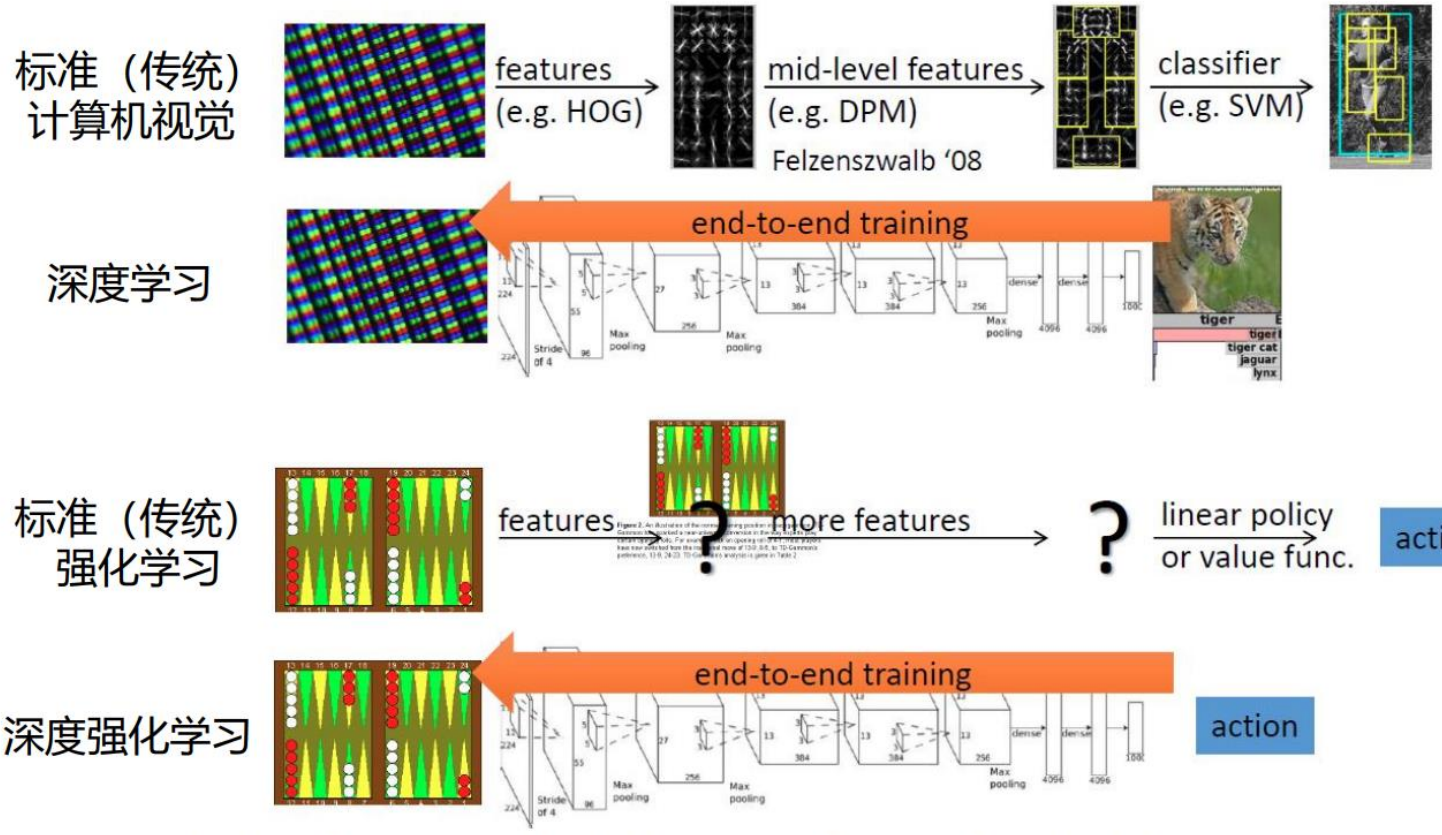
- 利用深度神经网络进行价值函数和策略近似
- 从而使强化学习算法能够以端到端的方式解决复杂问题



$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{E}}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)\right) \nabla_{\theta_i} Q(s, a; \theta_i)\right]$$

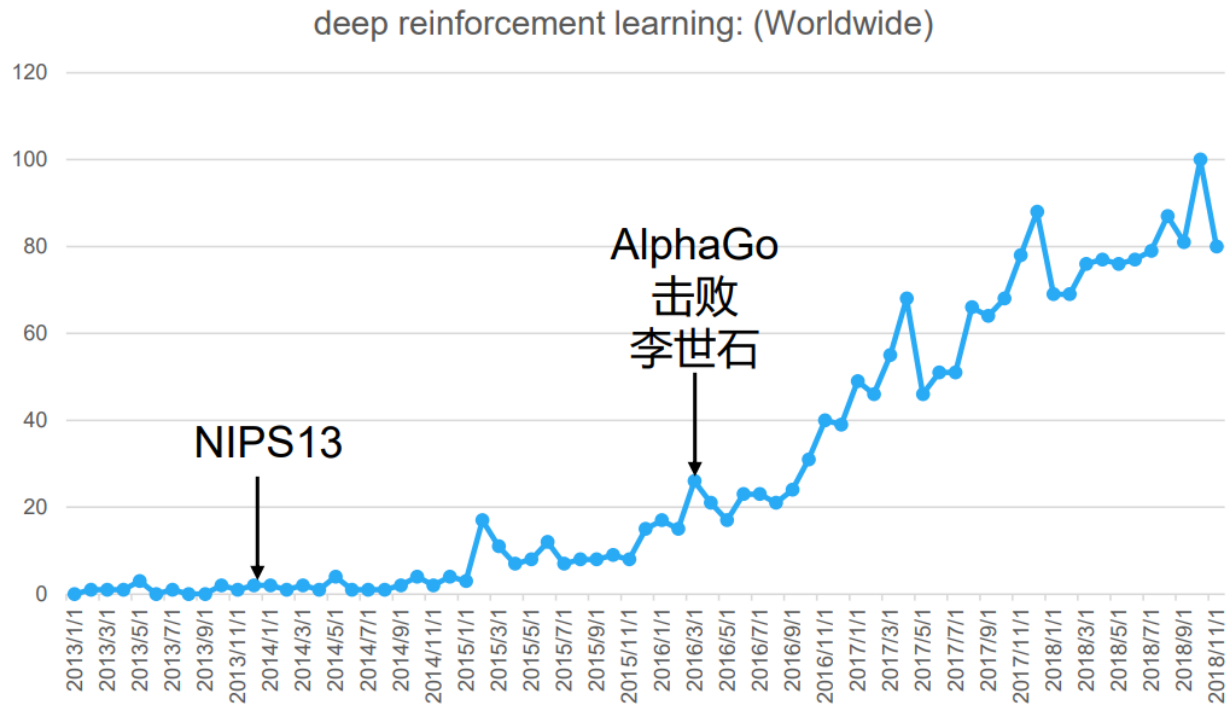
Q函数的参数通过神经网络反向传播学习

端到端强化学习



- 深度强化学习使强化学习算法能够以端到端的方式解决复杂问题
- 从一项实验室学术技术变成可以产生GDP的实际技术

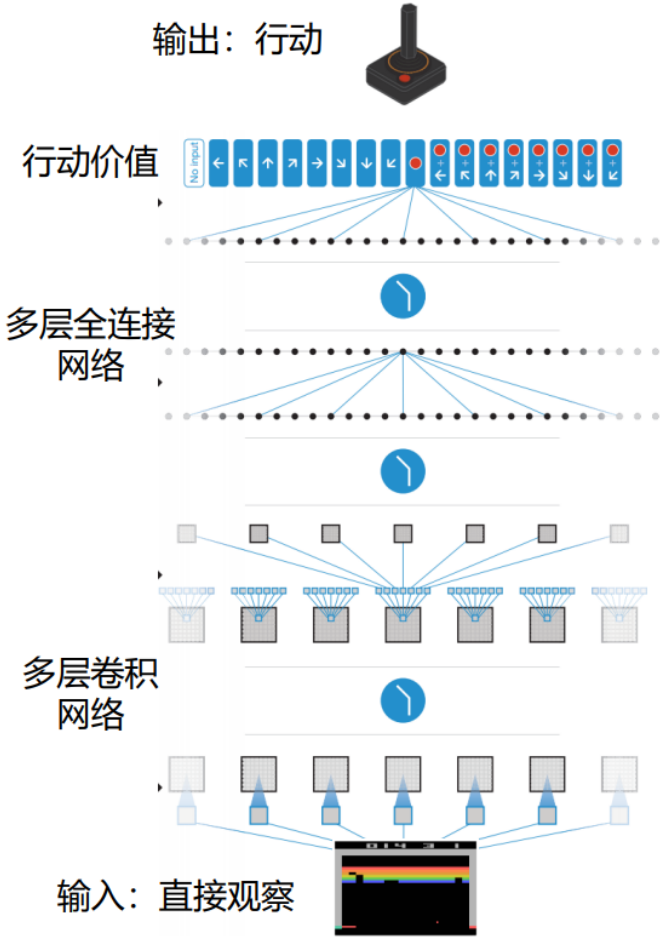
深度强化学习趋势



- Google搜索中词条“深度强化学习（deep reinforcement learning）”的趋势

深度强化学习带来的关键变化

- 将深度学习（DL）和强化学习（RL）结合在一起会发生什么？
  - 价值函数和策略变成了深度神经网络
  - 相当高维的参数空间
  - 难以稳定地训练
  - 容易过拟合
  - 需要大量的数据
  - 需要高性能计算
  - CPU（用于收集经验数据）和GPU（用于训练神经网络）之间的平衡
  - …
- 这些新的问题促进着深度强化学习算法的创新



深度Q网络（DQN）

- Q 学习算法学习一个由 θ 作为参数的函数 $Q_\theta(s,a)$
  - 更新方程 $Q_\theta(s_t,a_t) \leftarrow Q_\theta(s_t,a_t) + \alpha(r_t + \gamma \max_{a'} Q_\theta(s_{t+1},a') - Q_\theta(s_t,a_t))$

直观想法

- 使用神经网络来逼近 $Q_\theta(s,a)$，面对算法不稳定问题
  - 连续采样得到的 $\{(s_t,a_t,s_{t+1},r_t)\}$ 不满足独立分布
  - $\{(s_t,a_t,s_{t+1},r_t)\}$ 为状态-动作-下一状态-回报输入
  - $Q_\theta(s,a)$ 的频繁更新

解决办法

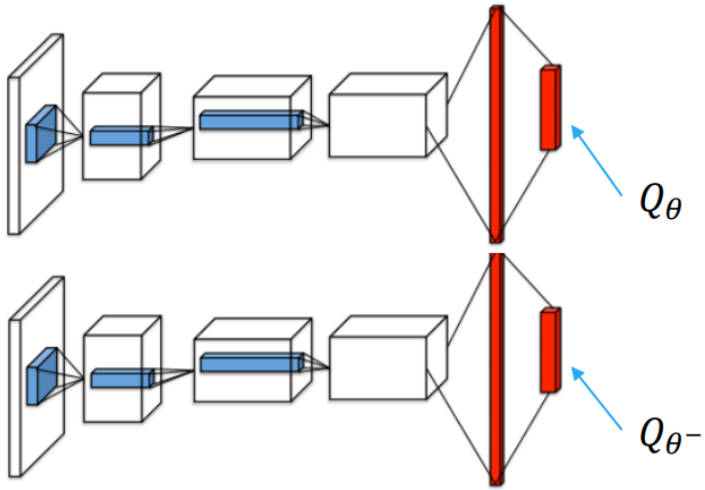
- 经验回放：均匀采样和优先经验回放
- 使用双网络结构：评估网络（evaluation network）和目标网络（target network）

"Human-Level Control Through Deep Reinforcement Learning"，Mnih, Kavukcuoglu, Silver et al. (2015)

目标网络

- 目标网络 $Q_{\theta^-}(s,a)$
  - 使用较旧的参数，记为 θ^-，每隔 C 步和训练网络的参数同步一次。
  - 第i次迭代的损失函数为

$$L_i(\theta_i) = \mathbb{E}_{s_t,a_t,s_{t+1},r_t,p_t\sim D}[\frac{1}{2}\omega_t(\underbrace{r_t + \gamma \max_{a'} Q_{\theta_i^-}(s_{t+1},a')}_{target} - Q_{\theta_i}(s_t,a_t))^2]$$

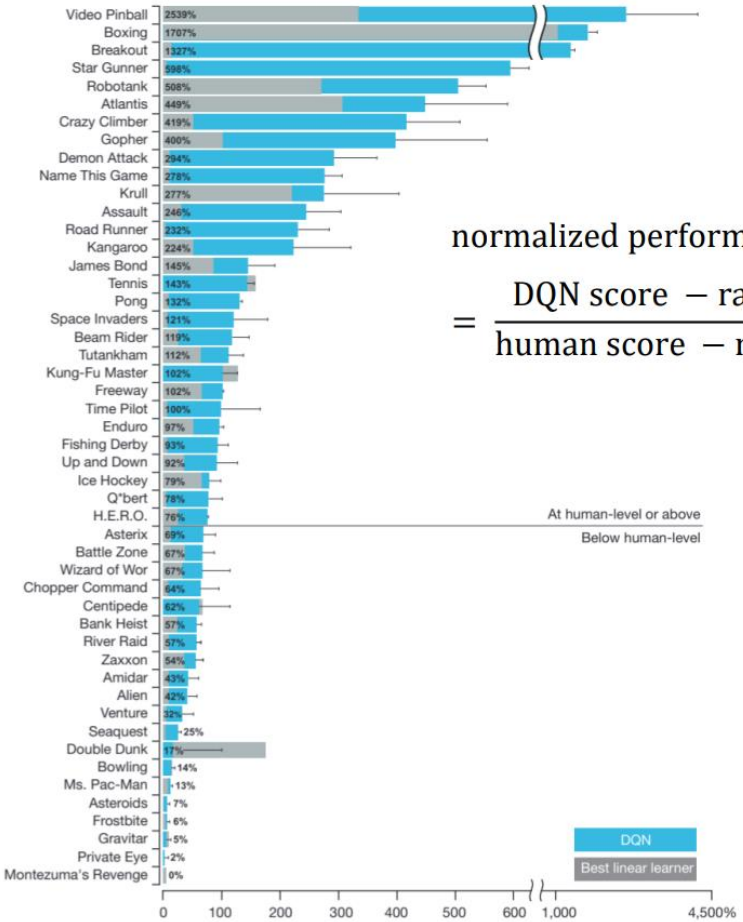


评估网络（evaluation network）

目标网络（target network）是评估网络几轮前的拷贝，不更新

"Human-Level Control Through Deep Reinforcement Learning"，Mnih, Kavukcuoglu, Silver et al. (2015)

在 Atari 环境中的实验结果



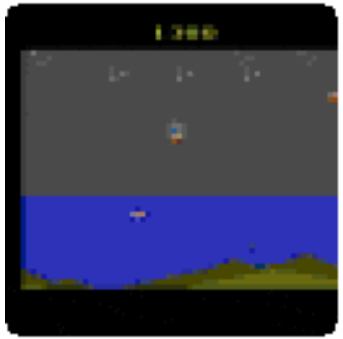
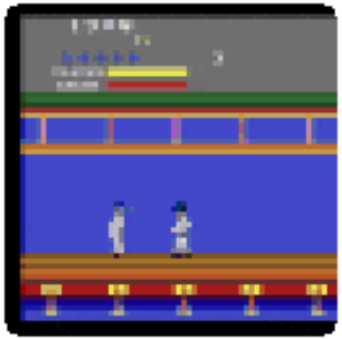
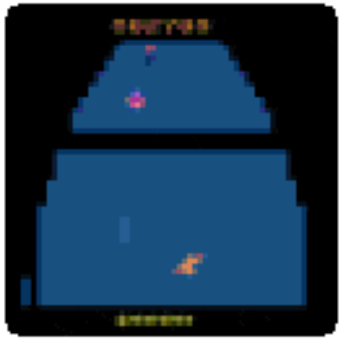
$$\text{normalized performance} = \frac{\text{DQN score} - \text{random play score}}{\text{human score} - \text{random play score}}$$

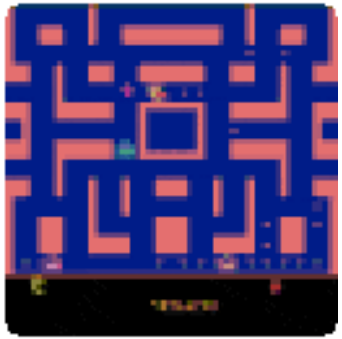
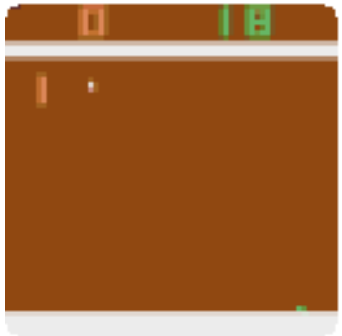
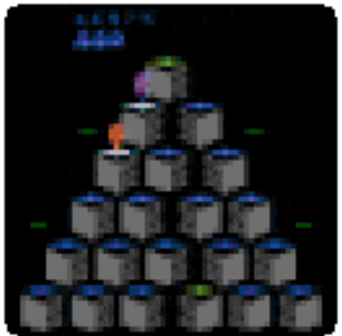
The performance of DQN is normalized with respect to a professional human games tester (that is, 100% level)

"Human-Level Control Through Deep Reinforcement Learning", Mnih, Kavukcuoglu, Silver et al. (2015)






Published as a conference paper at ICLR 2016

CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING

Timothy P. Lillicrap,* Jonathan J. Hunt,* Alexander Pritzel, Nicolas Heess,
Tom Erez, Yuval Tassa, David Silver & Daan Wierstra
Google Deepmind

DDPG

DDPG (Deep DPG) is a model-free, off-policy, actor-critic algorithm that combines:

- **DPG** (Deterministic Policy Gradients, Silver et al., ‘14): works over continuous action domain, not learning-based

- **DQN** (Deep Q-Learning**,** Mnih et al., ‘13): learning-based, doesn’t work over continuous action domain

Method

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Discussion of Experiment Results

- Target Networks and Batch Normalization are crucial
- DDPG is able to learn tasks over continuous domain, with better performance than DPG, but the variance in performance is still pretty high
- Q values estimated are quite accurate (compared to the true expected reward) in simple tasks, but not so accurate for more complicated tasks

Conclusion

- DDPG = DPG + DQN
- Big Idea is to bypass finding the local max of Q in DQN by jointly training a second neural network (actor) to predict the local max of Q.
- Tricks that made DDPG possible:
  - Replay buffer, target networks (from DQN)
  - Batch normalization, to allow transfer between different RL tasks with different state scales
  - Directly add noise to policy output for exploration, due to continuous action domain
- Despite these tricks, DDPG can still be sensitive to hyperparameters. TD3 and SAC offer better stability.

A cool application of DDPG: Wayve

Learning to Drive in a Day (Alex Kendall et al, 2018)

We selected a simple continuous action domain model-free reinforcement learning algorithm: deep deterministic policy gradients (DDPG) [8], to show that an off-the-shelf reinforcement learning algorithm with no task-specific adaptation is capable of solving the MDP posed in Section III-A.

DDPG consists of two function approximators: a critic $Q\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, which estimates the value $Q(s, a)$ of the expected cumulative discounted reward upon using action a in state s, trained to satisfy the Bellman equation

$$Q(s_t, a_t) = r_{t+1} + \gamma(1 - d_t)Q(s_{t+1}, \pi(s_{t+1})),$$

under a policy given by the actor $\pi\colon \mathcal{S} \to \mathcal{A}$, which attempts to estimate a Q-optimal policy $\pi(s) = \text{argmax}_a Q(s, a)$; here $(s_t, a_t, r_{t+1}, d_{t+1}, s_{t+1})$ is an experience tuple, a transition

2017

TRPO

PPO

Trust Region Policy Optimization

John Schulman JOSCHU@EECS.BERKELEY.EDU
Sergey Levine SLEVINE@EECS.BERKELEY.EDU
Philipp Moritz PCMORITZ@EECS.BERKELEY.EDU
Michael Jordan JORDAN@CS.BERKELEY.EDU
Pieter Abbeel PABBEEL@CS.BERKELEY.EDU
University of California, Berkeley, Department of Electrical Engineering and Computer Sciences

Emergence of Locomotion Behaviours in Rich Environments

Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin Riedmiller, David Silver
DeepMind

Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov
OpenAI

Policy Gradient (REINFORCE)

REINFORCE(s_0, π_θ)

Initialize π_θ to anything

Loop forever (for each episode)

Generate episode $s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T, a_T, r_T$ with π_θ

Loop for each step of the episode $n = 0, 1, \ldots, T$

$G_n \leftarrow \sum_{t=0}^{T-n} \gamma^t r_{n+t}$

Update policy: $\theta \leftarrow \theta + \alpha \gamma^n G_n \nabla \log \pi_\theta(a_n|s_n)$

Return π_θ

In practice, update on each batch(trajectory)

* Use the same notation in the paper

$$\max_\theta J(\pi_\theta) \doteq \mathop{\mathrm{E}}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]$$

$$g = \nabla_\theta J(\pi_\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t \nabla_\theta \log \pi_\theta(a_t|s_t) A^{\pi_\theta}(s_t, a_t) \right]$$

Problem?

- **Unstable update**
  - Step size is very important:
    - If step size is too large:
      - Large step → bad policy
      - Next batch is generated from current bad policy → collect bad samples
      - Bad samples → worse policy

      (compare to supervised learning: the correct label and data in the following batches may correct it)
    - If step size is too small: the learning process is slow
- **Data Inefficiency**
  - On-policy method: for each new policy, we need to generate a completely new trajectory
  - The data is thrown out after just one gradient update
  - As complex neural networks need many updates, this makes the training process very slow

Trust Region Policy Optimization (TRPO)

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t \right] \\ \text{subject to} \quad & \hat{\mathbb{E}}_t[\mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]] \leq \delta \end{aligned}$$

Common trick in optimization: Lagrangian Dual

$$\underset{\theta}{\text{maximize}} \, \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t - \beta \, \mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)] \right]$$

TRPO uses a hard constraint rather than a penalty because it is hard to choose a single value of β that performs well across different problems—or even within a single problem, where the characteristics change over the course of learning

Proximal Policy Optimization (PPO)

TRPO use conjugate gradient decent to handle the constraint

Hessian Matrix → expensive both in computation and space

Idea:
The constraint helps in the training process. However, maybe the constraint is not a strict constraint:
Does it matter if we only break the constraint just a few times?

What if we treat it as a "soft" constraint? Add proximal value to objective function?

PPO with Adaptive KL Penalty

$$L^{KLPEN}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t - \beta \, \text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)] \right]$$

Hard to pick β value → use adaptive β

Compute $d = \hat{\mathbb{E}}_t[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]]$

- If $d < d_{\text{targ}}/1.5$, $\beta \leftarrow \beta/2$
- If $d > d_{\text{targ}} \times 1.5$, $\beta \leftarrow \beta \times 2$

Still need to set up a KL divergence target value …

PPO with Adaptive KL Penalty

Algorithm 4 PPO with Adaptive KL Penalty

Input: initial policy parameters θ_0, initial KL penalty β_0, target KL-divergence δ
for $k = 0, 1, 2, \ldots$ **do**
  Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
  Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
  Compute policy update

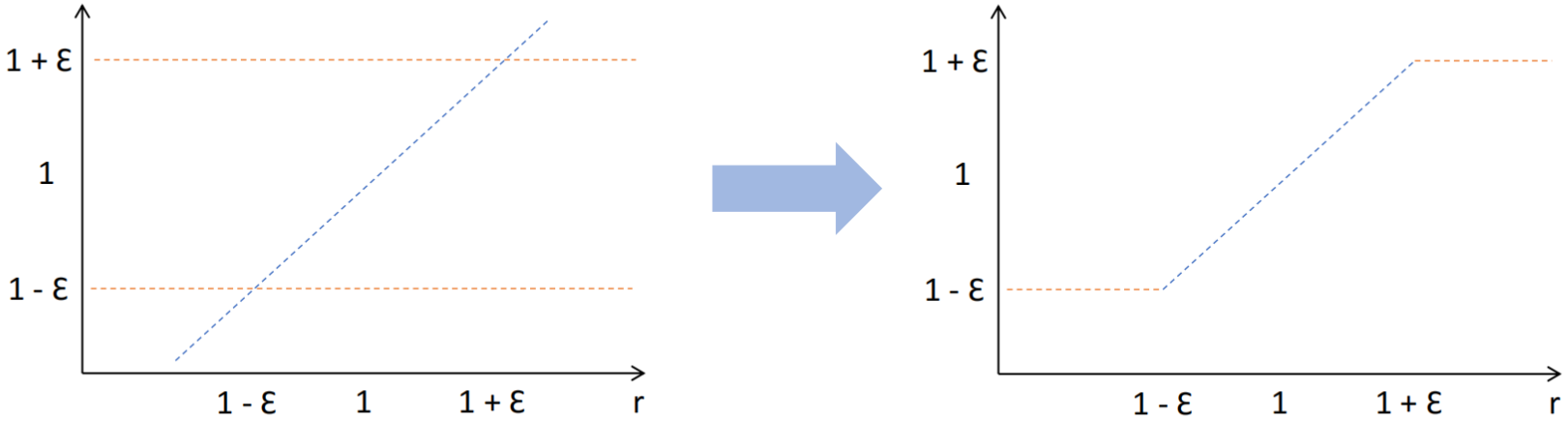
$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}(\theta) - \beta_k \bar{D}_{KL}(\theta || \theta_k)$$

  by taking K steps of minibatch SGD (via Adam)
  if $\bar{D}_{KL}(\theta_{k+1} || \theta_k) \geq 1.5\delta$ **then**
    $\beta_{k+1} = 2\beta_k$
  else if $\bar{D}_{KL}(\theta_{k+1} || \theta_k) \leq \delta / 1.5$ **then**
    $\beta_{k+1} = \beta_k / 2$
  end if
end for

PPO with Clipped Objective

$$\underset{\theta}{\text{maximize}} \quad \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right] \qquad r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}$$

Fluctuation happens when r changes too quickly → limit r within a range?



$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

PPO with Clipped Objective

Algorithm 5 PPO with Clipped Objective

Input: initial policy parameters θ_0, clipping threshold ϵ
for $k = 0, 1, 2, \ldots$ **do**
Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
Compute policy update

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}^{CLIP}(\theta)$$

by taking K steps of minibatch SGD (via Adam), where

$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_k} \left[\sum_{t=0}^{T} \left[\min(r_t(\theta)\hat{A}_t^{\pi_k}, \mathrm{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t^{\pi_k}) \right] \right]$$

end for

PPO in practice

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t\left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)\right]$$

Surrogate objective function

a squared-error loss for "critic"

$$(V_\theta(s_t) - V_t^{\text{targ}})^2$$

entropy bonus to ensure sufficient exploration

encourage "diversity"

* c1, c2: empirical values, in the paper, c1=1, c2=0.01

Performance

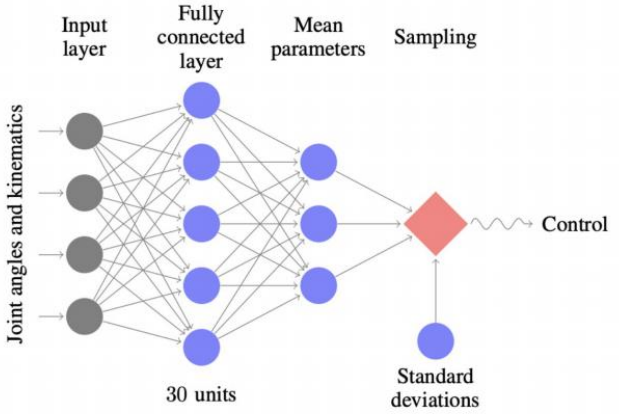
No clipping or penalty:	$L_t(\theta) = r_t(\theta)\hat{A}_t$
Clipping:	$L_t(\theta) = \min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta)), 1-\epsilon, 1+\epsilon)\hat{A}_t$
KL penalty (fixed or adaptive)	$L_t(\theta) = r_t(\theta)\hat{A}_t - \beta\,\text{KL}[\pi_{\theta_{\text{old}}}, \pi_\theta]$

Results from continuous control benchmark. Average normalized scores (over 21 runs of the algorithm, on 7 environments)

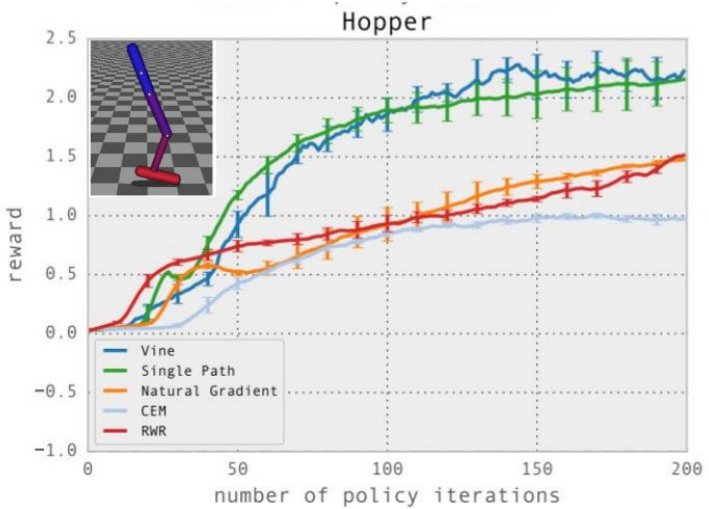
algorithm	avg. normalized score
No clipping or penalty	-0.39
Clipping, $\epsilon = 0.1$	0.76
Clipping, $\epsilon = 0.2$	**0.82**
Clipping, $\epsilon = 0.3$	0.70
Adaptive KL $d_{\text{targ}} = 0.003$	0.68
Adaptive KL $d_{\text{targ}} = 0.01$	0.74
Adaptive KL $d_{\text{targ}} = 0.03$	0.71
Fixed KL, $\beta = 0.3$	0.62
Fixed KL, $\beta = 1.$	0.71
Fixed KL, $\beta = 3.$	0.72
Fixed KL, $\beta = 10.$	0.69

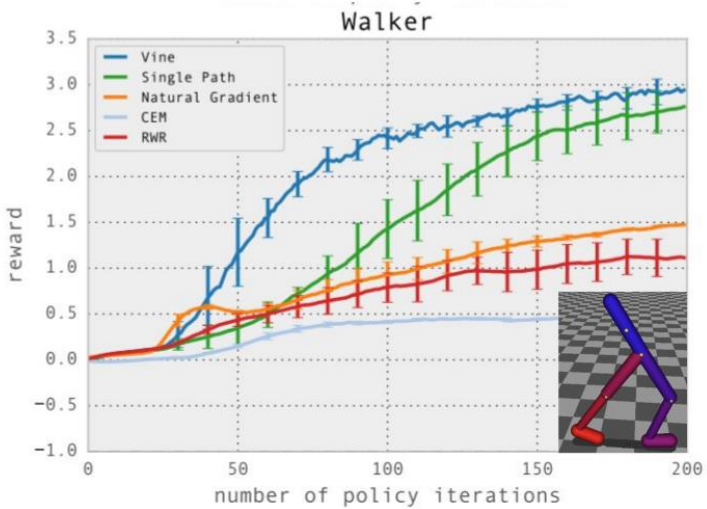
Performance

Experiments (TRPO)

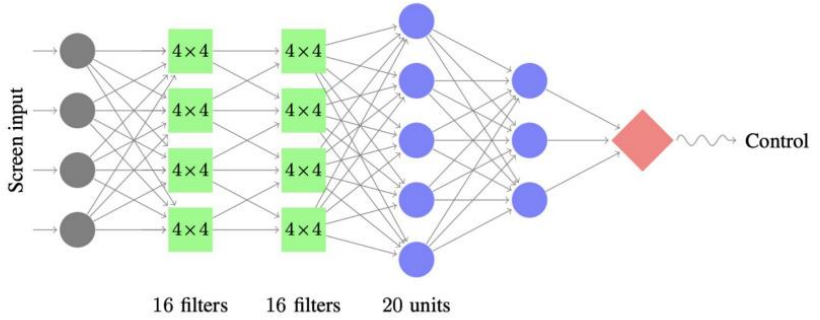


- Simulated Robotic Locomotion tasks
  - Hopper: 12-dim state space
  - Walker: 18-dim state space
  - rewards: encourage fast and stable running (hopper); encourage smooth walke (walker)

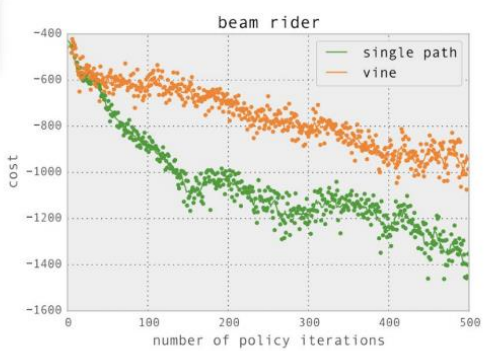


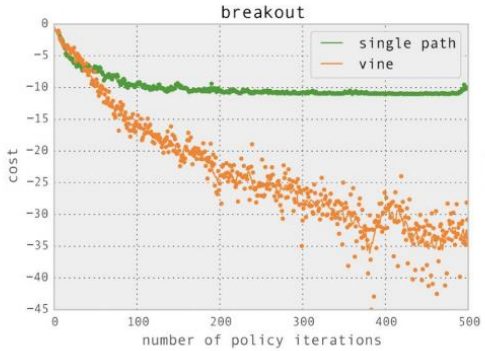


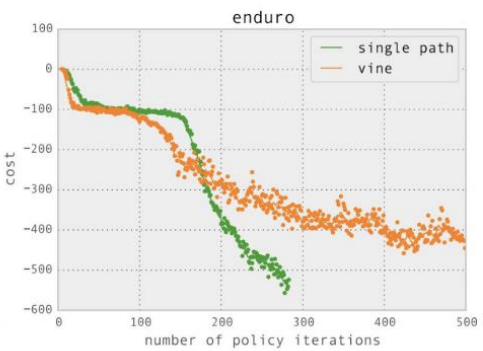
Experiments (TRPO)



- Atari games (discrete action space) - 0 / 1







	B. Rider	*Breakout*	*Enduro*	*Pong*	*Q*bert*	*Seaquest*	*S. Invaders*
Random	354	1.2	0	−20.4	157	110	179
Human (Mnih et al., 2013)	7456	31.0	368	−3.0	18900	28010	3690
Deep Q Learning (Mnih et al., 2013)	4092	168.0	470	20.0	1952	1705	581
UCC-I (Guo et al., 2014)	5702	380	741	21	20025	2995	692
TRPO - single path	1425.2	10.8	534.6	20.9	1973.5	1908.6	568.4
TRPO - vine	859.5	34.2	430.8	20.9	7732.5	788.4	450.2

Limitations of TRPO

- Hard to use with architectures with multiple outputs, e.g., policy and value function (need to weight different terms in distance metric)

- Empirically performs poorly on tasks requiring deep CNNs and RNNs, e.g., Atari benchmark (more suitable for locomotion)

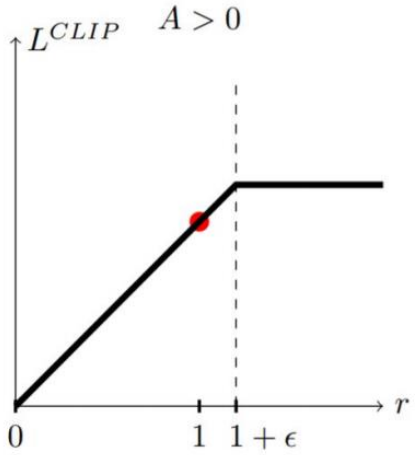
- Conjugate gradients makes implementation more complicated than SGD

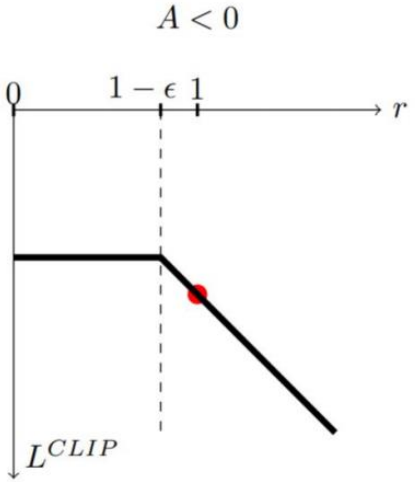
Proximal Policy Optimization (PPO)

- Clipped surrogate objective

TRPO: $$L^{CPI}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right] = \hat{\mathbb{E}}_t\left[r_t(\theta)\hat{A}_t\right]$$

PPO: $$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$





Proximal Policy Optimization (PPO)

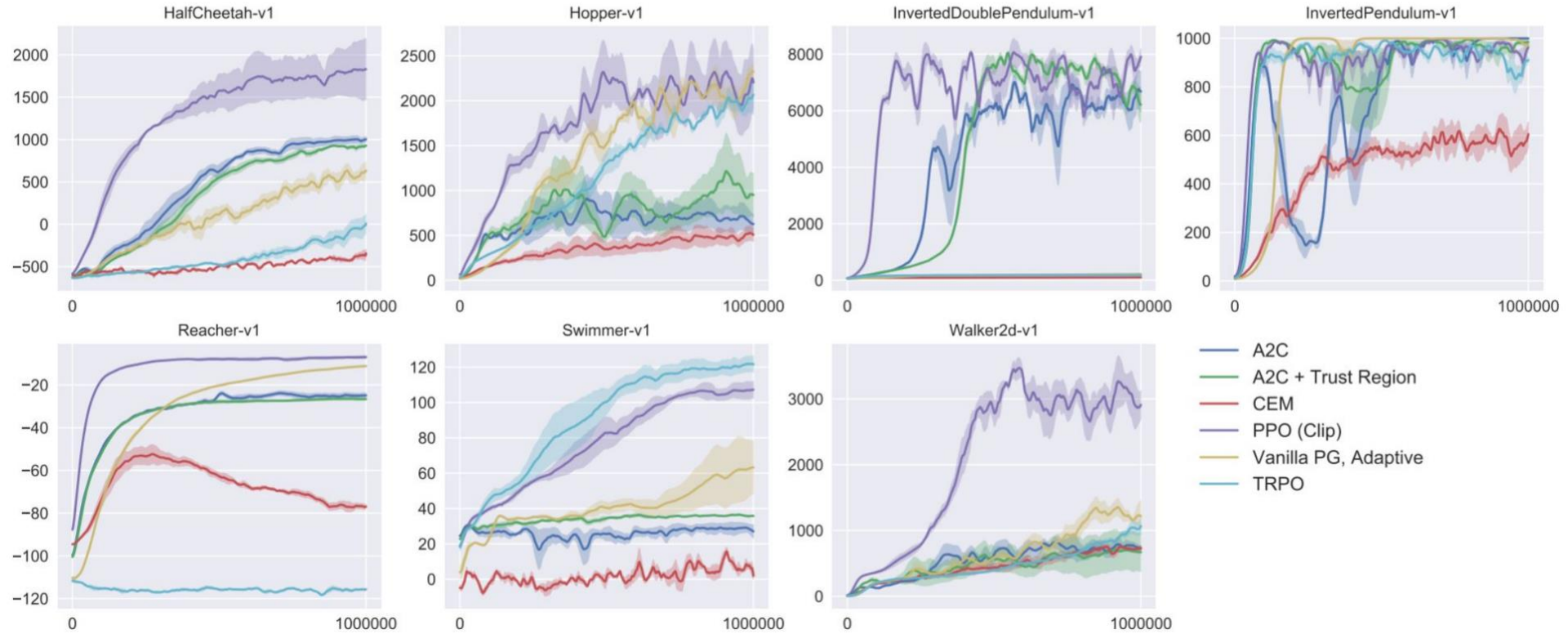
- Adaptive KL Penalty Coefficient

- Using several epochs of minibatch SGD, optimize the KL-penalized objective

$$L^{KLPEN}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t - \beta \, \text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)] \right]$$

- Compute $d = \hat{\mathbb{E}}_t[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]]$
  - If $d < d_{\text{targ}}/1.5$, $\beta \leftarrow \beta/2$
  - If $d > d_{\text{targ}} \times 1.5$, $\beta \leftarrow \beta \times 2$

Experiments (PPO)



Takeaways

- Trust region optimization guarantees the monotonic policy improvement.

- PPO is a first-order approximation of TRPO that is simpler to implement and achieves better empirical performance (both locomotion and Atari games).

2018

SAC

Soft Actor-Critic:
Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor

Tuomas Haarnoja [1] **Aurick Zhou** [1] **Pieter Abbeel** [1] **Sergey Levine** [1]

[1]Berkeley Artificial Intelligence Research, University of California, Berkeley, USA.

Prior Work: Soft Q-Learning

- Soft Q-Learning (Haarnoja et al., 2017)
  - off-policy algorithms under MaxEnt RL objective
  - Learns Q* directly
  - sample policy from exp(Q*) is intractable for continuous actions
  - use approximate inference methods to sample
    - Stein variational gradient descent
  - not true actor-critic

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t))$$

$$Q_{\text{soft}}(\mathbf{s}_t, \mathbf{a}_t) \leftarrow r_t + \gamma \, \mathbb{E}_{\mathbf{s}_{t+1} \sim p_{\mathbf{s}}} \left[V_{\text{soft}}(\mathbf{s}_{t+1}) \right], \ \forall \mathbf{s}_t, \mathbf{a}_t$$

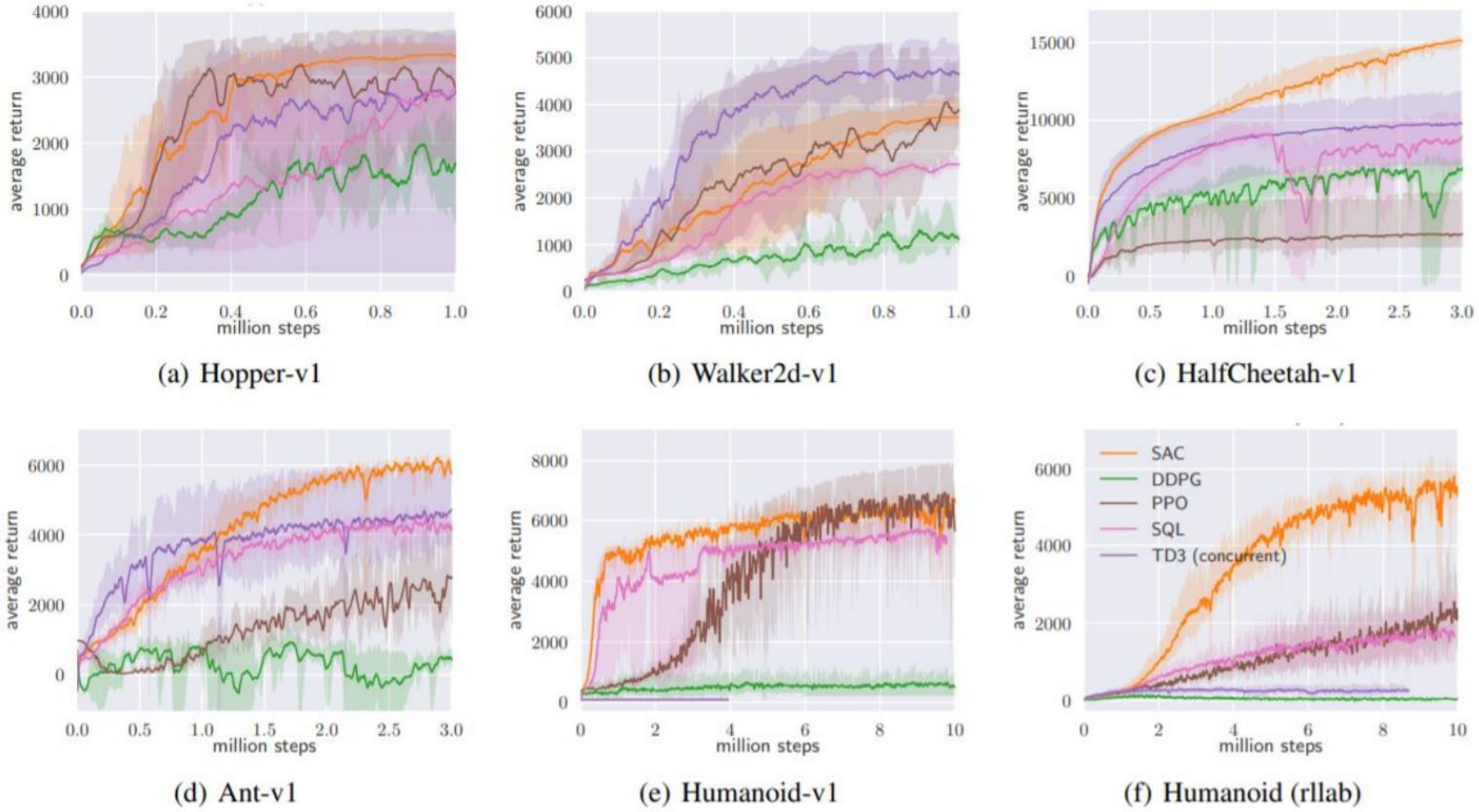
$$V_{\text{soft}}(\mathbf{s}_t) \leftarrow \alpha \log \int_{\mathcal{A}} \exp\left(\frac{1}{\alpha} Q_{\text{soft}}(\mathbf{s}_t, \mathbf{a}') \right) d\mathbf{a}', \ \forall \mathbf{s}_t$$

SAC: Contributions

- One of the most efficient model-free algorithms
  - SOTA off-policy
  - well suited for real world robotics learning
- Can learn stochastic policy on continuous action domain
- Robust to noise

- Ingredients:
  - Actor-critic architecture with seperate policy and value function networks
  - Off-policy formulation to reuse of previously collected data for efficiency
  - Entropy-constrained objective to encourage stability and exploration

SAC: Results

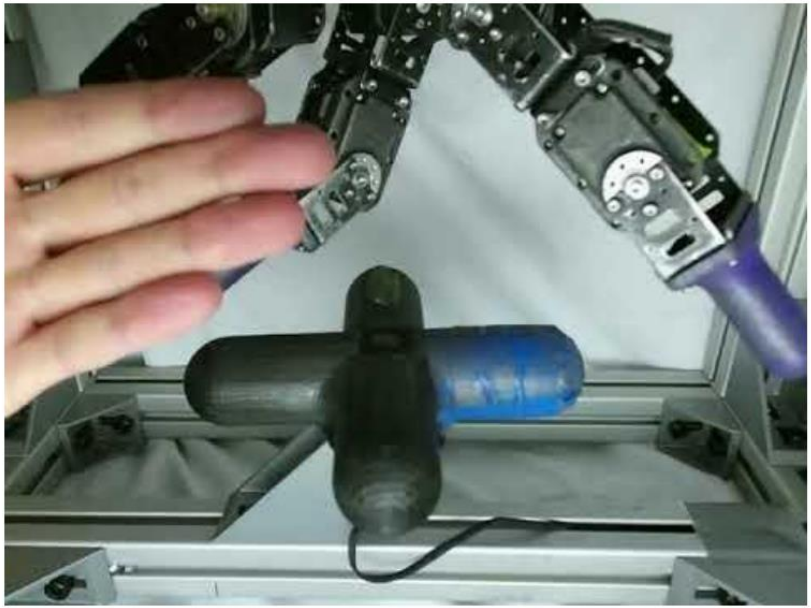


(a) Hopper-v1 (b) Walker2d-v1 (c) HalfCheetah-v1

(d) Ant-v1 (e) Humanoid-v1 (f) Humanoid (rllab)

Real World Robots

- Dexterous Hand Manipulations
- 20 hour end-to-end learning
- valve position as input: SAC 3 hours vs. PPO 7.4 hours

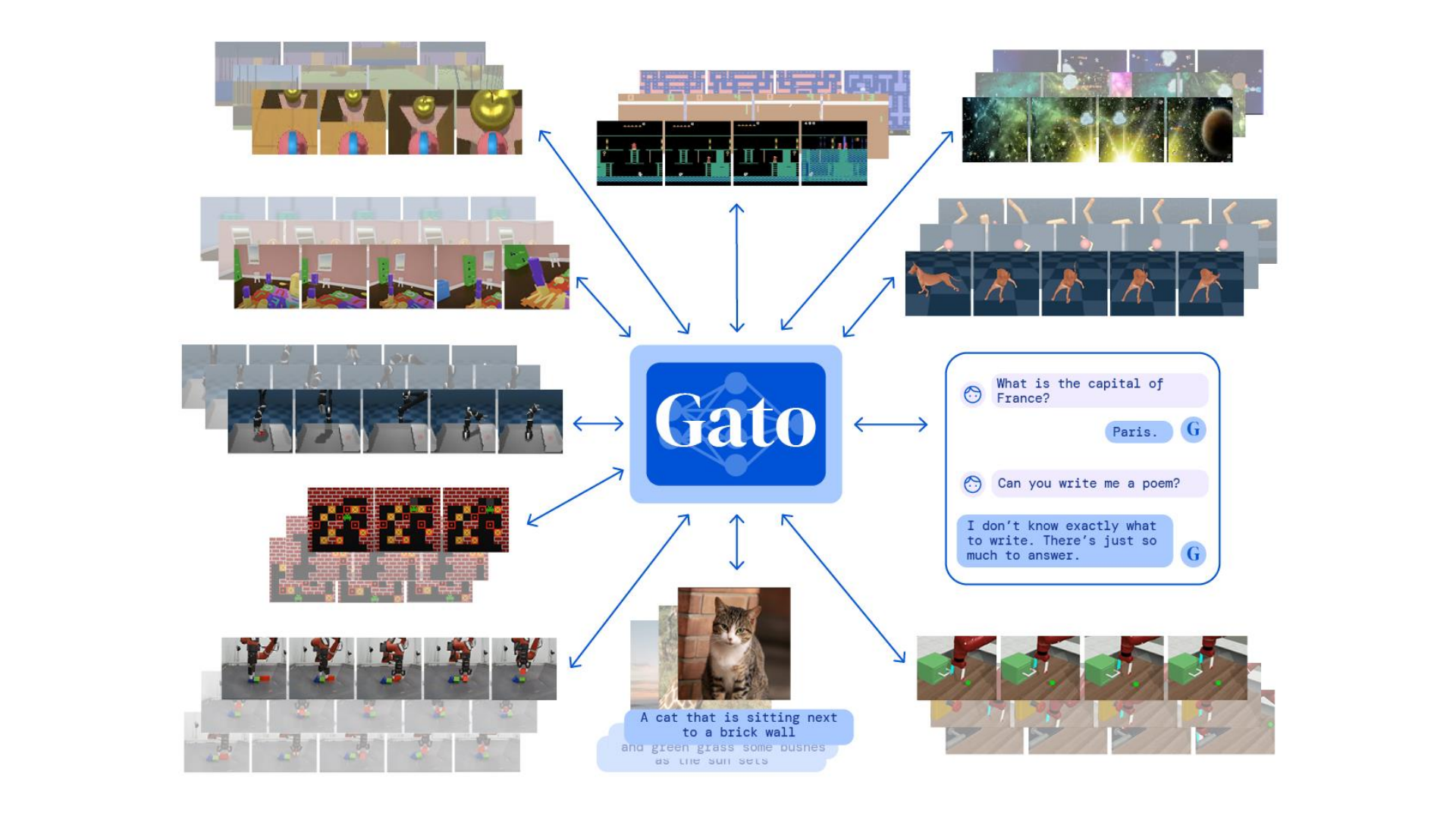


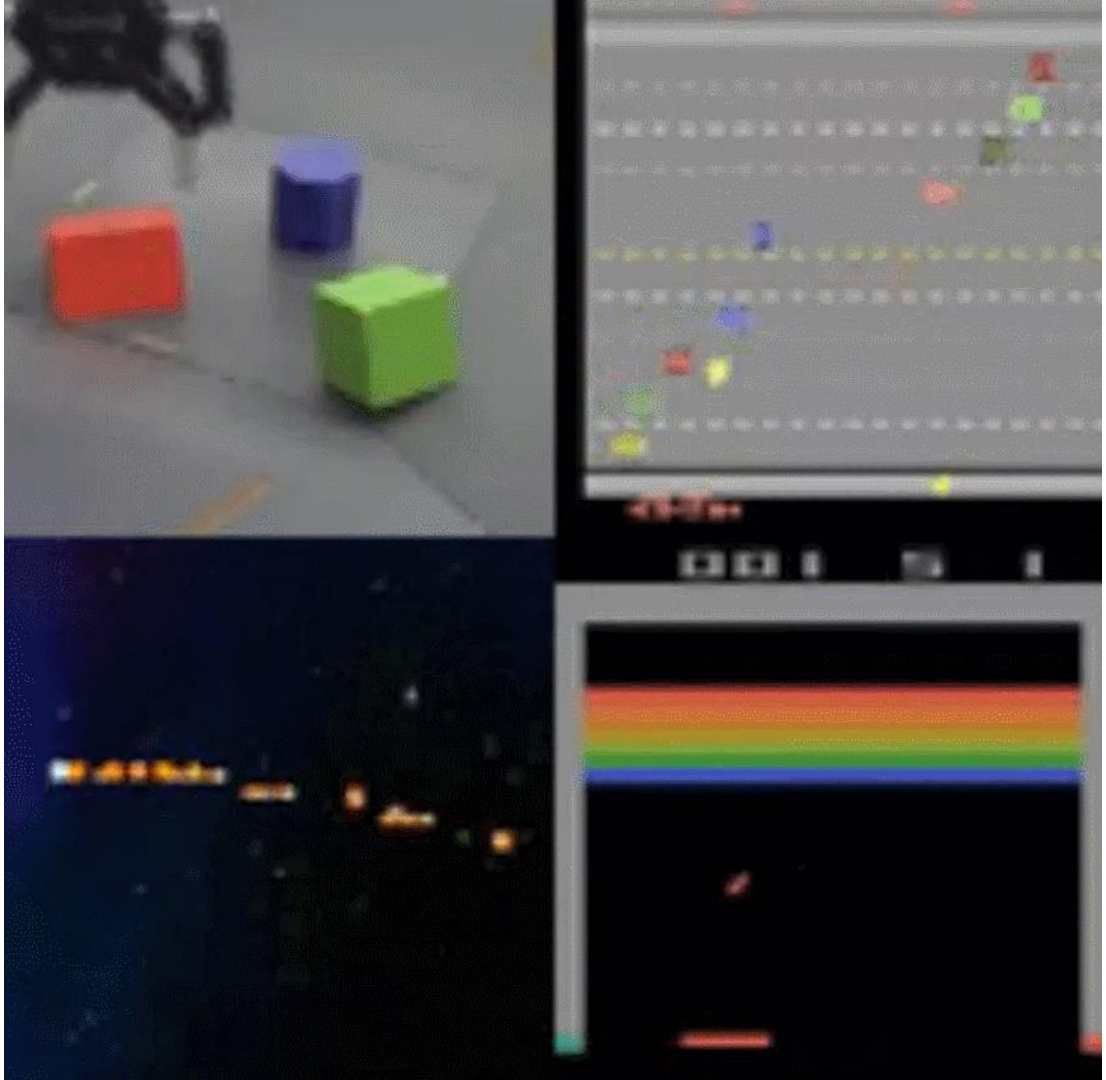
DeepMind

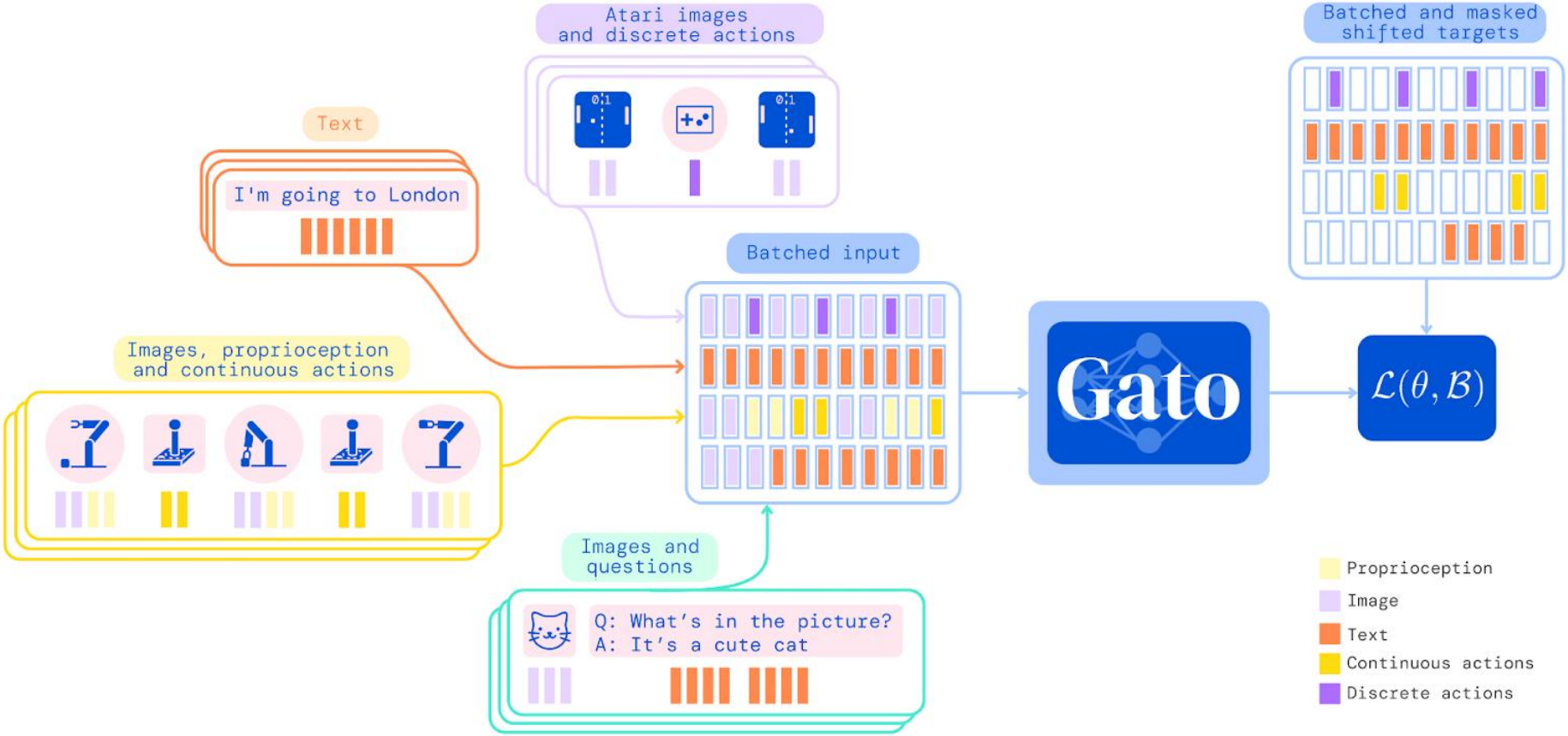
2022-5-19

A Generalist Agent

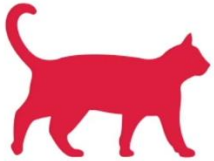
Scott Reed[*,†], Konrad Żołna[*], Emilio Parisotto[*], Sergio Gómez Colmenarejo[†], Alexander Novikov, Gabriel Barth-Maron, Mai Giménez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, Tom Eccles, Jake Bruce, Ali Razavi, Ashley Edwards, Nicolas Heess, Yutian Chen, Raia Hadsell, Oriol Vinyals, Mahyar Bordbar and Nando de Freitas[†]


Gato
What is the capital of France?
Paris.
Can you write me a poem?
I don't know exactly what to write. There's just so much to answer.
A cat that is sitting next to a brick wall

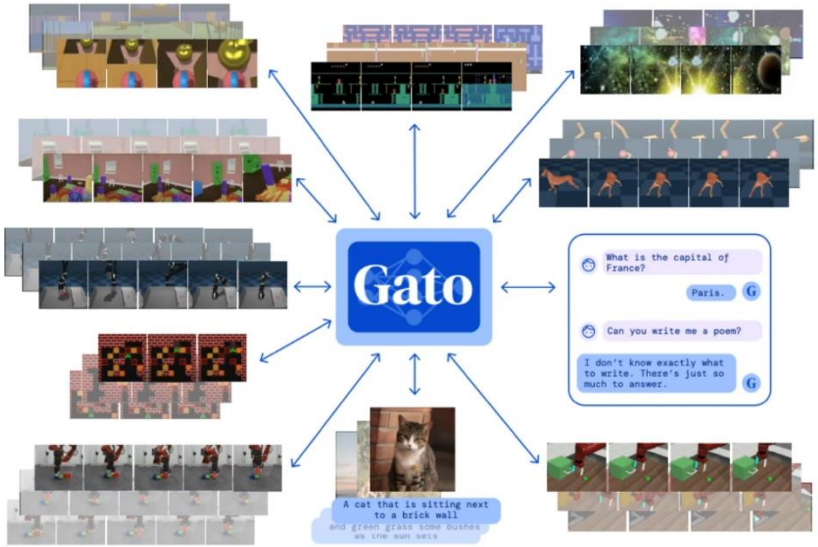



Atari images
and discrete actions
Batched and masked
shifted targets
Text
I'm going to London
Batched input
Images, proprioception
and continuous actions
Gato
$\mathcal{L}(\theta, \mathcal{B})$
Images and
questions
Q: What's in the picture?
A: It's a cute cat
Proprioception
Image
Text
Continuous actions
Discrete actions

What is Gato?



What is the paper setting up to demonstrate?



- Gato is a scalable generalist agent.
- It uses a single transformer/neural network with exactly the same weights for multiple tasks.
- It was trained on 604 various tasks.
- The number of parameters of Gato is 1.18B.

Benefits of using a single neural network

- Diversity of training data.
- Decreasing the necessity for handcrafting policy models.
- Generic models have proven to outperform than domain-specific approaches.

Methodology

- Step 1: Tokenization of Multimodal Data
- Step 2: Sequencing Data
- Step 3: Embedding input tokens and setting output targets
- Step4: Training

Step 1: Tokenization of Multimodal Data

- SentencePiece is used to encode text.
- Images are first processed into raster order sequence.
- Discrete values are flattened into row-major order integer sequences. The tokenized result is an integer sequence in the range [0, 1024].
- Continuous values are flattened into sequences of floating-point values in row-major order. The discrete integers are then shifted to the range of [32000, 33024).

Step 2: Sequencing Data

- Text tokens in the same order as the raw input text.
- Images patch tokens in raster order.
- Tensors in row-major order.
- Nested structures in lexicographical order by key.
- Agent episodes as time steps in time order.
- Agent timesteps as observation tokens.

Step 3: Embedding input tokens and setting output targets

- Tokens belonging to image patches → vector per patch.
- Tokens belonging to text, discrete or continuous-valued observations, or actions → a learned vector embedding space.

Step 4: Training objectives

- Two key modules:
- **Parameterized embedding function:** transforms tokens to token embeddings.
- **Sequence model:** outputs a distribution over the next discrete token.

Given a sequence of tokens $S_{1:L}$ and parameters θ, they model the data using the chain rule of probability:

$$\log p_\theta(s_1, \ldots, s_L) = \sum_{l=1}^{L} \log p_\theta(s_l | s_1, \ldots, s_{l-1})$$

The training loss for a batch B can then be written as,

$$\mathcal{L}(\theta, \mathcal{B}) = -\sum_{b=1}^{|\mathcal{B}|} \sum_{l=1}^{L} m\,(b,t) \log p_\theta \left(s_l^{(b)} | s_1^{(b)}, \ldots, s_{l-1}^{(b)} \right)$$

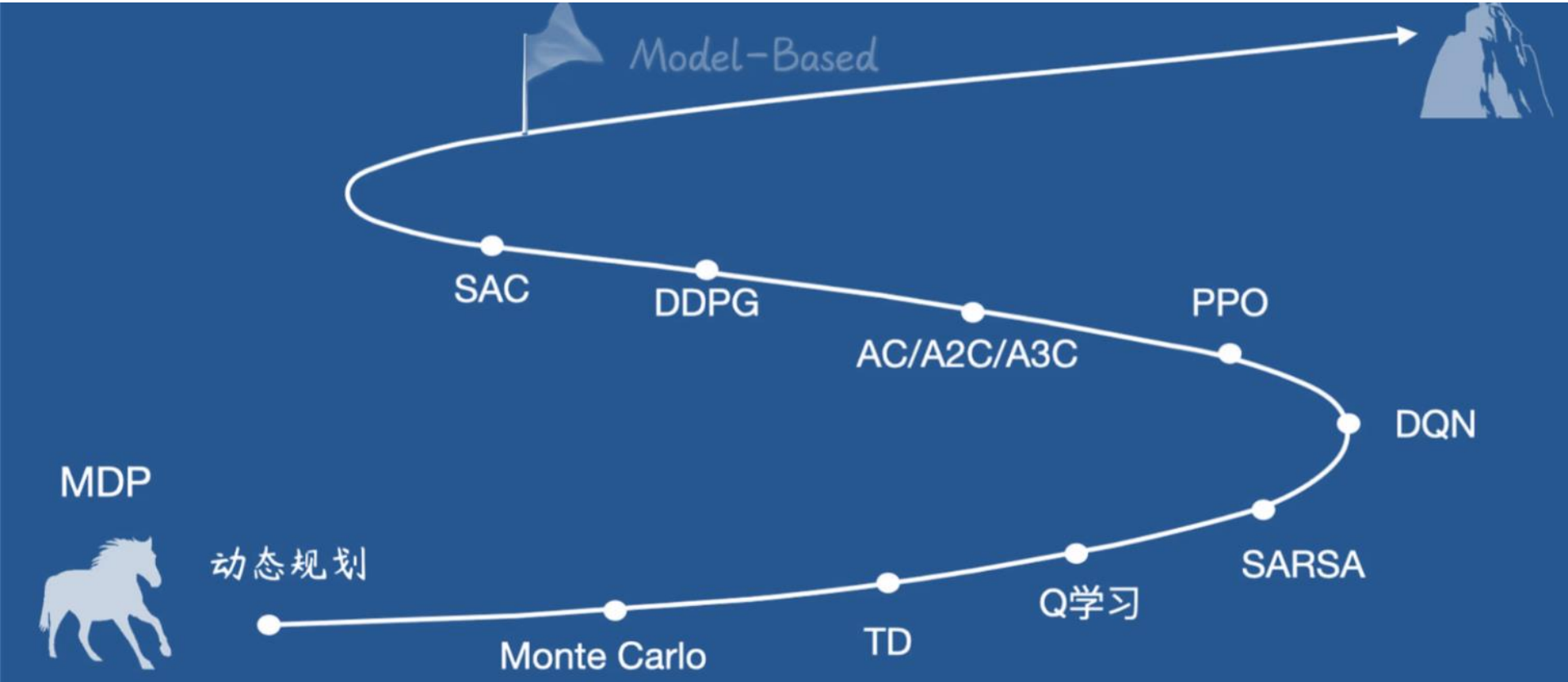
提纲

一、萌芽：从DP到Q Learning

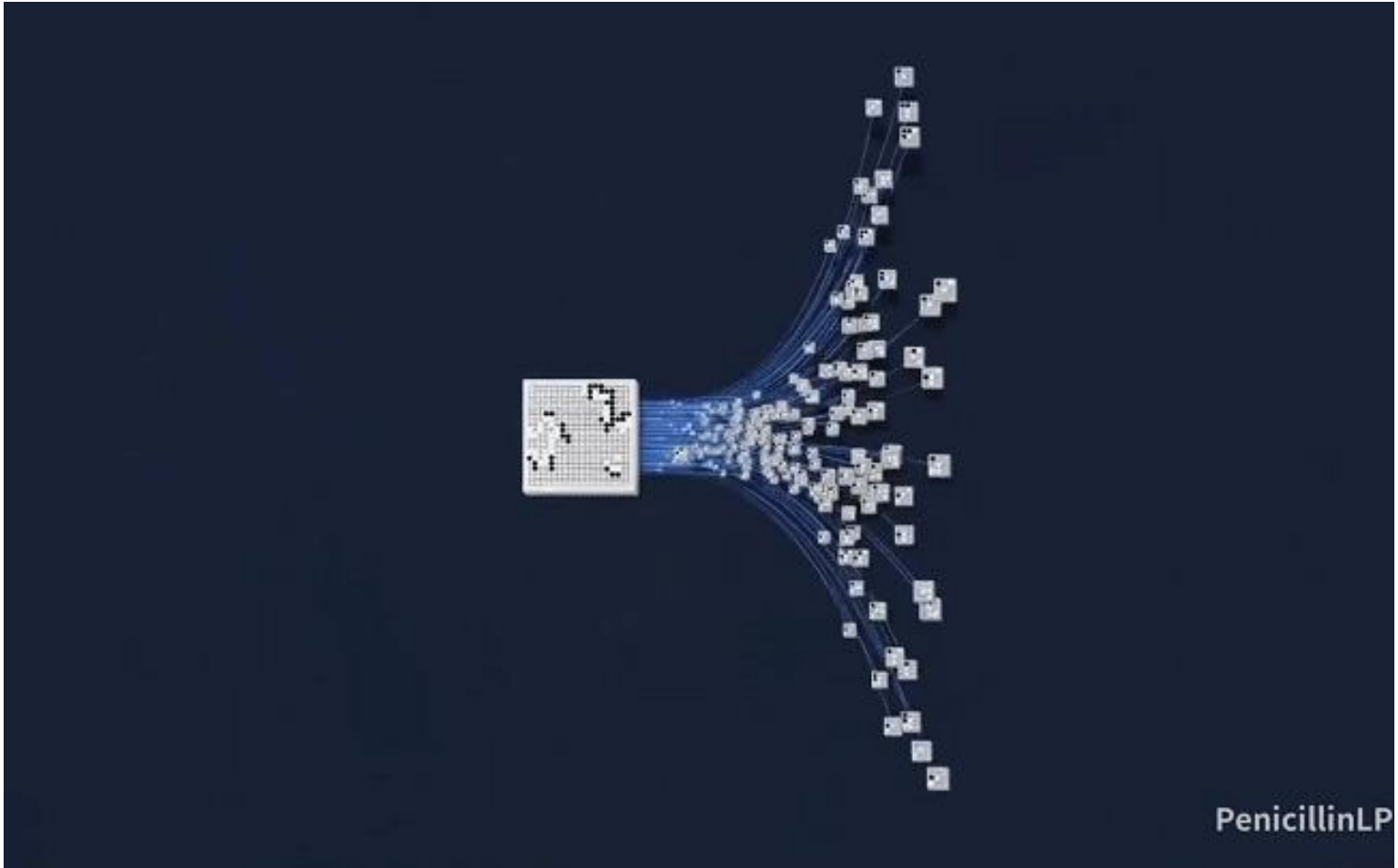
二、大成：从DQN到GATO

三、巅峰：从AlphaGo到MuZero

强化学习


Model-Based
SAC
DDPG
AC/A2C/A3C
PPO
DQN
MDP
动态规划
Monte Carlo
TD
Q学习
SARSA

蒙特卡洛树搜索(Monte Carlo Tree Search)



Deepmind

2016 Nature AlphaGo

ARTICLE

doi:10.1038/nature16961

Mastering the game of Go with deep neural networks and tree search

David Silver[1]*, Aja Huang[1]*, Chris J. Maddison[1], Arthur Guez[1], Laurent Sifre[1], George van den Driessche[1], Julian Schrittwieser[1], Ioannis Antonoglou[1], Veda Panneershelvam[1], Marc Lanctot[1], Sander Dieleman[1], Dominik Grewe[1], John Nham[2], Nal Kalchbrenner[1], Ilya Sutskever[2], Timothy Lillicrap[1], Madeleine Leach[1], Koray Kavukcuoglu[1], Thore Graepel[1] & Demis Hassabis[1]

2017 Nature AlphaGo Zero

ARTICLE

doi:10.1038/nature24270

Mastering the game of Go without human knowledge

David Silver[1]*, Julian Schrittwieser[1]*, Karen Simonyan[1]*, Ioannis Antonoglou[1], Aja Huang[1], Arthur Guez[1], Thomas Hubert[1], Lucas Baker[1], Matthew Lai[1], Adrian Bolton[1], Yutian Chen[1], Timothy Lillicrap[1], Fan Hui[1], Laurent Sifre[1], George van den Driessche[1], Thore Graepel[1] & Demis Hassabis[1]

2018 Science AlphaZero

RESEARCH

COMPUTER SCIENCE

A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play

David Silver[1,2]*†, Thomas Hubert[1]*, Julian Schrittwieser[1]*, Ioannis Antonoglou[1], Matthew Lai[1], Arthur Guez[1], Marc Lanctot[1], Laurent Sifre[1], Dharshan Kumaran[1], Thore Graepel[1], Timothy Lillicrap[1], Karen Simonyan[1], Demis Hassabis[1]†

2020 Nature MuZero

Article

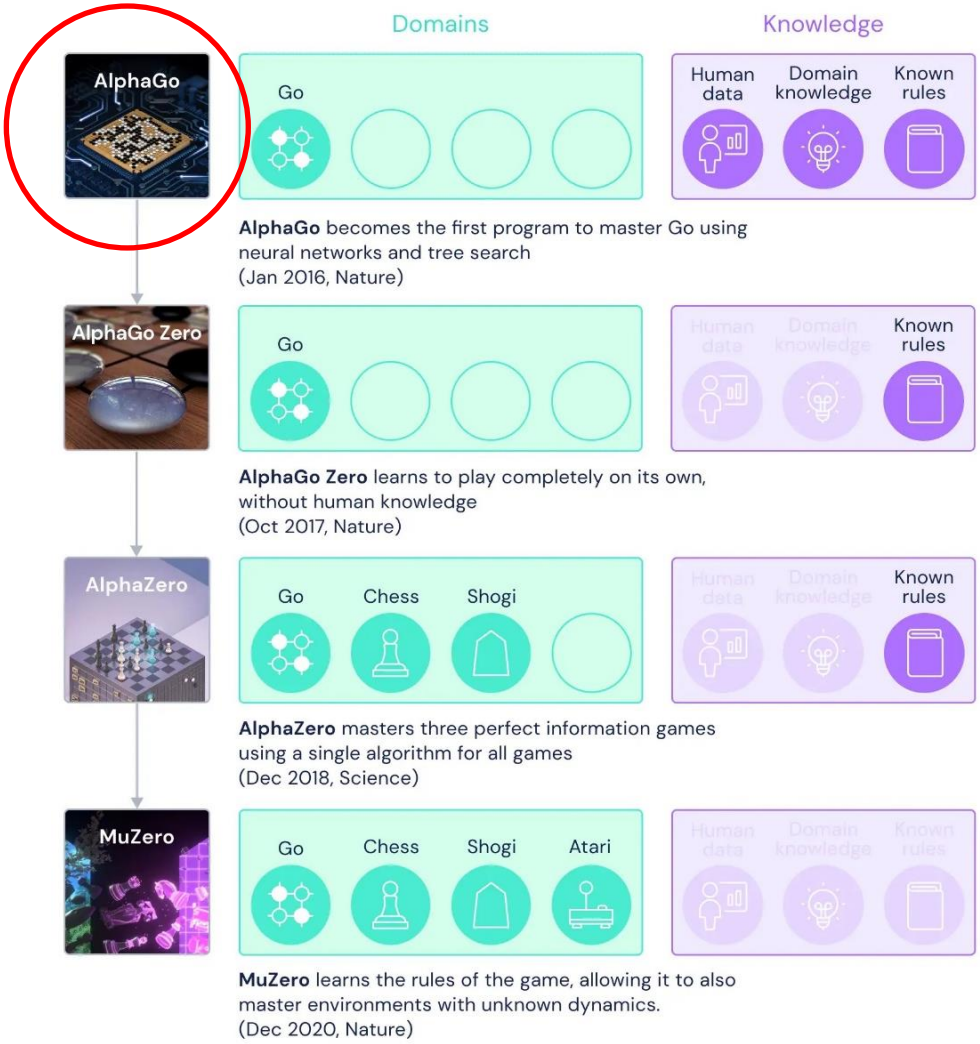
Mastering Atari, Go, chess and shogi by planning with a learned model

https://doi.org/10.1038/s41586-020-03051-4

Received: 3 April 2020

Accepted: 7 October 2020

Julian Schrittwieser[1,3], Ioannis Antonoglou[1,2,3], Thomas Hubert[1,3], Karen Simonyan[1], Laurent Sifre[1], Simon Schmitt[1], Arthur Guez[1], Edward Lockhart[1], Demis Hassabis[1], Thore Graepel[1,2], Timothy Lillicrap[1] & David Silver[1,2,3]✉

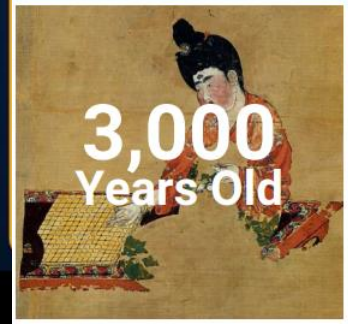

Domains
Knowledge
AlphaGo
Go
Human data
Domain knowledge
Known rules
AlphaGo becomes the first program to master Go using neural networks and tree search
(Jan 2016, Nature)
AlphaGo Zero
Go
Known rules
AlphaGo Zero learns to play completely on its own, without human knowledge
(Oct 2017, Nature)
AlphaZero
Go
Chess
Shogi
Known rules
AlphaZero masters three perfect information games using a single algorithm for all games
(Dec 2018, Science)
MuZero
Go
Chess
Shogi
Atari
MuZero learns the rules of the game, allowing it to also master environments with unknown dynamics.
(Dec 2020, Nature)

Go 围棋

碁 go ご

Game Start

Game End




围地多的一方获得胜利
PenicillinLP
bilibili

Complexity of Go

State-space Complexity

$3^{361} \approx 10^{172}$

Tic-Tac-Toe: 10^{4} Chess: 10^{43}

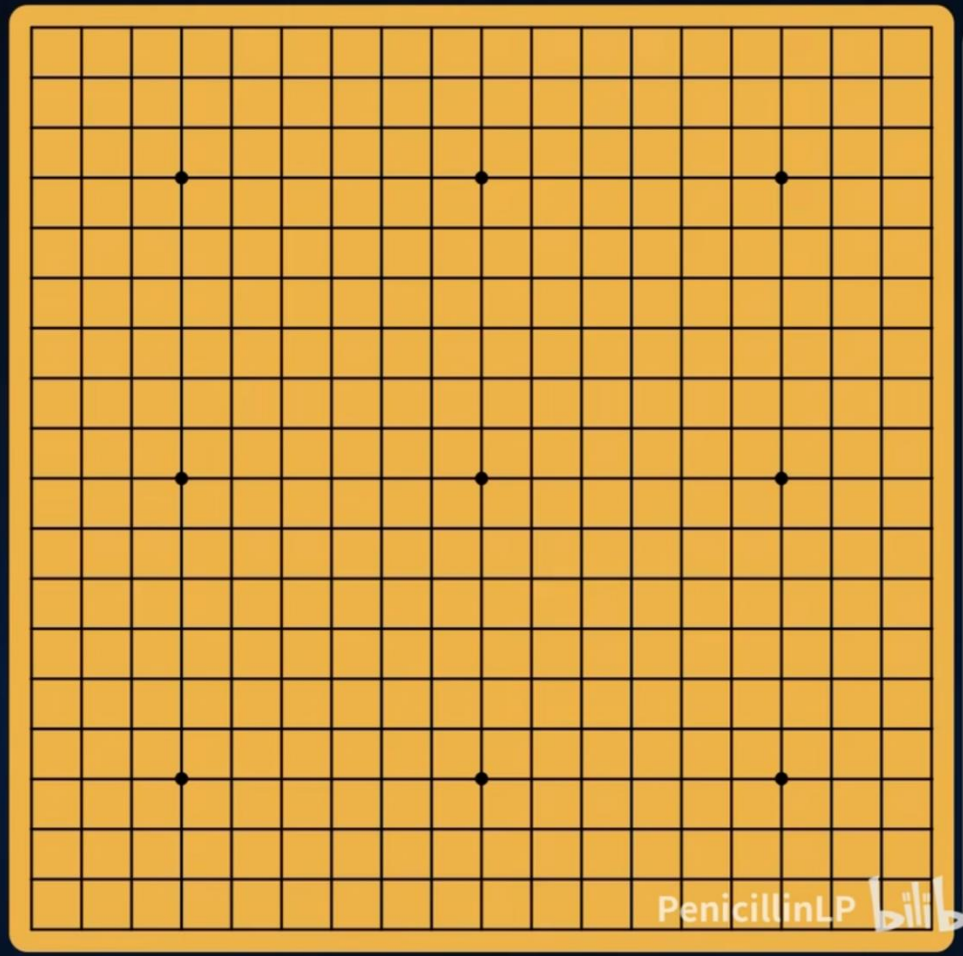
Game-tree Size

$361! \approx 10^{768}$

Game-tree Complexity

$250^{150} \approx 10^{360}$

Tic-Tac-Toe: 10^{5} Chess: 10^{123}



Complexity of Go

State-space Complexity

$3^{361} \approx 10^{172}$

Tic-Tac-Toe: 10^{4} Chess: 10^{43}

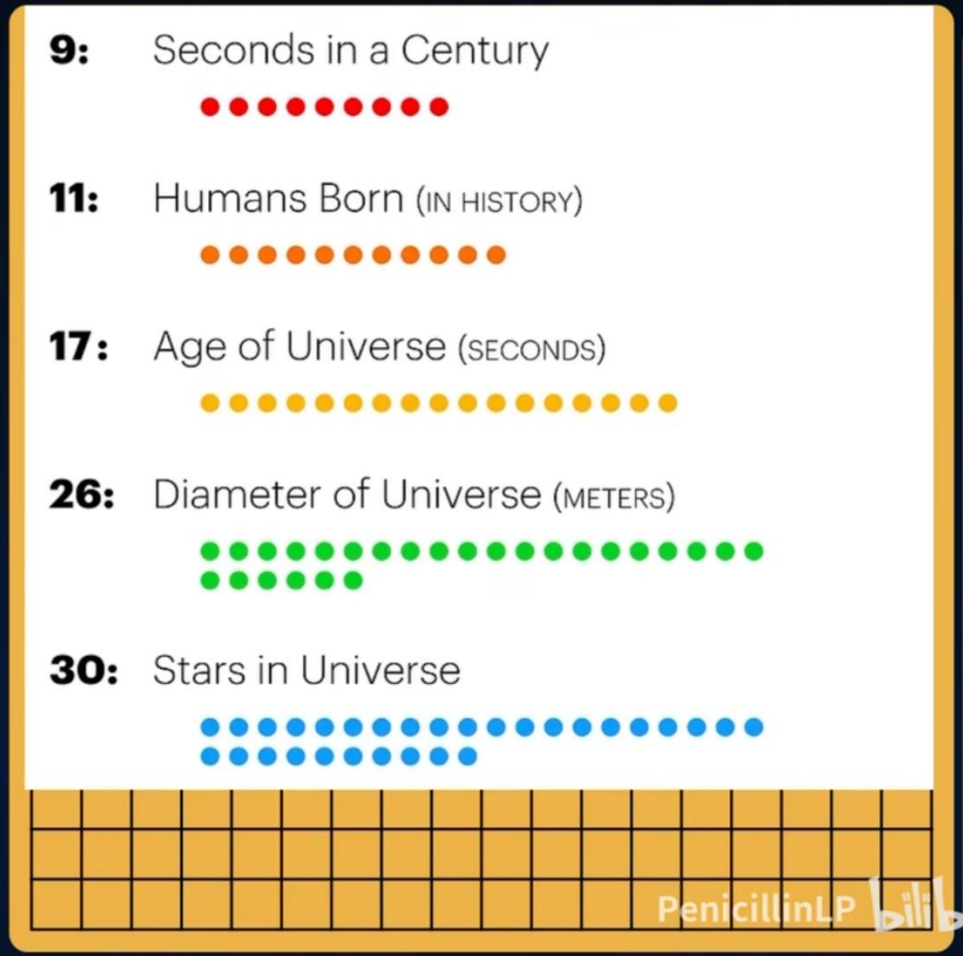
Game-tree Size

$361! \approx 10^{768}$

Game-tree Complexity

$250^{150} \approx 10^{360}$

Tic-Tac-Toe: 10^{5} Chess: 10^{123}



Complexity of Go

State-space Complexity

$3^{361} \approx 10^{172}$

Tic-Tac-Toe: 10^{4} Chess: 10^{43}

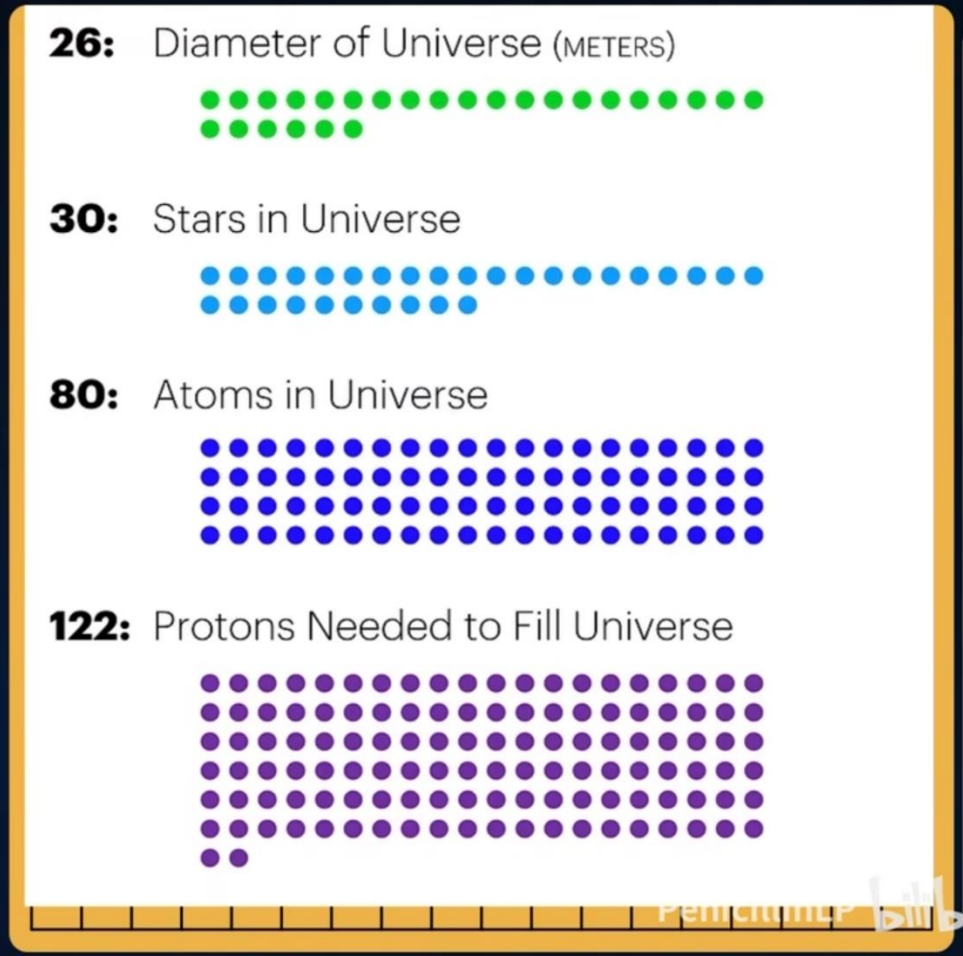
Game-tree Size

$361! \approx 10^{768}$

Game-tree Complexity

$250^{150} \approx 10^{360}$

Tic-Tac-Toe: 10^{5} Chess: 10^{123}



Complexity of Go

State-space Complexity

$3^{361} \approx 10^{172}$

Tic-Tac-Toe: 10^{4} Chess: 10^{43}

Game-tree Size

$361! \approx 10^{768}$

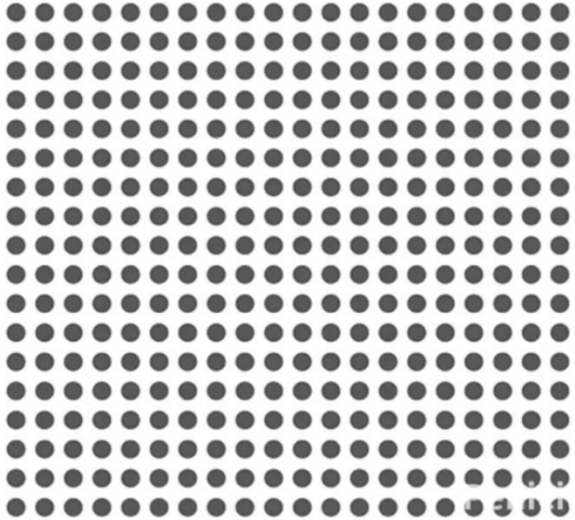
Game-tree Complexity

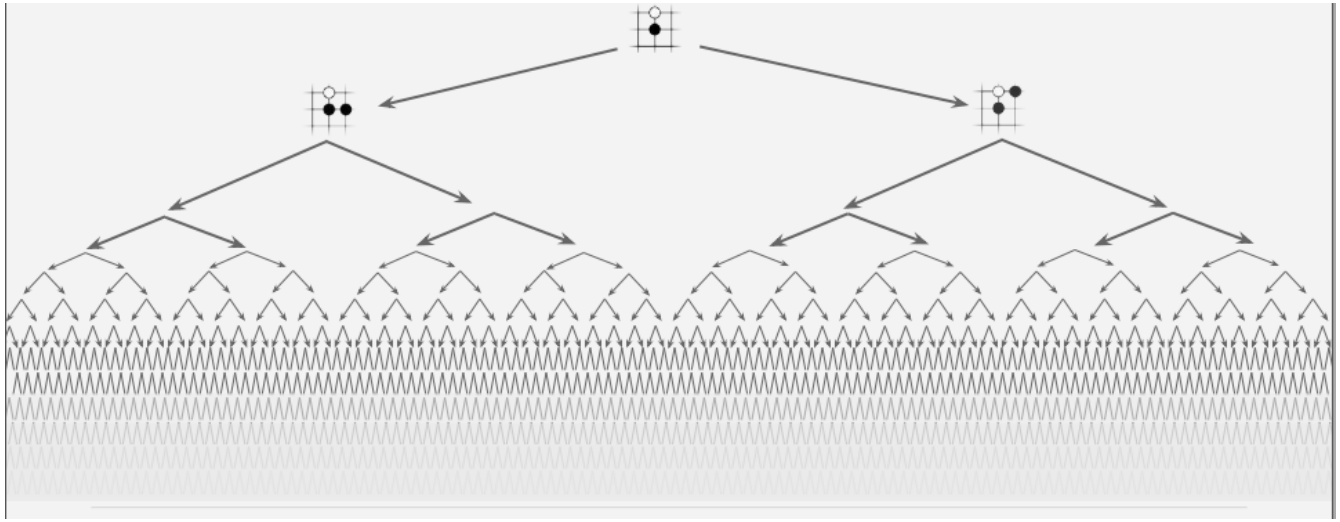
$250^{150} \approx 10^{360}$

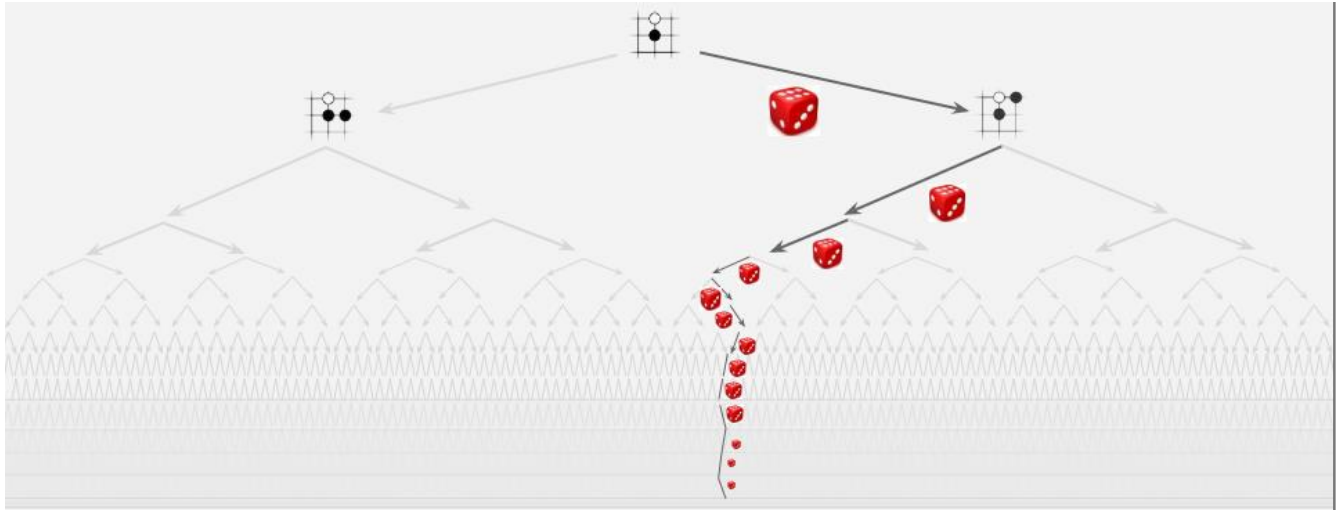
Tic-Tac-Toe: 10^{5} Chess: 10^{123}

122: Protons Needed to Fill Universe

360: Go Game-Tree Complexity

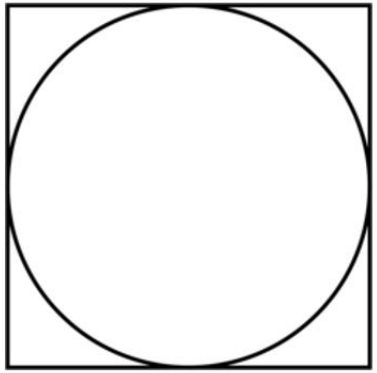


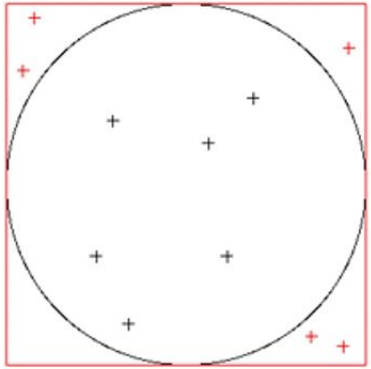


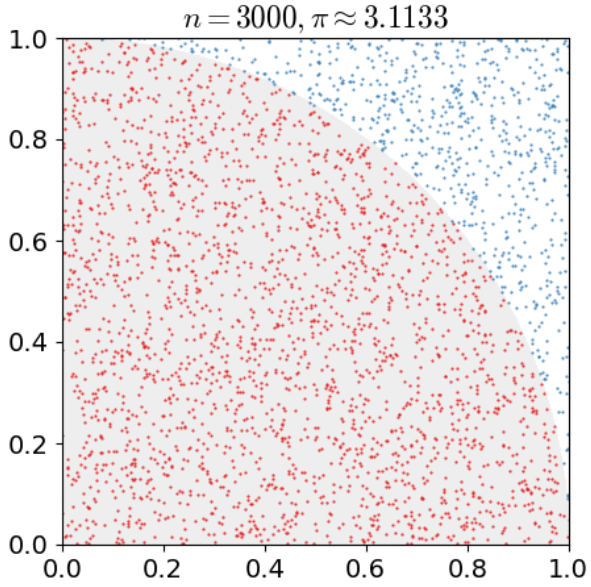


蒙特卡洛方法

- 蒙特卡洛方法（Monte-Carlo methods）是一类广泛的计算算法。生活中处处都是MC方法。
  - 依赖于重复随机抽样来获得数值结果
- 例如，计算圆的面积



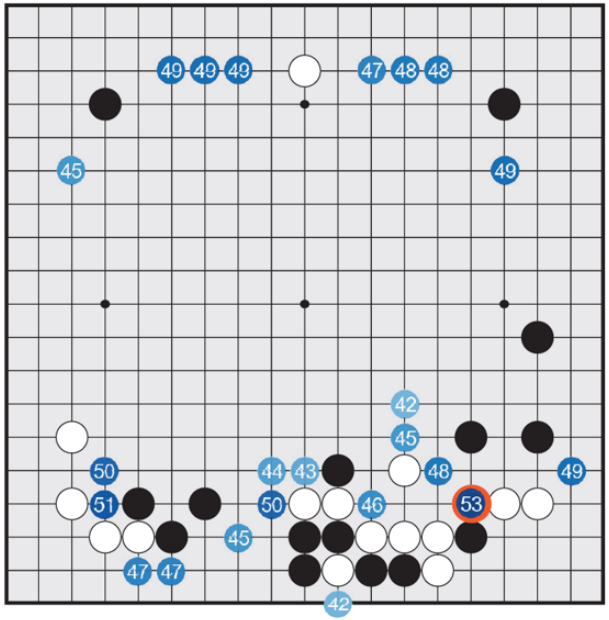


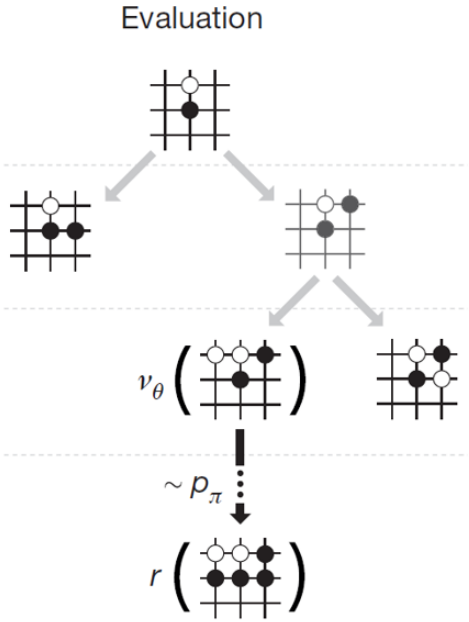


$$\text{Circle Surface} = \text{Square Surface} \times \frac{\#\text{points in circle}}{\#\text{points in total}}$$

蒙特卡洛方法

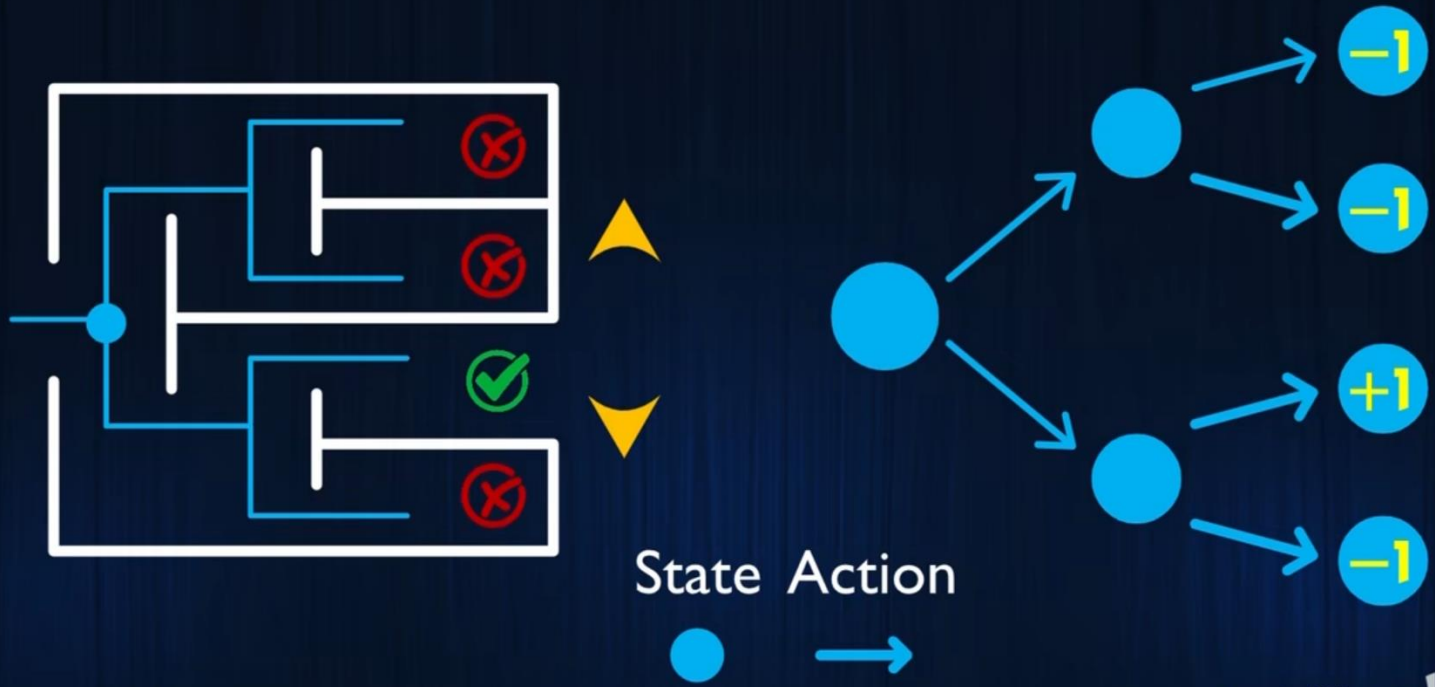
- 围棋对弈：估计当前状态下的胜率





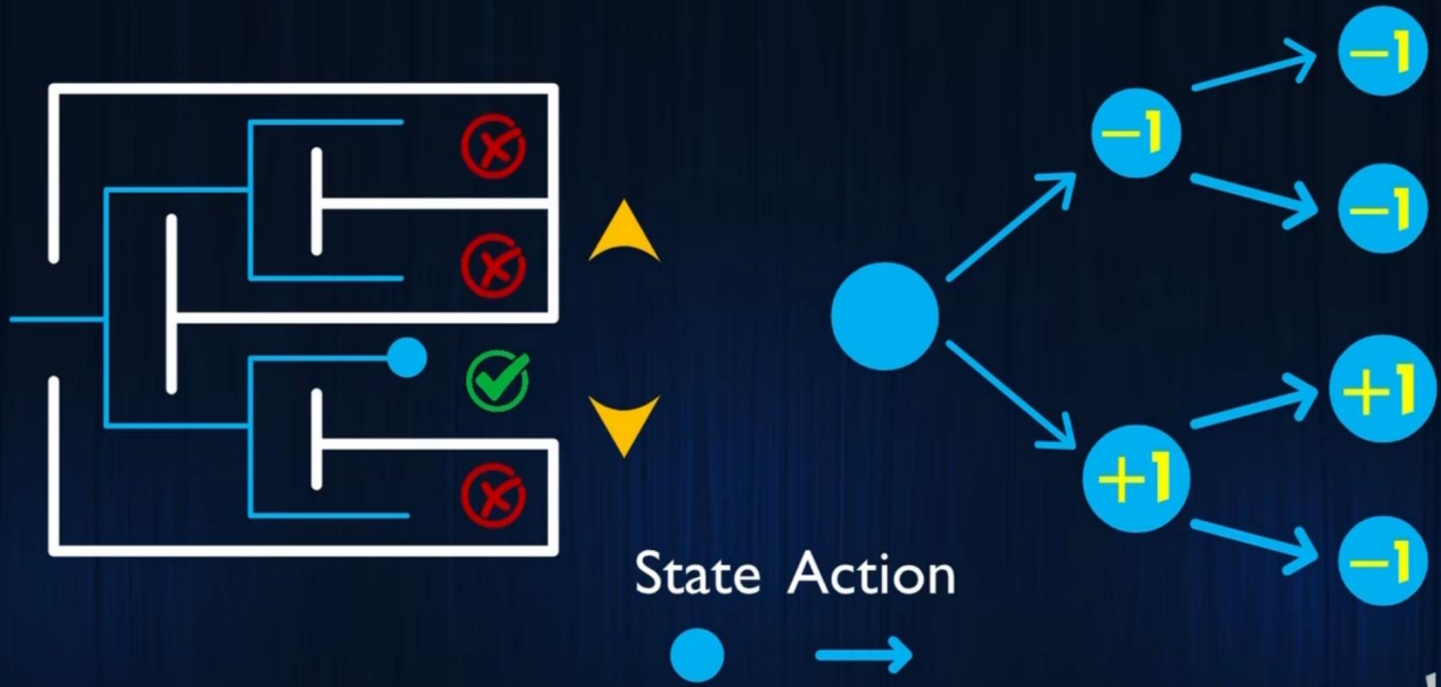
$$\text{Win Rate}(s) = \frac{\#\text{win simulation cases started from } s}{\#\text{simulation cases started from } s \text{ in total}}$$

Monte Carlo Tree Search A Naive Tree Search Algorithm



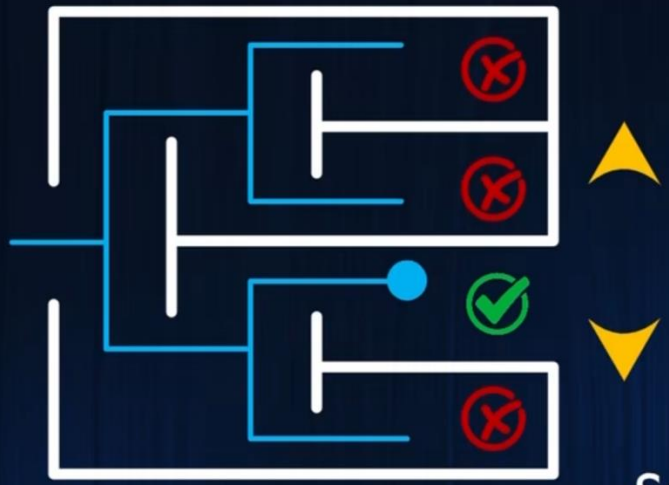
PenicillinLP bilibili

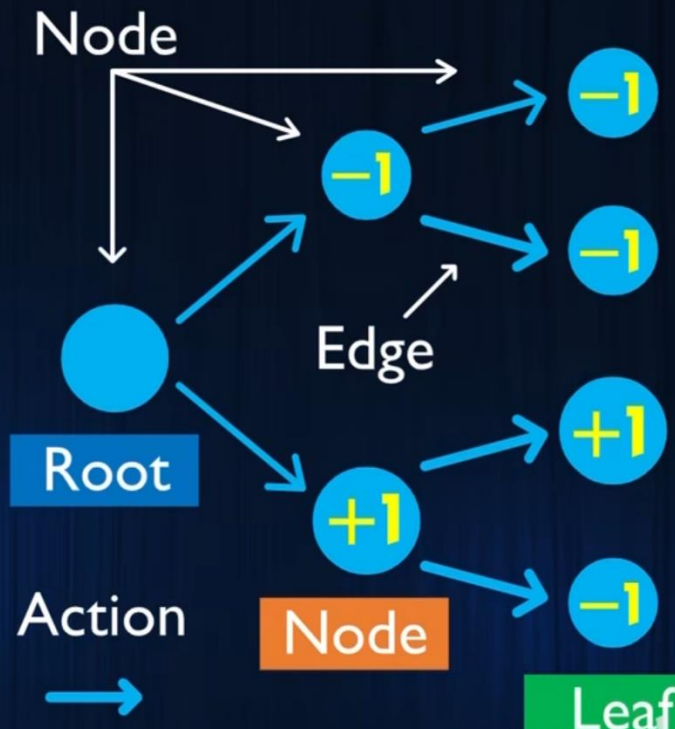
Monte Carlo Tree Search A Naive Tree Search Algorithm

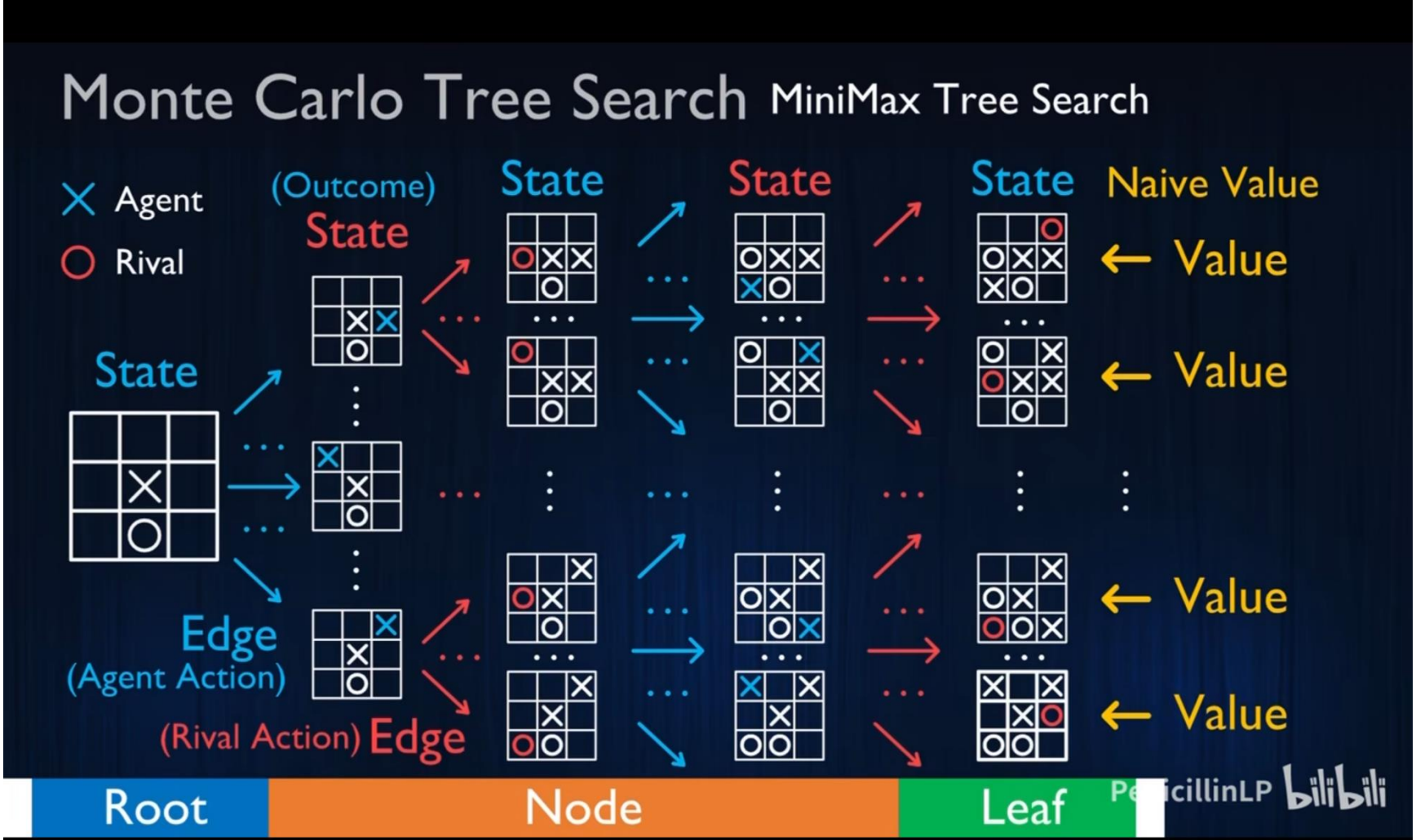


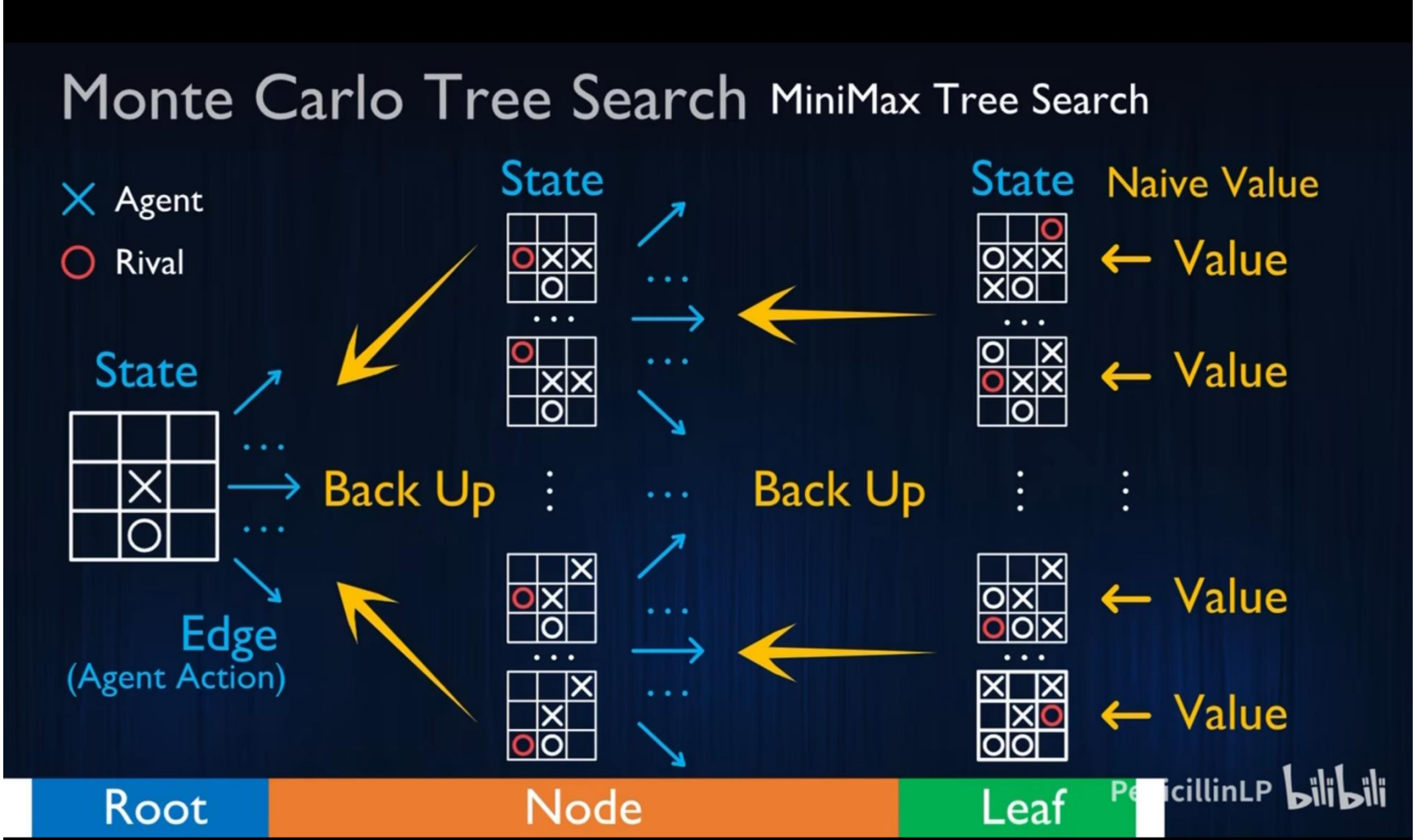
PenicillinLP bilibili

Monte Carlo Tree Search A Naive Tree Search Algorithm



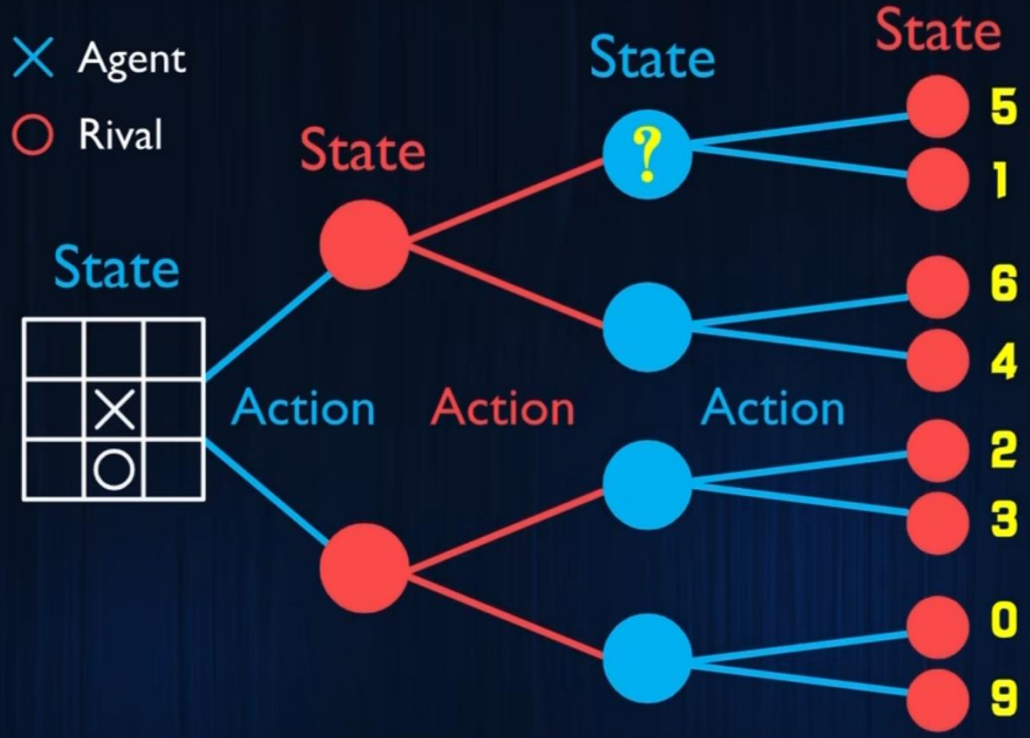



Monte Carlo Tree Search
MiniMax Tree Search
Agent
Rival
(Outcome)
State
State
State
State
Naive Value
State
Value
Value
Edge
(Agent Action)
Value
(Rival Action)
Edge
Value
Root
Node
Leaf
PenicillinLP
bilibili


Monte Carlo Tree Search
MiniMax Tree Search
Agent
Rival
State
State
Naive Value
Value
Value
State
Back Up
Back Up
Edge
(Agent Action)
Value
Value
Root
Node
Leaf
PenicillinLP
bilibili

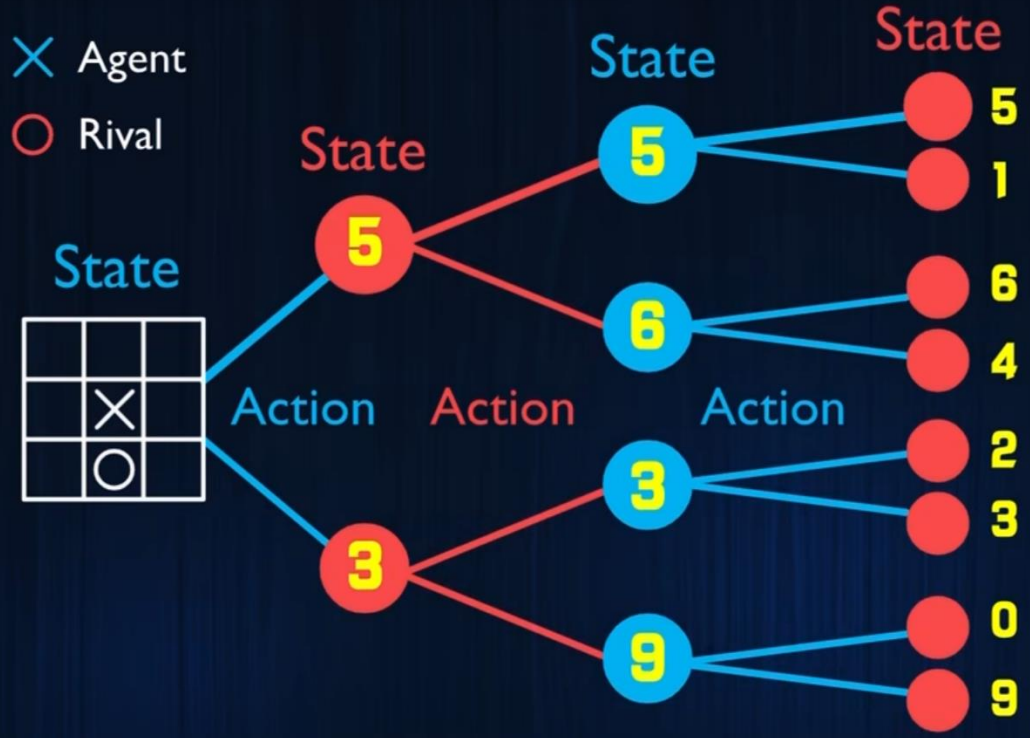
Monte Carlo Tree Search

MiniMax Tree Search — MiniMax



PenicillinLP bilibili

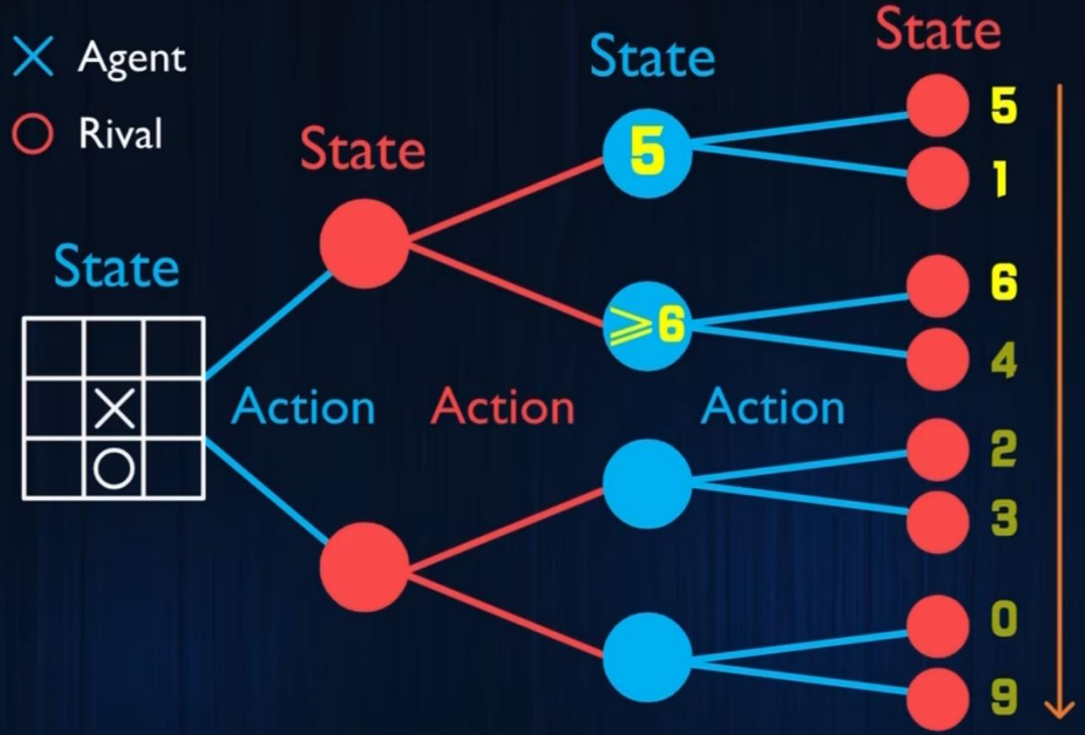
Monte Carlo Tree Search MiniMax Tree Search — MiniMax



PenicillinLP bilibili

Monte Carlo Tree Search

MiniMax Tree Search — MiniMax



MiniMax

+

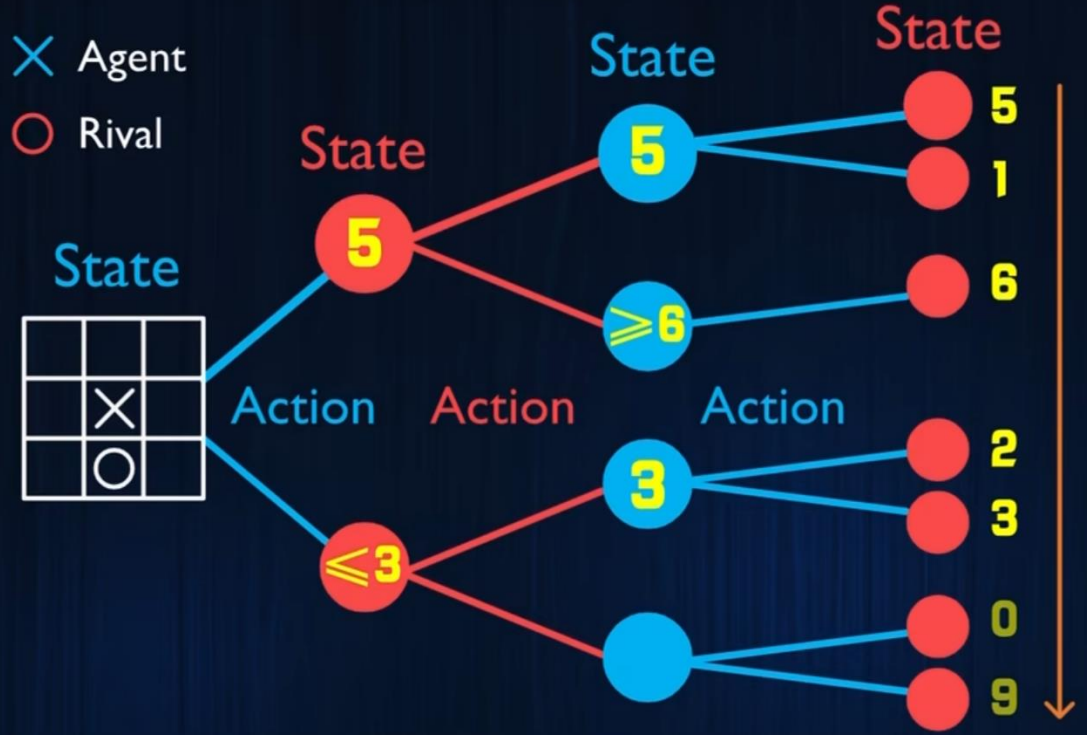
Alpha-Beta Pruning

=

Deep Blue, 1997

Monte Carlo Tree Search

MiniMax Tree Search — MiniMax



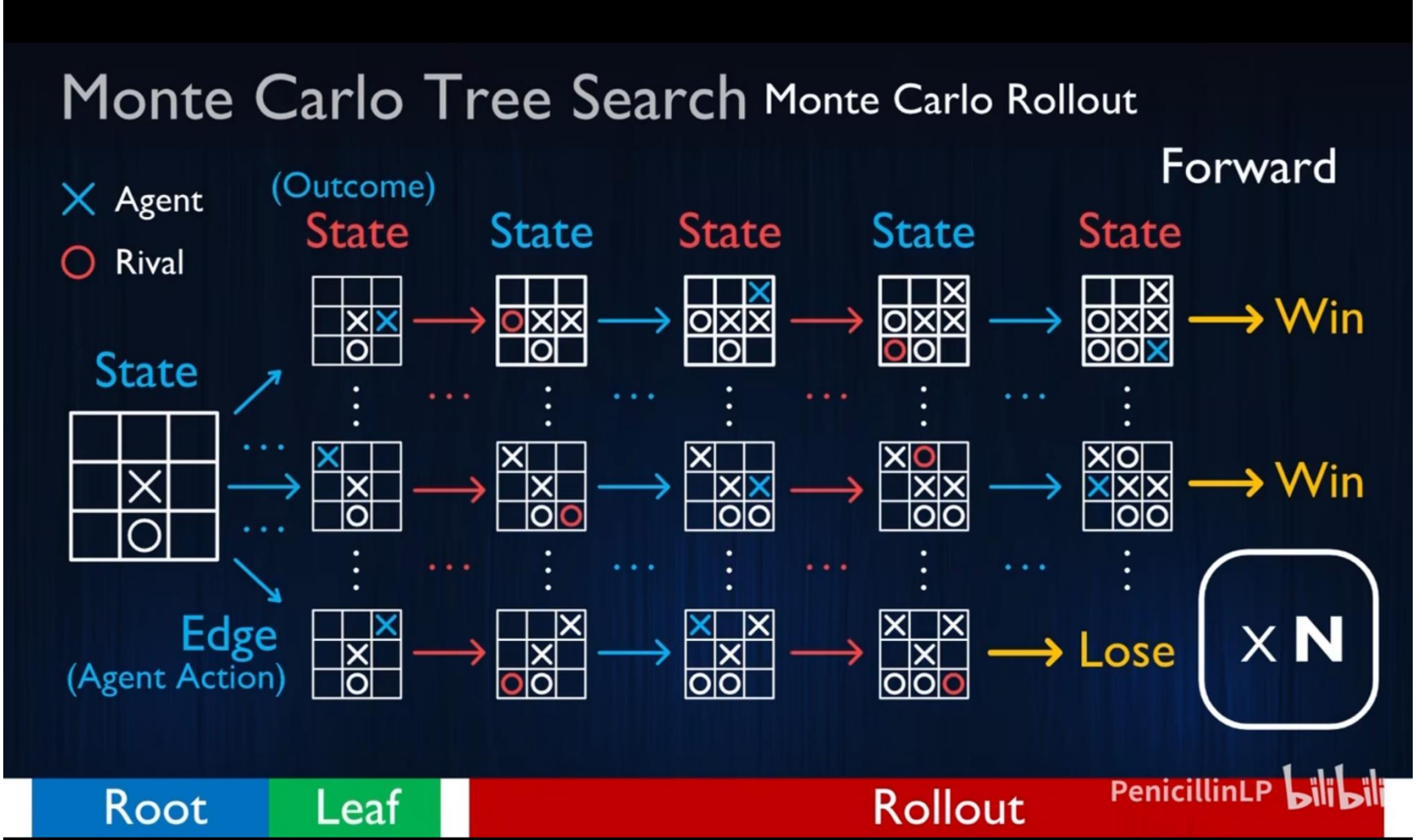
MiniMax

+

Alpha-Beta Pruning

=

Deep Blue, 1997


Monte Carlo Tree Search
Monte Carlo Rollout
Forward
Agent
Rival
(Outcome)
State
State
State
State
State
Win
State
Win
Edge
(Agent Action)
Lose
X N
Root
Leaf
Rollout
PenicillinLP
bilibili

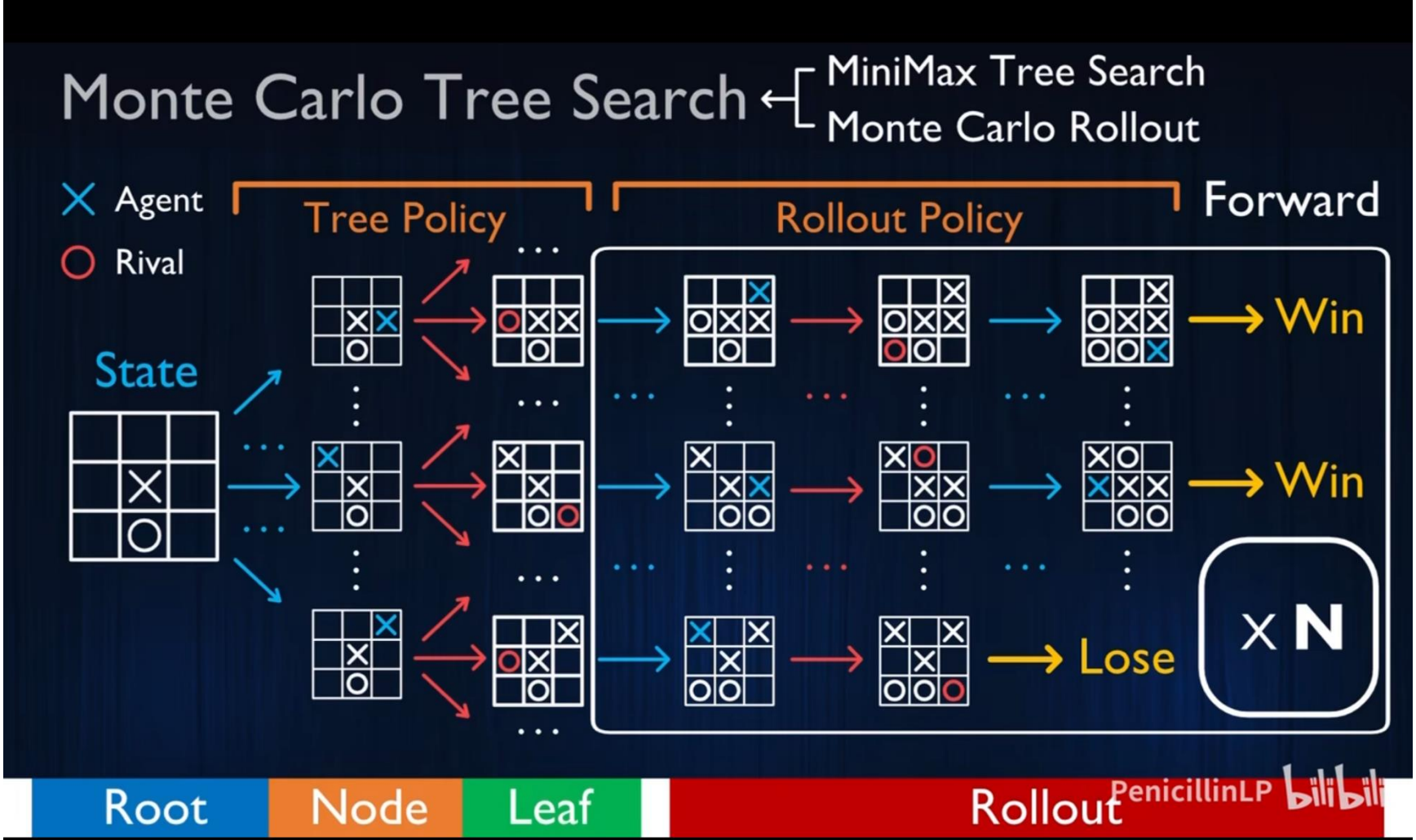
Monte Carlo Tree Search Monte Carlo Rollout Rollout Policy

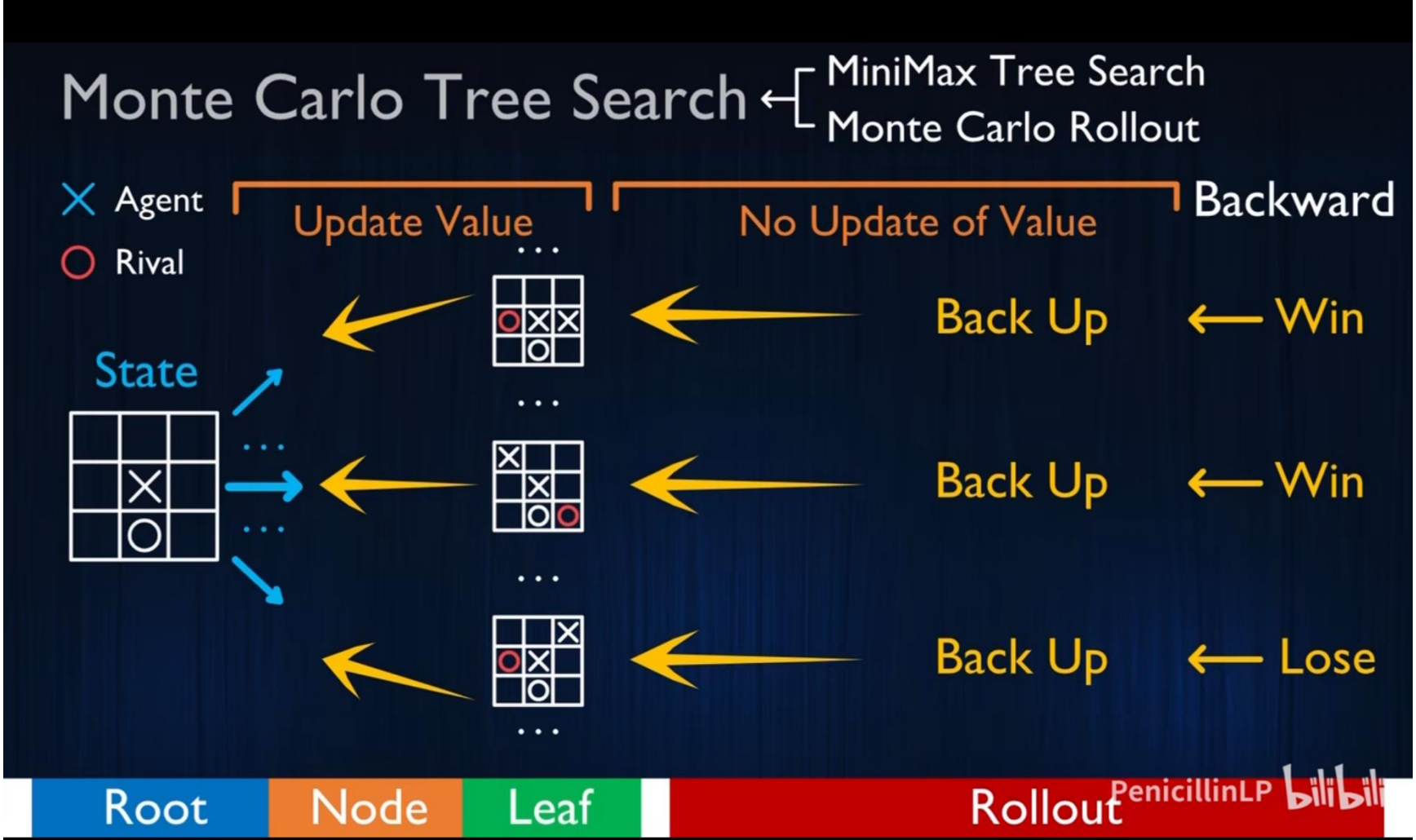
Backward

Monte Carlo Rollout

+

MiniMax Tree Search


Monte Carlo Tree Search
MiniMax Tree Search
Monte Carlo Rollout
Agent
Rival
Tree Policy
Rollout Policy
Forward
State
Win
Win
Lose
X N
Root
Node
Leaf
Rollout
PenicillinLP
bilibili


Monte Carlo Tree Search
MiniMax Tree Search
Monte Carlo Rollout
Agent
Rival
Update Value
No Update of Value
Backward
State
Back Up
Win
Back Up
Win
Back Up
Lose
Root
Node
Leaf
Rollout
PenicillinLP
bilibili

Monte Carlo Tree Search ← MiniMax Tree Search + Monte Carlo Rollout

X Agent

O Rival

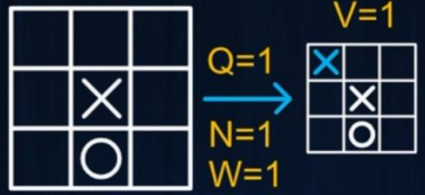
Tree Policy (UCT) :

$$A = \text{argmax}[Q(s,a)+c*\sqrt{\ln(N(s))/N(s,a)}]$$

Rollout Policy: Random

Selection ⟶ Simulation ⟶ Backup

State



Back Up ⟵ Win

Backup Q(s,a):

$Q(s,a) = W(s,a)/N(s,a)$

Backup W(s,a):

$W(s,a) = \sum V(S_{leaf})$

Backup $V(S_{leaf})$:

1 if win; -1 if lose

Root | Leaf | Rollout

PenicillinLP bilibili

Monte Carlo Tree Search ← MiniMax Tree Search, Monte Carlo Rollout

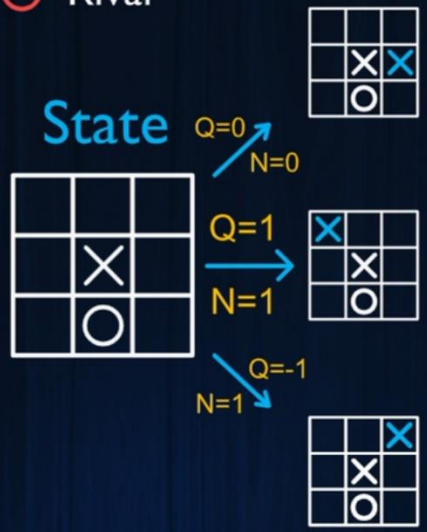
X Agent

O Rival

Tree Policy (UCT) :

$A = \text{argmax}[Q(s,a)+c*\sqrt{\ln(N(s))/N(s,a)}]$

Rollout Policy: Random



Selection ⟶ Simulation ⟶ Backup

One Iteration

Root Leaf

PenicillinLP bilibili

Monte Carlo Tree Search ← MiniMax Tree Search, Monte Carlo Rollout

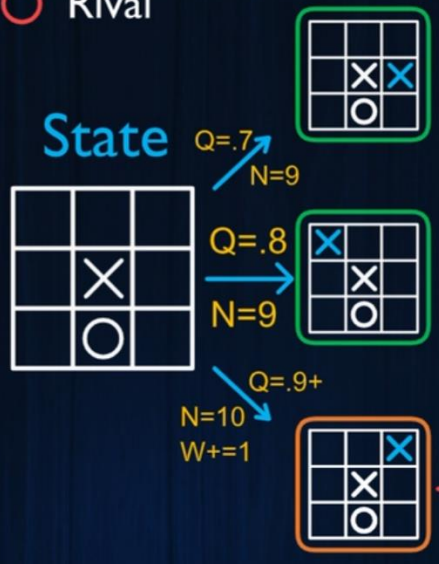
X Agent

O Rival

Tree Policy (UCT) :

$A = \text{argmax}[Q(s,a)+c*\sqrt{\ln(N(s))/N(s,a)}]$

Rollout Policy: Random



Selection → Simulation → Backup

if N=10 → (optional) Expansion

One Iteration

Back Up ← Win

Root | Node | Leaf | Rollout

Monte Carlo Tree Search ← MiniMax Tree Search + Monte Carlo Rollout

X Agent

O Rival

Take action:

Maximum Q(s,a)

Maximum N(s,a)

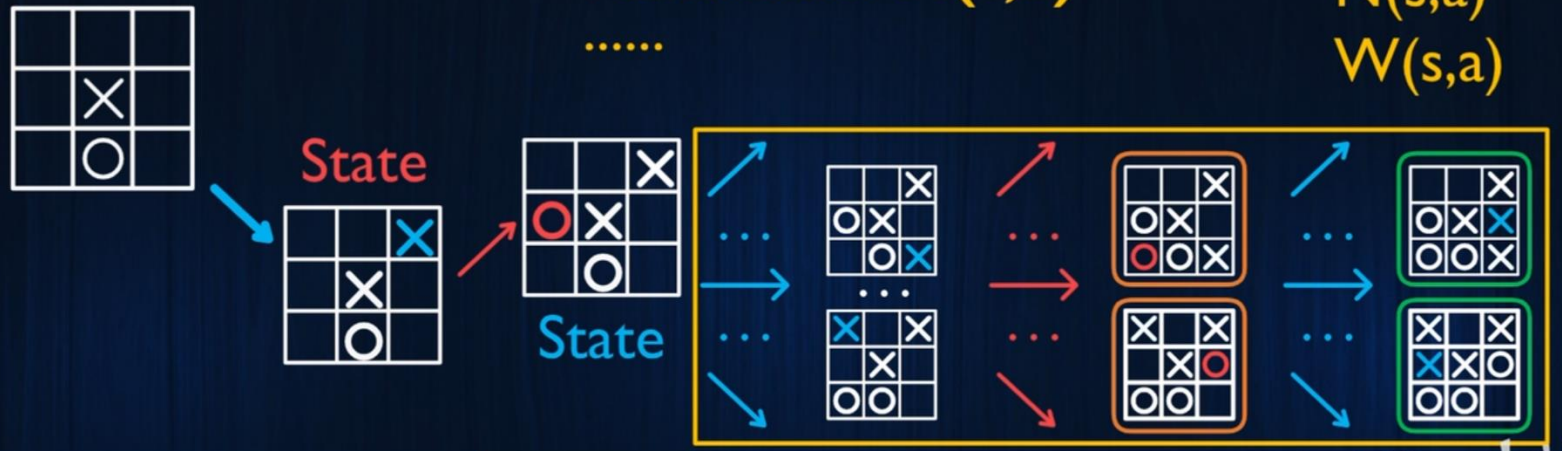
.....

After N Iterations

Q(s,a)

N(s,a)

W(s,a)



Significance of MCTS

AlphaGo

MCTS
+
Deep NN

2006, Coulom
2006, Kocsis & Szepesvári
2006, Hinton & Salakhutdinov

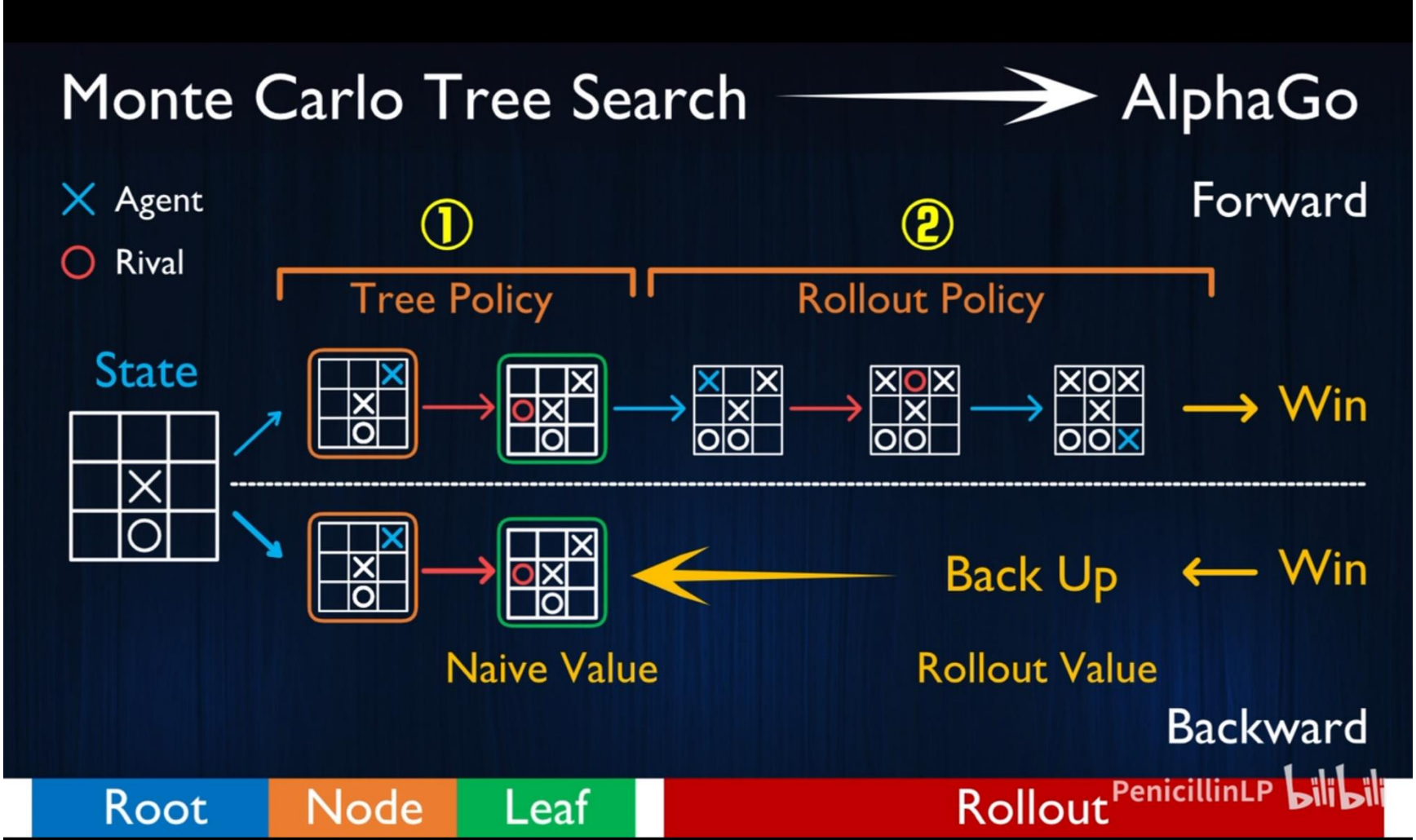
Rank System
of Go (one example)

Professional dan だん
1p - 9p 段

Amature dan だん
1d - 7/8/9d 段

Amature kyu きゅう
10k - 1k 級

PenicillinLP


Monte Carlo Tree Search
AlphaGo
Agent
Rival
Forward
①
②
Tree Policy
Rollout Policy
State
Win
Back Up
Win
Naive Value
Rollout Value
Backward
Root
Node
Leaf
Rollout
PenicillinLP
bilibili

AlphaGo Monte Carlo Tree Search + Deep Neural Network

Supervised Learning of Policy Networks

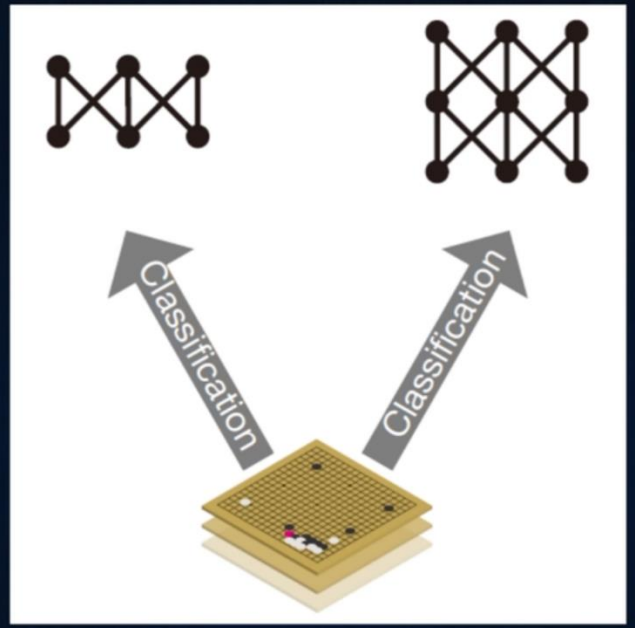
$$\Delta\sigma \propto \frac{\partial \log p_\sigma(a|s)}{\partial \sigma}$$

Rollout Policy $p_\pi(a|s)$

Linear Softmax
of small pattern features

Action time: 2 μs
Accuracy: 24.2 %

Human Expert
State-Action Pairs
8 million (s,a)



SL Policy $p_\sigma(a|s)$

Deep Convolutional
Neural Network

Action time: 3 ms
Accuracy: 57.0 %

Human Expert
State-Action Pairs
30 million (s,a)

AlphaGo Monte Carlo Tree Search + Deep Neural Network

Reinforcement Learning of Policy Networks and Value Networks

RL Policy $p_\rho(a|s)$

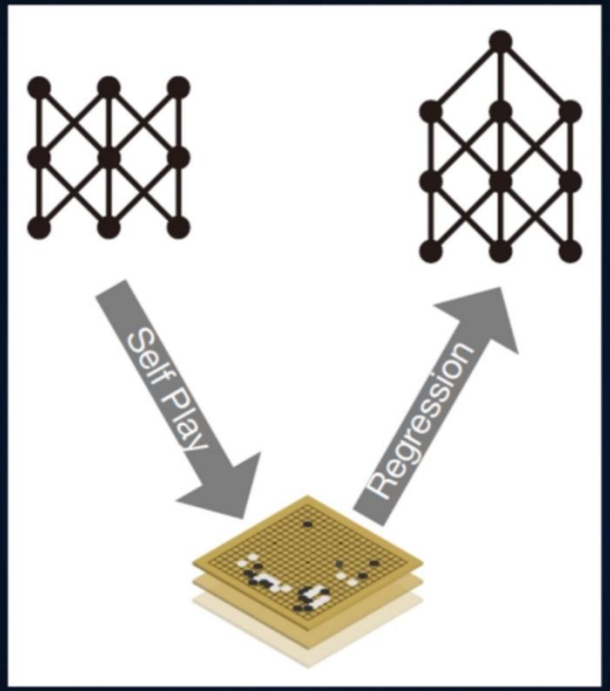
Deep Convolutional Neural Network

ρ initialized as σ

Self Play (s,a,z)

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

z = ± 1



Value $v_\theta(s)$

Deep Convolutional Neural Network

Self Play of $p_\rho(a|s)$
30 million (s,z)

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

Training MSE 0.226
Testing MSE 0.234

AlphaGo Monte Carlo Tree Search + Deep Neural Network

X Agent

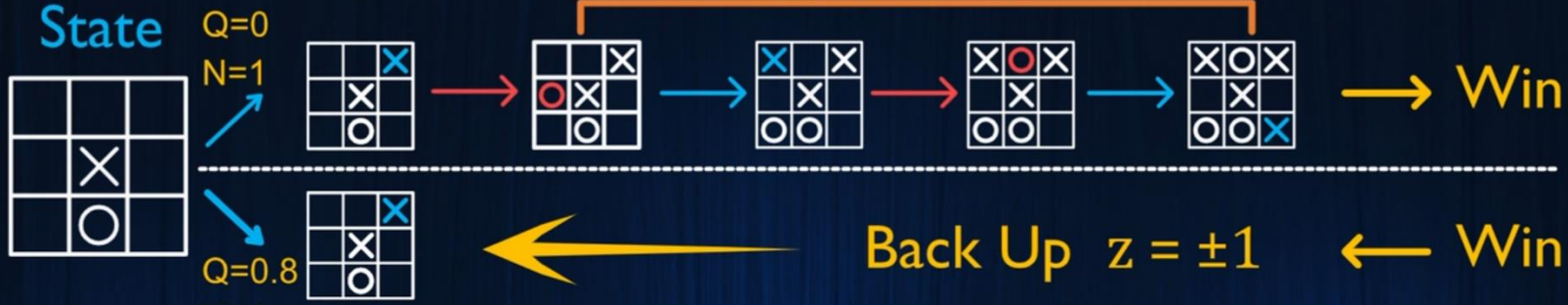
O Rival

① Tree Policy (with SL Policy) :

A = argmax[Q(s,a)+u(s,a)]

$$u(s,a) \propto \frac{p_\sigma(a|s)}{1+N(s,a)}$$

② Rollout Policy $p_\pi(a|s)$



Root | Leaf | Rollout

PenicillinLP bilibili

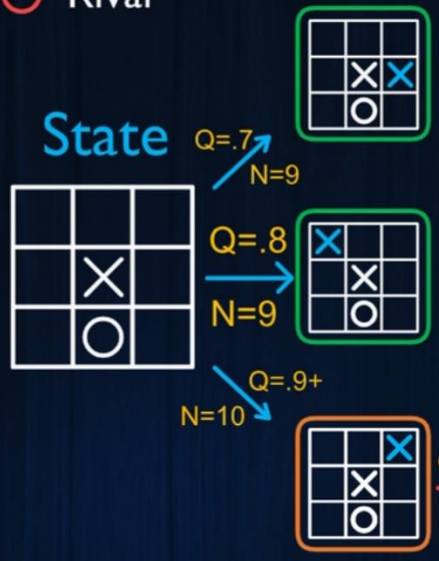
AlphaGo Monte Carlo Tree Search + Deep Neural Network

X Agent

O Rival

① Tree Policy (with SL Policy) :
A = argmax[Q(s,a)+u(s,a)]

② Rollout Policy:
$p_\pi(a|s)$



Selection → Simulation → Backup

(optional) Expansion

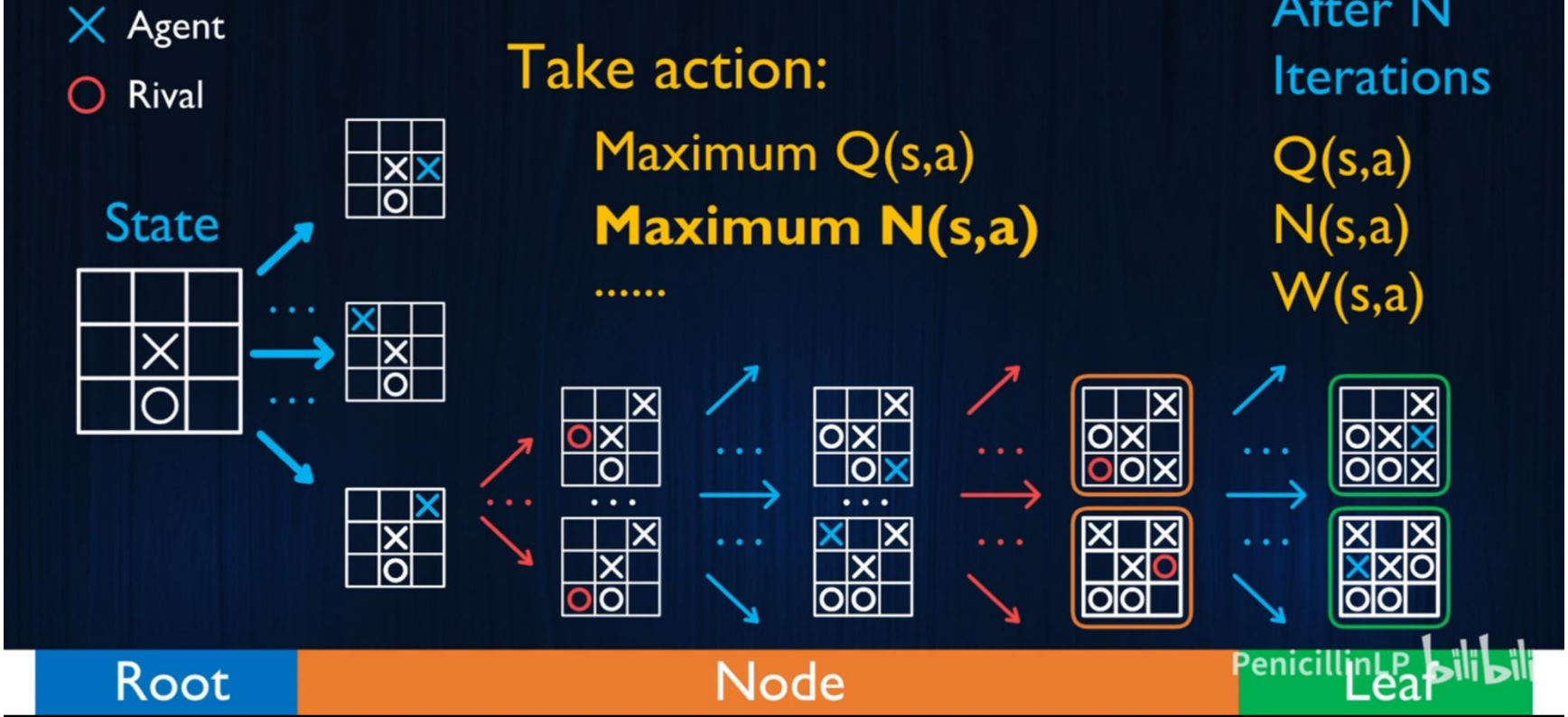
One Iteration

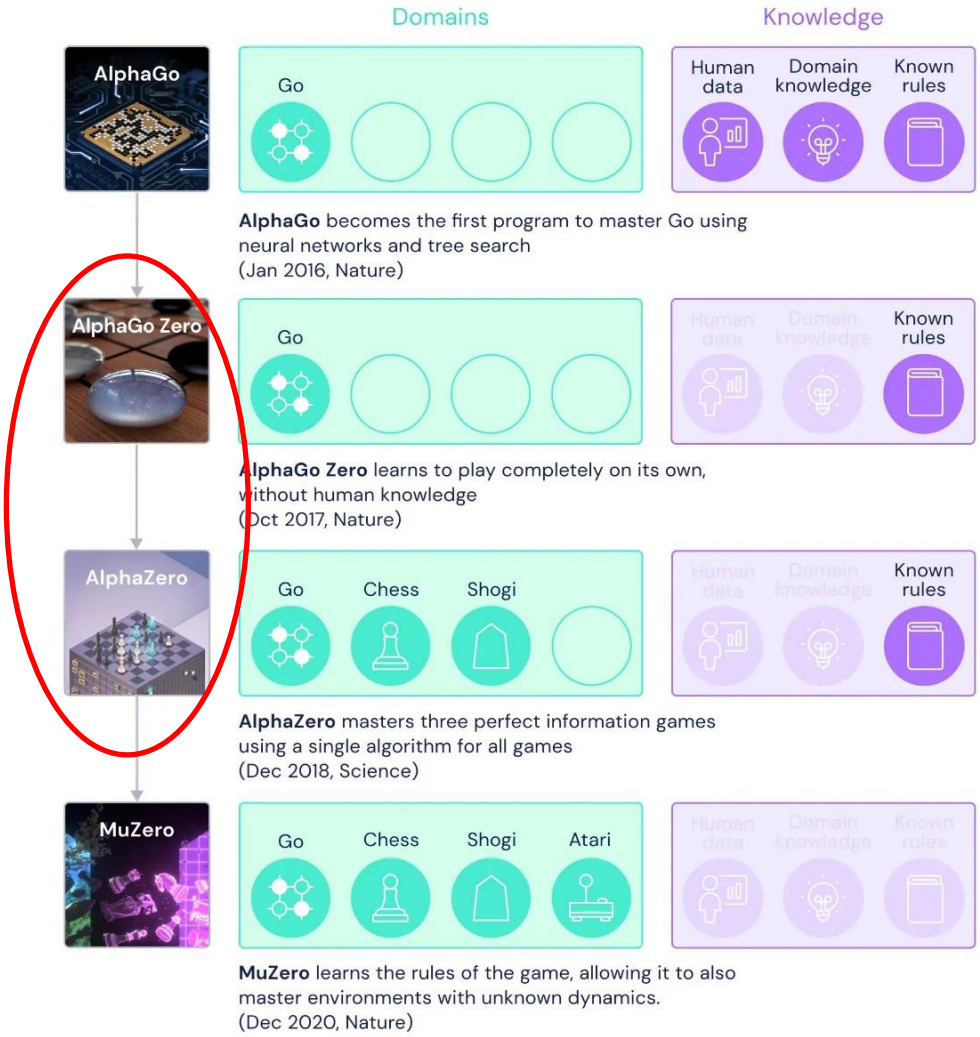
Back Up ← Win

③ $V = (1-\lambda)z + \lambda v_\theta(s)$

Root | Node | Leaf | Rollout

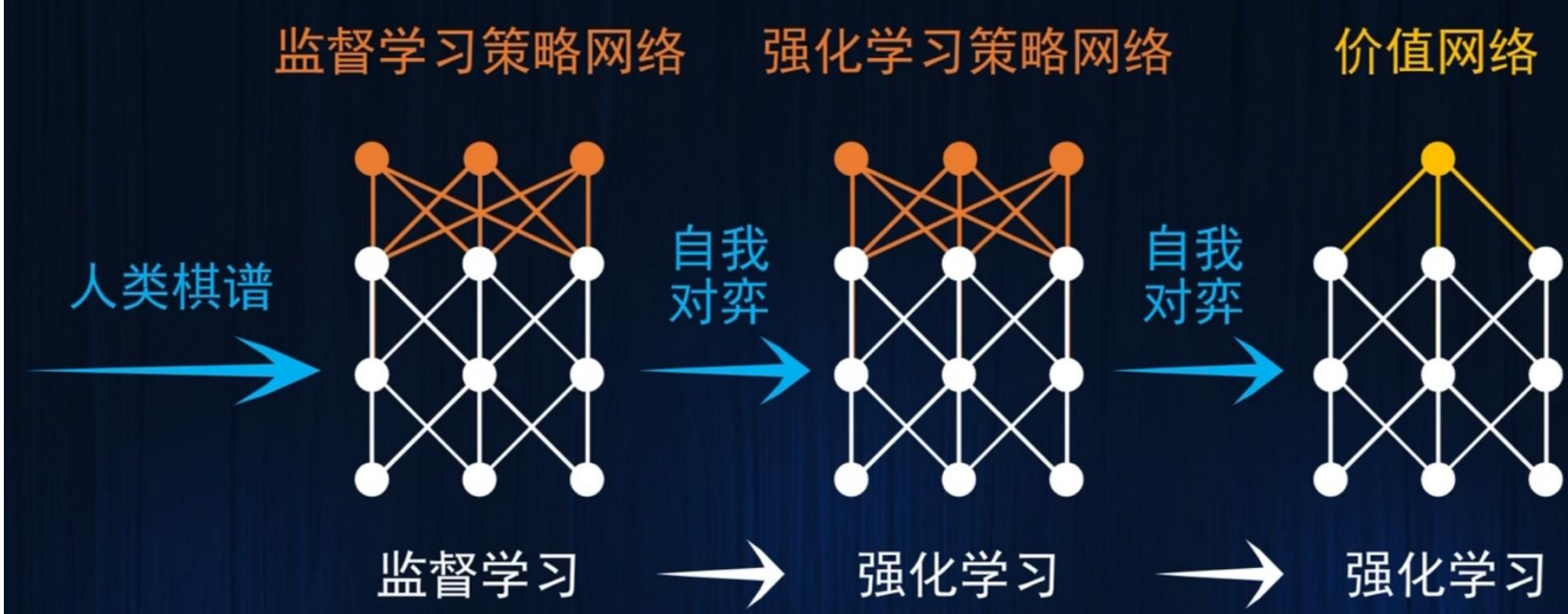
AlphaGo Monte Carlo Tree Search + Deep Neural Network




Domains
Knowledge
AlphaGo
Go
Human data
Domain knowledge
Known rules
AlphaGo becomes the first program to master Go using neural networks and tree search
(Jan 2016, Nature)
AlphaGo Zero
Go
Known rules
AlphaGo Zero learns to play completely on its own, without human knowledge
(Oct 2017, Nature)
AlphaZero
Go
Chess
Shogi
Known rules
AlphaZero masters three perfect information games using a single algorithm for all games
(Dec 2018, Science)
MuZero
Go
Chess
Shogi
Atari
MuZero learns the rules of the game, allowing it to also master environments with unknown dynamics.
(Dec 2020, Nature)

AlphaGo

PenicillinLP bilibili



利用的是策略网络和价值网络进行蒙特卡洛树搜索

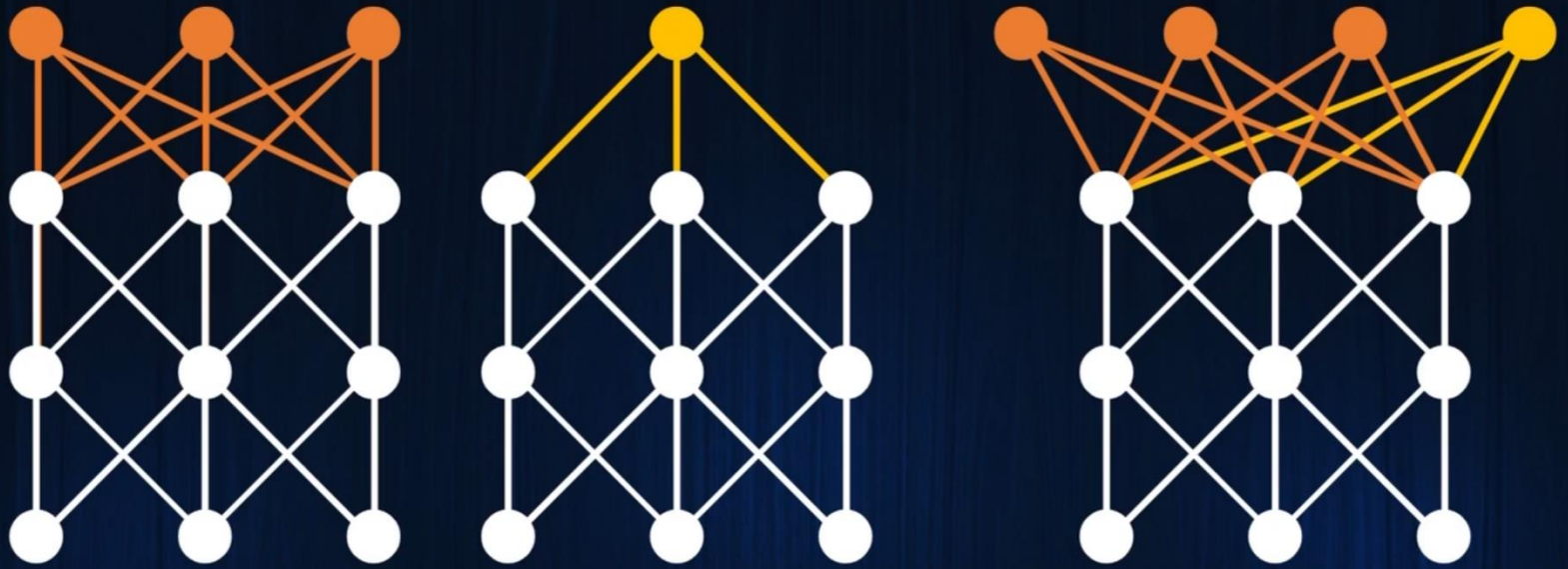
AlphaGo

PenicillinLP bilibili

蒙特卡洛树搜索
Monte Carlo Tree Search

利用的是策略网络和价值网络进行蒙特卡洛树搜索

AlphaGo → AlphaGo Zero



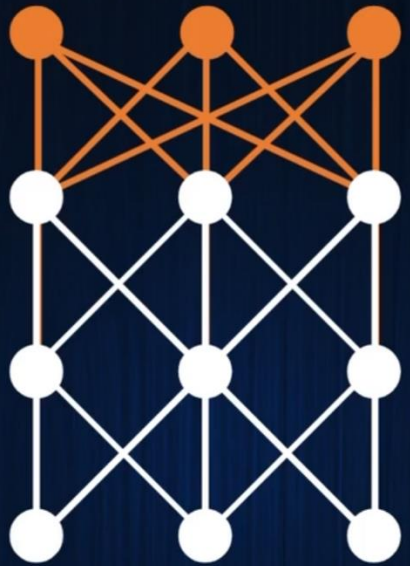
只用了一个神经网络

纯监督学习

PenicillinLP bilibili

一. 完全靠别人教

人类棋谱



AlphaGo

监督学习策略网络

这样得到的策略网络并不强

监督学习 强化学习PenicillinLP bilibili

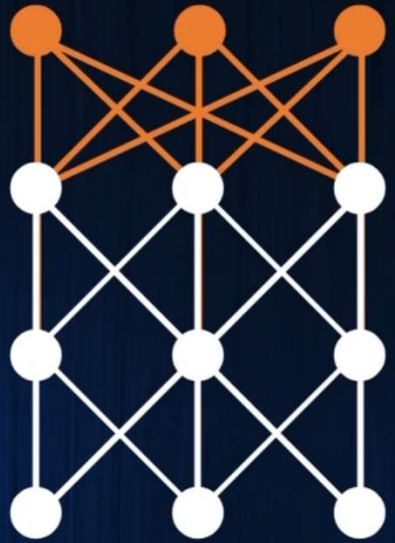
二. 师傅领进门，修行在个人

人类棋谱

监督学习

自我对弈

强化学习



AlphaGo

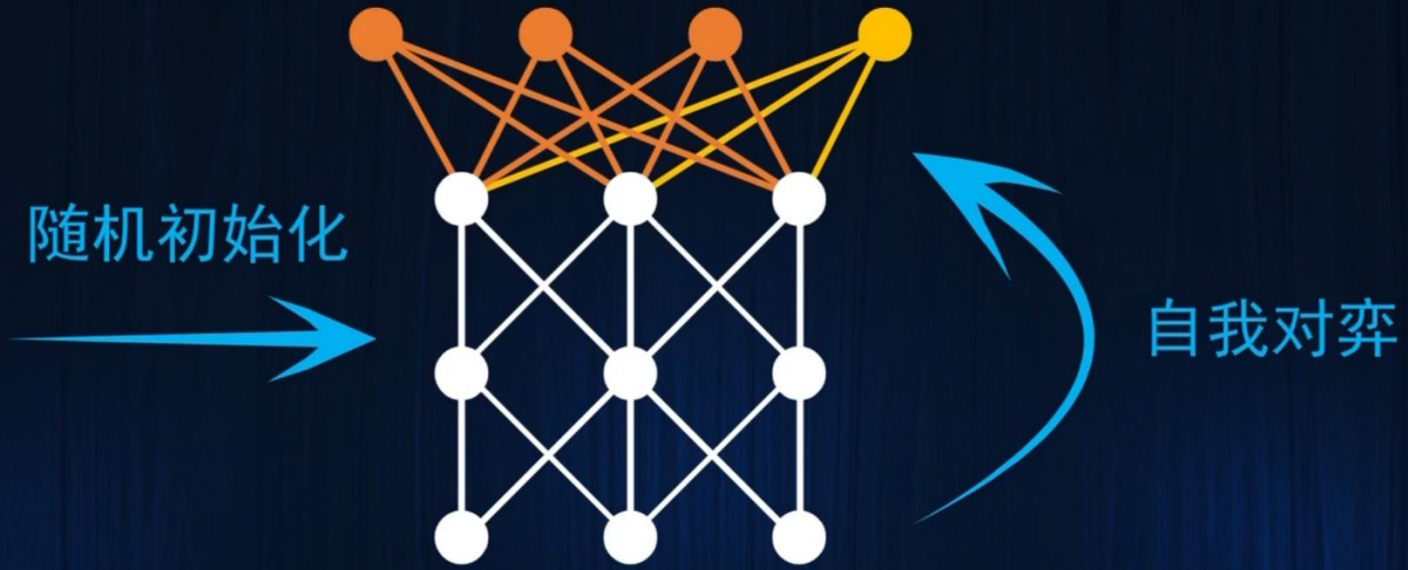
强化学习
策略网络

这样得到的就是AlphaGo中的强化学习策略网络

纯强化学习

PenicillinLP bilibili

三. 完全靠自己琢磨



正所谓思而不学则殆

PenicillinLP bilibili

纯监督学习

一. 完全靠别人教 

监督学习 强化学习

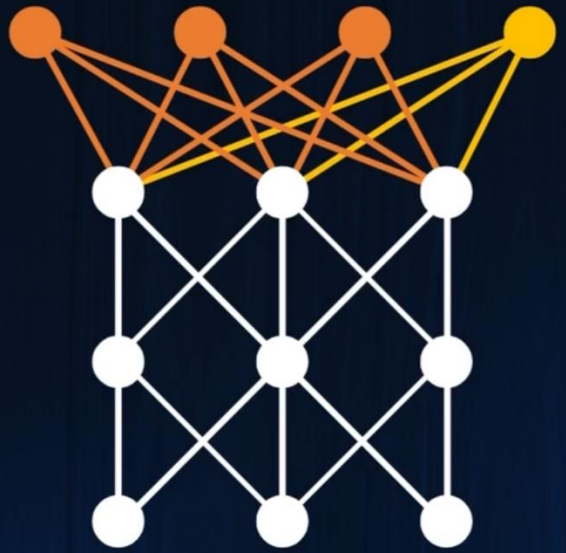
二. 师傅领进门，修行在个人 

纯强化学习

三. 完全靠自己琢磨 

AlphaGo Zero使用了第四种学习方法

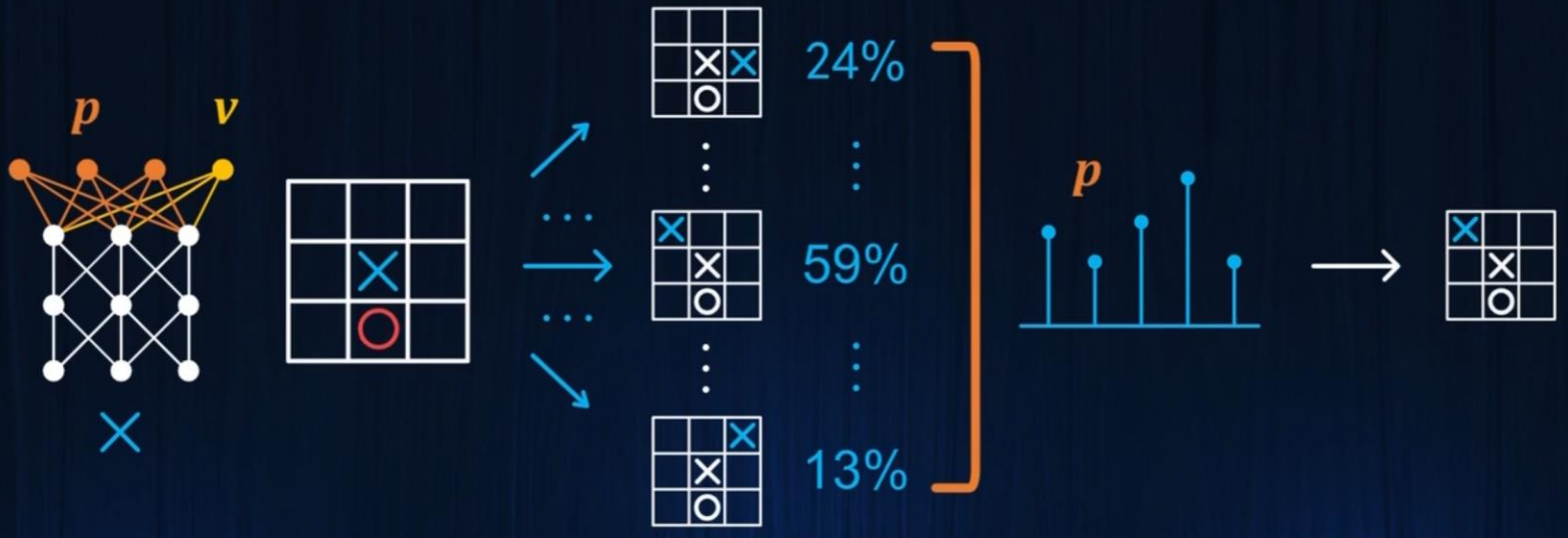
PenicillinLP bilibili



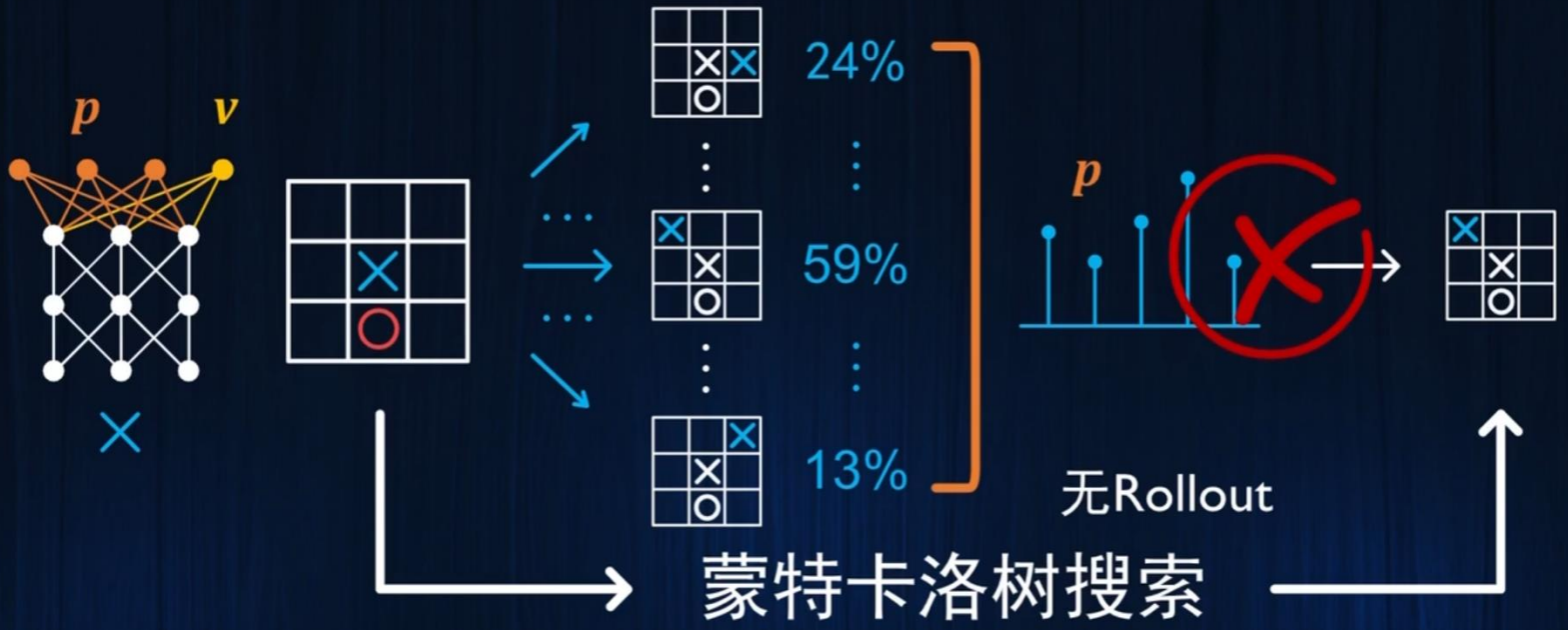
蒙特卡洛树搜索

AlphaGo Zero这个神经网络的老师

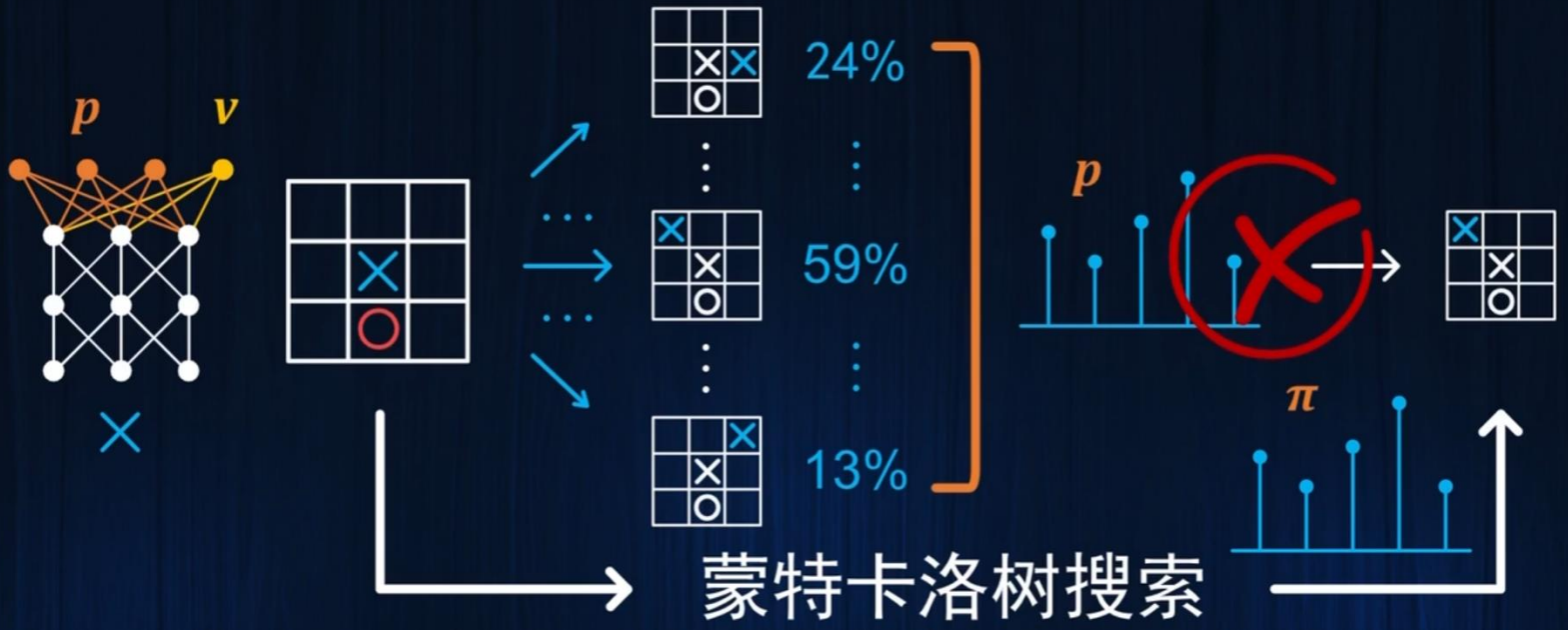
PenicillinLP bilibili



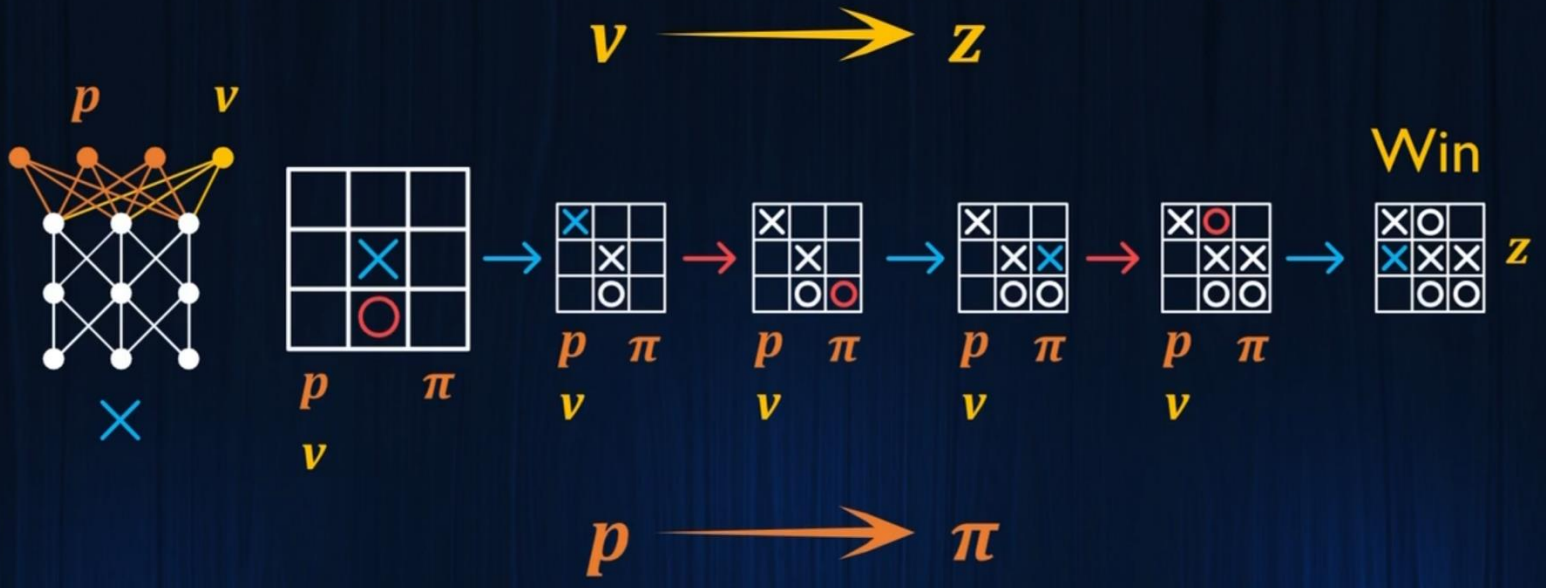
这样就和AlphaGo训练强化学习策略网络的时候一样



AlphaGo Zero没有人类训练，也就没有Rollout策略



我们利用蒙特卡洛树搜索选择行动，直到游戏结束



如果最后赢了，那就提高这个行动的概率

PenicillinLP

一. π 比 p 强

p

π 蒙特卡洛树搜索

二. p 越强，π 越强

PenicillinLP bilibili

Alpha

深度神经网络

+

蒙特卡洛树搜索

Zero

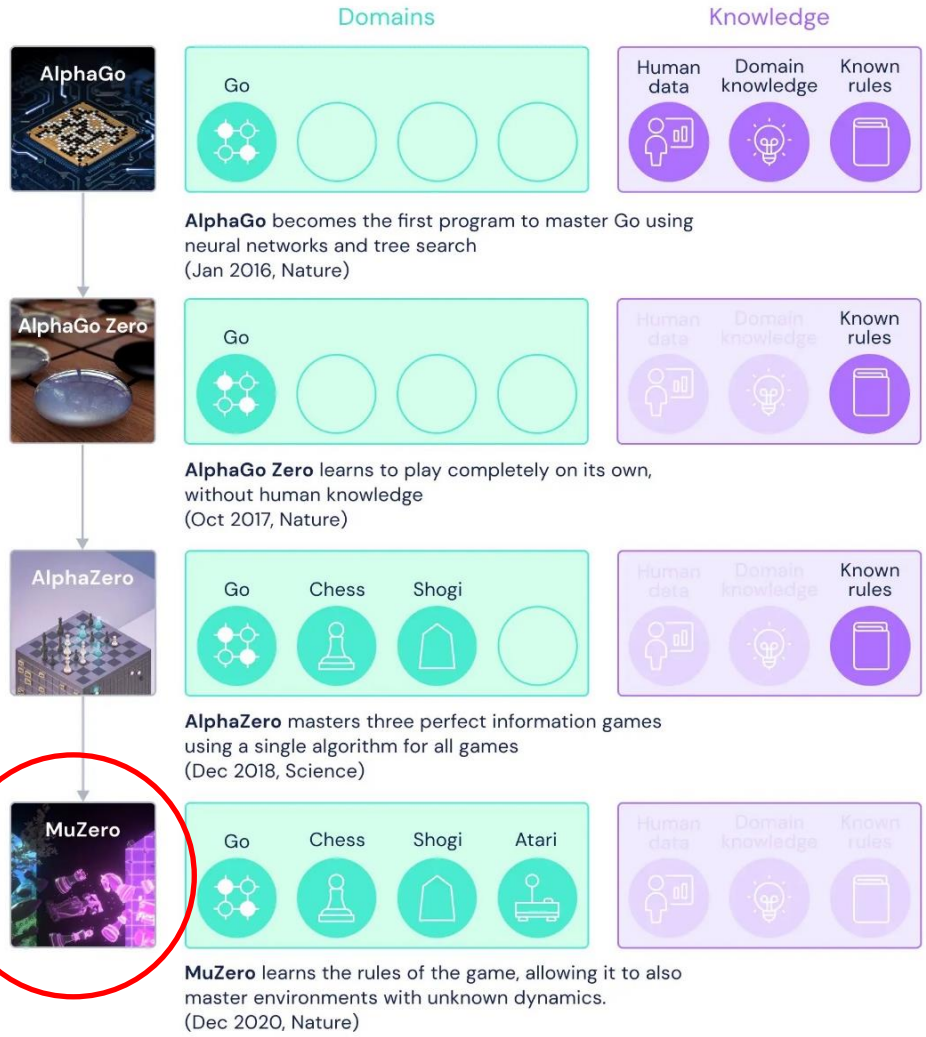
不需要人类知识

AlphaGo

AlphaZero

AlphaGo Zero

AlphaZero这篇文章也就是用这三个例子呈现了这一思想而已


Domains
Knowledge
AlphaGo
Go
Human data
Domain knowledge
Known rules
AlphaGo becomes the first program to master Go using neural networks and tree search
(Jan 2016, Nature)
AlphaGo Zero
Go
Known rules
AlphaGo Zero learns to play completely on its own, without human knowledge
(Oct 2017, Nature)
AlphaZero
Go
Chess
Shogi
Known rules
AlphaZero masters three perfect information games using a single algorithm for all games
(Dec 2018, Science)
MuZero
Go
Chess
Shogi
Atari
MuZero learns the rules of the game, allowing it to also master environments with unknown dynamics.
(Dec 2020, Nature)

MuZero

表征网络　　机制网络　　隐状态空间

深度神经网络+蒙特卡洛树搜索

预测网络

围棋规则

PenicillinLP

一、终局和胜负的判断

二、提子

三、由谁落子，以及允许的落子

PenicillinLP

规则

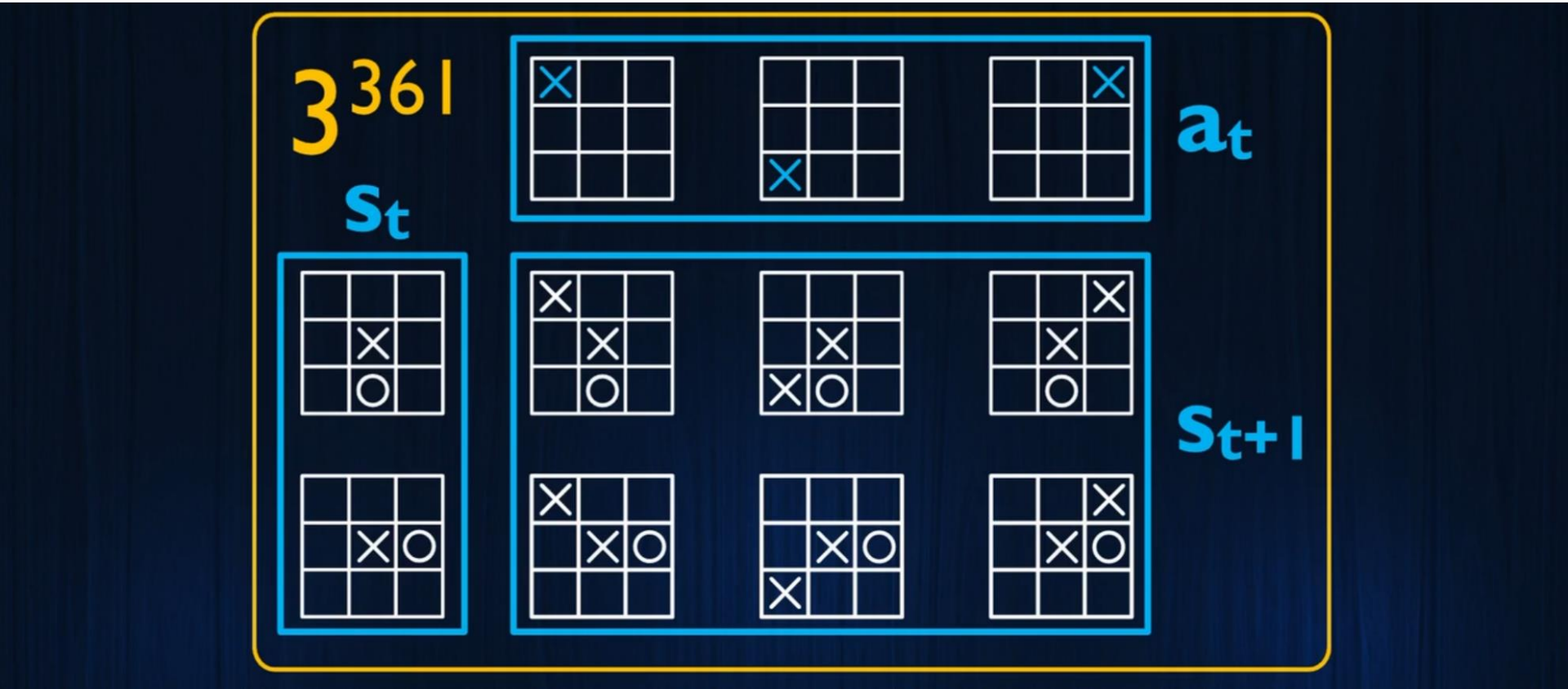
一、最终状态以及奖励的判断

二、游戏机制，Dynamics

$(s_t, a_t) \rightarrow s_{t+1}$

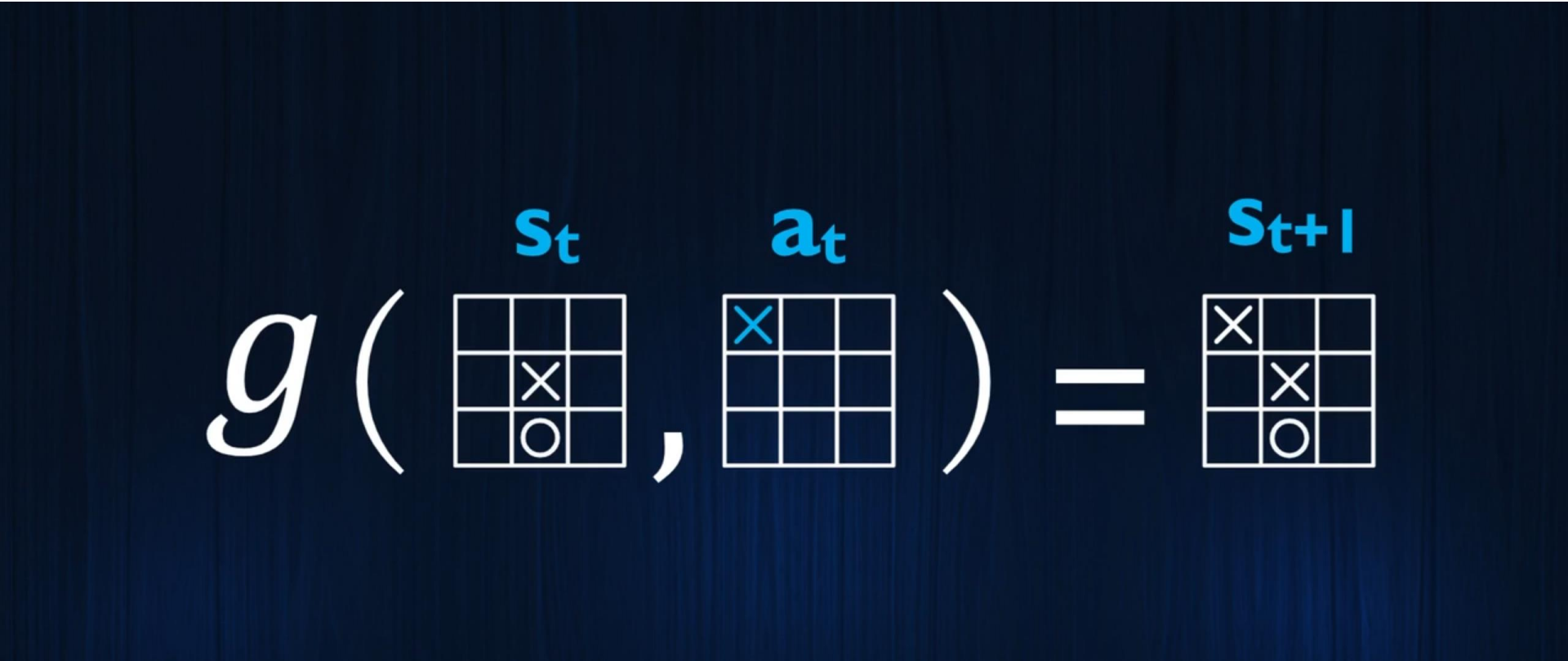
三、采取行动的Agent & 允许的行动

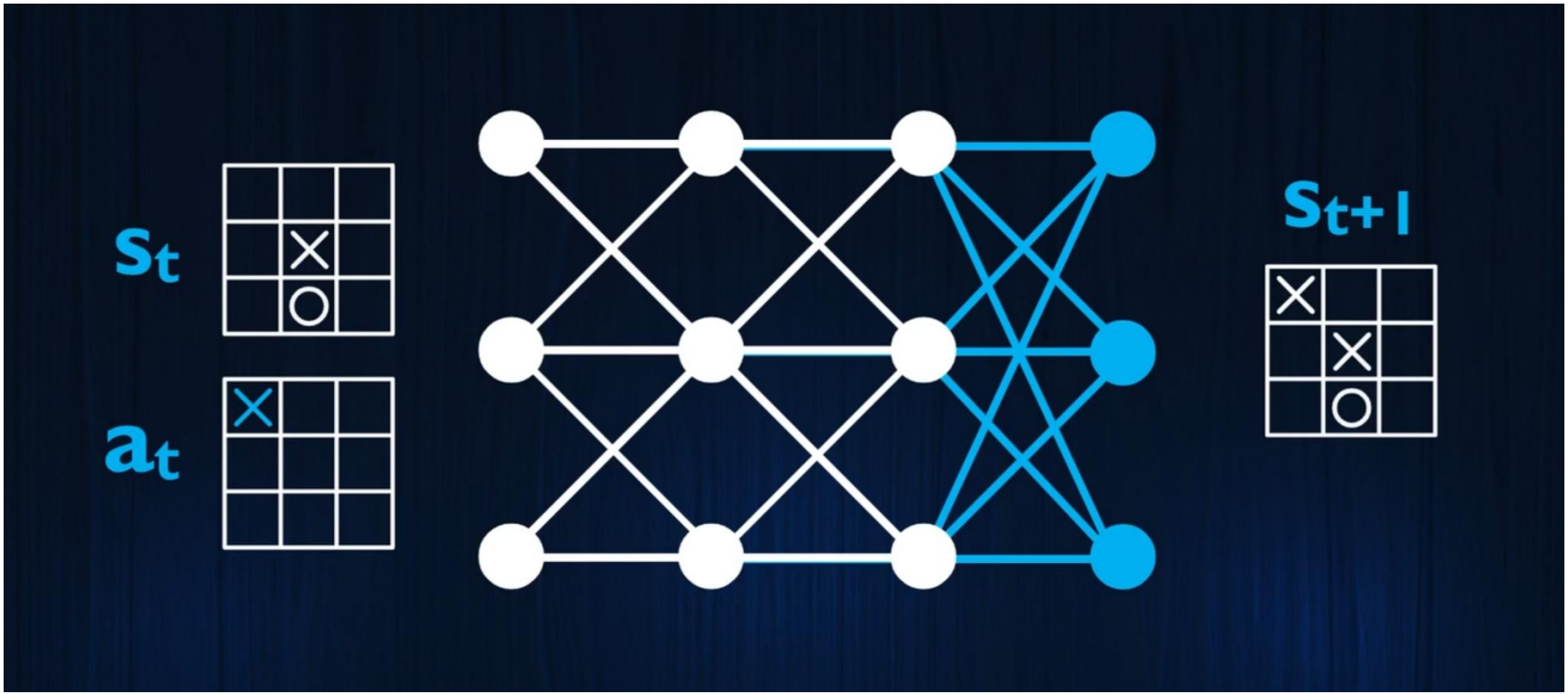
列表法



$$2^{360}\ \text{Byte} = 2^{320}\ \text{TB}$$

函数法




s_t
a_t
s_{t+1}

隐状态 Hidden State

表征网络

Representation Network

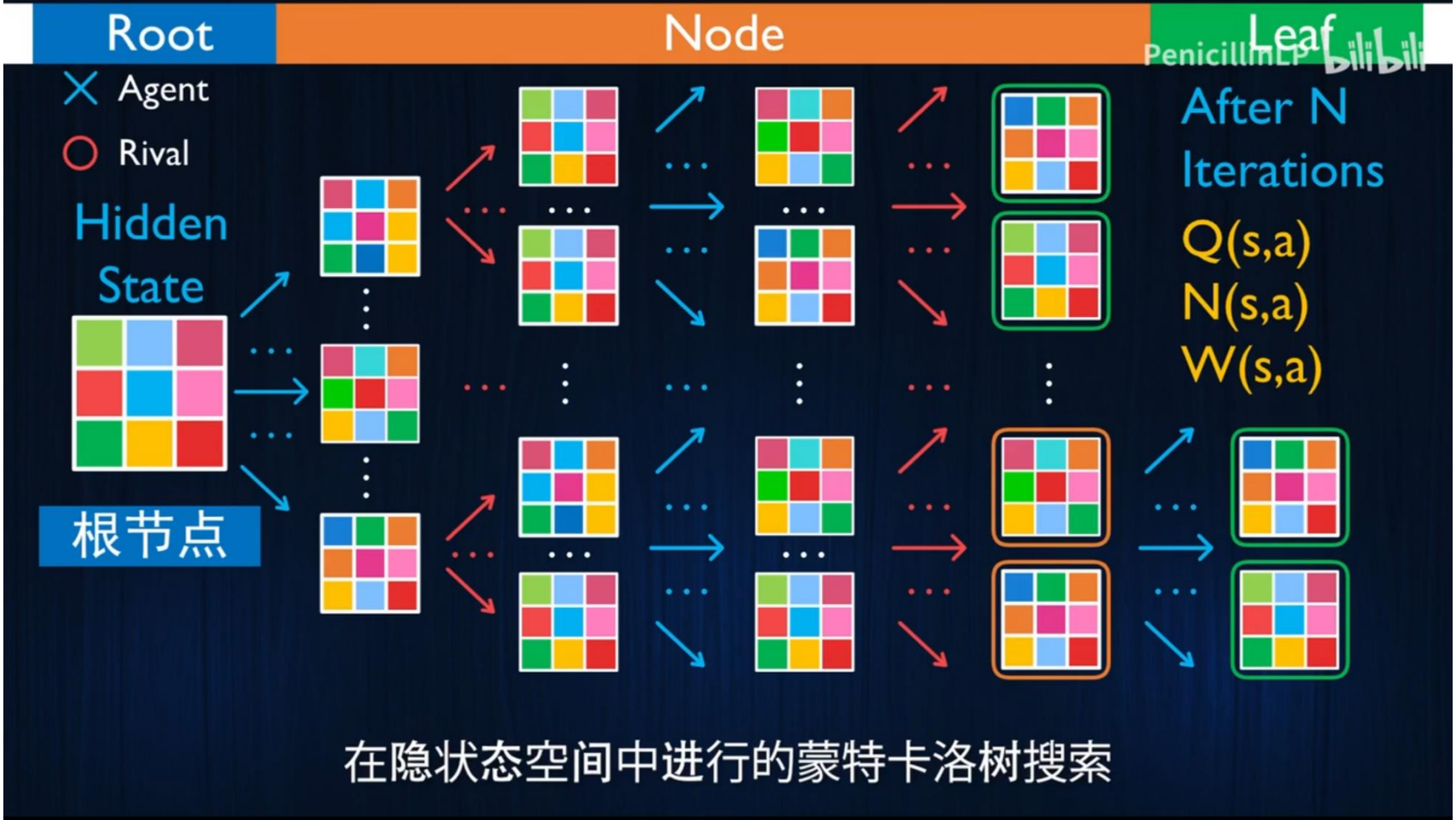
h

0	0	0
0	1	0
0	0	0

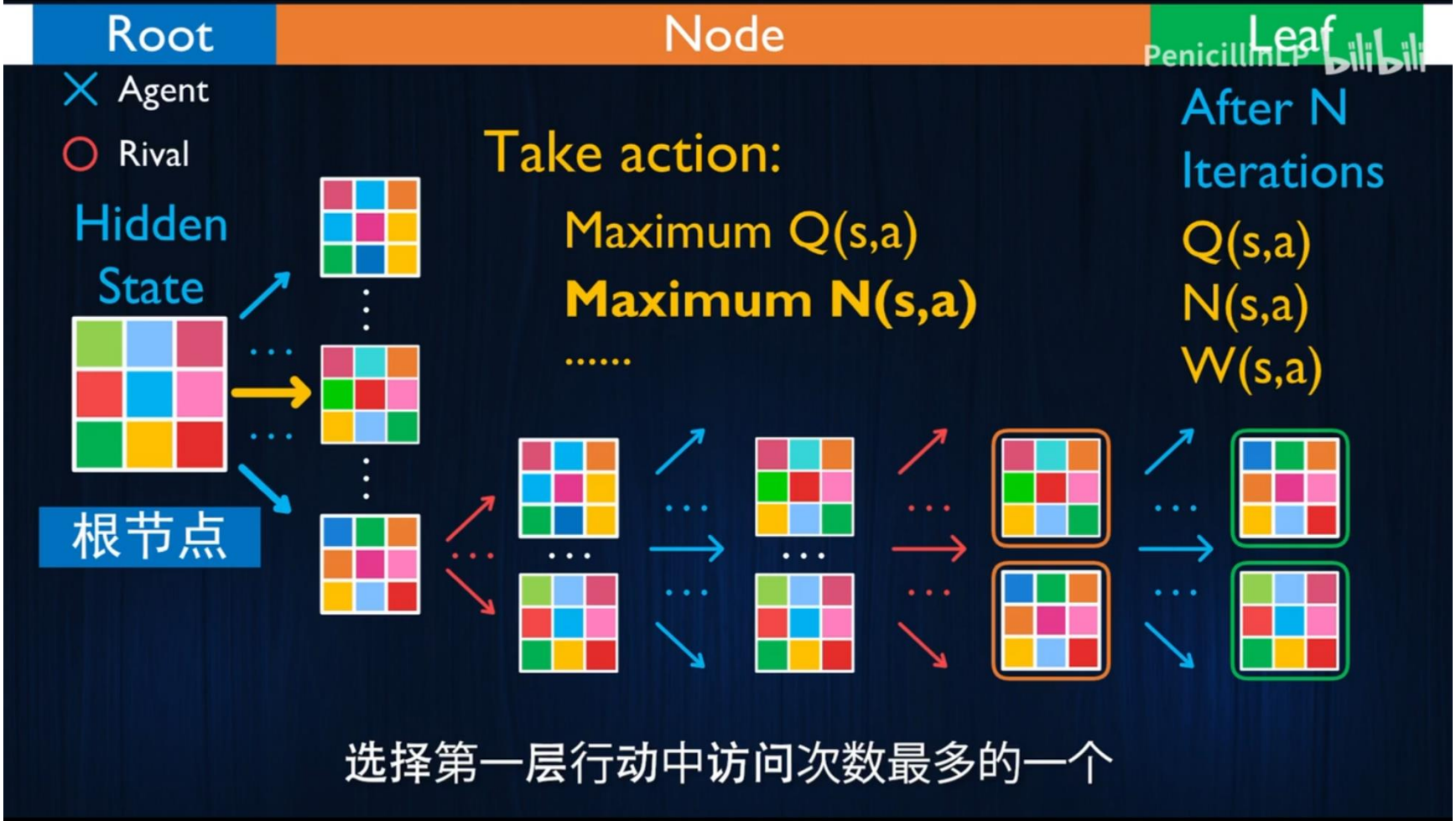
0	0	0
0	0	0
0	1	0

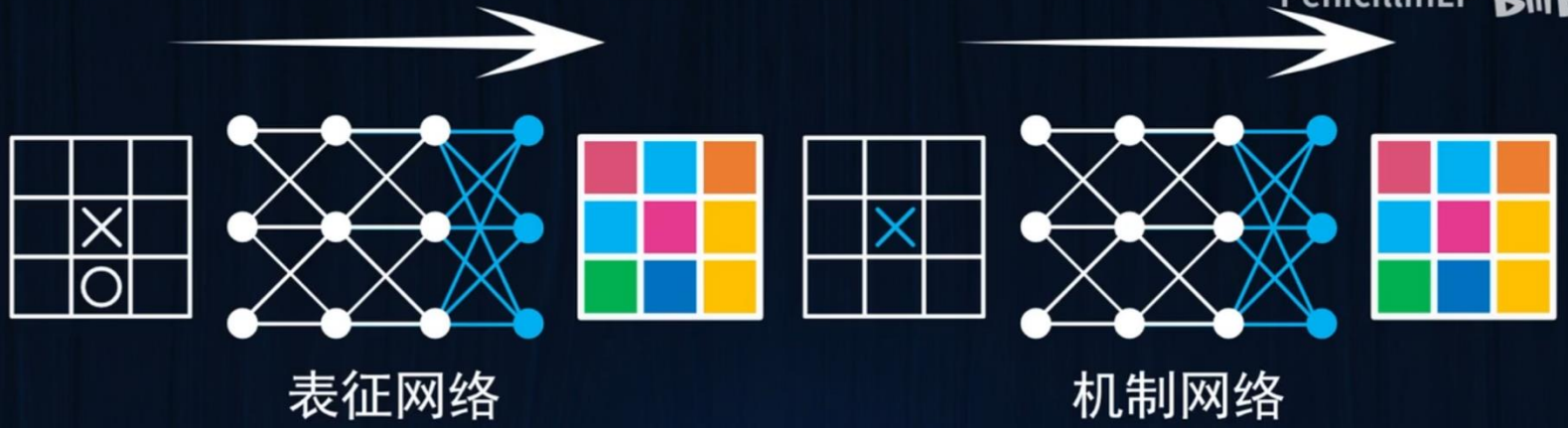
1	1	1
1	0	1
1	0	1

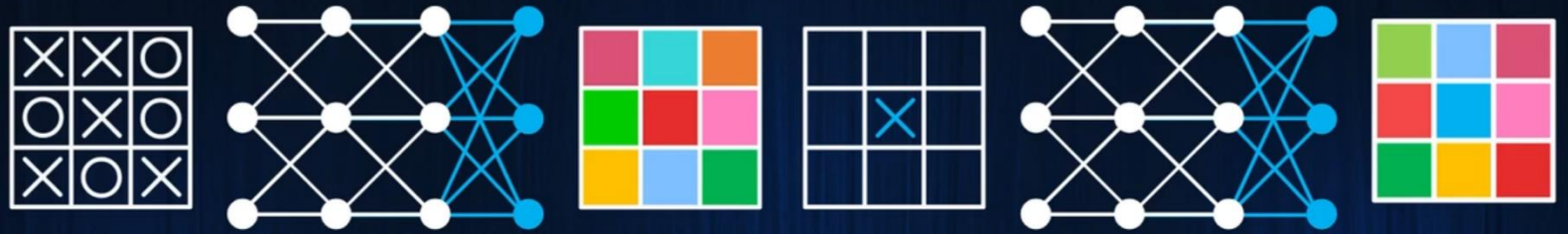
表征网络把一个真实的棋盘状态转换成一个抽象的隐状态



在隐状态空间中进行的蒙特卡洛树搜索


Root
Node
Leaf
PenicillinLP
Agent
Rival
Hidden State
根节点
Take action:
Maximum Q(s,a)
Maximum N(s,a)
.....
After N Iterations
Q(s,a)
N(s,a)
W(s,a)
选择第一层行动中访问次数最多的一个





把隐状态和行动作为机制网络的输入

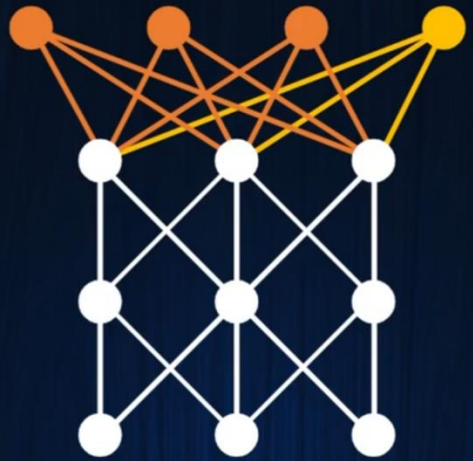
MuZero

PenicillinLP bilibili

表征网络 h

Representation Network

预测网络 f



Prediction Network

机制网络 g

Dynamics Network


PenicillinLP bilibili
AlphaGo
AlphaGo Zero
AlphaZero
MuZero
AlphaGo开创了结合深度神经网络和蒙特卡洛树搜索这一思想


PenicillinLP bilibili
AlphaGo
AlphaGo Zero
92.1%
AlphaZero
MuZero
Alpha(Go)Zero进一步提出利用蒙特卡洛树搜索来训练深度神经网络


AlphaGo
AlphaGo Zero
AlphaZero
MuZero
PenicillinLP
bilibili
开创了在隐状态空间进行蒙特卡洛树搜索的算法

谢谢大家