TechX 深度强化学习课程（六）

何舜成

宽带网数字媒体实验室
清华大学自动化系

2018 年 7 月 23 日

目录

基于模型的方法

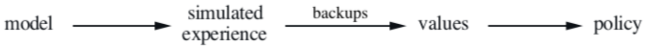
模型与规划

两种不同程度的模型

- 分布模型——精确地知道概率分布 $p(s', r \mid s, a)$
- 采样模型——输入 (s, a) 可以得到 (s', r)

分布模型强于采样模型，但采样模型可能较分布模型更容易获取。（掷骰子）

基于先前 RL 架构，策略的选取和改进都是建立在估值的基础上。有了环境模型，我们可以通过模拟的方法提高值函数的精确度，这是基于模型的方法（model-based）的框架。



模型与规划

基于模型学习值函数与先前的算法并无区别（下图），利用模型，和利用与环境的交互学习值函数是几乎等价的。但这两个优点保证了基于模型的算法的必要性

- 环境无法指定初始状态
- 与环境的交互代价大

Do forever:

1. Select a state, $S \in \mathcal{S}$, and an action, $A \in \mathcal{A}(s)$, at random
2. Send S, A to a sample model, and obtain
   a sample next reward, R, and a sample next state, S'
3. Apply one-step tabular Q-learning to S, A, R, S':
   $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$

图 1: Alg: Random Sample Tabular Q-planning

Dyna-Q

Dyna-Q 是一种经典算法，较之通常的 RL 算法，加入了模型学习和用模型学习值函数的模块。

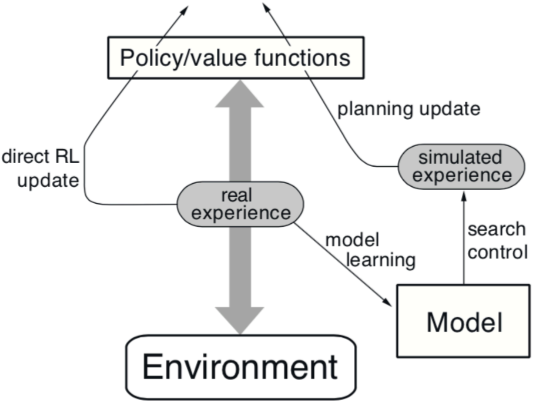


图 2: Dyna-Q 算法结构

Dyna-Q

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

Do forever:

(a) $S \leftarrow$ current (nonterminal) state

(b) $A \leftarrow \epsilon\text{-greedy}(S,Q)$

(c) Execute action A; observe resultant reward, R, and state, S'

(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$

(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)

(f) Repeat n times:

- $S \leftarrow$ random previously observed state
- $A \leftarrow$ random action previously taken in S
- $R, S' \leftarrow Model(S,A)$
- $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$

图 3: Alg: Tabular Dyna-Q

示例：迷宫

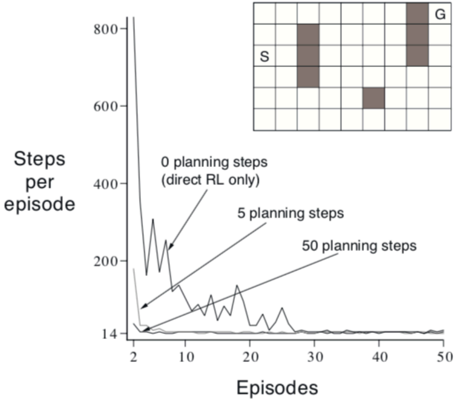


图 4: 模型对性能的提升

基于模型方法的局限性

- 更多 RL 问题的环境模型是随机的（非确定性）
- 环境模型过于复杂以至于难以学习
- 一旦模型学习差，反而妨碍值函数的估计
- 计算开销大

强化学习方法小结

要解决的问题：**MDP**

RL 算法共性：

- 对状态，或动作进行估计（求值函数）
- 估计时利用状态轨迹反推
- 使用广义的策略迭代法

强化学习方法小结

算法的两个维度：深度和广度

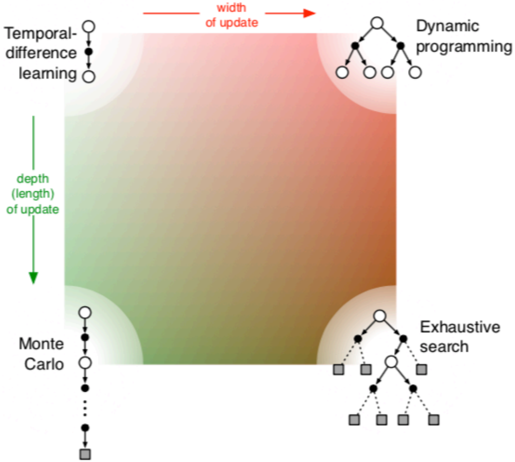


图 5: RL 算法分类

强化学习方法小结

算法的多种维度

- 广度
- 深度
- on-policy / off-policy
- 回报的形式
- 动作值函数 / 状态值函数
- 动作选择策略
- 是否使用模型
- **函数逼近的方法**

值函数逼近

函数逼近方法

Tabular 方法的含义：查表，维护一张从 s 到 $V(s)$ 的表格（或从 (s, a) 到 $Q(s, a)$），本质是学习一种映射。

能否用带参数的函数，如 $\hat{v}(s, \boldsymbol{w})$ 或 $\hat{q}(s, a, \boldsymbol{w})$ 去逼近 $v_\pi(s)$ 和 $q_\pi(s, a)$? 这种方法称为**函数逼近**方法。

当 MDP 的状态空间很大，甚至是连续量时，函数逼近方法显示出了与 Tabular 方法巨大的优点：

- 节省存储空间
- 能推广和泛化

在最近几年 RL 的进展中，几乎全用到了函数逼近方法。

状态值函数估计

对于状态值函数 $\hat{v}(s, Vw)$，评价对 $v_\pi(s)$ 的估计是否准确，可以采用加权的均方误差。

定义状态 s 的权重 $\mu(s)$，反应在策略 π 下，一个 episode 中，状态 s 被访问的概率。满足 $\forall s, \mu(s) \geq 0, \sum_s \mu(s) = 1$。

加权的均方误差记为 $\overline{VE}$

$$\overline{VE} = \sum_s \mu(s)(v_\pi(s) - \hat{v}(s, \boldsymbol{w}))^2 \tag{1}$$

满足对于所有可能的 $\boldsymbol{w}$，都有 $\overline{VE}(\boldsymbol{w}) \geq \overline{VE}(\boldsymbol{w}^*)$ 的 $\boldsymbol{w}^*$ 是最优参数。

应用 SGD

为了优化 $\overline{VE}$，一个可行的方法是在已有的样本上，用 SGD 减小预测误差。

$$\begin{aligned} \boldsymbol{w}_{t+1} &= \boldsymbol{w}_t - \frac{1}{2}\alpha\nabla\left(v_\pi(S_t) - \hat{v}(S_t, \boldsymbol{w}_t)\right)^2 & (2) \\ &= \boldsymbol{w}_t + \alpha\left(v_\pi(S_t) - \hat{v}(S_t, \boldsymbol{w}_t)\right)\nabla\hat{v}(S_t, \boldsymbol{w}_t) & (3) \end{aligned}$$

但真值 $v_\pi(S_t)$ 通常是未知的，我们需要用其他值 U_t 替代它。若 $\mathbb{E}[U_t \mid S_t = s] = v_\pi(S_t)$，那么 U_t 是无偏的。$\boldsymbol{w}_t$ 可以保证收敛到一个局部最优解。

应用 SGD

联想到 MC 和 TD 方法，容易推导出相应的算法来

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^d \rightarrow \mathbb{R}$

Initialize value-function weights $\mathbf{w}$ as appropriate (e.g., $\mathbf{w} = \mathbf{0}$)
Repeat forever:
  Generate an episode $S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T$ using π
  For $t = 0, 1, \ldots, T-1$:
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[G_t - \hat{v}(S_t,\mathbf{w})\big]\nabla\hat{v}(S_t,\mathbf{w})$

图 6: Alg: Gradient Monte Carlo

应用 SGD

由于 TD 方法不能保证目标是无偏的，因此称为半梯度方法。同样的还可以推导出相应的 n-step 方法。

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \rightarrow \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$

Initialize value-function weights $\mathbf{w}$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose $A \sim \pi(\cdot|S)$
    Take action A, observe R, S'
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})\big]\nabla\hat{v}(S,\mathbf{w})$
    $S \leftarrow S'$
  until S' is terminal

图 7: Alg: Semi-gradient TD(0)

半梯度 Sarsa

Input: a differentiable function $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$

Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
Repeat (for each episode):
    $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)
    Repeat (for each step of episode):
        Take action A, observe R, S'
        If S' is terminal:
            $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
            Go to next episode
        Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., ε-greedy)
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
        $S \leftarrow S'$
        $A \leftarrow A'$

图 8: Alg: Semi-gradient Sarsa

Policy Gradient

策略参数化

之前的方法基本都是先估计值函数，然后按不同的方法构造策略。策略的本质是 s 到 a 的映射。有了参数化的工具，我们可以直接将策略参数化，此时的策略为 $\pi(a \mid s, \boldsymbol{\theta})$。

为了改进策略，需要定义一个评价函数 $J(\boldsymbol{\theta})$，改进策略时用梯度方法：

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla J(\boldsymbol{\theta}_t) \tag{4}$$

评价函数一般采用参数化的值函数。

参数化的好处

某些情况下，最优策略是随机策略，$\epsilon - greedy$ 等方法无法做到。如下例：走廊问题

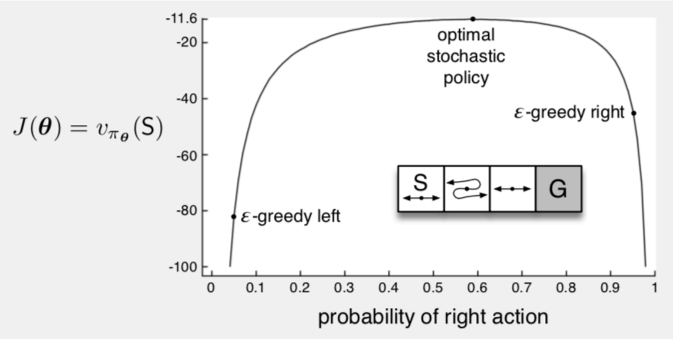


图 9: 最优策略是随机策略

Policy Gradient 定理

在 episode 长度有限的情况下，一般取 $J(\boldsymbol{\theta}) = v_{\pi_{\boldsymbol{\theta}}}(s_0)$，$s_0$ 是某一固定的初始状态（大多数问题中初始状态都是相同的）。

Policy Gradient 定理计算了 $J(\boldsymbol{\theta})$ 的梯度方向：

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a q_\pi(s, a) \nabla \pi(a \mid s, \boldsymbol{\theta}) \tag{5}$$

证明略。

直观上看，越是经常出现的状态，梯度的比重越大，越是 Q 值越高的动作，梯度也越大。

REINFORCE 算法

从 PG 定理可以推断出梯度的形式

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}\left[\sum_a q_\pi(S_t, a)\nabla\pi(a \mid S_t, \boldsymbol{\theta})\right] \tag{6}$$

经过变形，可以写成与 Q 函数无关的形式

$$\begin{aligned}\nabla J(\boldsymbol{\theta}) &= \mathbb{E}_\pi\left[\sum_a \pi(a \mid S_t, \boldsymbol{\theta}) q_\pi(S_t, a) \frac{\nabla\pi(a \mid S_t, \boldsymbol{\theta})}{\pi(a \mid S_t, \boldsymbol{\theta})}\right] \\ &= \mathbb{E}_\pi\left[q_\pi(S_t, A_t)\frac{\nabla\pi(a \mid S_t, \boldsymbol{\theta})}{\pi(a \mid S_t, \boldsymbol{\theta})}\right] \\ &= \mathbb{E}_\pi\left[G_t\frac{\nabla\pi(a \mid S_t, \boldsymbol{\theta})}{\pi(a \mid S_t, \boldsymbol{\theta})}\right]\end{aligned} \tag{7}$$

REINFORCE 算法

应用 SGD，参数按下式更新

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha G_t \nabla \log \pi(A_t \mid S_t, \boldsymbol{\theta}_t) \tag{8}$$

完整算法如下

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$
Repeat forever:
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
    For each step of the episode $t = 0, \ldots, T-1$:
        $G \leftarrow$ return from step t
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t, \boldsymbol{\theta})$

图 10: Alg: REINFORCE

REINFORCE with Baseline

这是 REINFORCE 算法的一个推论，设 $b(s)$（与 a 无关）是一个 baseline，将其放入 PG 定理

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a (q_\pi(s,a) - b(s)) \nabla \pi(a \mid s, \boldsymbol{\theta}) \tag{9}$$

可以发现这个梯度与原梯度等价，因为

$$\sum_a b(s) \nabla \pi(a \mid s, \boldsymbol{\theta}) = b(s) \nabla \sum_a \pi(a \mid s, \boldsymbol{\theta}) = b(s) \nabla 1 = 0 \tag{10}$$

自然的选择是将 $\hat{v}(s, \boldsymbol{w})$ 作为 baseline。这将极大地减小训练时的方差。

Actor-Critic 方法

若将带 baseline 的 REINFORCE 算法中的 G_t 换成 $G_{t:t+1}$，这就成为了 AC 方法。AC 方法中，状态值函数不只是作为 baseline 出现，而是结合了单步回报用以评判策略的好坏。AC 方法的更新式为

$$\begin{aligned}\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha(G_{t:t+1} - \hat{v}(S_t, \boldsymbol{w}))\nabla \log \pi(A_t \mid S_t, \boldsymbol{\theta}_t) \\ &= \boldsymbol{\theta}_t + \alpha(R_{t+1} + \gamma\hat{v}(S_{t+1}, \boldsymbol{w}) - \hat{v}(S_t, \boldsymbol{w}))\nabla \log \pi(A_t \mid S_t, \boldsymbol{\theta}_t)\end{aligned} \tag{11}$$

一般记 $\delta_t = R_{t+1} + \gamma\hat{v}(S_{t+1}, \boldsymbol{w}) - \hat{v}(S_t, \boldsymbol{w})$

Actor-Critic 方法

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$

Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$
Repeat forever:
    Initialize S (first state of episode)
    $I \leftarrow 1$
    While S is not terminal:
        $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
        Take action A, observe S', R
        $\delta \leftarrow R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} I \delta \nabla_{\mathbf{w}} \hat{v}(S,\mathbf{w})$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla_{\boldsymbol{\theta}} \ln \pi(A|S,\boldsymbol{\theta})$
        $I \leftarrow \gamma I$
        $S \leftarrow S'$

图 11: Alg: One-step Actor-Critic

参数化方法总结

上面介绍了三大类算法

- 只参数化值函数：梯度 Sarsa
- 只参数化策略：REINFORCE
- 策略和值函数都参数化：Actor-Critic 算法

参数化最为流行的做法是应用人工神经网络。在此基础上发展出了一大批优秀的算法。

AlphaGo 原理剖析

问题特点

围棋是目前最为复杂的棋类游戏，历史悠久，曾被认为是人类智慧的制高点

- 规则简单
- 状态空间庞大

1997 年“深蓝”战胜人类国际象棋世界冠军卡斯帕罗夫，但在围棋领域，AI 在 2015 年以前一直没有取得太大进步。

从 2015 年起，AlphaGo 的出现让围棋 AI 取得历史性突破

- 2015 年 AlphaGo 战胜樊麾二段，次年在 Nature 发布论文
- 2016 年 AlphaGo 击败顶尖棋手李世乭
- 2017 年 AlphaGo 击败世界第一棋手柯洁

蒙特卡洛树搜索

蒙特卡洛树搜索（MCTS）4 步骤

- **选择**：从根节点开始，利用树策略（如 $\epsilon - greedy$ 或 UCB）选择动作
- **展开**：当动作选择达到了某一个叶节点时，将叶节点展开
- **模拟**：采用 rollout 策略一直进行下去，直到终结状态，获得回报
- **备份**：将回报值往前传，更新每一条边的 $Q(s, a)$

蒙特卡洛树搜索

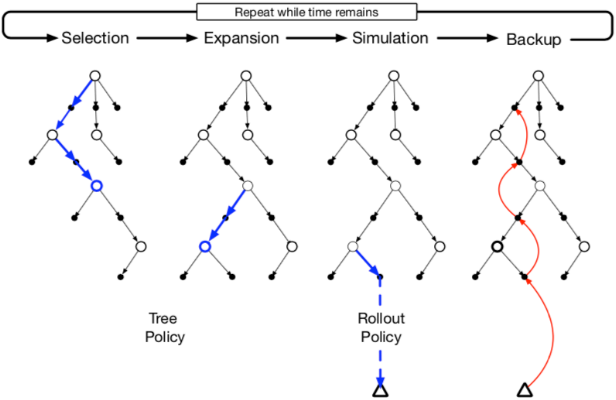


图 12: MCTS 图解

AlphaGo 完整算法

第一步：学习人类下棋策略

利用 KGS 围棋服务器的 3000 万样本训练了一个模仿人类落子偏好的策略 $\pi_\sigma(a \mid s)$，σ 是卷积神经网络的参数集合。这是一个监督学习问题，根据策略的似然度来执行 SGD

$$\Delta\sigma \propto \frac{\partial \log \pi_\sigma(a \mid s)}{\partial \sigma} \tag{12}$$

同时还训练了一个体量更小，运算速度更快的 rollout 策略 $\pi_w(a \mid s)$。

AlphaGo 完整算法

第二步：自我对战强化策略

有了初始的下棋策略，就可以自我对战一步步改进。此时策略记为 $\pi_\rho(a \mid s)$，网络结构与 π_σ 相同，并且参数初始化为 $\rho = \sigma$。每次对战时，对手采用从以前的训练过程中随机抽取的策略。当棋局结束时，依据胜负情况给出回报 $r(s_T) = \pm 1$，然后令 $z_t = r(s_T), t < T$，更新时采用下式

$$\Delta\rho \propto \frac{\partial \log \pi_\rho(a_t \mid s_t)}{\partial \rho} z_t \tag{13}$$

这一步用到了哪种 RL 算法?

AlphaGo 完整算法

第三步： 训练状态值函数

这一步目的是估计 $v_\pi(s)$，同样利用神经网络 $v_{theta}(s)$，网络结构与 π_σ 类似，只是最后几层结构有区别。不同于以往的强化学习算法，这里采用了监督学习的回归方法，即等到所有的自我对战结束之后，再来学习 s 到 z 的映射。

为了最小化 MSE，参数更新采用如下规则

$$\Delta\theta \propto \frac{\partial v_\theta(s)}{\partial\theta}(z - v_\theta(s)) \tag{14}$$

AlphaGo 完整算法

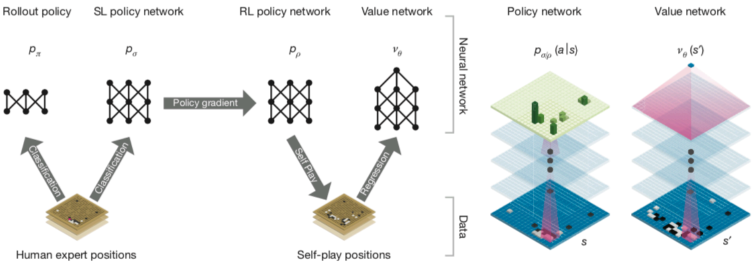


图 13: AlphaGo 前三步

AlphaGo 完整算法

第四步：MCTS

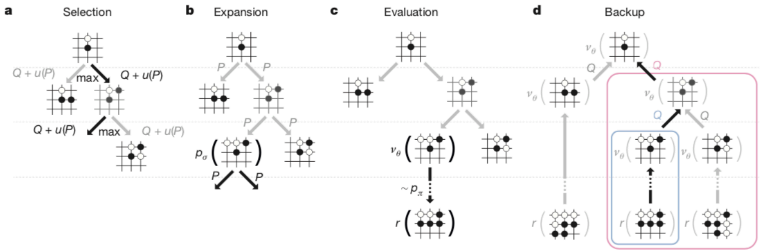


图 14: AlphaGo 的 MCTS 图解

在路径选择时，动作的选取既要考虑 Q 值，又要考虑探索

$$a_t = \arg\max_a (Q(s_t, a) + u(s_t, a)) \tag{15}$$

AlphaGo 完整算法

其中

$$u(s,a) \propto \frac{P(s,a)}{1+N(s,a)} \tag{16}$$

$P(s,a)$ 是先验概率比如 $\pi_\rho(a \mid s)$，$N(s,a)$ 是这条边的访问计数。

Backup 的过程中，首先依据状态值函数和 rollout 的结果确定叶节点的值

$$V(s_L) = (1-\lambda)v_\theta(s_L) + \lambda z_L \tag{17}$$

然后回溯更新每条边的 Q 值

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{n} 1(s,a,i) V(s_L^i) \tag{18}$$

$1(s,a,i)$ 表示第 i 次模拟是否访问到了 (s,a)。MCTS 过程结束后，agent 将选择最常访问的 a。

常用 RL 环境

Gym

除去一些特殊问题需要自建环境，目前科研领域常用的 RL 环境有

- MUJUCO——连续动作，简单模型
- 经典控制模型
- Box2D——二维图像，连续动作
- Atari——离散动作

所有常用环境可以在 OpenAI 的Gym里找到。