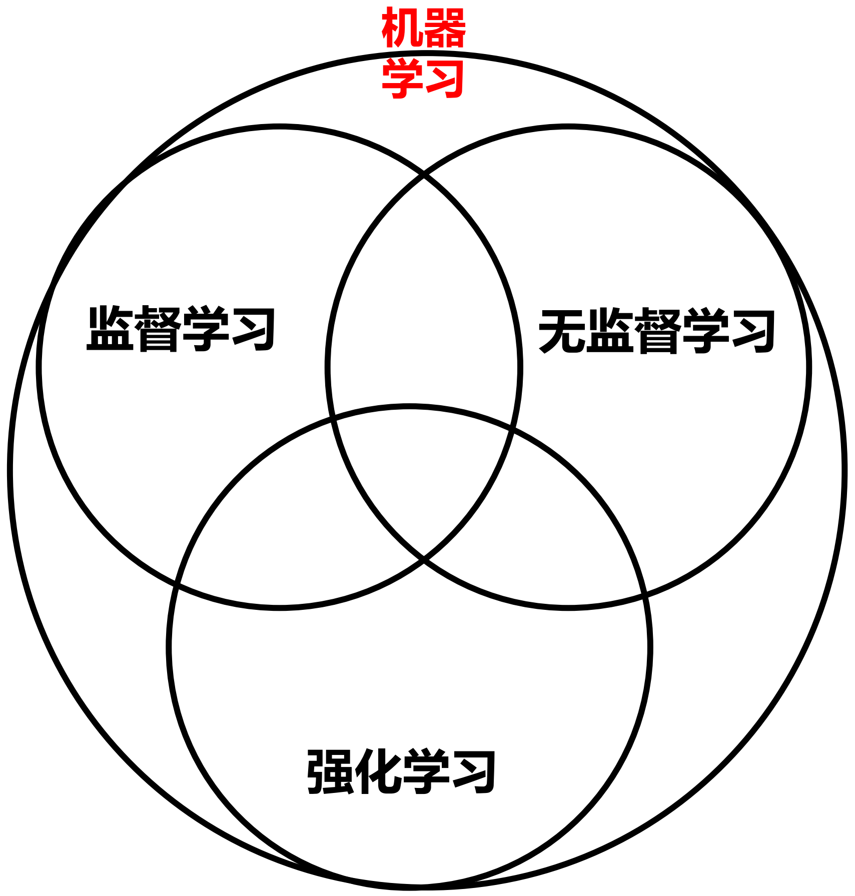
Reinforcement Learning

- 背景介绍
- 马 尔 科 夫 决 策 过 程
- 改 进 算 法

背景介绍

Introduction to RL



□ 预测

- ➢ 监督学习：根据数据预测所需输出
  - 数据独立同分布
  - 有正确的标签
- ➢ 无监督学习：生成数据实例
  - 数据独立同分布
  - 只有数据的特征，无标签

□ 决策

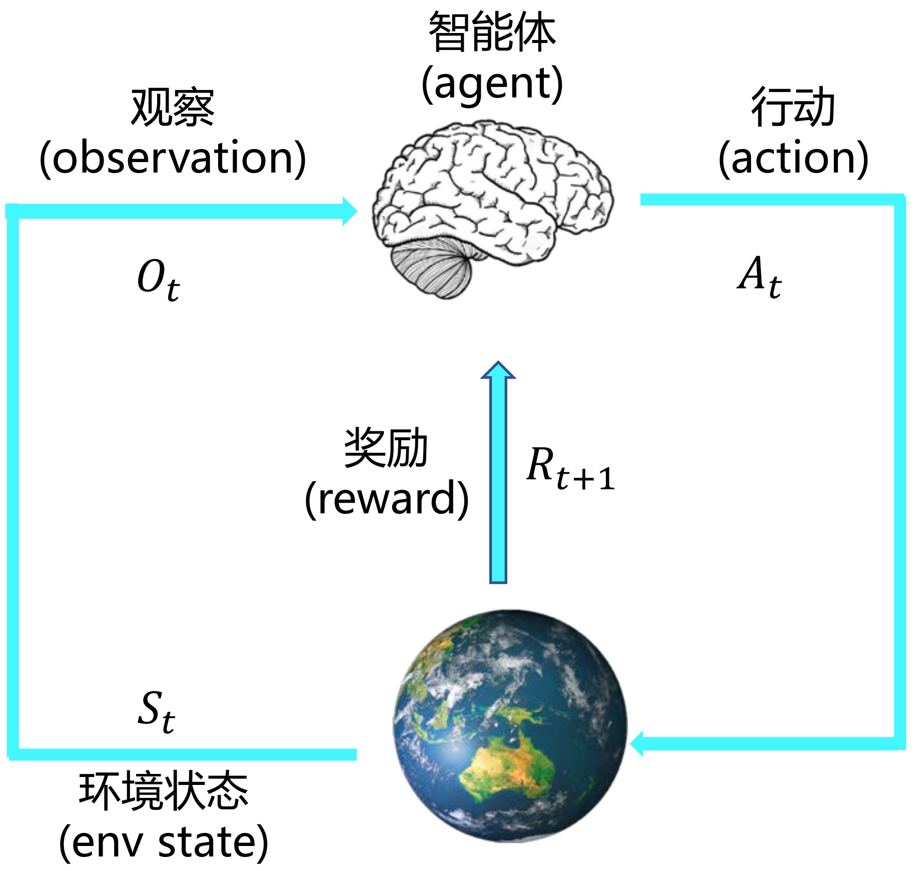
- ➢ 强化学习：在动态环境中采取行动
  - 数据是时序相关的，不 i.i.d
  - 没有正确的标签，只有每一步的 reward

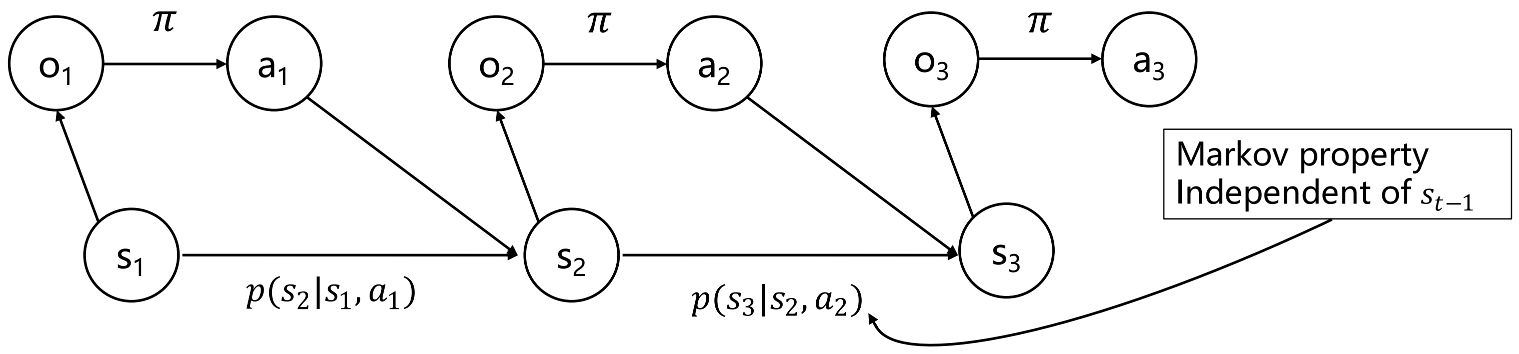
Introduction to RL

a computational approach to learning whereby an agent tries to maximize the total amount of reward it receives while interacting with a complex and uncertain environment.

---Sutton and Barto

通过从交互学习来实现目标的计算方法 (Learning from interactions with environment)




o_1 $\xrightarrow{\pi}$ a_1
o_2 $\xrightarrow{\pi}$ a_2
o_3 $\xrightarrow{\pi}$ a_3
s_1
s_2
s_3
$p(s_2|s_1,a_1)$
$p(s_3|s_2,a_2)$
Markov property
Independent of s_{t-1}

Full observability: $O_t = S_t$

Introduction to RL

- **历史 (History)** the sequence of observations, actions, rewards, which is used to construct the agent state

$$H_t = O_0, A_0, R_1, O_1, A_1, R_2, \dots, \; O_{t-1}, A_{t-1}, R_t, O_t$$

- **智能体状态 (Agent State)** a function of History

$$S_t = f(H_t) \begin{cases} S_t = O_t & (might\ not\ be\ enough) \\ S_t = H_t & (might\ be\ too\ large) \\ S_t = u(S_{t-1}, A_{t-1}, R_t, O_t) & (how\ to\ pick\ or\ learn\ u) \end{cases}$$

- **策略 (Policy)** a map function from state to action
  - 确定性策略 (Deterministic policy)

$$a = \pi(s)$$

  - 随机策略 (Stochastic policy)

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

- **奖励 (Reward)** a scalar feedback signal, Indicate how well agent is doing at step t

Introduction to RL

- **价值函数 (Value function)** expected discounted sum of future rewards under a particular policy π

  - 状态价值函数（state value function）

$$v^{\pi}(s) = E_{\pi}[G_t|S_t = s] = E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right]$$

  - 状态-动作价值函数（state-action value function）

$$q^{\pi}(s,a) = E_{\pi}[G_t|S_t = s, A_t = a] = E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a\right]$$

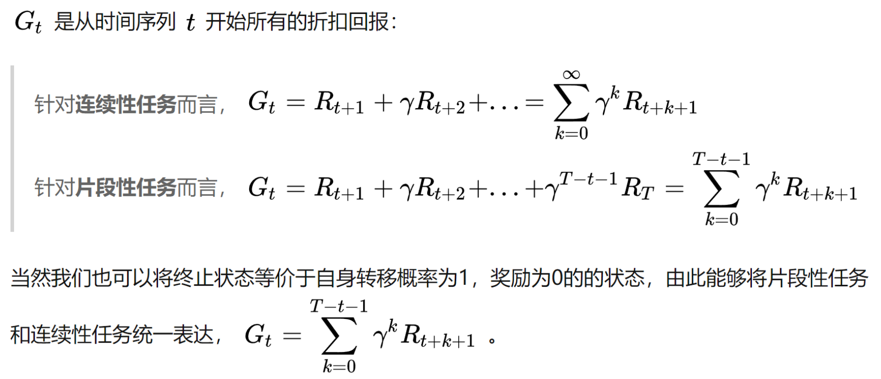
- **环境的模型 (Model)** agent`s state representation of the environment

  - Predict the next state

$$\mathrm{P}_{ss'}^{a} = P(S_{t+1} = s'|S_t = s, A_t = a)$$

  - Predict the next reward

$$R_s^a = R(R_{t+1}|S_t = s, A_t = a)$$



G_t 是从时间序列 t 开始所有的折扣回报：

> 针对**连续性任务**而言， $G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$
>
> 针对**片段性任务**而言， $G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-t-1} R_T = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$

当然我们也可以将终止状态等价于自身转移概率为1，奖励为0的的状态，由此能够将片段性任务和连续性任务统一表达， $G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$ 。

预测和控制 Prediction & Control

在强化学习里，我们经常需要先解决关于预测（prediction）的问题，而后在此基础上解决关于控制（Control）的问题。

- 预测：给定一个策略，评价未来。可以看成是求解在给定策略下的价值函数（value function）的过程。How well will I (an agent) do if I (the agent) follow a specific policy?
- 控制：找到一个好的策略来最大化未来的奖励。

马尔科夫决策过程

马尔可夫决策过程（Markov Decision Processes，MDPs）是对强化学习问题的数学描述，要求环境是全观测的。

几乎所有的RL问题都能用MDPs来描述：

> **最优控制问题**可以描述成**连续MDPs**；
> **部分观测环境**可以转化成**POMDPs**；
> 赌博机问题是只有一个状态的MDPs。

Finite Markov Decision Process

The history of states: $h_t = \{s_1, s_2, \ldots, s_t\}$

State s_t is Markovian if and only if:

$$p(s_{t+1}|h_t) = p(s_{t+1}|s_t)$$

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

Finite Markov Decision Process

MDP can be represented as a tuple: (S, A, P, R, γ)

1. S is a set of states
2. A is a set of actions
3. P is transition model: $P(\mathrm{S}_{t+1} = s' | S_t = s, A_t = a)$
4. R is reward function $R(s, a) = E[R_{t+1} | S_t = s, A_t = a]$
5. γ is discount factor $\gamma \in [0,1]$

Markov Decision Process

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

- 状态价值函数（state value function）

$$v^\pi(s) = E_\pi[G_t|S_t = s] = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots |S_t = s]$$

- 状态-动作价值函数（state-action value function）

$$q^\pi(s,a) = E_\pi[G_t|S_t = s, A_t = a] = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots |S_t = s, A_t = a]$$

- Relation between $v^\pi(s)$ and $q^\pi(s,a)$

$$v^\pi(s) = \sum_{a\in A} \pi(a|s) q^\pi(s,a)$$

$$q^\pi(s,a) = R(s,a) + \gamma \sum_{s'\in S} P(s'|s,a) v^\pi(s')$$

Decision Making in MDP

Prediction

- evaluate a given policy
- Input a MDP and a policy π, output value function $v^{\pi}(s)$

Control

- search for the optimal policy
- Input a MDP, output optimal value function v^* and optimal policy π^*

Prediction in MDP

用贝尔曼期望方程求解Prediction问题

$$v = \mathcal{R} + \gamma \mathcal{P} v$$

假设状态集合为 $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ ，那么：

$$\begin{bmatrix} v(s_1) \\ \vdots \\ v(s_n) \end{bmatrix} = \begin{bmatrix} \mathcal{R}_{s_1} \\ \vdots \\ \mathcal{R}_{s_n} \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{s_1 s_1} & \cdots & \mathcal{P}_{s_1 s_n} \\ \vdots & & \vdots \\ \mathcal{P}_{s_n s_1} & \cdots & \mathcal{P}_{s_n s_n} \end{bmatrix} \begin{bmatrix} v(s_1) \\ \vdots \\ v(s_n) \end{bmatrix}$$

贝尔曼方程本质上是一个**线性方程**，可以直接解：

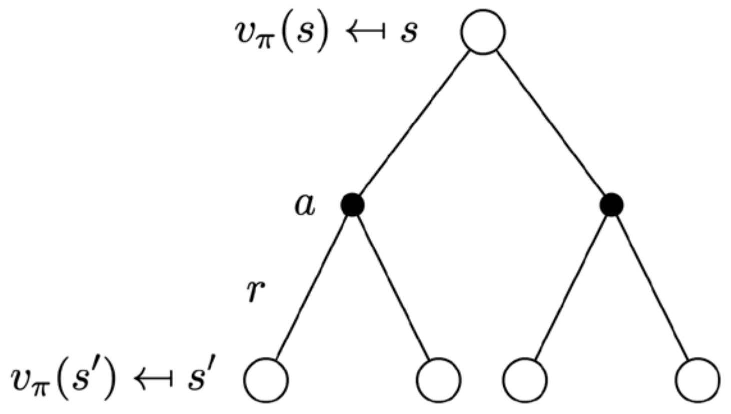
$$(1 - \gamma \mathcal{P}) v = \mathcal{R}$$

$$v = (1 - \gamma \mathcal{P})^{-1} \mathcal{R}$$

实际上，**计算复杂度是** $O(n^3)$ ， n 是状态数量。因此直接求解仅适用于小规模的MRPs。大规模MRP的求解通常使用迭代法。常用的迭代方法有：动态规划（Dynamic Programming, DP）、蒙特卡洛评估（Monte-Carlo evaluation）、时序差分学习（Temporal-Difference, TD），后文会逐步讲解这些方法。

Bellman Expectation Equation

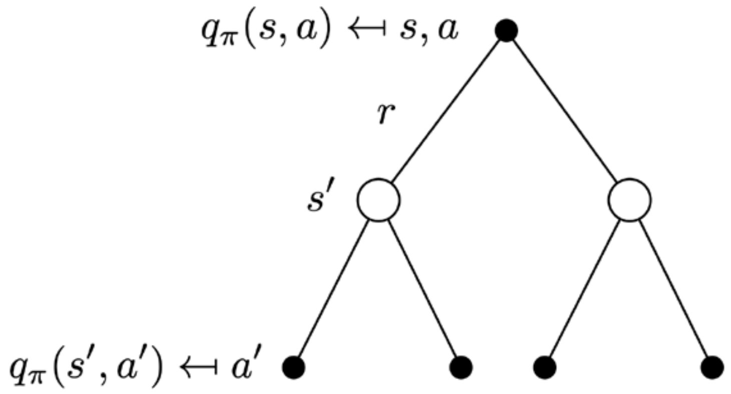
$$
\begin{aligned}
v^{\pi}(s) &= E_{\pi}[G_t|S_t = s] = E_{\pi}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots |S_t = s] \\
&= E_{\pi}[R_{t+1} + \gamma G_{t+1}|S_t = s] \\
&= E_{\pi}[R_{t+1} + \gamma v^{\pi}(S_{t+1})|S_t = s] \\
&= \sum_{a\in A} \pi(a|s) \left(R(s,a) + \gamma \sum_{s'\in S} P(s'|s,a) v^{\pi}(s') \right)
\end{aligned}
$$



$v_\pi(s) \leftarrow\!\shortmid s$
a
r
$v_\pi(s') \leftarrow\!\shortmid s'$

The action-value function can similarly be decomposed

$$
\begin{aligned}
q^{\pi}(s,a) &= E_{\pi}[R_{t+1} + \gamma q^{\pi}(S_{t+1}, A_{t+1})|S_t = s, A_t = a] \\
&= R(s,a) + \gamma \sum_{s'\in S} P(s'|s,a) \sum_{a'\in A} \pi(a'|s') q^{\pi}(s',a')
\end{aligned}
$$



$q_\pi(s,a) \leftarrow\!\shortmid s,a$
r
s'
$q_\pi(s',a') \leftarrow\!\shortmid a'$

Control in MDP

用贝尔曼最优方程求解Control问题 Find Optimal Policy π

The optimal value function

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

The optimal policy

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

An optimal policy can be found by maximizing over $q^*(s, a)$

$$\pi^*(a|s) = \begin{cases} 1, & if\ a = \arg\max_{a \in A} q^*(s, a) \\ 0, & otherwise \end{cases}$$ <—最优的策略

Noting that $q^*(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) v^*(s')$

Bellman Optimality Equation

If improvements stops, we have

$$q^{\pi}(s, \pi'(s)) = \max_{a \in A} q^{\pi}(s, a) = q^{\pi}(s, \pi(s)) = v^{\pi}(s)$$

Thus we get Bellman optimality equations:

$$v^*(s) = \max_a q^*(s, a)$$

$$q^*(s, a) = R(s, a) + \gamma \textstyle\sum_{s' \in S} P(s'|s, a) v^*(s')$$

then

$$v^*(s) = \max_a \left(R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) v^*(s') \right)$$

$$q^*(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) \max_{a'} q^*(s', a')$$

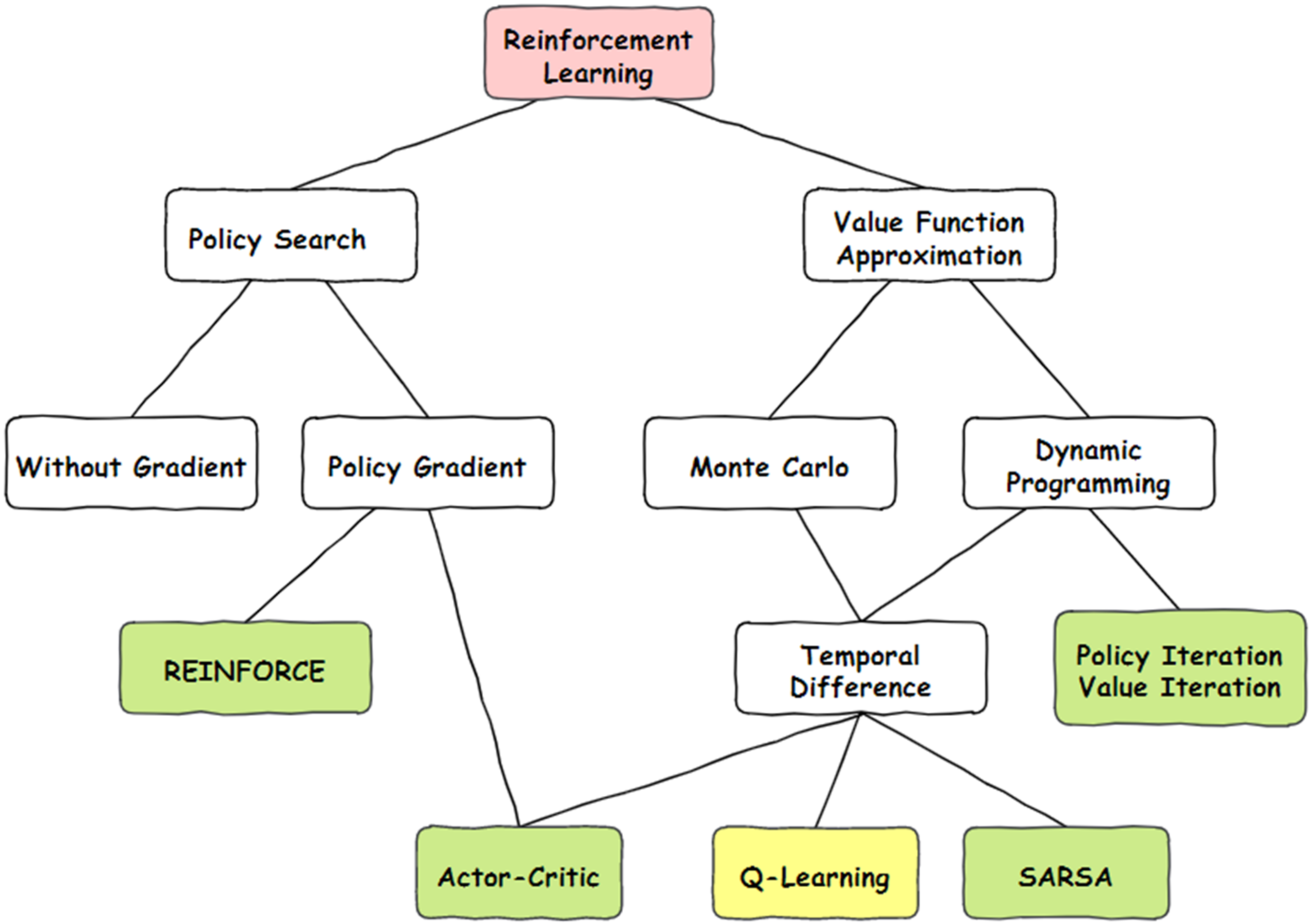
2.4.13 求解Bellman最优方程

Bellman最优方程是非线性的，没有固定的解决方案，通过一些迭代方法来解决：价值迭代、策略迭代、Q学习、Sarsa等。后续会逐步讲解展开。

- **Bellman最优方程与Bellman方程的关系**

1. 贝尔曼最优方程本质上就是利用了 π_* 的特点，将求期望的算子转化成了 $\max\limits_a$
2. 在贝尔曼期望方程中， π 是已知的，而在贝尔曼最优方程中， π_* 是未知的
3. 解贝尔曼期望方程的过程即对应了评价，解贝尔曼最优方程的过程即对应了优化

改进算法


Reinforcement Learning
Policy Search
Value Function Approximation
Without Gradient
Policy Gradient
Monte Carlo
Dynamic Programming
REINFORCE
Temporal Difference
Policy Iteration
Value Iteration
Actor-Critic
Q-Learning
SARSA

Model-based 情况下的Prediction 与 Control的改进

Dynamic Programming动态规划：

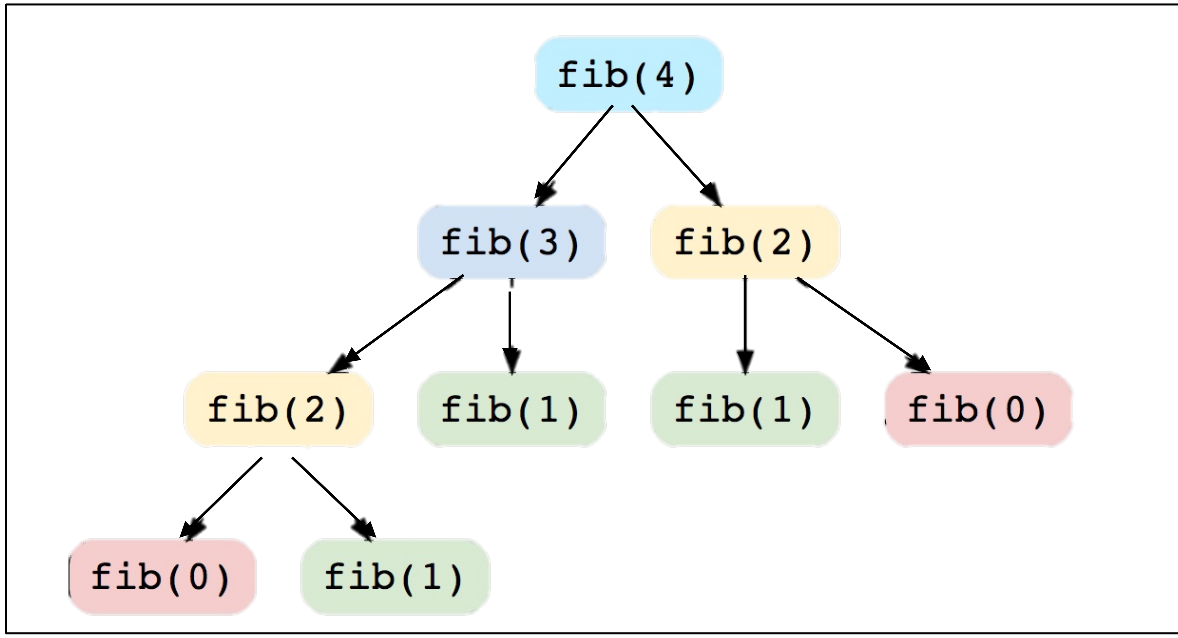
1-Prediction：减少计算开销

2-Control：迭代求解最优策略

Dynamic Programming

Characteristics of DP

1. **overlapping subproblems**
   - finding the solution involves solving the same subproblem multiple times
2. **optimal substructure property**
   - the overall optimal solution can be constructed from the optimal solutions of its subproblems



MDP satisfies both properties

Value Iteration for MDP Control

Iteration on the Bellman optimality backup

$$v(s) \leftarrow \max_{a \in A}\left(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)v\,(s')\right)$$

Algorithm:

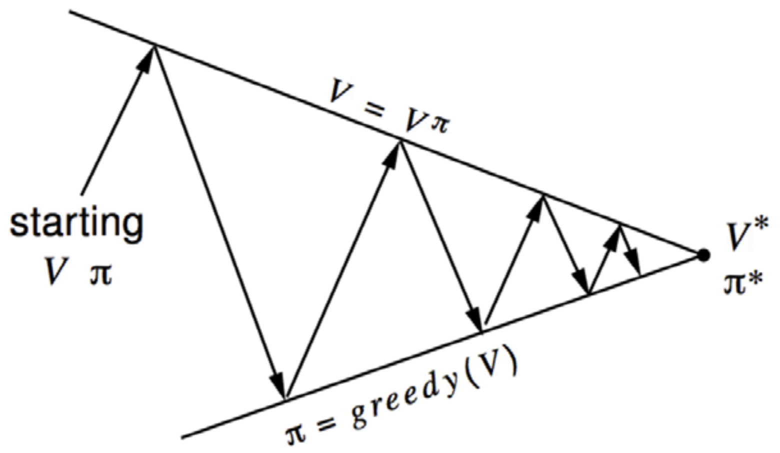
```
1. Initialize k=1 and $v_0(s) = 0$ for all states s
2. For k=1:K
    1. For each state s
        $q_{k+1}(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)v_k\,(s')$
        $v_{k+1}(s) = \max_a q_{k+1}(s,a)$
    2. k = k+1
3. To retrieve the optimal policy after the value iteration
```

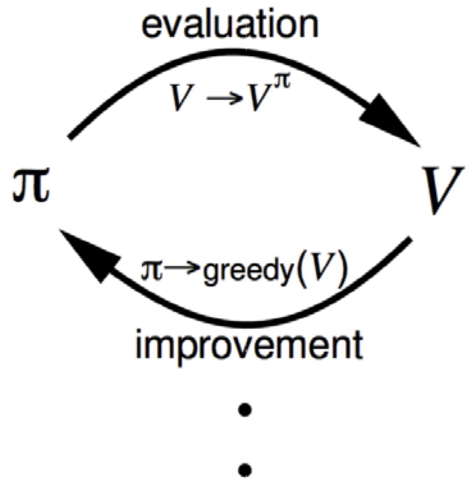
$$\pi(s) = \arg\max_{a \in A}\left(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)v_{k+1}\,(s')\right)$$

Policy Iteration for MDP Control

Iterate between the two steps:

1. Evaluate the policy π (compute v given current π)
2. Improve policy π by acting greedily with respect to v^{π}
    1. $q^{\pi_t}(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v^{\pi_t}(s')$
    2. $\pi_{t+1}(s) = \arg\max_{a} q^{\pi_t}(s,a)$





策略评价：求 v_π 。使用方法：迭代式策略评价

策略提升：提升策略 $\pi^{'} \geq \pi$ 。使用方法：贪婪策略提升

随机初始化一个 V 和 π ，通过策略评价求得当前策略下的 $V = V^\pi$ ，然后做一次贪婪的策略提升 $\pi = \text{greedy}(V)$ ，此时的 V 函数就不是基于策略 π 下的，再做一次策略评价，使得 V 函数与现在策略一致……如此反复多次，最终得到最优策略 π^* 和最优状态价值函数 V^* 。

本质上就是使用当前策略产生新的样本，然后使用新的样本更好的估计策略的价值，然后利用策略的价值更新策略，然后不断反复。理论可以证明最终策略将收敛到最优。

Model-free RL and Model-based RL

In many real world problems, we may not know the MDP model or it is just too big to use

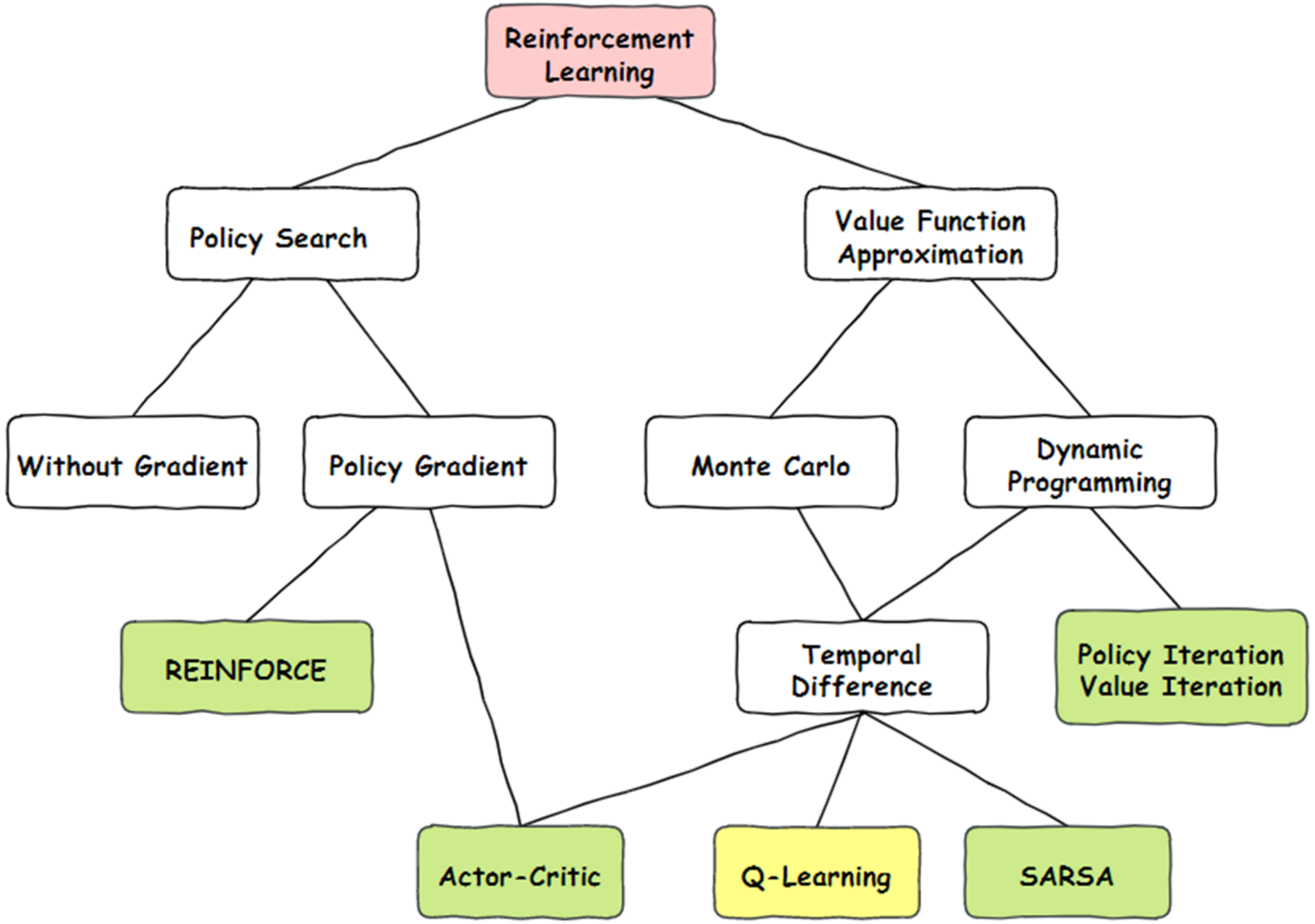
- Chess: 10^{47} states
- Game of Go: 10^{170} states
- Robot Arm: continuous state and action space

Model-free RL can solve the problems by letting the agent interact with the environment and collecting trajectories/episodes:

$$\{S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T, S_T\}$$

Two Model-free methods:

1. Monte Carlo method
2. Temporal Difference (TD) learning


Reinforcement Learning
Policy Search
Value Function Approximation
Without Gradient
Policy Gradient
Monte Carlo
Dynamic Programming
REINFORCE
Temporal Difference
Policy Iteration Value Iteration
Actor-Critic
Q-Learning
SARSA

model free情况下
用蒙特卡洛法进行Prediction

Monte Carlo Policy Evaluation

Recall:

- Agent interacts with the environment to collect many trajectories $\tau: \{S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T, S_T\}$ by following a policy π
- Return: $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \dots$
- We want to compute $v^\pi(s) = E_{\tau \sim \pi}[G_t | S_t = s]$

MC policy evaluation uses empirical mean return instead of expected return

Algorithm:

Every time step t that state s is visited in an trajectory:

$$N(s) \leftarrow N(s) + 1$$
$$R(s) \leftarrow R(s) + G_t$$
$$v(s) = \frac{R(s)}{N(s)}$$

By law of large numbers, $v(s) \to v^\pi(s) \;\; as \;\; N(s) \to \infty$

Incremental MC Update

Incremental mean: $\mu_n = \frac{1}{n}\sum_{i=1}^{n} x_i = \mu_{n-1} + \frac{1}{n}(x_n - \mu_{n-1})$

Algorithm:

```
1. For every trajectory $\{S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T, S_T\}$
    2. For each state $S_t$ with computed return $G_t$
```

$$N(S_t) \leftarrow N(S_t) + 1$$

$$v(S_t) \leftarrow v(S_t) + \frac{1}{N(S_t)}(G_t - v(S_t))$$

For non-stationary problems, we use

$$v(S_t) \leftarrow v(S_t) + \alpha\,(G_t - v(S_t))$$

MC does not require MDP model, no bootstrapping, and even does not require the state is Markovian, but it can only apply to trajectories that are complete and have terminal state.

model free情况下
用时序差分法进行Prediction

Temporal-Difference Learning

TD learns from incomplete episodes, by sampling and bootstrapping

Our goal is still learn v^{π} from experience under policy π

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

Instead of using G_t to update $v(S_t)$, TD(0) use $R_{t+1} + \gamma v(S_{t+1})$ which is called TD target

$\delta_t = R_{t+1} + \gamma v(S_{t+1}) - v(S_t)$ is called TD error

n-step TD prediction

$$\begin{cases} n = 1(TD(0)) & G_t^{(1)} = R_{t+1} + \gamma v(S_{t+1}) \\ n = 2 & G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 v(S_{t+2}) \\ & \vdots \\ n = \infty(MC) & G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-1} R_T \end{cases}$$

DP, MC and TD

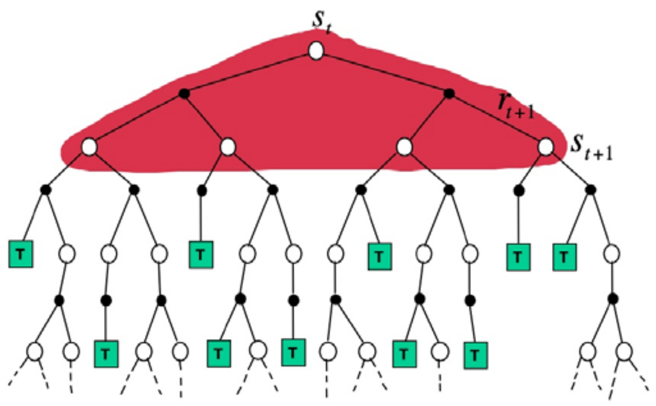
Bootstrapping

Update involves an estimate
- DP bootstraps
- MC does not bootstrap
- TD bootstraps

Sampling

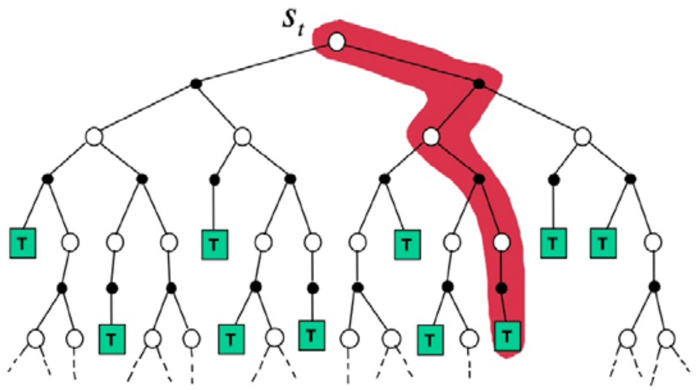
Update samples an expectation
- DP does not sample
- MC samples
- TD samples

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



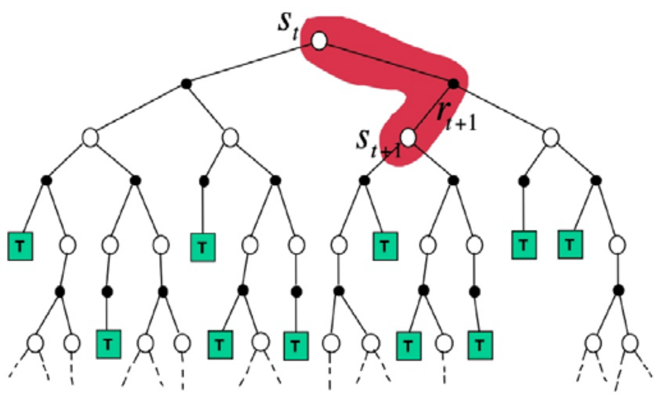
Dynamic Programming

$v(S_t) \leftarrow v(S_t) + \alpha(G_t - v(S_t))$



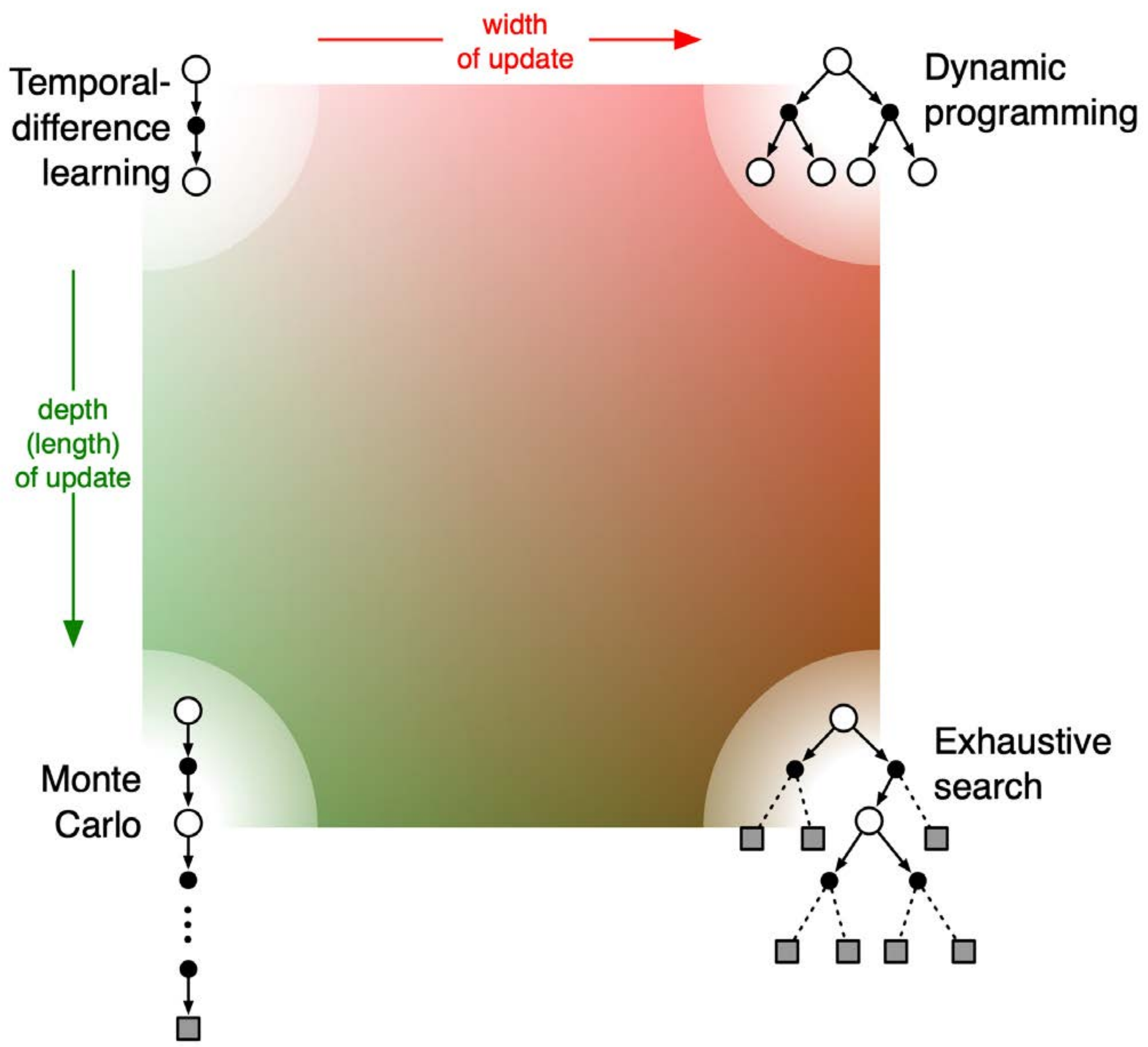
Monte Carlo

$TD(0) : v(S_t) \leftarrow v(S_t) + \alpha(R_{t+1} + \gamma v(s_{t+1}) - v(S_t))$



Temporal Difference Learning

Unified View of Reinforcement Learning



- **对比分析MC和TD的优缺点**

1. MC对比TD之一

> TD在知道结果之前可以学习，MC必须等到最后结果才能学习；
> TD可以在没有结果时学习，可以在持续进行的环境里学习；
> TD算法有多个驱动力——MC算法只有奖励值作为更新的驱动力，TD算法有奖励值和状态转移作为更新的驱动力。

2. MC对比TD之二

MC零偏差（bias）；高方差（variance）；收敛性较好（即使采用函数逼近）；对初始值不敏感；简单、容易理解和使用；随着样本数量的增加，方差逐渐减小，趋近于0。

TD有一些偏差；低方差；表格法下 $\mathrm{TD}\left(0\right)$ **收敛到** $v_\pi\left(s\right)$ **（函数逼近时不一定）；对初始值更敏感（用到了贝尔曼方程）；通常比MC更高效；随着样本数量的增加，偏差逐渐减少，趋近于0。**

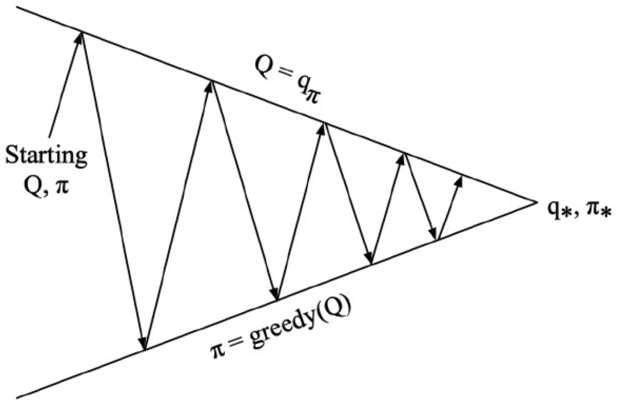
3. MC对比TD之三

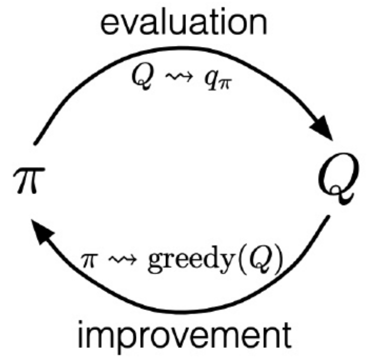
TD算法使用了MDP问题的马尔可夫属性，**在Markov环境下更有效**；但是MC算法并不利用马尔可夫属性，**通常在非Markov环境下更有效。**

model free情况下的Control

Model-free Control for MDP

Now we want to optimize the value function of a unknown MDP




evaluation
$Q \rightsquigarrow q_\pi$
π
Q
$\pi \rightsquigarrow \text{greedy}(Q)$
improvement

Policy Iteration for a known MDP

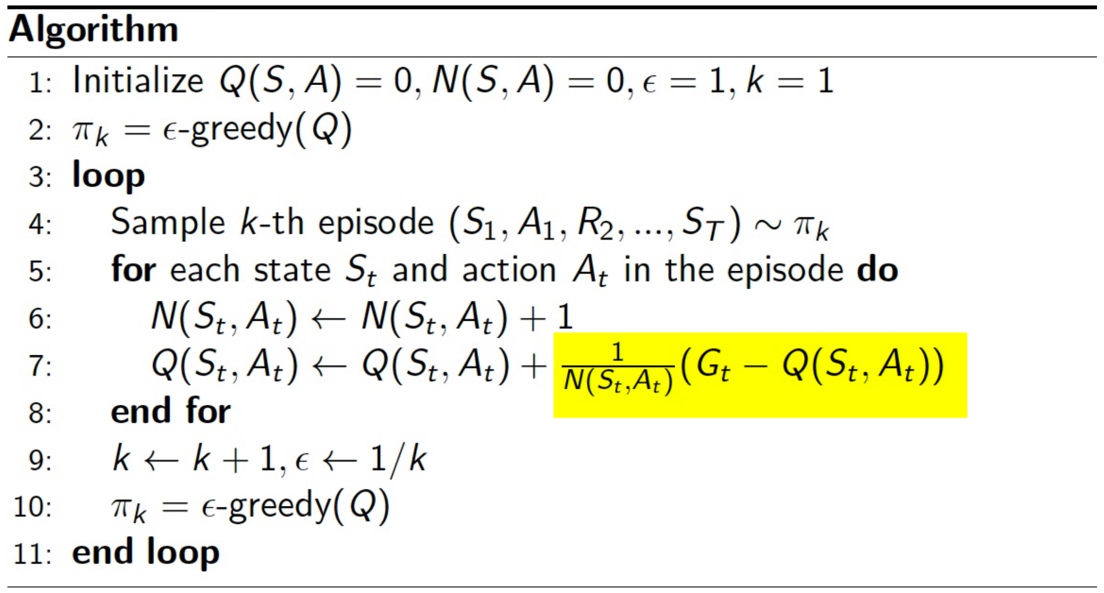
$$q^{\pi_t}(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v^{\pi_t}(s')$$
$$\pi_{t+1}(s) = \arg\max_a q^{\pi_t}(s,a)$$

use MC or TD
to do the policy evaluation

policy improvement is the same

MC with ϵ–Greedy Exploration

we need to trade off between exploration and exploitation



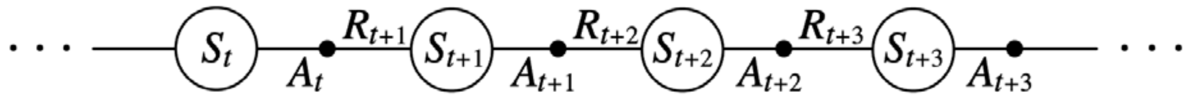
Algorithm

```
 1: Initialize $Q(S,A) = 0, N(S,A) = 0, \epsilon = 1, k = 1$
 2: $\pi_k = \epsilon$-greedy$(Q)$
 3: loop
 4:     Sample $k$-th episode $(S_1, A_1, R_2, ..., S_T) \sim \pi_k$
 5:     for each state $S_t$ and action $A_t$ in the episode do
 6:         $N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$
 7:         $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}(G_t - Q(S_t, A_t))$
 8:     end for
 9:     $k \leftarrow k + 1, \epsilon \leftarrow 1/k$
10:     $\pi_k = \epsilon$-greedy$(Q)$
11: end loop
```

ϵ–Greedy Exploration

$$\pi(a|s) = \begin{cases} \dfrac{\epsilon}{|A|} + 1 - \epsilon, & if\ a^* = \arg\max_{a \in A} q(s,a) \\ \dfrac{\epsilon}{|A|}, & otherwise \end{cases}$$

Sarsa: TD Control

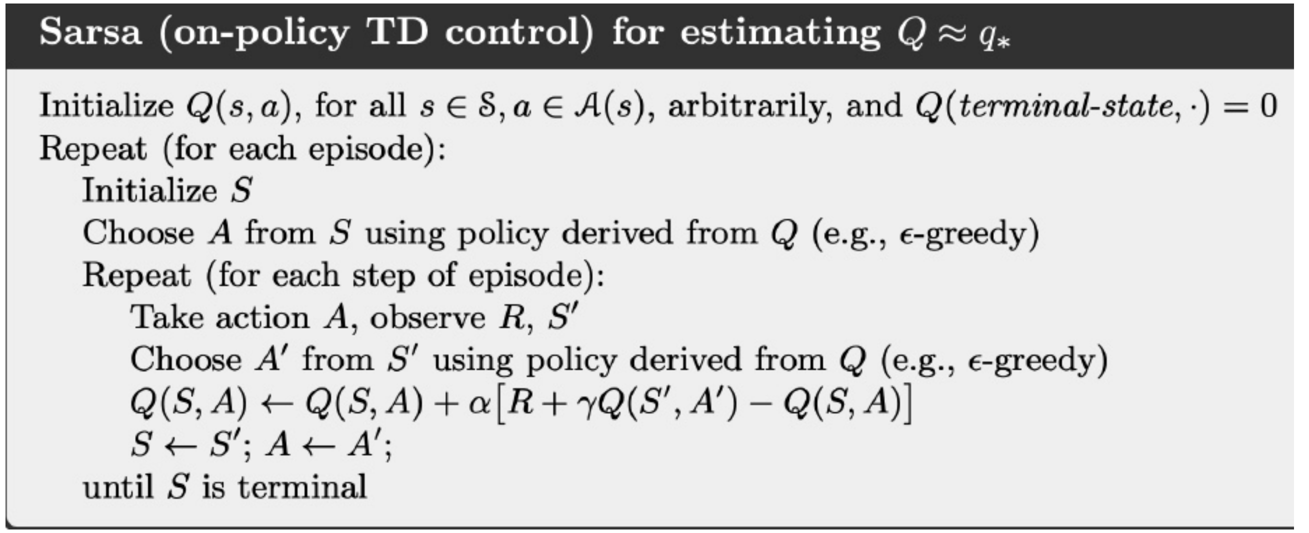


The update is done after every transition, no need to wait until the trajectory is finished.

ϵ-greedy policy for one step to choose action A', then bootstrap the action value function to do the update:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

we can also do n-step Sarsa



Sarsa (on-policy TD control) for estimating $Q \approx q_*$

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

On-policy vs. Off-policy

On-policy learning

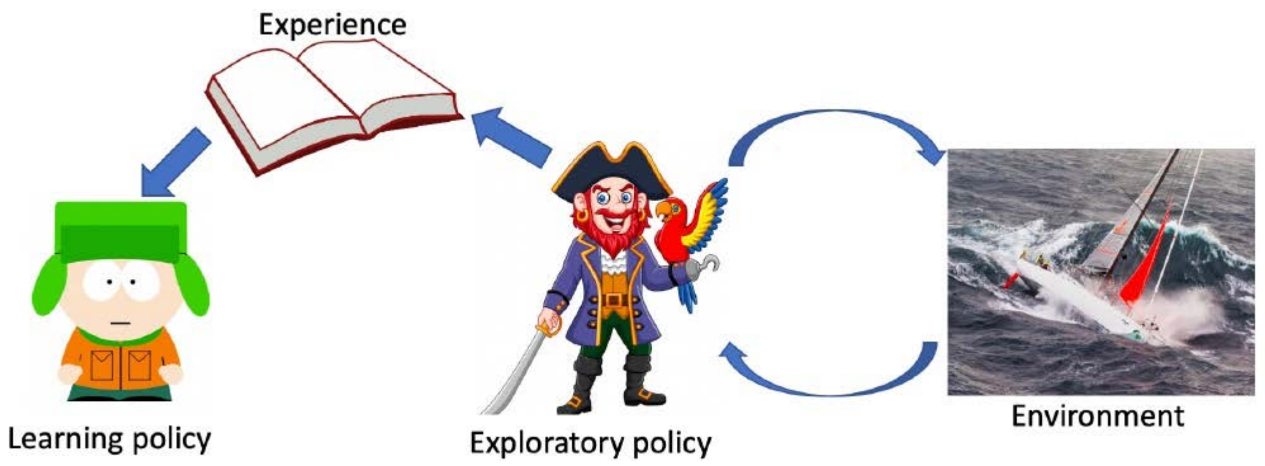
Learn about policy π from the experience collected from π

- Sarsa is on-policy learning

Off-policy learning

Learn about policy π from the experience collected from another policy μ

- target policy π can learn to become optimal
- behavior policy μ can be more exploratory



Q-learning: Off-policy Control

The target policy π is greedy on $Q(s,a)$

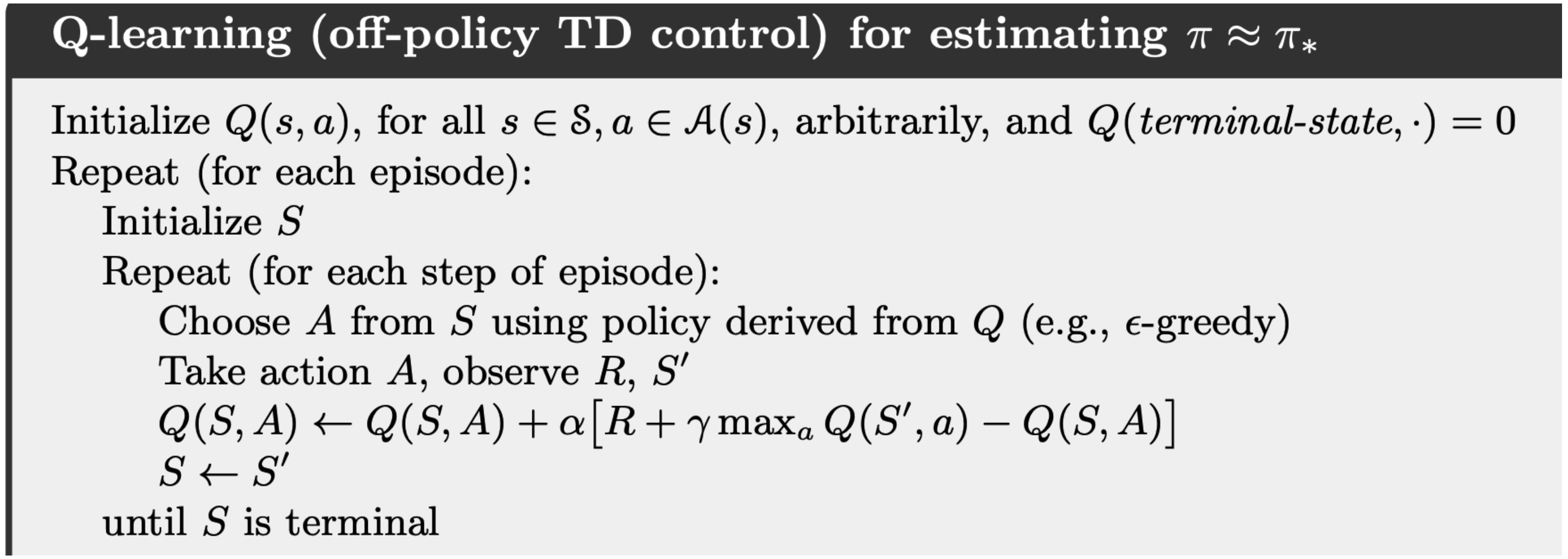
$$\pi(S_{t+1}) = \arg\max_{a'} Q(s_{t+1}, a')$$

Sarsa still use ϵ-greedy on $Q(s,a)$ here

The behavior policy μ could be totally random, but we let it improve by following ϵ-greedy on $Q(s,a)$

Thus the Q-learning update is:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$



Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

```
Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, 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., $\epsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
        $S \leftarrow S'$
    until $S$ is terminal
```

Comparison of Sarsa and Q-Learning

Sarsa: On-Policy TD control

Choose action A_t from S_t using policy derived from Q with ϵ-greedy

Take action A_t, observe R_{t+1} and S_{t+1}

Choose action A_{t+1} from S_{t+1} using policy derived from Q with ϵ-greedy

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

Q-Learning: Off-Policy TD control

Choose action A_t from S_t using policy derived from Q with ϵ-greedy

Take action A_t, observe R_{t+1} and S_{t+1}

Then `imagine' A_{t+1} as $\arg\max_{a'} Q(S_{t+1}, a')$ in the update target

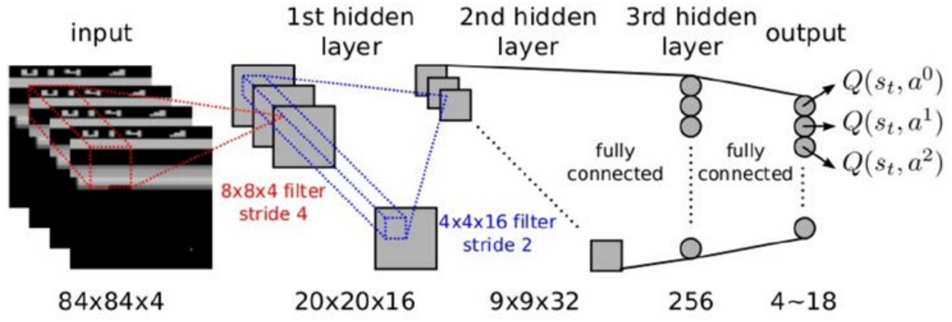
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$

Deep Q-Learning

Use network to approximate $Q^{\pi}(s,a)$

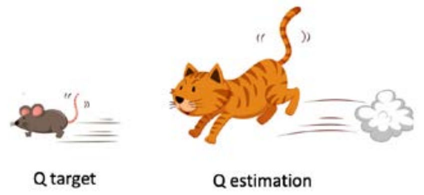
Two challenges:

- correlation between samples
- non-stationary targets



Solutions of Deep Q-Network (DQN):

- Experience Replay
- Fixed Q targets



s_1, a_1, r_1, s_2
s_2, a_2, r_2, s_3
s_3, a_3, r_3, s_4
...
s_t, a_t, r_t, s_{t+1}

Replay memory D

Sample a batch from D for training

Let a different set of parameter θ^{-} be the set of weights used in the target, and θ be the weights that are being updated (in practice, DQN uses a copy of θ as θ^{-} and freezes it for some training steps

$$\nabla_\theta J(\theta) = \left(R_{t+1} + \gamma \max_a \hat{Q}(s_{t+1}, a, \theta^{-}) - \hat{Q}(s_t, a_t, \theta) \right) \nabla_\theta \hat{Q}(s_t, a_t, \theta)$$

Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. nature, 2015, 518(7540): 529-533.

To Be Continued ...