Reinforcement Learning

- Introduction to RL
- Multi-armed Bandits
- Finite Markov Decision Process
- Model-free RL
- Function Approximation and DQN
- Policy-based RL
- Model-based RL

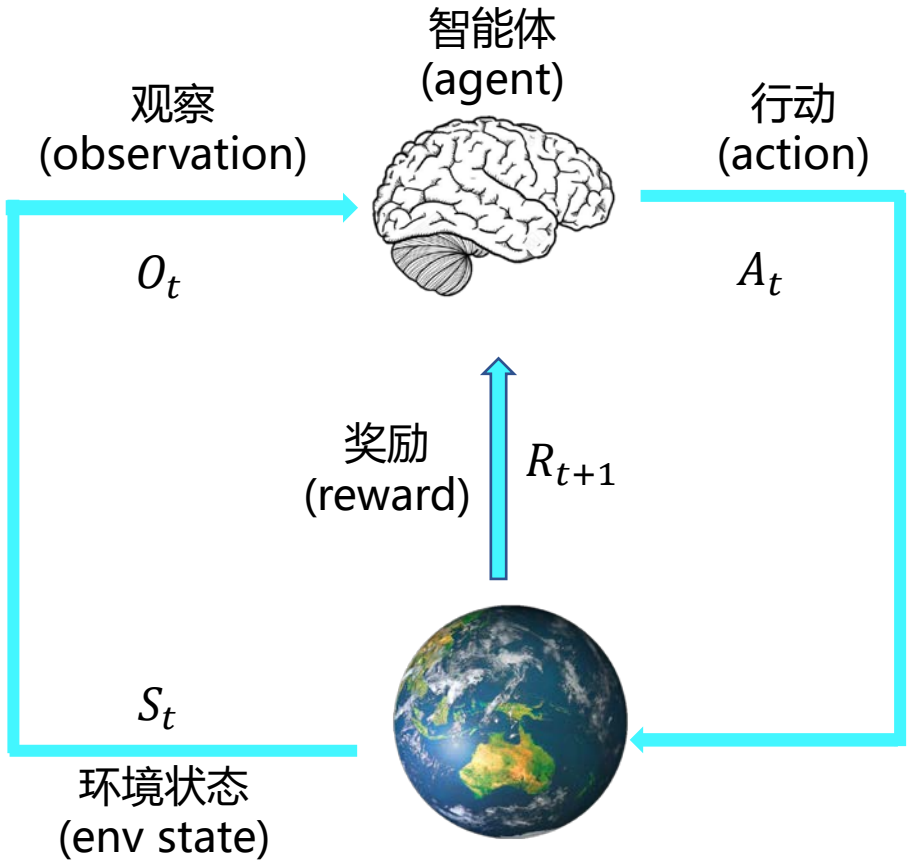
Reference:
https://github.com/zhoubolei/introRL
(2021 DeepMind x UCL)强化学习系列讲座
David Silver's slides, CS285 Reinforcement Learning
The Multi-Armed Bandit Problem and Its Solutions (lilianweng.github.io)
Reinforcement Learning: An Introduction(Richard S. Sutton and Andrew G. Barto)

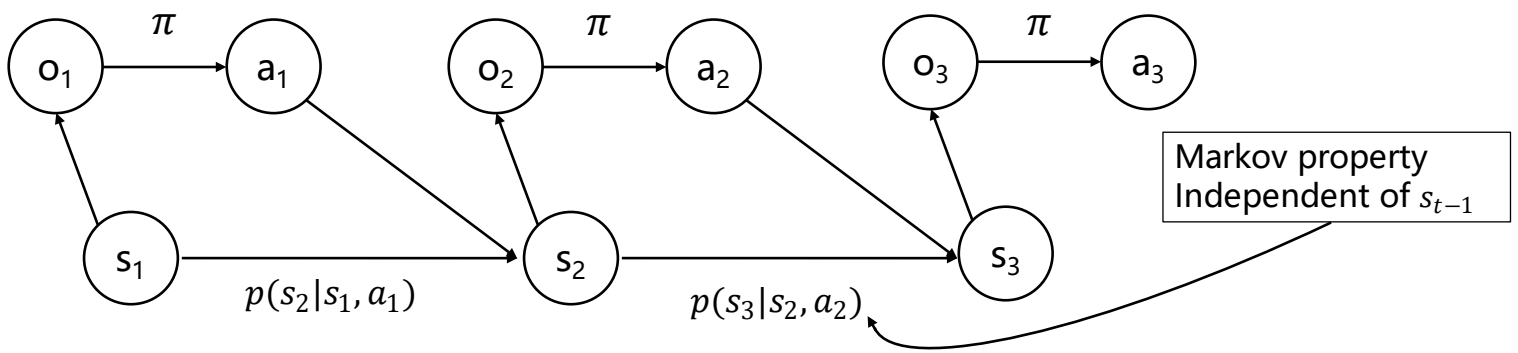
Introduction to RL

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)

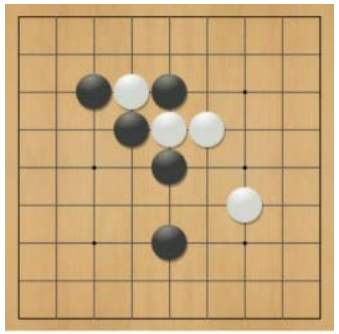




Full observability: $O_t = S_t$

Rules of Go

- Usually played on 19x19 board
- Simple rules, complex strategy
- Black and white place down stones alternately
- Surrounded stones are captured and removed
- The player with more territory wins the game



Introduction to RL

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

$$H_t = O_0, \mathrm{A}_0, R_1, O_1, A_1, R_2, \dots, \ O_{t-1}, A_{t-1}, R_t, O_t$$

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

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

- **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

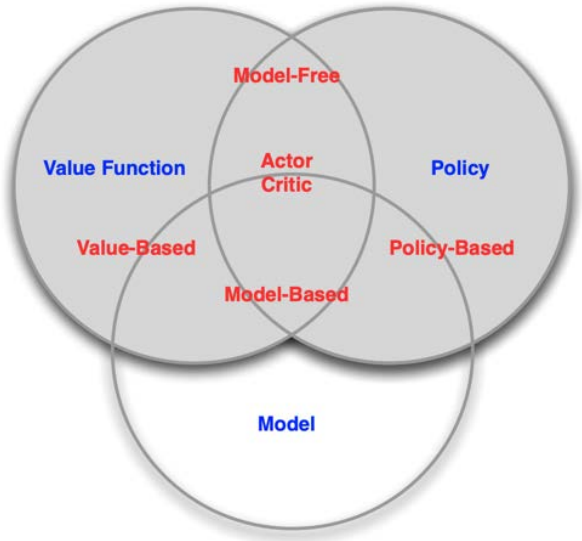
$$\mathrm{P}_{ss'}^{a} = P(S_{t+1} = s'|S_t = s, A_t = a)$$

  - Predict the next reward

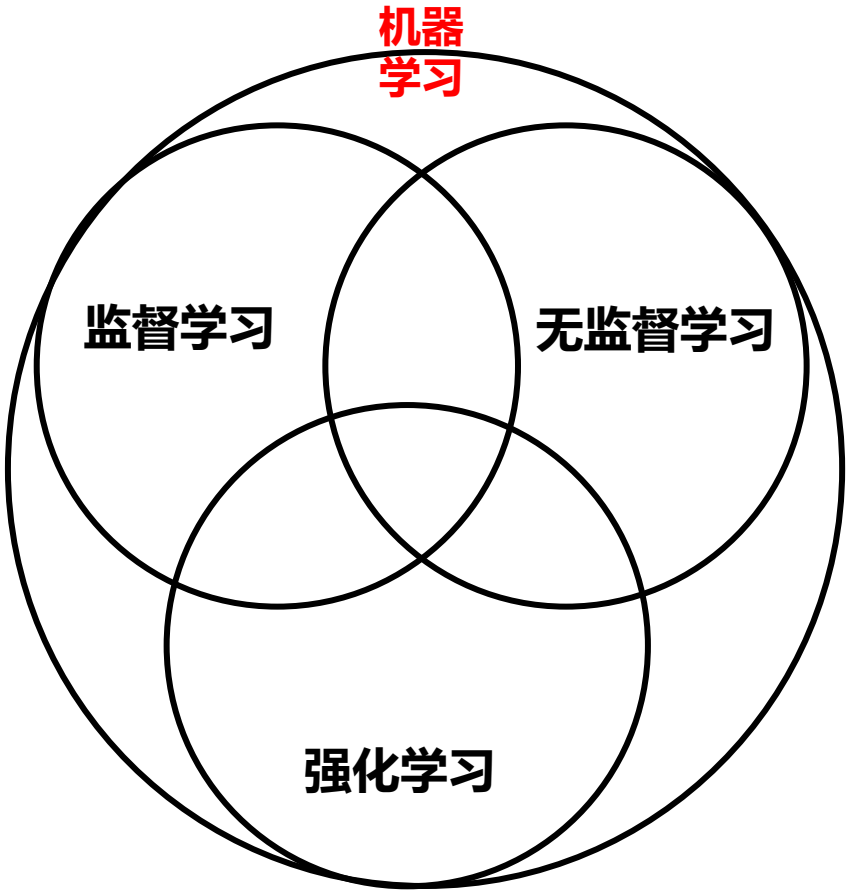
$$R_s^a = R(R_{t+1}|S_t = s, A_t = a)$$

Introduction to RL

- Value-based RL
  - Explicit: value function
  - Implicit: Policy (can derive a policy from value function)
- Policy-based RL
  - Explicit: policy
  - No value function
- Actor-Critic RL
  - Explicit: policy and value function
- Model-based RL
  - Explicit: model
  - May of may not have policy and/or value function
- Model-free RL
  - Explicit: value function and/or policy function
  - No model



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

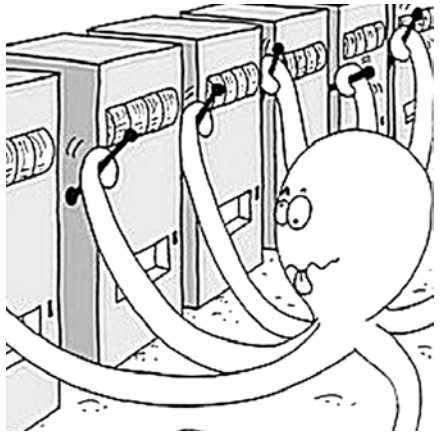
Multi-armed Bandits

a set of distributions $\{R_a | a \in A\}$

- A is a set of actions (or "arms")
- R_a is distribution on rewards given action a
- At each step t, the agent select an action $A_t \in A$,
- The environment generates a reward $R_t \sim R_{A_t}$

The goal is to maximize $E[\sum_{t=1}^{T} R_t]$

Exploration vs Exploitation



The environment is assumed to have only a single state/no state

- actions no longer have long-term consequences in the env
- actions still do impact immediate reward

The action value for action a is the expected reward: $q(a) = E[R_t | A_t = a]$

The optimal value is $v^* = \max_{a \in A} q(a)$

The optimal action $a^* \in \arg\max_{a \in A} q(a)$

Regret of an action a is $\Delta_a = v^* - q(a)$

Theorem (Lai and Robbins)

Asymptotic total regret is at least logarithmic in number of steps

$$\lim_{t \to \infty} L_t \geq \log t \sum_{a | \Delta_a > 0} \frac{\Delta_a}{KL(\mathcal{R}_a || \mathcal{R}_{a*})}$$

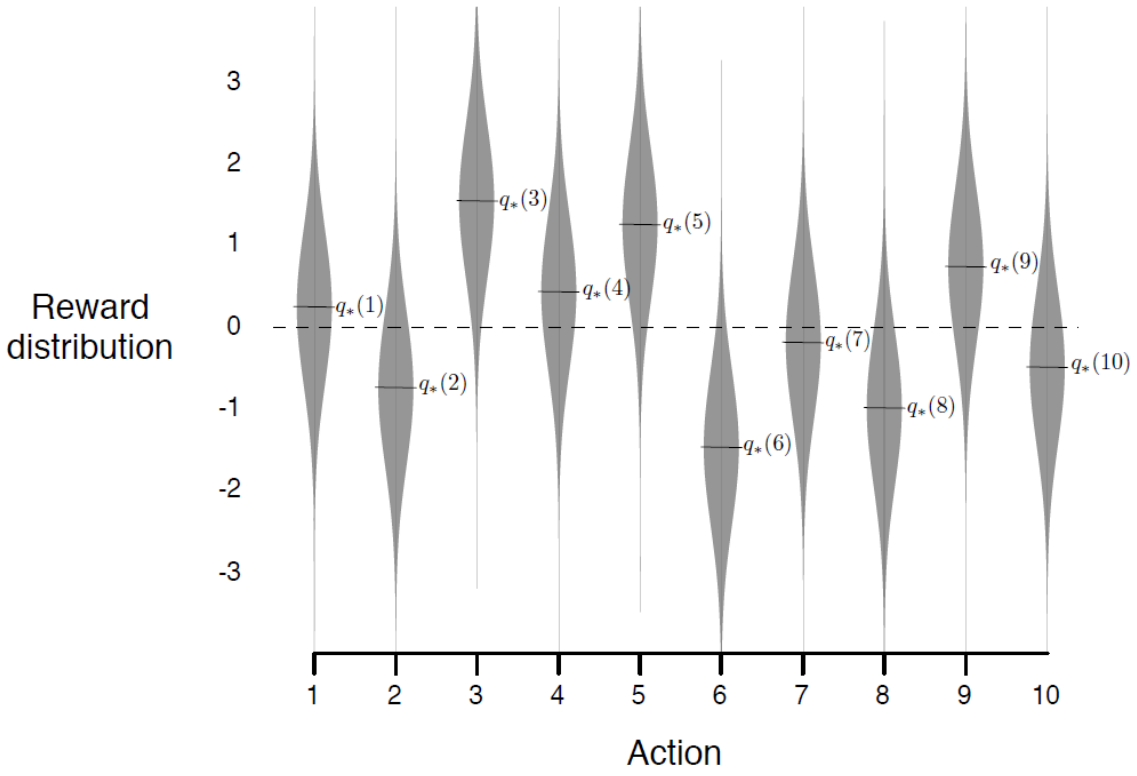
(Note: $\mathrm{KL}(\mathcal{R}_a || \mathcal{R}_{a*}) \propto \Delta_a^2$)

- Note that **regret grows at least logarithmically**

We want to minimize the total regret: $L_T = \sum_{t=1}^{T} (v^* - q(A_t)) = \sum_{t=1}^{T} \Delta_{A_t}$

maximize the expected cumulative reward = minimize total regret

Multi-armed Bandits



$$Q_t(a) = \frac{t\text{时刻前通过执行动作a得到的收益总和}}{t\text{时刻前执行动作}a\text{的次数}}$$

$$= \frac{\sum_{i=1}^{t-1} R_i \mathbb{I}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{I}_{A_i=a}} = \frac{1}{N_t(a)} \sum_{i=1}^{t-1} R_i \mathbb{I}_{A_i=a}$$

The simplest action selection rule is to select one of the actions with the highest estimated value:

$$A_t = \arg\max_a Q_t(a)$$

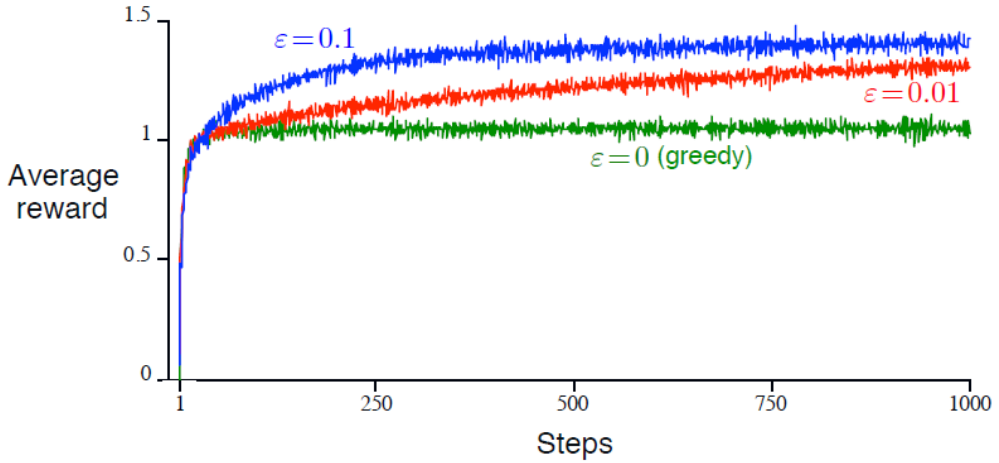
Multi-armed Bandits

Based on how we do exploration, there several ways to solve the multi-armed bandits

- No exploration: the most naive approach and a bad one.
- Exploration at random: ϵ-greedy
- Exploration smartly with preference to uncertainty: UCB

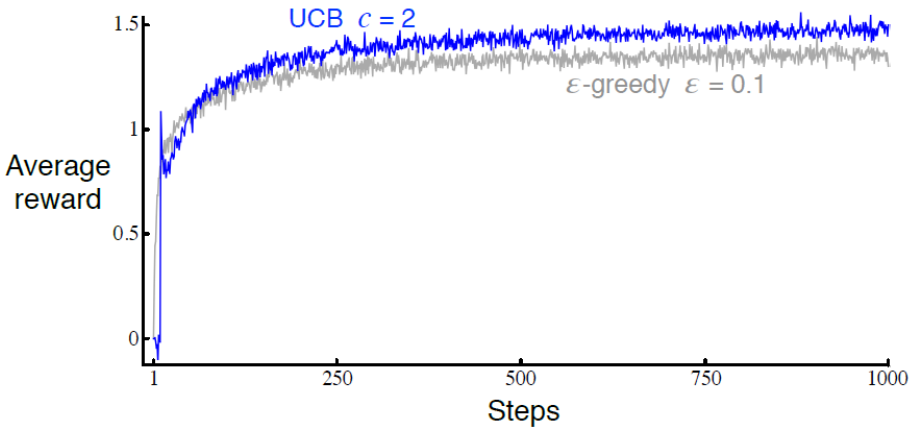
ϵ–Greedy Exploration

$$\pi_t(a) = \begin{cases} \frac{\epsilon}{|A|} + 1 - \epsilon, & if\ Q_t(a) = \arg\max_{b \in A} Q(b) \\ \frac{\epsilon}{|A|}, & otherwise \end{cases}$$



Upper Confidence Bound(UCB)

$$A_t = \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right]$$



Upper Confidence Bounds

Estimate an upper confidence $U_t(a)$ for each action value, such that $q(a) \leq Q_t(a) + U_t(a)$ with high probability

Select action maximizing upper confidence bound (UCB)

$$A_t = \arg\max_a [Q_t(a) + U_t(a)]$$

Using Hoeffding`s Inequality to bandits with bounded rewards, if $R_t \in [0,1]$, then

$$P\big(Q_t(a) + U_t(a) \leq q(a)\big) \leq e^{-2N_t(a)U_t(a)^2}$$

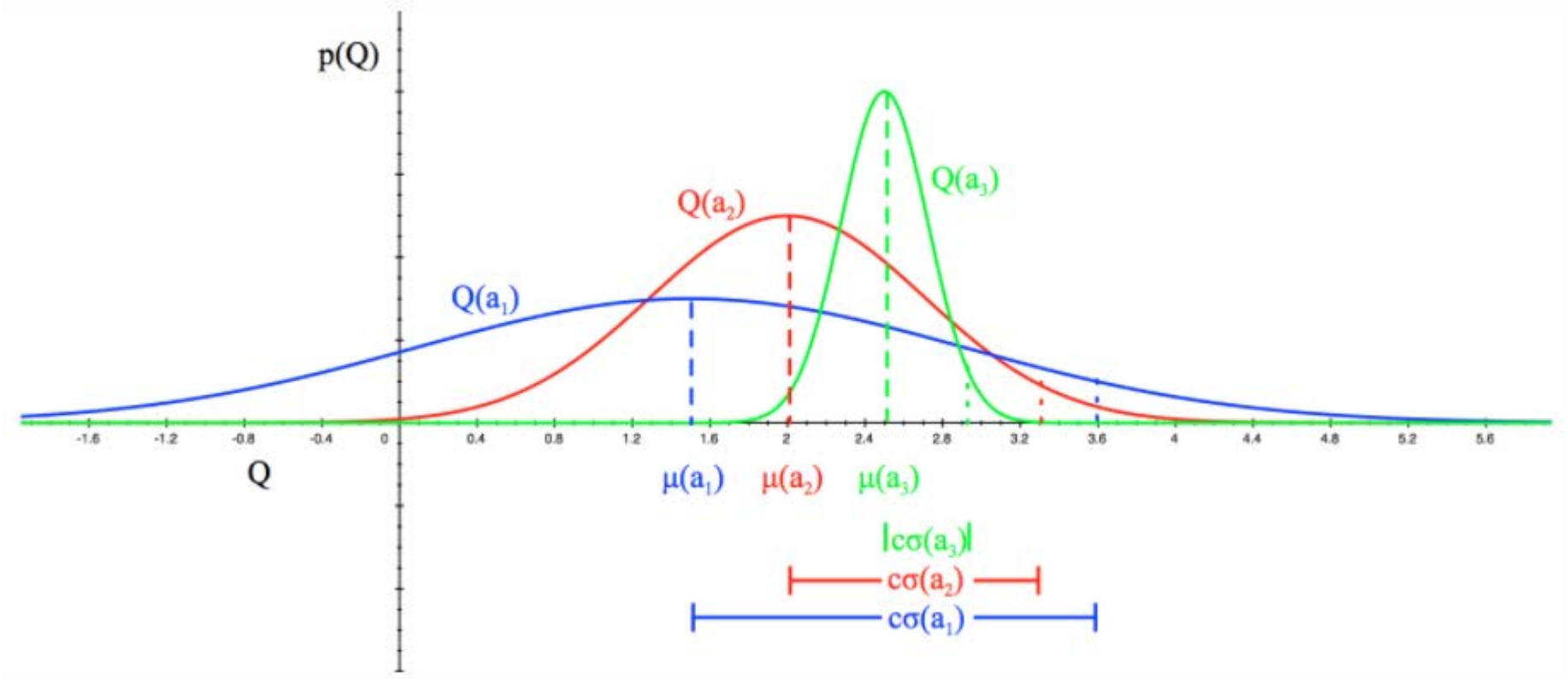
$$e^{-2N_t(a)U_t(a)^2} \leq p \quad \Longrightarrow \quad U_t(a) = \sqrt{\frac{-\ln p}{2N_t(a)}}$$

Idea: reduce p as we observe more rewards, e.g., $p = 1/t$

$$U_t(a) = \sqrt{\frac{\ln t}{2N_t(a)}}$$

Bayesian Bandits

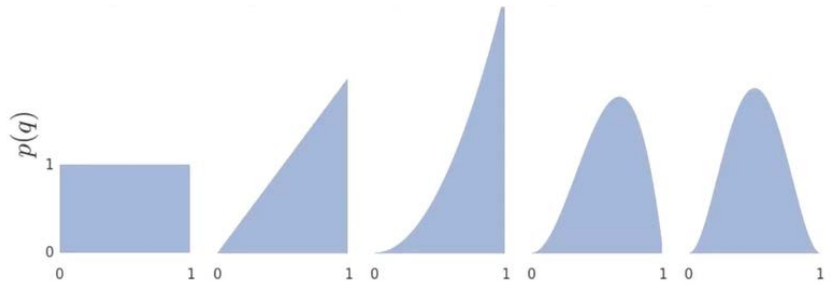
- We could adopt Bayesian approach and model distributions over values $P(q(a)|\theta_t)$
- This is interpreted as our belief that, e.g. $q(a) \sim N(\mu, \sigma)$
- Allows us to inject prior knowledge θ_0
- We can then use posterior belief to guide exploration



Bayesian Bandits

- Consider bandits with Bernoulli reward distribution: rewards are 0 or +1
- For each action, the prior could be a beta distribution $Beta(\alpha_0, \beta_0)$, e.g. $\alpha_0 = \beta_0 = 1$
- The posterior of $q(a)$ is also a beta distribution, we can update by
  - $\alpha_t = \alpha_{t-1} + 1, if\ R_t = 0$
  - $\beta_t = \beta_{t-1} + 1, if\ R_t = 1$

Suppose: $R_1 = +1$, $R_2 = +1$, $R_3 = 0$, $R_4 = 0$



Thompson sampling (Thompson 1933):

- Sample $Q_t(a) \sim p_t(q(a)), \forall a$
- Select action maximising sample, $A_t = \underset{a \in \mathcal{A}}{\text{argmax}}\ Q_t(a)$

Thompson sampling is sample-based probability matching

$$\pi_t(a) = \mathbb{E}\left[\mathcal{I}(Q_t(a) = \max_{a'} Q_t(a'))\right]$$
$$= p\left(q(a) = \max_{a'} q(a')\right)$$

For Bernoulli bandits, Thompson sampling achieves Lai and Robbins lower bound on regret, and therefore is **optimal**

Probability matching

Select action a according to the probability (belief) that a is optimal

$$\pi_t(a) = p\left(q(a) = \max_{a'} q(a') \mid \mathcal{H}_{t-1}\right)$$

Probability matching is optimistic in the face of uncertainty:
Actions have higher probability when either the estimated value is high, or the uncertainty is high

Can be difficult to compute $\pi(a)$ analytically from posterior (but can be done numerically)

Gradient Bandit Algorithms

Can we learn policy $\pi(a)$ directly, instead of learning values?

For instance, define action preferences $H_t(a)$ and a policy

$$\Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \doteq \pi_t(a)$$

The preferences are not values: they are just learnable policy parameters

Idea: update policy parameters such that expected value increases (gradient ascend)

$$\nabla_\theta \, \mathbb{E}[R_t|\pi_\theta] = \nabla_\theta \sum_a \pi_\theta(a) \overbrace{\mathbb{E}[R_t|A_t = a]}^{= q(a)} = \sum_a q(a) \, \nabla_\theta \pi_\theta(a) = \sum_a q(a) \, \frac{\pi_\theta(a)}{\pi_\theta(a)} \, \nabla_\theta \pi_\theta(a)$$

$$= \sum_a \pi_\theta(a) \, q(a) \, \frac{\nabla_\theta \pi_\theta(a)}{\pi_\theta(a)} \qquad = \mathbb{E}\left[R_t \, \frac{\nabla_\theta \pi_\theta(A_t)}{\pi_\theta(A_t)} \right] = \mathbb{E}\left[R_t \nabla_\theta \log \pi_\theta(A_t) \right]$$

can be sampled

$$\theta = \theta + \alpha R_t \nabla_\theta \log \pi_\theta(A_t)$$

Gradient Bandit Algorithms

$$H_{t+1}(a) = H_t(a) + \alpha R_t \frac{\partial \log \pi_t(A_t)}{\partial H_t(a)}$$
$$= H_t(a) + \alpha R_t(\mathcal{I}(a = A_t) - \pi_t(a))$$

$$\Longrightarrow$$

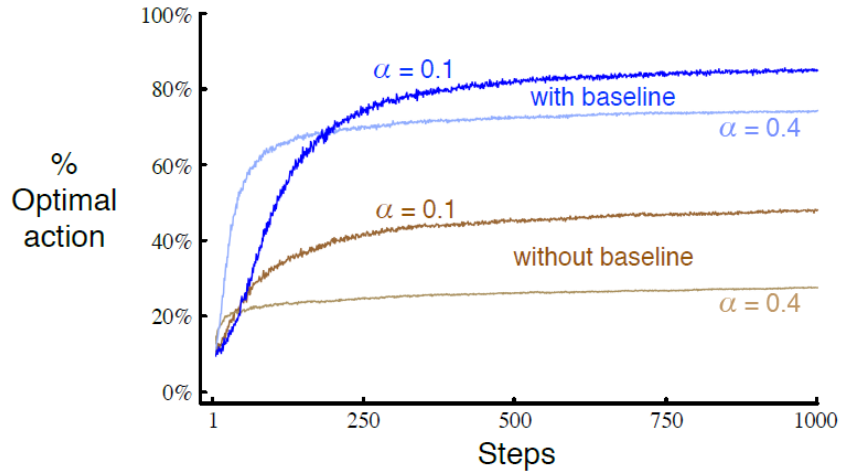
$$H_{t+1}(A_t) = H_t(A_t) + \alpha R_t(1 - \pi_t(A_t))$$
$$H_{t+1}(a) = H_t(a) - \alpha R_t \pi_t(a) \qquad \text{if } a \neq A_t$$

policy gradients with baselines

Note that $\sum_a \pi_\theta(a) = 1$. Therefore, for any b, $\sum_a b \nabla_\theta \pi_\theta(a) = b \nabla_\theta \sum_a \pi_\theta(a) = 0$

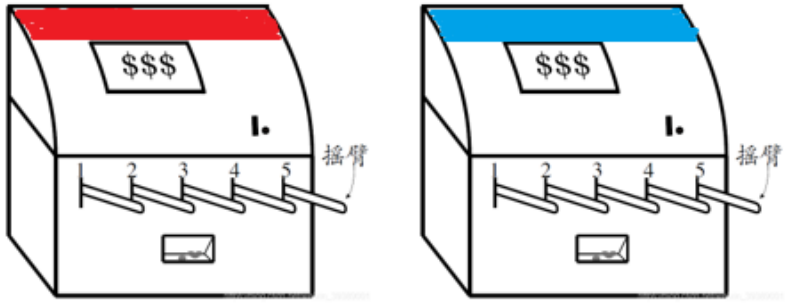
This means we can subtract a **baseline**, and instead use $\theta = \theta + \alpha(R_t - b)\nabla_\theta \log \pi_\theta(A_t)$

Baselines do not change the expected update, but they do change variance



Average performance of the gradient bandit algorithm with and without a reward baseline on the 10-armed testbed when the $q_*(a)$ are chosen to be near +4 rather than near zero.

Associative Search (Contextual Bandits)



Contextual Bandits is an example of an associative search task, so called because it involves both trial-and-error learning to search for the best actions, and association of these actions with the situations in which they are best.

Associative search tasks are intermediate between the k-armed bandit problem and the full reinforcement learning problem.

They are like the full reinforcement learning problem in that they involve learning a policy, but they are also like our version of the k-armed bandit problem in that each action affects only the immediate reward. If actions are allowed to affect the next situation as well as the reward, then we have the full reinforcement learning problem.

Finite Markov Decision Process

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

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: $P(S_{t+1} = s' | S_t = s, A_t = a)$
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

- 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）

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

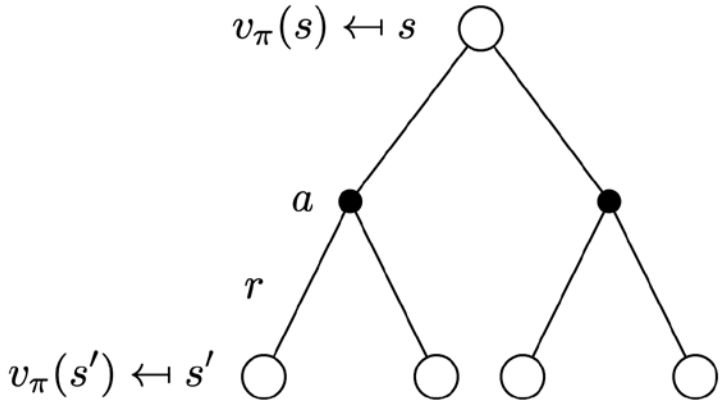
- Relation between $v^{\pi}(s)$ and $q^{\pi}(s,a)$

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

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

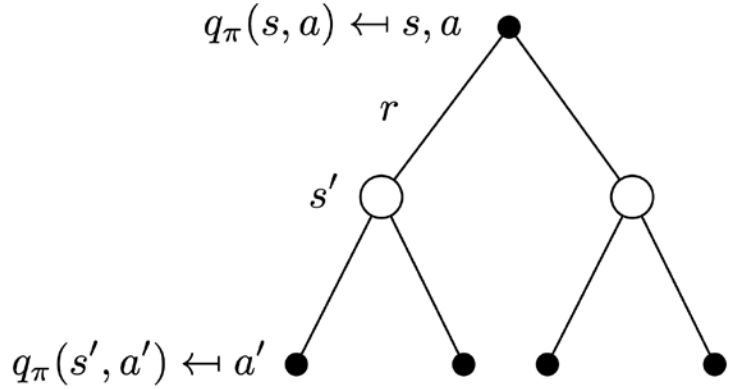
Bellman Expectation Equation

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



The action-value function can similarly be decomposed

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



Decision Making in MDP

Prediction

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

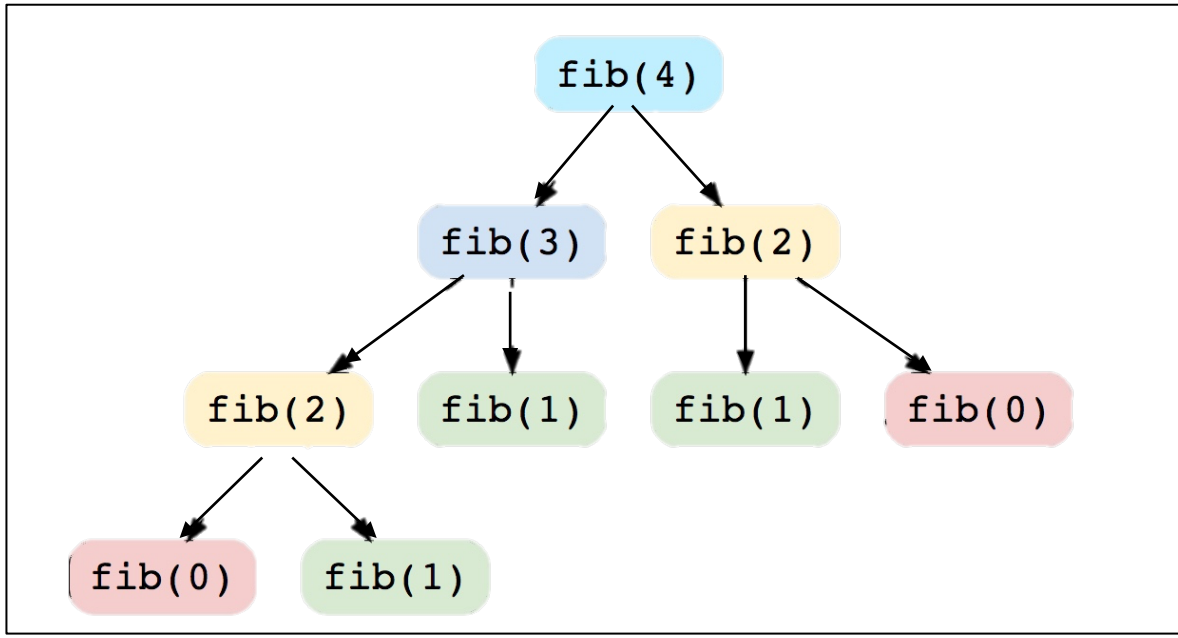
Control

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

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

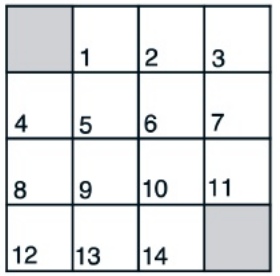
Policy evaluation in MDP

Iteration on Bellman expectation backup

$$v_{t+1}(s) = \sum_{a \in A} \pi(a|s) \left(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v_t(s') \right)$$

Convergence: $v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_\pi$

Evaluating a Random Policy in the Small Gridworld



$R_t = -1$
on all transitions

Transition is deterministic given the action
Two terminal states (two shaded squares)

t=0

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

t=1

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

t=2

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

t=3

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

t=10

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

t=∞

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

Find Optimal Policy

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 $q^*(s, a)$

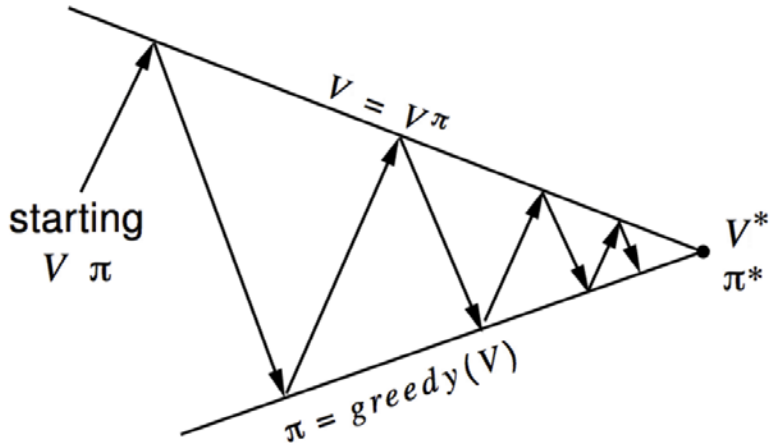
$$\pi^*(a|s) = \begin{cases} 1, & if\ a = \arg\max_{a \in A} q^*(s, a) \\ 0, & otherwise \end{cases}$$

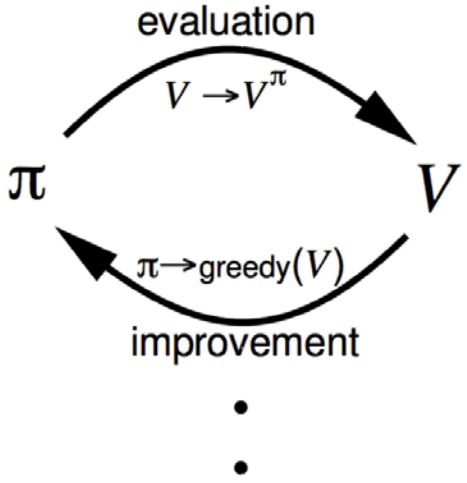
Noting that $q^*(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) v^*(s')$

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. $\pi_{t+1}(s) = \arg\max_a q^{\pi_t}(s,a)$





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:

$$v^{*}(s) = \max_{a} q^{*}(s,a)$$

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

then

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

Used for Value Iteration

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

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

$$q_{k+1}(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v_k\,(s')$$

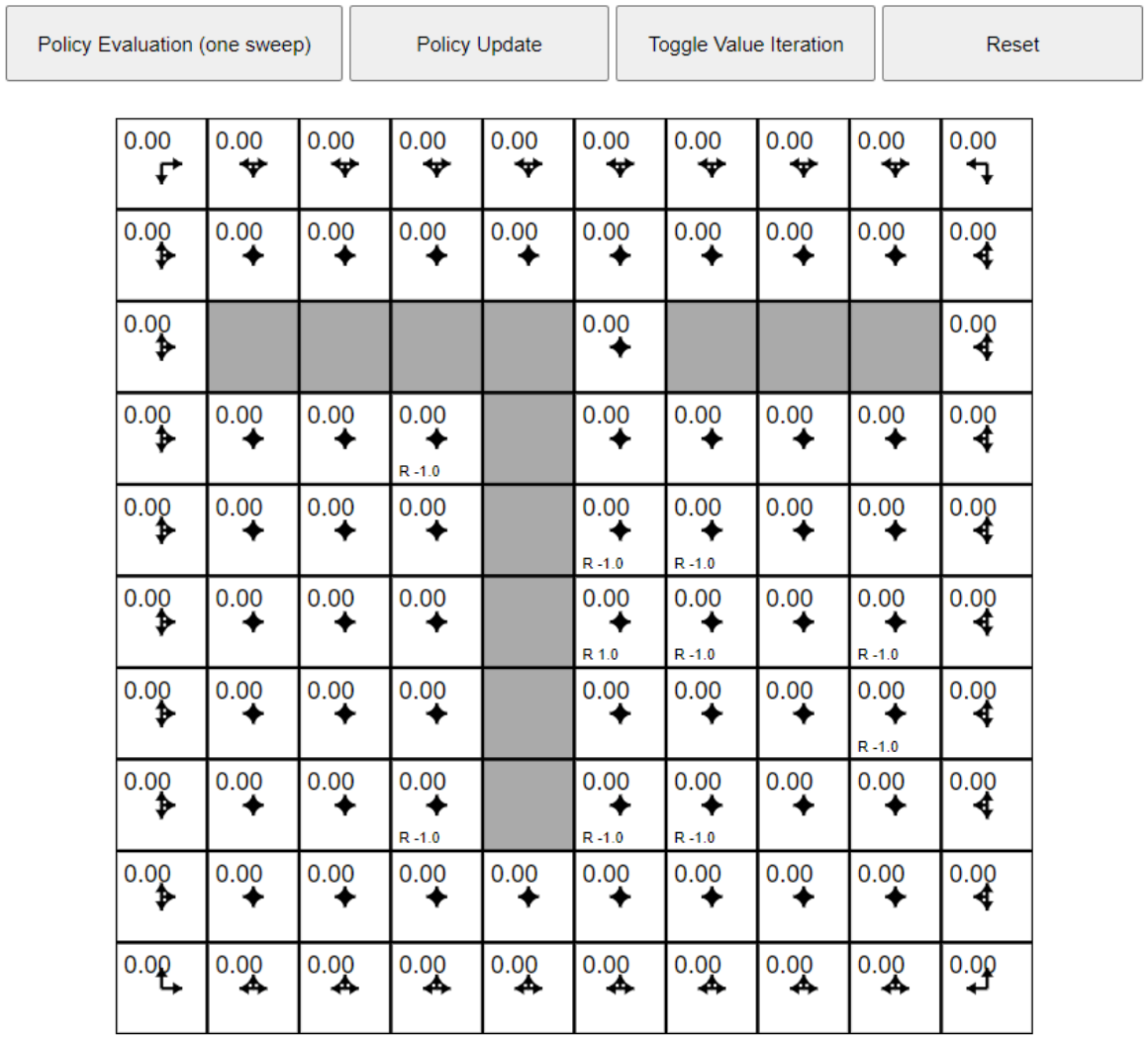
$$v_{k+1}(s) = \max_a q_{k+1}(s,a)$$

    2. k = k+1

3. To retrieve the optimal policy after the value iteration

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

Gridworld with Dynamic Programming (stanford.edu)



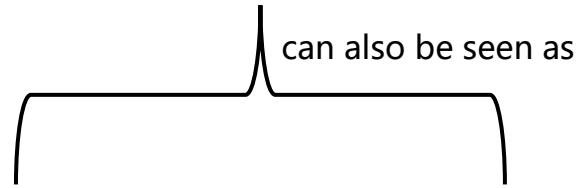
Comparison between PI and VI

Policy Iteration

policy evaluation + policy improvement

Value Iteration

finding optimal value function + one policy extraction



truncated policy evaluation (reassigning $v(s)$ after just one step)

policy improvement (max operation)

Summary for Prediction and Control in MDP

Table: Dynamic Programming Algorithms

Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectation Equation	Iterative Policy Evaluation
Control	Bellman Expectation Equation	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

Convergence of Bellman iteration

Value function space

- Consider a vector space $\mathcal{V}$ with $|S|$ dimension over value functions
- Each point in this space fully specifies a state-value function v

∞-norm: $\left|\left|u - v\right|\right|_\infty = \max_s |u(s) - v(s)|$

Bellman expectation operator: $T^\pi V(s) = \sum_a \pi(a|s) \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s')\right]$

Bellman optimality operator: $T^* V(s) = \max_a \{R(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s')\}$

The operator is a γ-contraction: $\|T(u) - T(v)\|_\infty \leq \gamma \|u - v\|_\infty$

Contraction Mapping Theorem

For any metric $\mathcal{V}$ space that is complete under an operator $T(v)$, where T is a γ-contraction

- T converges to a unique fixed point
- At a linear convergence rate of γ

https://people.clas.ufl.edu/kees/files/ContractionMapping.pdf

Convergence of Bellman iteration

$$\|T^{\pi}V - T^{\pi}U\|_{\infty} = \max_{s} \gamma \sum_{s'} \Pr(s'|s,\pi(s))|V(s') - U(s')|$$

$$\leq \gamma \Big(\sum_{s'} \Pr(s'|s,\pi(s)) \Big) \max_{s'} |V(s') - U(s')| \leq \gamma \|U - V\|_{\infty}.$$

$$\begin{aligned}
\|TV - TU\|_{\infty} = & \max_{s} |\max_{a}\{R(s,a) + \gamma \sum_{s'} \Pr(s'|s,a)V(s')\} \\
& - \max_{a}\{R(s,a) + \gamma \sum_{s'} \Pr(s'|s,a)U(s')\}| \\
\leq & \max_{s,a} |R(s,a) + \gamma \sum_{s'} \Pr(s'|s,a)V(s') \\
& - R(s,a) - \gamma \sum_{s'} \Pr(s'|s,a)V(s')| \\
= & \gamma \max_{s,a} |\sum_{s'} \Pr(s'|s,a)(V(s') - U(s'))| \\
\leq & \gamma \Big(\sum_{s'} \Pr(s'|s,a) \Big) \max_{s'} |(V(s') - U(s'))| \leq \gamma \|V - U\|_{\infty}.
\end{aligned}$$

Model-free 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 methods:

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

Monte Carlo Policy Evaluation

Recall:

- Agent interacts with the environment to collect many trajectories τ: $\{S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T, S_T\}$ 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, $v(s) \rightarrow v^{\pi}(s) \ \ as \ \ N(s) \rightarrow \infty$

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.

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

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

Temporal-Difference Learning

如果价值函数数组V在一幕内没有改变（例如，蒙特卡洛方法就是如此），则蒙特卡洛误差可以写为TD误差之和

$$
\begin{aligned}
G_t - V(S_t) &= R_{t+1} + \gamma G_{t+1} - V(S_t) + \gamma V(S_{t+1}) - \gamma V(S_{t+1}) \\
&= \delta_t + \gamma\big(G_{t+1} - V(S_{t+1})\big) \\
&= \delta_t + \gamma\delta_{t+1} + \gamma^2\big(G_{t+2} - V(S_{t+2})\big) \\
&= \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} + \cdots + \gamma^{T-t-1}\delta_{T-1} + \gamma^{T-t}\big(G_T - V(S_T)\big) \\
&= \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} + \cdots + \gamma^{T-t-1}\delta_{T-1} + \gamma^{T-t}\big(0 - 0\big) \\
&= \sum_{k=t}^{T-1} \gamma^{k-t}\delta_k .
\end{aligned}
$$

We have:

$\delta_t \doteq R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t)$

$V_{t+1}(S) = V_t(S) + \alpha[R_{t+1} + V_t(S') - V_t(S)]$ ⇒ U_t

Therefore

$G_t - V_t(S_t) = R_{t+1} + \gamma G_{t+1} - V_t(S_t) + \gamma V_t(S_{t+1}) - \gamma V_t(S_{t+1})$

$= \delta_t + \gamma[G_{t+1} - V_t(S_{t+1})]$

$= \delta_t + \gamma[G_{t+1} - V_{t+1}(S_{t+1})] + \gamma[V_{t+1}(S_{t+1}) - V_t(S_{t+1})]$

$= \delta_t + \gamma[G_{t+1} - V_{t+1}(S_{t+1})] + \gamma U_{t+1}$ {这里之所以是U_{t+1}是因为括号中是$t+1$时刻的值S_{t+1}}

接下来，就可以使用递推式了

$G_t - V_t(S_t) = \delta_t + \gamma[G_{t+1} - V_{t+1}(S_{t+1})] + \gamma U_{t+1}$

$= \delta_t + \gamma\delta_{t+1} + \gamma^2[G_{t+2} - V_{t+2}(S_{t+2})] + \gamma U_{t+1} + \gamma^2 U_{t+2}$

$= \cdots$

$= \sum_{k=t}^{T-1}[\gamma^{k-t}\delta_k + \gamma^{k-t+1}U_{k+1}]$

DP, MC and TD

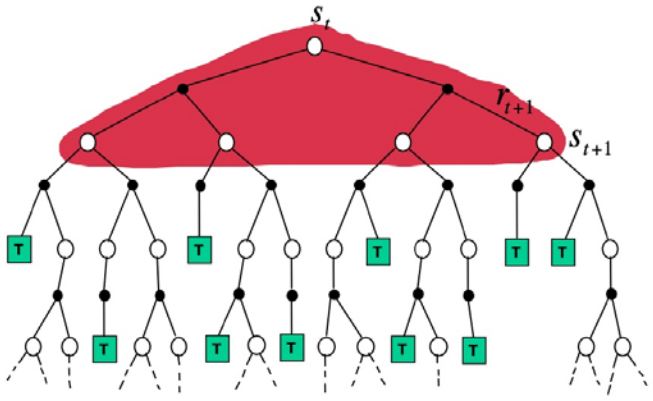
Bootstrapping

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

Sampling

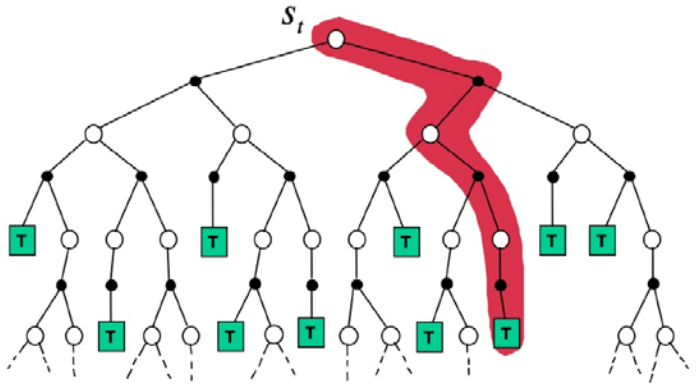
Update samples an expectation
- DP does not sample
- MC samples
- TD samples

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



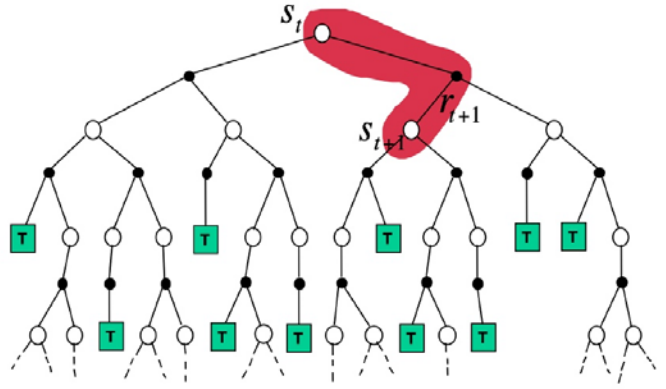
Dynamic Programming

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



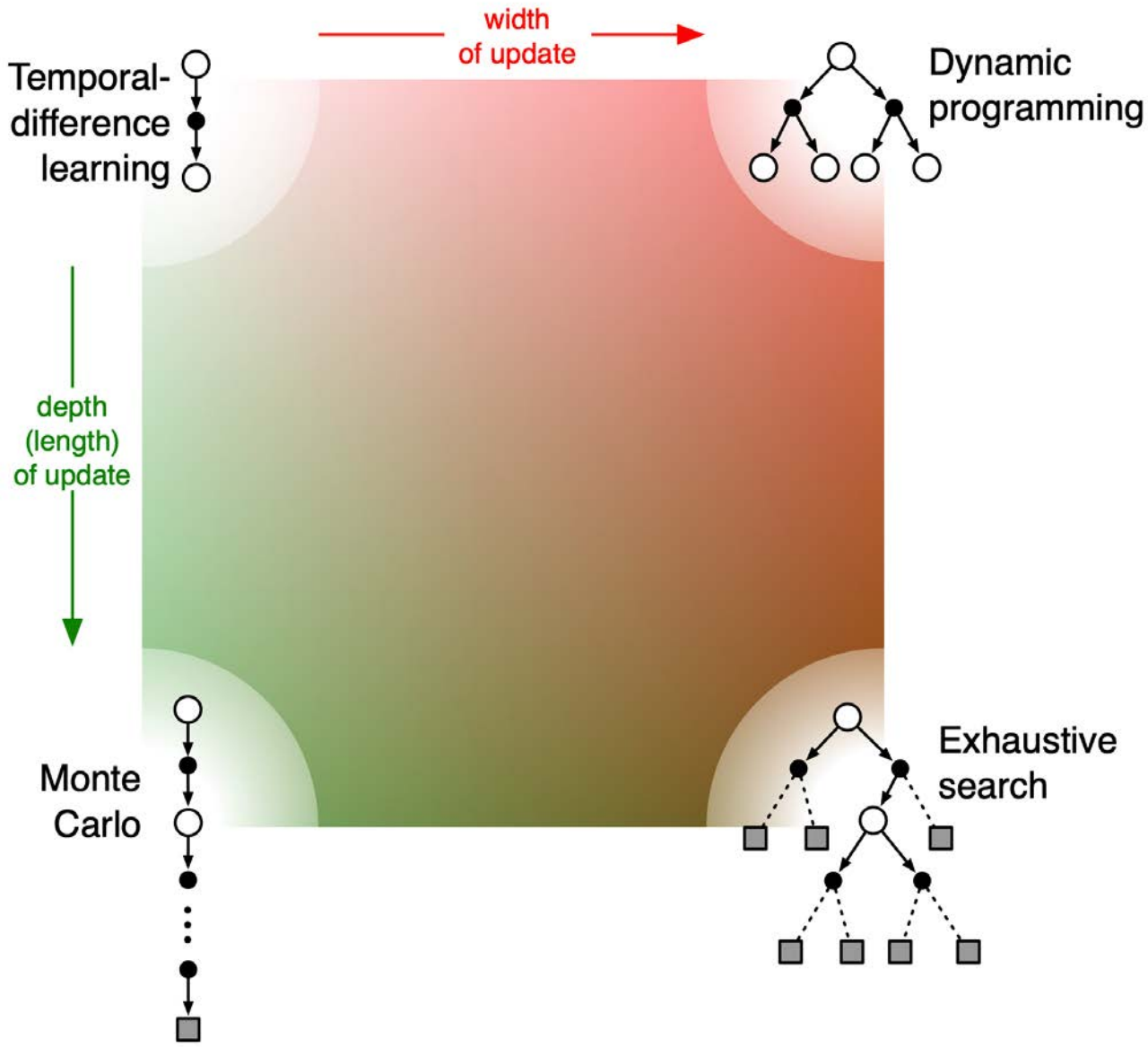
Monte Carlo

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



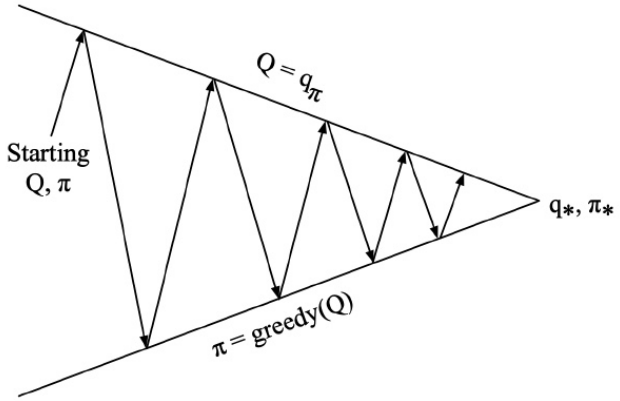
Temporal Difference Learning

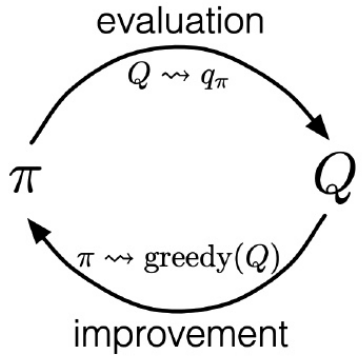
Unified View of Reinforcement Learning



Model-free Control for MDP

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





Policy Iteration for a known MDP

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

$$\pi_{t+1}(s) = \arg\max_a q^{\pi_t}(s,a)$$

we don't know

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

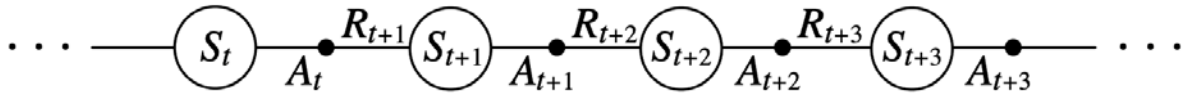
Algorithm

Initialize $Q(S, A) = 0, N(S, A) = 0, \epsilon = 1, k = 1$
$\pi_k = \epsilon$-greedy(Q)
loop
  Sample k-th episode $(S_1, A_1, R_2, ..., S_T) \sim \pi_k$
  for each state S_t and action A_t in the episode **do**
    $N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}(G_t - Q(S_t, A_t))$
  end for
  $k \leftarrow k + 1, \epsilon \leftarrow 1/k$
  $\pi_k = \epsilon$-greedy(Q)
end loop

ϵ–Greedy Exploration

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

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 $Q(\textit{terminal-state}, \cdot) = 0$
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

On-policy vs. Off-policy

On-policy learning

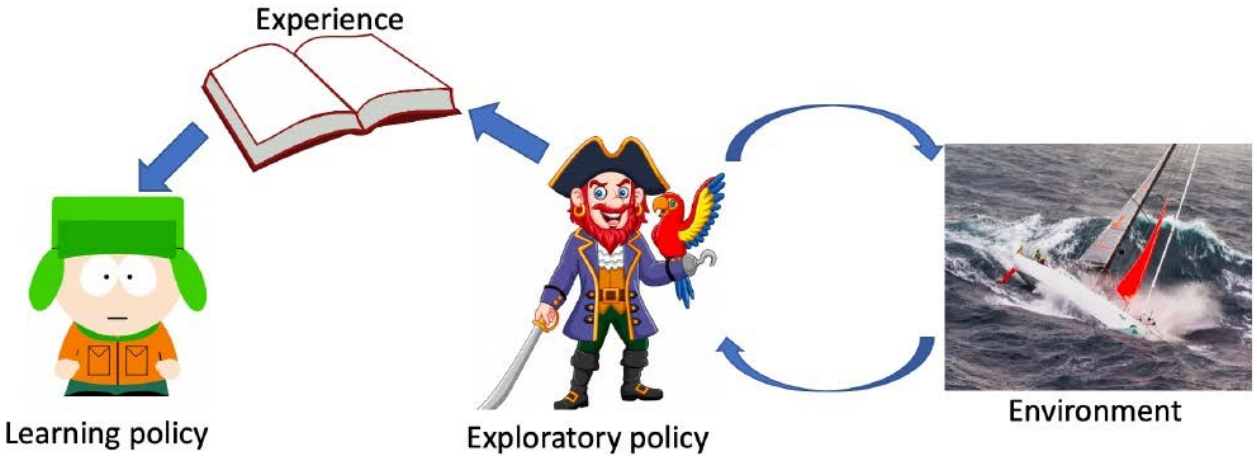
Learn about policy π from the experience collected from π

- 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., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
    $S \leftarrow S'$
  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

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

Maximization Bias and Double Learning

我们回顾一下Q-learning的target值：$y_t = R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$,

这个max操作会使Q value越来越大，甚至高于真实值，原因用数学公式来解释的话就是：对于两个随机变量X_1和X_2，有 $E[\max(X_1, X_2)] \geq \max(E[X_1], E[X_2])$

$$
\begin{aligned}
\max_{a'} Q_w(s_{t+1}, a') &= Q_w(s_{t+1}, \arg\max_{a'} Q_w(s_{t+1}, a')) \\
&= E[G_t | s_{t+1}, \arg\max_{a'} Q_w(s_{t+1}, a'), w] \\
&\geq max(E[G_t | s_{t+1}, a_1, w], E[G_t | s_{t+1}, a_2, w], \cdots) \\
&= max(Q_w(s_{t+1}, a_1), Q_w(s_{t+1}, a_2), \cdots)
\end{aligned}
$$

不等式右边才是我们真正想求的值，问题就出在选择action和计算target Q value用的是同一个估计。而且随着候选动作的增加，过高估计的程度也会越严重。

Maximization Bias and Double Learning

Example 6.7: Maximization Bias Example The small MDP shown inset in Figure 6.5 provides a simple example of how maximization bias can harm the performance of TD control algorithms. The MDP has two non-terminal states A and B. Episodes always start in A with a choice between two actions, left and right. The right action transitions immediately to the terminal state with a reward and return of zero. The left action transitions to B, also with a reward of zero, from which there are many possible actions all of which cause immediate termination with a reward drawn from a normal distribution with mean -0.1 and variance 1.0. Thus, the expected return for any trajectory starting with left is -0.1, and thus taking left in state A is always a mistake. Nevertheless, our control methods may favor left because of maximization bias making B appear to have a positive value. Figure 6.5 shows that Q-learning with ε-greedy action selection initially learns to strongly favor the left action on this example. Even at asymptote, Q-learning takes the left action about 5% more often than is optimal at our parameter settings ($\varepsilon = 0.1$, $\alpha = 0.1$, and $\gamma = 1$).

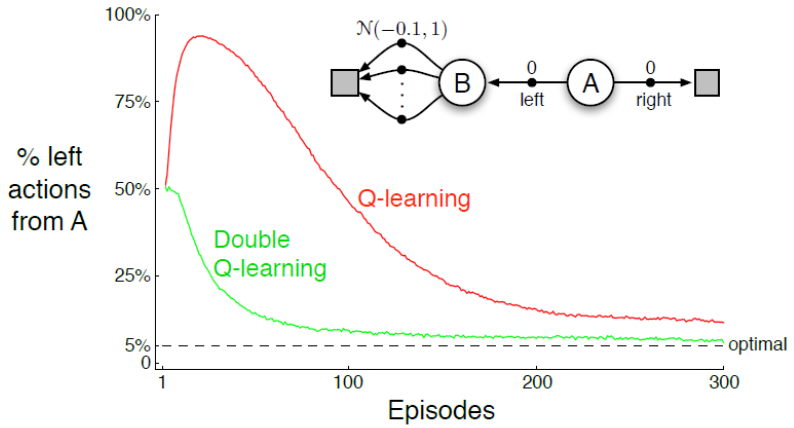


Figure 6.5: Comparison of Q-learning and Double Q-learning on a simple episodic MDP (shown inset). Q-learning initially learns to take the left action much more often than the right action, and always takes it significantly more often than the 5% minimum probability enforced by ε-greedy action selection with $\varepsilon = 0.1$. In contrast, Double Q-learning is essentially unaffected by maximization bias. These data are averaged over 10,000 runs. The initial action-value estimates were zero. Any ties in ε-greedy action selection were broken randomly.

将这些样本划分为两个集合，并用它们学习两个独立的对真实价值$q(a)$的估计$Q_1(a)$和$Q_2(a)$，

那么我们接下来就可以使用其中一个估计，比$Q_1(a)$，来确定最大的动作$A^* = \arg\max_a Q_1(a)$，

再用另外一个来计算其价值的估计$Q_2(A^*) = Q_2(\arg\max_a Q_1(a))$。

Function Approximation

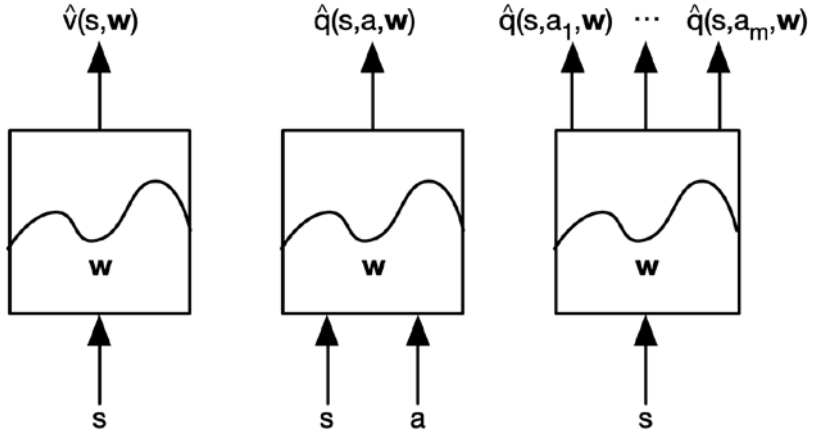
Why do we need function approximation?

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

How to do?

generalization

$$\hat{v}(s, \mathrm{w}) \approx v^{\pi}(s)$$
$$\hat{q}(s, a, w) \approx q^{\pi}(s, a)$$
$$\hat{\pi}(s, \theta) \approx \pi(a|s)$$



Many possible function approximators:
- Linear functions
- Decision trees
- Neural networks

Value Function Approximation

Approximate the state value function

$$\hat{V}(s;w) \approx V^{\pi}(s)$$

Minimize the MSE between the estimate and the true value (suppose we know)

$$J(\theta) = E_{\pi}\left[\frac{1}{2}\left(V^{\pi}(s) - \hat{V}(s;w)\right)^2\right]$$

Stochastic gradient descend

$$w \leftarrow w - \alpha \nabla_w J(w) = w + \alpha\left(V^{\pi}(s) - \hat{V}(s;w)\right)\nabla_w \hat{V}(s;w)$$

Value Function Approximation

But actually we do not have the true value $V^{\pi}(s)$, so we replace it

For MC,

$$w \leftarrow w - \alpha \nabla_w J(w) = w + \alpha \left(G_t - \hat{V}(s_t; w) \right) \nabla_w \hat{V}(s_t; w)$$

For TD(0),

$$w \leftarrow w - \alpha \nabla_w J(w) = w + \alpha \left(R_{t+1} + \gamma \hat{V}(s_{t+1}; w) - \hat{V}(s_t; w) \right) \nabla_w \hat{V}(s_t; w)$$

Convergence of Linear TD(0)

TD方法在表格型数据可以收敛到真实的V^{π}，但是当使用线性函数近似的情况下，虽然也具有收敛性的，但并不是收敛到通常我们所说的最优值，而是它附近的一个值。

$$\overline{\mathrm{VE}}(\mathbf{w}_{\mathrm{TD}}) \leq \frac{1}{1-\gamma} \min_{\mathbf{w}} \overline{\mathrm{VE}}(\mathbf{w})$$

收敛性需要用额外的方法来证明。

Corresponding to every state s, there is a real-valued vector

$$\mathbf{x}(s) \doteq (x_1(s), x_2(s), \ldots, x_d(s))^{\top}$$

$$\hat{v}(s,\mathbf{w}) \doteq \mathbf{w}^{\top}\mathbf{x}(s) \doteq \sum_{i-1}^{d} w_i x_i(s)$$

The update at each time t is

$$\begin{aligned}\mathbf{w}_{t+1} &\doteq \mathbf{w}_t + \alpha\left(R_{t+1} + \gamma \mathbf{w}_t^{\top}\mathbf{x}_{t+1} - \mathbf{w}_t^{\top}\mathbf{x}_t\right)\mathbf{x}_t \\ &= \mathbf{w}_t + \alpha\left(R_{t+1}\mathbf{x}_t - \mathbf{x}_t\left(\mathbf{x}_t - \gamma\mathbf{x}_{t+1}\right)^{\top}\mathbf{w}_t\right),\end{aligned}$$

where here we have used the notational shorthand $\mathbf{x}_t = \mathbf{x}(S_t)$. Once the system has reached steady state, for any given $\mathbf{w}_t$, the expected next weight vector can be written

$$\mathbb{E}[\mathbf{w}_{t+1}|\mathbf{w}_t] = \mathbf{w}_t + \alpha(\mathbf{b} - \mathbf{A}\mathbf{w}_t),$$

where

$$\mathbf{b} \doteq \mathbb{E}[R_{t+1}\mathbf{x}_t] \in \mathbb{R}^d \quad \text{and} \quad \mathbf{A} \doteq \mathbb{E}\left[\mathbf{x}_t\left(\mathbf{x}_t - \gamma\mathbf{x}_{t+1}\right)^{\top}\right] \in \mathbb{R}^{d\times d}$$

From (9.10) it is clear that, if the system converges, it must converge to the weight vector $\mathbf{w}_{\mathrm{TD}}$ at which

$$\begin{aligned}&\mathbf{b} - \mathbf{A}\mathbf{w}_{\mathrm{TD}} = \mathbf{0} \\ \Rightarrow \quad &\mathbf{b} = \mathbf{A}\mathbf{w}_{\mathrm{TD}} \\ \Rightarrow \quad &\mathbf{w}_{\mathrm{TD}} \doteq \mathbf{A}^{-1}\mathbf{b}.\end{aligned}$$

This quantity is called the *TD fixed point*. In fact linear semi-gradient TD(0) converges to this point.

Sutton R S. Learning to predict by the methods of temporal differences[J]. Machine learning, 1988, 3(1): 9-44.

Convergence of Linear TD(0)

$$\mathbb{E}[\mathbf{w}_{t+1}|\mathbf{w}_t] = (\mathbf{I} - \alpha\mathbf{A})\mathbf{w}_t + \alpha\mathbf{b}.$$

$$\begin{aligned}
\mathbf{A} &= \sum_s \mu(s) \sum_a \pi(a|s) \sum_{r,s'} p(r, s'|s, a)\mathbf{x}(s)\big(\mathbf{x}(s) - \gamma\mathbf{x}(s')\big)^\top \\
&= \sum_s \mu(s) \sum_{s'} p(s'|s)\mathbf{x}(s)\big(\mathbf{x}(s) - \gamma\mathbf{x}(s')\big)^\top \\
&= \sum_s \mu(s)\mathbf{x}(s)\left(\mathbf{x}(s) - \gamma\sum_{s'} p(s'|s)\mathbf{x}(s')\right)^\top \\
&= \mathbf{X}^\top\mathbf{D}(\mathbf{I} - \gamma\mathbf{P})\mathbf{X},
\end{aligned}$$

关于这个矩阵正定性的证明，用到了两个矩阵相关的定理。

1. 任意矩阵 M 是正定的 $\Leftrightarrow$ 对称矩阵 $S = M + M^\top$ 是正定的。(Sutton, 1988)
2. 对于任意一个对称实矩阵 S，如果它的所有对角线元素为正且大于对应的非对角线元素绝对值之和，那么它是正定的。(Varga, 1962)

Sutton R S. Learning to predict by the methods of temporal differences[J]. Machine learning, 1988, 3(1): 9-44.

The Deadly Triad for the Danger of Instability and Divergence

Function approximation A powerful, scalable way of generalizing from a state space much larger than the memory and computational resources (e.g., linear function approximation or ANNs).

Bootstrapping Update targets that include existing estimates (as in dynamic programming or TD methods) rather than relying exclusely on actual rewards and complete returns (as in MC methods).

Off-policy training Training on a distribution of transitions other than that produced by the target policy. 如动态规划中所做的，遍历整个状态空间并且均匀地更新所有状态而不理会目标策略就是一个离轨策略训练的例子。

Convergence of Control Methods

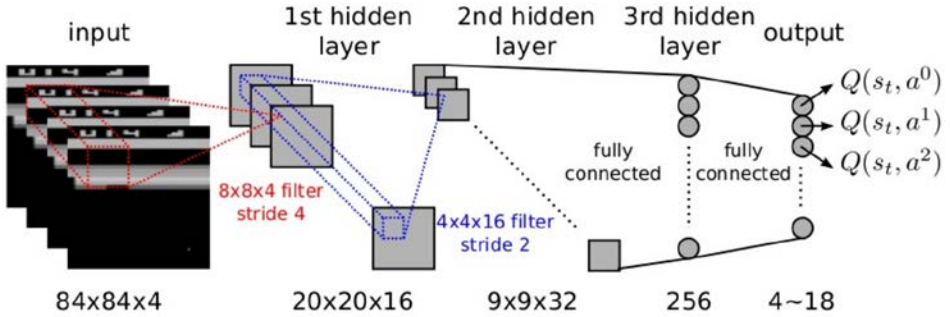
Algorithm	Table Lookup	Linear	Non-Linear
Monte-Carlo Control	$\surd$	$(\surd)$	X
Sarsa	$\surd$	$(\surd)$	X
Q-Learning	$\surd$	X	X

$(\surd)$ moves around the near-optimal value function

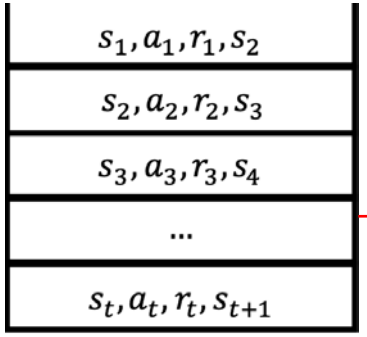
Deep Q-Learning

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



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.

Policy-based RL

Now that our goal is to find an optimal policy, why not we just parameterize the policy as $\pi_\theta(a|s)$

Compared with Value-based RL

Advantages:

- better convergence properties
- more efficient in high-dimensional or continuous action space
- can learn stochastic policies, while value-based RL can not

Disadvantages:

- usually converges to a local optimum
- high variance when evaluating a policy (interact with the environment, not sampling enough episodes)

Policy-based RL

Softmax Policy

For discrete action space

$$\pi_\theta(a|s) = \frac{\exp f_\theta(s,a)}{\sum_{a'} \exp f_\theta(s,a')}$$

Gaussian Policy

For continuous action space

mean is a function of state features $\mu(s) = f_\theta(s)$

variance may be fixed σ^2 or also be parameterized

$a \sim N(\mu(s), \sigma^2)$

Likelihood ratio trick (will use later):

$$\nabla_\theta \pi_\theta(a|s) = \pi_\theta(s,a) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} = \pi_\theta(a|s) \underbrace{\nabla_\theta \log \pi_\theta(a|s)}$$

score function

Policy Gradient for MDP

τ is a trajectory sampled from the environment following policy π_θ

$$\tau = (s_0, a_0, r_1, \dots, s_{T-1}, a_{T-1}, r_T, s_T) \sim (\pi_\theta, P(s_{t+1}|s_t, a_t))$$

The policy objective is defined as

$$J(\theta) = E_{\tau\sim\pi_\theta}[\sum_t \gamma^t R(s_t, a_t)] = \sum_\tau P(\tau, \theta) G(\tau) \quad \text{where} \quad P(\tau, \theta) = \mu(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t|s_t) P(s_{t+1}|s_t, a_t)$$

The goal is

$$\theta^* = \arg\max_\theta J(\theta)$$

Take the gradient with respect to θ:

Likelihood ratio trick

$$\nabla_\theta J(\theta) = \nabla_\theta \sum_\tau P(\tau, \theta) G(\tau) = \sum_\tau \nabla_\theta P(\tau, \theta) G(\tau) = \sum_\tau P(\tau, \theta) G(\tau) \nabla_\theta \log P(\tau, \theta)$$

Approximate with empirical estimate of m sample trajectories:

$$\nabla_\theta J(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} G(\tau_i) \nabla_\theta \log P(\tau_i, \theta)$$

Policy Gradient for MDP

Now we have

$$\nabla_\theta J(\theta) \approx \frac{1}{m}\sum_{i=1}^{m} G(\tau_i)\nabla_\theta \log P(\tau_i, \theta)$$

Decompose $\nabla_\theta \log P(\tau, \theta)$

$$\nabla_\theta \log P(\tau, \theta) = \nabla_\theta \log\left[\mu(s_0)\prod_{t=0}^{T-1}\pi_\theta(a_t|s_t)p(s_{t+1}|s_t, a_t)\right] = \sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_t|s_t)$$

Thus

$$\nabla_\theta J(\theta) \approx \frac{1}{m}\sum_{i=1}^{m} G(\tau_i)\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)$$

We do not need to know the transition model !
Policy gradient is model-free

REINFORCE

REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

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
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t, \boldsymbol{\theta})$

Williams R J. Simple statistical gradient-following algorithms for connectionist reinforcement learning[J]. Machine learning, 1992, 8(3): 229-256.

Problem of MC Policy Gradient

$$\nabla_\theta J(\theta) \approx \frac{1}{m}\sum_{i=1}^{m} G(\tau_i) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^i|s_t^i)$$

Because $G(\tau_i) = \sum_t \gamma^t R(s_t, a_t)$

- unbiased but has large variance

Causality: policy at time t' cannot affect reward at time t when $t < t'$

$$\nabla_\theta J(\theta) \approx \frac{1}{m}\sum_{i=1}^{m} \sum_{t'=0}^{T-1} \gamma^{t'} R(s_t^i, a_t^i) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^i|s_t^i)$$

$$\approx \frac{1}{m}\sum_{i=1}^{m} \sum_{t'=t}^{T-1} \gamma^{t'} R(s_t^i, a_t^i) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^i|s_t^i)$$

$$= \frac{1}{m}\sum_{i=1}^{m} \sum_{t=0}^{T-1} G_t^i \nabla_\theta \log \pi_\theta(a_t^i|s_t^i)$$

ways to address the problem:

- use temporal causality
- include a baseline (talk about it later)
- use critic (talk about it later)

Include a Baseline

$$\nabla_\theta J(\theta) = E_\tau \left[\sum_{t=0}^{T-1} G_t \nabla_\theta \log \pi_\theta(a_t|s_t) \right]$$

We subtract a baseline $b(s)$ from the policy gradient to reduce variance

$$\nabla_\theta J(\theta) = E_\tau \left[\sum_{t=0}^{T-1} (G_t - b(s_t)) \nabla_\theta \log \pi_\theta(a_t|s_t) \right]$$

A good baseline is the expected return

$$b(s_t) = E[R_t + \gamma R_{t+1} + \cdots + \gamma^{T-1-t} R_{T-1}]$$

We have

$$E_\tau[\nabla_\theta \log \pi_\theta(a_t|s_t)\, b(s_t)] = 0$$

$$Var_\tau[\nabla_\theta \log \pi_\theta(a_t|s_t)\, (G_t - b(s_t))] < Var_\tau[\nabla_\theta \log \pi_\theta(a_t|s_t)\, G_t]$$

Vanilla Policy Gradient Algorithm with Baseline

procedure POLICY GRADIENT(α)

Initialize policy parameters θ and baseline values $b(s)$ for all s, e.g. to 0

for iteration $= 1, 2, \ldots$ **do**

Collect a set of m trajectories by executing the current policy π_θ

for each time step t of each trajectory $\tau^{(i)}$ **do**

Compute the *return* $G_t^{(i)} = \sum_{t'=t}^{T-1} r_{t'}$

Compute the *advantage estimate* $\hat{A}_t^{(i)} = G_t^{(i)} - b(s_t)$

Re-fit the baseline to the empirical returns by updating $\mathbf{w}$ to minimize

$$\sum_{i=1}^{m} \sum_{t=0}^{T-1} \| b(s_t) - G_t^{(i)} \|^2$$

Update policy parameters θ using the policy gradient estimate $\hat{g}$

$$\hat{g} = \sum_{i=1}^{m} \sum_{t=0}^{T-1} \hat{A}_t^{(i)} \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)})$$

with an optimizer like SGD ($\theta \leftarrow \theta + \alpha \cdot \hat{g}$) or Adam

return θ and baseline values $b(s)$

Using Critic to Reduce Variance

$$\nabla_\theta J(\theta) = E_\tau \left[\sum_{t=0}^{T-1} G_t \nabla_\theta \log \pi_\theta(a_t|s_t) \right]$$

G_t is actually a unbiased but noisy estimate of $Q^{\pi_\theta}(s_t, a_t)$ from Monte Carlo method.

we can use the idea from value-based RL and use a critic to estimate $Q^{\pi_\theta}(s_t, a_t)$

$$Q_w(s, a) \approx Q^{\pi_\theta}(s, a)$$

Therefore, the update becomes

$$\nabla_\theta J(\theta) = E_{\pi_\theta} \left[\sum_{t=0}^{T-1} Q_w(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t|s_t) \right]$$

Actor: policy function with parameters θ

Critic: state-action value function with parameters w,
and it is actually solving policy evaluation, so previous methods can be used: MC, TD

Actor-Critic with Baseline

$$\nabla_\theta J(\theta) = E_{\pi_\theta}\left[\sum_{t=0}^{T-1} Q_w\,(s_t, a_t)\,\nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

Recall the definition of Q-function and V-function, we can find that value function can naturally be a great baseline as $V^\pi(s) = \mathrm{E}_{a\sim\pi}[Q^\pi\,(s,a)]$

We define Advantage function: $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$

So the update becomes:

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[A^{\pi_\theta}(s,a)\nabla_\theta \log \pi_\theta(a|s)]$$ omit the sum just for convenience

We also call it as Advantage Actor-Critic (A2C)

One more thing, we do not need to build two models for Q-function and V-function (as you can do this).
Actually we can get a estimate of Advantage function by TD error $\delta^{\pi_\theta} = R(s,a) + \gamma V^{\pi_\theta}(s') - V^{\pi_\theta}(s)$

$$E_{\pi_\theta}[\delta^{\pi_\theta}|s,a] = E_{\pi_\theta}[r + \gamma V^{\pi_\theta}(s')|s,a] - V^{\pi_\theta}(s) = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s) = A^{\pi_\theta}(s,a)$$

Just need one set of parameters to estimate V

Deep Deterministic Policy Gradient

DDPG can be considered as a continuous action version of DQN integrated the idea of policy gradient.

$$\text{DQN: } a^* = \arg\max_a Q^*(s,a)$$

$$\text{DDPG: } a^* = \arg\max_a Q^*(s,a) \approx Q_\phi(s, \mu_\theta(s))$$

$\mu_\theta(s)$ is a deterministic policy giving the action that maximizes $Q_\phi(s,a)$.

DDPG's objective:

$$\text{Q-target: } y(r,s',d) = r + \gamma(1-d)Q_{\phi_{target}}(s', \mu_{\theta_{target}}(s'))$$

$$\text{Q-function: } \min_\phi E_{s,r,s',d\sim D}\left[Q_\phi(s,a) - y(r,s',d)\right]$$

$$\text{Policy: } \max_\theta E_{s\sim D}\left[Q_\phi(s, \mu_\theta(s))\right]$$

Reply buffer D and target networks for both Q-network are the same as DQN.

Model-based Reinforcement Learning

Model is a representation of the environment (MDP), generally includes transition and reward

$$S_{t+1} \sim P(S_{t+1}|S_t, A_t)$$
$$R_{t+1} \sim R(R_{t+1}|S_t, A_t)$$

Learn the model from $\{S_1, A_1, R_2, S_2, A_2, R_3, \dots, S_T\}$ supervised learning

$$S, A \to S', R$$
$$S, A \to S', R$$
$$\vdots$$
$$S, A \to S', R$$

$s, a \to r$ is a regression problem

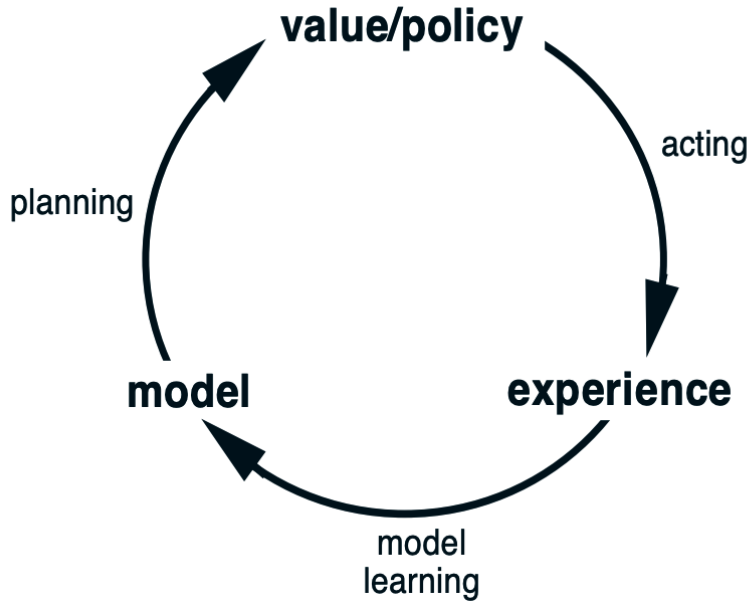
$s, a \to s'$ is a density estimation problem

Model-based Reinforcement Learning

If we have the model of the environment, we can do planning

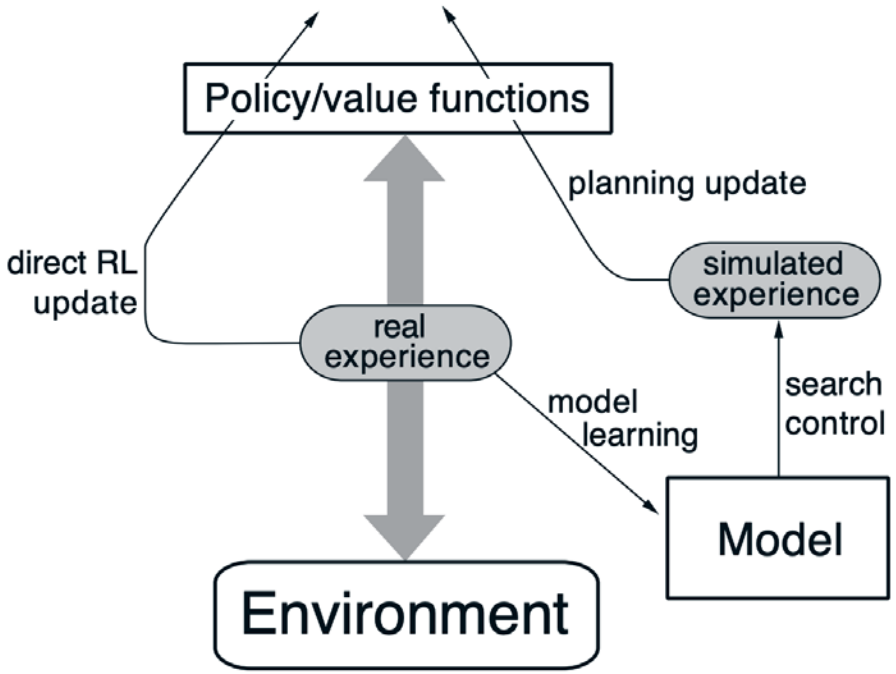
$$\text{experience} \xrightarrow{learning} \text{model} \xrightarrow{planning} \text{better policy}$$

Dynamic programming is a kind of planning



Dyna

A structure that integrates learning, planning and reacting.



two roles of real experience:

- improve the value and policy
- improve the model

Based on different RL models(MC, Sarsa, Q-learning) , there are many types of Dyna.

Here is Dyna based on Q-learning.

Tabular 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) $A \leftarrow \epsilon$-greedy(S,Q)

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

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

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

(f) Repeat n times:

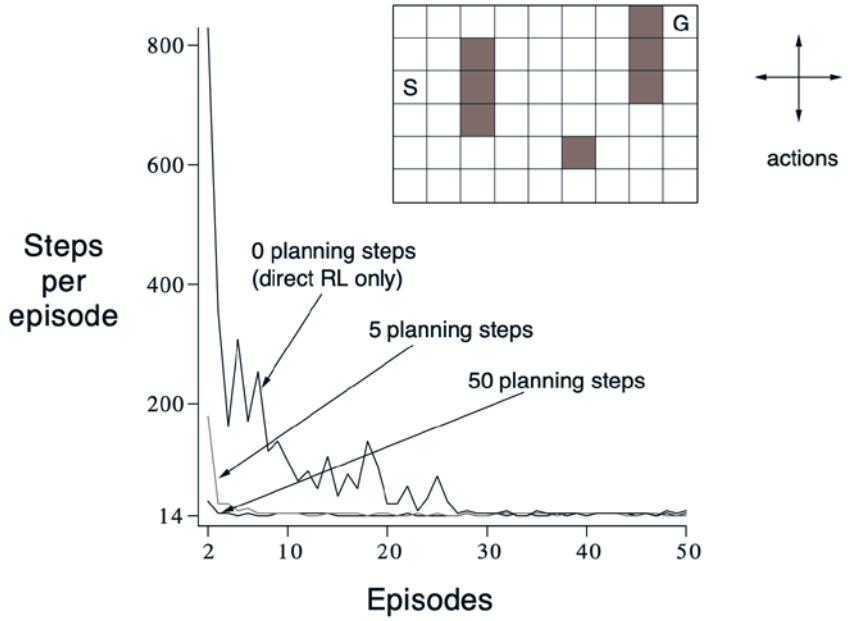
$S \leftarrow$ random previously observed state

$A \leftarrow$ random action previously taken in S

$R, S' \leftarrow Model(S,A)$

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

Dyna



Dyna-Q+ (increase exploration)

rewards: $r + \kappa\sqrt{\tau}$

- r: original reward
- κ: a small weight parameter
- τ: how long a state has not been visited