NOTES FOR REINFORCEMENT LEARNING

Yao Yao
realyao21002@gmail.com

1 Multi-arm Bandit: only one state

- ε-greedy. To find out the best arm by statistical analysis. At each step/round/time, choose the current best arm with probability (w.p.) $(1-\varepsilon)$ (**exploitation**) or randomly choose an arm w.p. ε (**exploration**). However, there is always a gap in regret.
- Upper Confidence Bound (UCB) method. To find out the best arm by statistical analysis, too. At each step/round/time, without a probability, choose the largest 'potential' arm that optimize the value function added with a **Chernoff-Hoeffding bound term**. This is to enable the reward gap to approach 0.
- Gradient-bandit method. To decide a distribution for pulling arms. A naive version of policy gradient method.

Elements: K-arm bandit, with mean $\mu_1,\dots,\mu_K$, action $A_t \in \mathscr{A} = \{1,\dots,K\}$ and R_t. At each time step compute estimation of mean - using rewards accumulated **before time T**

$$Q_T(a) = \frac{\Sigma_{t=1}^{T-1} R_t \mathbf{1}_{A_t=a}}{N_T(a)}, \text{ where } N_T(a) = \Sigma_{t=1}^{T-1}\mathbf{1}_{A_t=a}.$$

1.1 ε-greedy

Randomly pick $A_t \sim U\{1,\dots,K\}$ w.p. $\varepsilon \in (0,1)$, or pick $A_t = argmax_{a\in\mathscr{A}} Q_T(a)$ w.p. $1-\varepsilon$.
In application - generate an R.V.$\sim U(0,1)$ to ensure the exploration rate is $P(R.V. \le \varepsilon) = \varepsilon$.

$$: \text{if } R.V. \le \varepsilon, \text{ randomly choose an arm.}$$

There is an iteration in Q -

$$Q_{N+1} = \frac{(N-1)\cdot Q_N + R_N}{N} = Q_N + \frac{R_N - Q_N}{N},$$

where $N+1$ is the local time when **an arm** is being estimated, i.e. N is the last time an arm is pulled.

Bernoulli Bandit $r_i \sim Bernoulli(\mu_i)$

Define an R.V. $Y \sim U(0,1)$ and

$$X = \begin{cases} 1, & Y < \mu \\ 0, & Y \ge \mu \end{cases}$$

to ensure the reward of a Bernoulli arm is $r_i = 1$ w.p. μ_i

$$: P(X=1) = P(Y<\mu) = \mu,\ P(X=0) = P(Y \ge \mu) = 1-\mu.$$

Hyper Param Analysis

The larger ε, the more exploration, the faster it converges, but not necessarily converge to good solutions. The more exploitation, the more cumulative rewards, but the slower it converges.
Empirical good choice $\varepsilon = 0.1$ or $\varepsilon = 0.05$.

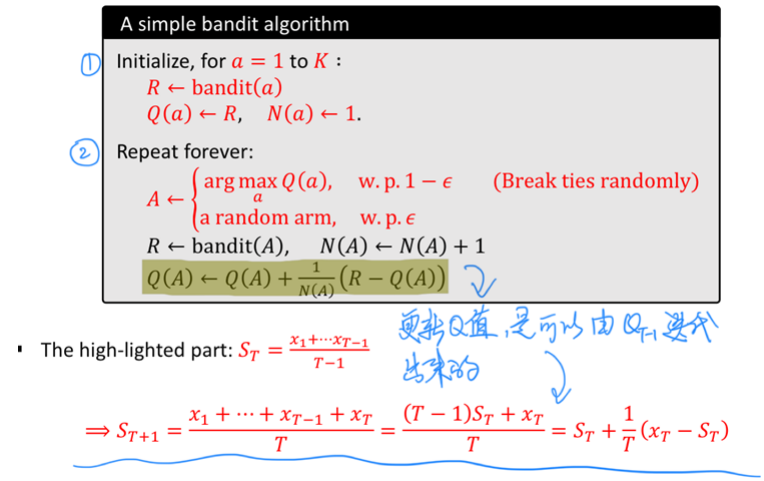

Figure 1: ε-greedy

1.2 Regret analysis: ε-greedy results in a gap

A measure of 'expected loss' compared to always pulling the optimal arm.

$$Regret(T) = T\mu^* - E[\Sigma_{k=1}^T R_{A_k}] = T\mu^* - E[\Sigma_{k=1}^T \mu_{A_k}] = T\mu^* - E[\Sigma_{a \in \mathscr{A}} N(a)\mu_a],$$

where $\mu^* = max_i\{\mu_i\}, \mu_{A_k}$ = estimated mean for arm A_k = some a, $N(a)$ = #times arm a is pulled.

If regret $\to 0$, then

$$Regret = T\mu^* - E[\Sigma_{k=1}^T \mu_{A_k}] \to 0,$$

$$AvgReward(T) = \frac{E[\Sigma_{k=1}^T \mu_{A_k}]}{T} \to \mu^*,$$

$$\frac{Regret(T)}{T} = \mu^* - AvgReward(T) \to 0.$$

In ε-greedy, assuming arm 10 as the true best

$$Regret = T\mu_{10} - \frac{9\varepsilon}{10} T\bar{\mu}_{other} - \frac{10 - 9\varepsilon}{10} T\mu_{10} = \frac{9\varepsilon T}{10}(\mu_{10} - \bar{\mu}_{other}),$$

$$AvgRegret(T) = \frac{Regret(T)}{T} = \frac{9\varepsilon}{10}(\mu_{10} - \bar{\mu}_{other}) > 0.$$

There is a non-diminishing gap.

A Naive Approach

Introducing **Chernoff-Hoeffding bound**:

$$P(|Q_n(a) - \mu_a| \le \delta) \ge 1 - 2e^{-2n\delta^2},$$

where n is the number of times arm a is pulled, μ_a is the true mean reward of arm a.

Then the confidence interval $\mu_a \in [Q_n(a) - \delta, Q_n(a) + \delta]$ is what we focus on. Suppose the gap between the true best and the true second best is known as Δ. Let $1 - 2e^{-2n(\Delta/3)^2} \ge 1 - p$. then $n \ge \frac{9\ln(2/p)}{2\Delta^2}$ would satisfies

$$\forall a \ne 10, Q_n(a) \le \mu_a + \Delta/3 < \mu_{10} - \Delta/3 \le Q_n(10) \text{ w.p. at least } 1 - p,$$

to ensure the true best arm 10 has the highest $Q_n(10)$ w.p. $1-10p$. There are 10 inequalities, where any of them does not hold w.p.

$$P(\text{any does not hold}) = P(\{1\text{st does not hold}\} \cup \cdots \cup \{10\text{th does not hold}\}) \leq 10p.$$

Then the approach is **exploration-then-exploitation** - pull each time for n times, and then always choose the arm with best estimated reward for $T-nk$ times. Note that

$$\text{let } p = 1/T, \text{ then } n = [\frac{9\ln(2T)}{2\Delta^2}].$$

Such approach has drawbacks that the user needs to know T as the number of trials, which also makes the exploration and exploitation completely separated and lack flexibility, and Δ as the gap.

1.3 The UCB Approach

This approach helps the Chernoff-Hoeffding bound inequality holds, to make the

$$\frac{Regret(T)}{T} = \mu^* - AvgReward(T) = O(1/\sqrt{T}) \to 0.$$

At each time step, choose an arm to maximize the **Upper Confidence Bound** of the previous confidence interval. Here the selected $\delta_t(a)$ varies at each time across arms, and is $\sqrt{\frac{2\ln t}{N_t(a)}}$ such that

$$1 - 2e^{-2n\delta_t^2} = 1 - 1/t^4.$$

The larger the time t grows, the higher the probability that $Q_t(a) + \delta_t(a)$ provides a true upper

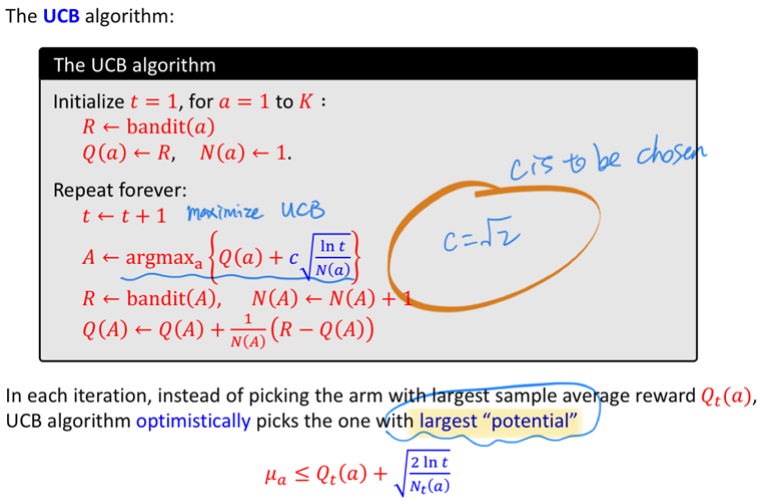

Figure 2: The UCB approach

bound for the true mean μ_a, although δ_t is changing through time.
The larger the pull number $N_t(a)$ grows, the tighter the upper bound $Q_t(a) + \delta_t(a)$ is compared with the true mean μ_a. Because $\delta_t(a)$ would decrease, letting the upper bound be closer to μ_a.
In each iteration, instead of picking the arm with largest $Q_t(a)$, UCB alg optimistically picks the one with **the largest 'potential'**.

- Every arm will be pulled infinitely. $\delta_t(a)$ gets larger even if arm a is never pulled, then arm a will be pulled until its ucb becomes the largest.

- Pulled suboptimal arm will be played less frequently, because $N_t(a)$ gives a penalty in $\delta_t(a)$.
- Overestimate rarely pulled arms. In particular, if $N_t(a) = 0$, then its ucb $= \infty$, making the unpulled arm pulled first. (However, in practice, we always initialize the first pulling of each arm.)

1.4 Gradient-Bandit

Direct optimization to optimize the distribution for pulling each arm. A naive version of the policy gradient method.

$$\begin{aligned} min_\pi & -\Sigma_{a=1}^K \mu_a \pi_a \\ s.t. & \pi \geq 0 \\ & \Sigma_a \pi_a = 1 \end{aligned}$$

Where, the objective is the negative expected reward when pulling arms according to the distribution π. To ensure $\pi_a \in [0,1]$, introduce a parameter θ_a for each arm and let

$$\pi_a(\theta) = \frac{exp(\theta_a)}{\Sigma_{b=1}^K exp(\theta_b)}.$$

The objective becomes $min_\theta\{f(\theta) : -\Sigma_{a=1}^K \mu_a \pi_a(\theta)\}$.
In such method, we update the distribution using gradient descent, which utilize the unknown μ_a. Although this is unknown, we can still have an unbiased estimator of the gradient without knowing μ_a. Fortunately,

$$\frac{\partial f(\theta)}{\partial \theta_a} = \Sigma_{b\neq a}^K \mu_b \pi_a(\theta)(\pi_b(\theta) - 0) + \mu_a \pi_a(\theta)(\pi_a(\theta) - 1) = E_{b\sim\pi(\theta), R=bandit(b)}[R \cdot (\pi_a(\theta) - \mathbf{1_{b=a}})].$$

The estimator means that, when you pull an arm b, calculate the partial derivative using the expectation for each arm a.
Then, conduct gradient descent step for each $\theta_a^{t+1} = \theta_a^t - \eta_t R_b(\pi_a^t - \mathbf{1_{a=b}})$, where π_a^t is computed at the beginning of each iteration.

Gradient-bandit algorithm

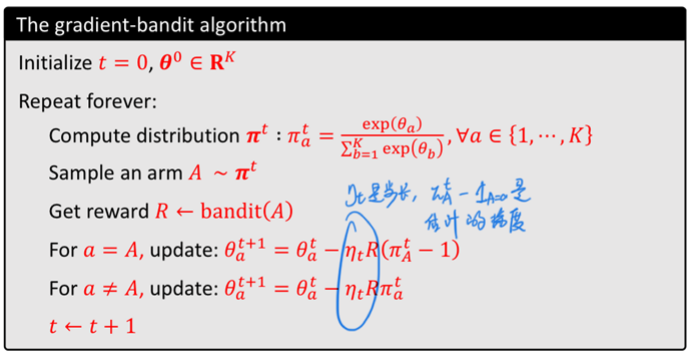

Figure 3: Gradient Bandit

- Taking smaller steps will converge to higher final average reward, but with slower convergence speed.
- Better than ε-greedy.

Add base-line to improve GB

The base-line is

$$\bar{R}_T = \frac{\Sigma_{t=1}^{T} R_t}{T}, \text{ to update baseline, } \bar{R}_{T+1} = \frac{\bar{R}_T \cdot T + R_{T+1}}{T+1} = \bar{R}_T + \frac{R_{T+1} - \bar{R}_T}{T+1}.$$

When conducting gradient descent,

$$\theta_a^{t+1} = \theta_a^t - \eta_t (R_b - \bar{R}_{t+1})(\pi_a^t - \mathbf{1}_{\mathbf{a}=\mathbf{b}}), \text{ when choosing arm } b.$$

Adding a baseline **does not change the unbiased-ness**.

- Ways to improve gradient-bandit
  - We can add a base-line to the gradient formula:

The gradient-bandit algorithm with base-line

Initialize $t = 0$, $\boldsymbol{\theta}^0 \in \mathbf{R}^K$, $\bar{R} = 0$

Repeat forever:

Compute distribution $\boldsymbol{\pi}^t$: $\pi_a^t = \frac{\exp(\theta_a)}{\Sigma_{b=1}^{K} \exp(\theta_b)}, \forall a \in \{1, \cdots, K\}$

Sample an arm $A \sim \boldsymbol{\pi}^t$

Get reward $R \leftarrow \text{bandit}(A)$

$\bar{R} = R + (R - \bar{R})/(t+1)$ $\quad \bar{R} = \frac{\bar{R} \cdot t + R}{t+1} = \bar{R} + \frac{R - \bar{R}}{t+1}$

For $a = A$, update: $\theta_a^{t+1} = \theta_a^t - \eta_t (R - \bar{R})(\pi_A^t - 1)$

For $a \neq A$, update: $\theta_a^{t+1} = \theta_a^t - \eta_t (R - \bar{R}) \pi_a^t$

$t \leftarrow t + 1$

Figure 4: gradient bandit with baseline

$$\nabla f_B(\theta) = \nabla f(\theta), \text{ and } \Sigma_{a=1}^{K} \pi_a(\theta) = 1 \Rightarrow \Sigma_{a=1}^{K} \frac{\partial \pi_a(\theta)}{\partial \theta_a} = 0.$$

Which means equally shifting the rewards of each arm by a constant B will not change the relative performance of the arms and will not change the gradients.
Adding an appropriate base-line will often improve the (overall) performance.

2 Markov Decision Process

The (finite) Markov Decision Process is written as

$$\mathscr{M}(\mathscr{S}, \mathscr{A}, r, \gamma, P).$$

2.1 MDP Elements

- State space $\mathscr{S}$ s.t. $s \in \mathscr{S}$ is a set.
- Action space $\mathscr{A}$ s.t. $a \in \mathscr{A}$. To distinguish between the action choice a and the real action $\hat{a}$.
- Reward function $r(\cdot, \cdot)$. Results from $s \in \mathscr{S}$ and $a \in \mathscr{A}$, $E[R] = r(s,a)$ or simplified $R = r(s,a)$.
- Transition probability P. A probability $P(s'|s,a)$, so $s' \sim P(\cdot|s,a)$. For each action a, we can also write $P_a(s,s')$, so $s' \sim P_a(s,\cdot)$.

- Discount factor $\gamma \in (0,1)$. We generally **don't expect a terminating state**, then a discount on future reward should be conducted. We would like to maximize the **expected discounted cumulative reward**

  $$E[\Sigma_{t=0}^{\infty}\gamma^t R_t] \text{ with } E[R_t|s_t,a_t] = r(s_t,a_t).$$

  In most cases, γ is close to 1, e.g., $\gamma = 0.95$.

- Policy $\pi : \mathscr{S} \to \Delta_{\mathscr{A}}$ is a mapping from the state space to a **distribution over the action space**. Under a policy π, the probability of an action a given a state s gives $a \sim \pi(\cdot|s)$. Given an **initial state distribution** ξ, we would like to find the optimal policy by maximizing

$$V^{\pi}(\xi) := E[\Sigma_{t=0}^{\infty}\gamma^t \cdot r(s_t,a_t)|s_0 \sim \xi, a_t \sim \pi(\cdot|s_t), s_{t+1} \sim P(\cdot|s_t,a_t)].$$

2.2 Estimating Cumulative Reward: Monte Carlo

Approach 1: generate random horizon, without computing γ

At each epoch, generate a random horizon H s.t. $H = X - 1$ with $X \sim Geom(1-\gamma)$, then $P(H \geq k) = \gamma^k$. After that, sample a finite sequence $\{s_t,a_t,R_t\}_{t=0}^{H}$ and compute $\tilde{V}_{epoch} = \Sigma_{t=0}^{H} R_t$. After epochs,take the average $\bar{V} = \frac{1}{h}\Sigma_{epoch=1}^{h}\tilde{V}_{epoch}$.
The estimator $\tilde{V}$ is unbiased,

$$E[\tilde{V}] = E[\Sigma_{t=0}^{\infty}\mathbf{1}_{t\leq H}R_t] = \Sigma_{t=0}^{\infty}P(H \geq t)E[R_t] = \Sigma_{t=0}^{\infty}\gamma^t E[R_t] = V^{\pi}(\xi).$$

Expected trajectory length is $E[H]+1 = \frac{1}{1-\gamma}$.

Alternative estimator: fixed horizon

Take a fixed horizon $H = [const \cdot \frac{ln(1/\varepsilon)}{1-\gamma}]$. Then sample a sequence $\{s_t,a_t,R_t\}_{t=0}^{H}$ and compute $\tilde{V} = \Sigma_{t=0}^{H}\gamma^t R_t$.
This estimator is biased, but the bias is very small:

$$|E[\tilde{V}] - V^{\pi}(\xi)| \leq P(\varepsilon).$$

Complimentary knowledge on horizon H

There are two types of horizon, relatively for finite and infinite MDP:

$$H = \begin{cases} \text{terminating state}, & \text{finite MDP} \\ \tau, & \text{infinite MDP} \end{cases}.$$

Fixed horizon may lead to an unexpected terminating state.

3 Value Functions

3.1 State Value Function

Definition

As the cumulative reward function is

$$V^{\pi}(\xi) = E_{s_0\sim\xi, a_t\sim\pi(\cdot|s_t), s_{t+1}\sim P(\cdot|s_t,a_t)}\Sigma_{t=0}^{\infty}\gamma^t r(s_t,a_t),$$

when the initial state s_0 is known and instead denoted by s, we can compute the state value function from such initial s.

$$V^{\pi}(s) = E_{a_t \sim \pi(\cdot|s_t), s_{t+1} \sim P(\cdot|s_t,a_t)}[\Sigma_{t=0}^{\infty} \gamma^t r(s_t, a_t)|s_0 = s].$$

Note that

$$V^{\pi}(\xi) = E_{s_0 \sim \xi}[V^{\pi}(s_0)].$$

Bellman Equation

$$\begin{aligned} V^{\pi}(s) &= \Sigma_{a \in A} \pi(a|s) \Sigma_{s' \in S} P(s'|s,a)[r(s,a) + \gamma V^{\pi}(s')] \\ &= \Sigma_{a \in A} \pi(a|s)[r(s,a) + \Sigma_{s' \in S} P(s'|s,a) \gamma V^{\pi}(s')], \forall s. \end{aligned}$$

In expectation form,

$$V^{\pi}(s) = E_{a \sim \pi(\cdot|s), s' \sim P(\cdot|s,a)}[r(s,a) + \gamma V^{\pi}(s')], \forall s.$$

$\mathbf{V}^{\pi}(\mathbf{s})$ in Terms of $\mathbf{Q}^{\pi}(\mathbf{s}, \mathbf{a})$

$$V^{\pi}(s) = \Sigma_{a \in A} \pi(a|s) Q^{\pi}(s,a) = E_{a \sim \pi(\cdot|s)} Q^{\pi}(s,a).$$

3.2 State Action Value Function (Q Function)

Definition

$$Q^{\pi}(s,a) := E_{s_{t+1} \sim P(\cdot|s_t,a_t), a_t \sim \pi(\cdot|s_t)}[\Sigma_{t=0}^{\infty} \gamma^t r(s_t, a_t)|s_0 = s, a_0 = a], \forall s, a.$$

Bellman Equation

$$\begin{aligned} Q^{\pi}(s,a) &= \Sigma_{s' \in S} P(s'|s,a) \Sigma_{a' \in A} \pi(a'|s')[r(s,a) + \gamma Q^{\pi}(s', a')] \\ &= r(s,a) + \Sigma_{s' \in S} \Sigma_{a' \in A} P(s'|s,a) \pi(a'|s') \gamma Q^{\pi}(s', a'). \end{aligned}$$

In expectation form,

$$Q^{\pi}(s,a) = r(s,a) + \gamma E_{s' \sim P(\cdot|s,a), a' \sim \pi(\cdot|s')}[Q^{\pi}(s', a')].$$

$\mathbf{Q}^{\pi}(\mathbf{s}, \mathbf{a})$ in Terms of $\mathbf{V}^{\pi}(\mathbf{s})$

$$\begin{aligned} Q^{\pi}(s,a) &= \Sigma_{s' \in S} P(s'|s,a)[r(s,a) + \gamma V^{\pi}(s')] \\ &= r(s,a) + \gamma E_{s' \sim P(\cdot|s,a)}[V^{\pi}(s')]. \end{aligned}$$

3.3 Optimal Value Function

To obtain the optimal policy π^*, maximize the cumulative reward function

$$max_{\pi} V^{\pi}(\xi) := E_{\xi, \pi, P}[\Sigma_{t=0}^{\infty} \gamma^t r(s_t, a_t)].$$

The obtained π^* will **simultaneously optimize all the value functions and Q functions**.
In such a policy,

$$\pi^*(a|s) = \begin{cases} 1, & a = a^*(s) \\ 0, & a \neq a^*(s) \end{cases}.$$

Where

$$a^* = argmax_{a\in A}\{Q^*(s,a)\} = argmax_{a\in A}\{r(s,a) + \gamma\Sigma_{s'\in S}P(s'|s,a)V^*(s')\}.$$

Optimal State Value Function

$$V^*(s) = max_{a\in A}\{r(s,a) + \gamma\Sigma_{s'\in S}P(s'|s,a)V^*(s')\} = max_{a\in A}\{Q^*(s,a)\}.$$

Optimal State Action Value Function

$$\begin{aligned} Q^*(s,a) &= r(s,a) + \gamma \cdot max_{a'\in A}\{\Sigma_{s'\in S}P(s'|s,a)Q^*(s',a')\} \\ &= r(s,a) + \gamma\Sigma_{s'\in S}P(s'|s,a) \cdot max_{a'\in A}\{Q^*(s',a')\} \end{aligned}$$

3.4 Notes on Solving the Optimal Policy

Note: Solving **Bellman Equations** for $V^*(s)$ or $Q^*(s,a)$ would derive the optimal policy π^*. However, how to solve those Bellman Equations depends on premise assumptions.

- If we assume known transitions and terminating state (i.e. with a horizon), we use dynamic programming, which problem is always called **planning**. This is the focus of section 4.
- If we meet with an infinite-horizon MDP without terminating state, we use value iteration or Q iteration.

4 Finite Horizon MDP: Dynamic Programming

The finite horizon maybe 1) time horizon, or 2) terminating state. (see 2.2)
Note: When setting up, invalid action will be presented by $reward = -\infty$. In practice, although all the states share the same action set, it is OK for different state to have different action sets.

4.1 Example: Grid World Navigation - Terminating State

- State $s = (i, j)$ as the position.
- Action $a \in \{R, D\}$.
- Transition. The transition $s, a \to s'$ is deterministic, i.e. $P(s'|s,a) = 1$. Therefore, to simplify, use the transition mapping $s' = trans(s,a)$. Note that

  $$trans((i,j,),R) = (i, min\{4, j+1\}),$$

  $$trans((i,j),D) = (min\{4, i+1\}, j).$$

  Then it can be seen that state $(4,4)$ is absorbing, which state once hit will not change forever.
- Reward $r(s,a)$ should take the **penalty for invalid state** and take the **terminating state** into consideration.

  $$r((i,j),R) = w[trans((i,j),R)] - \infty \cdot \mathbf{1}_{j=4},$$

  $$r((i,j),D) = w[trans((i,j),D)] - \infty \cdot \mathbf{1}_{i=4}$$

  (where w is the reward number on a grid (i,j)). For terminating state,

  $$r((4,4),a) = 0, \forall a \in \{R, D\}.$$

  Note: once hit the terminating state, the reward received afterwards would be 0 forever.

Solving Bellman Equations

In such a problem, the probability in the original Bellman Equations would be deterministic as 1 or 0, and we also set $\gamma = 1$ -

$$\begin{aligned} V^*(s) &= max_{a\in A}\{r(s,a) + \gamma V^*(s')\} \\ &= max_{a\in A}\{r(s,a) + V^*(trans(s,a))\}. \end{aligned}$$

In order to conduct **dynamic programming**, start from the terminating state $(4,4)$ with $V^*(4,4) = 0$. Then, in the Bellman Equation above, the $r(s,a)$ and the $V^*(s')$ are all known, enabling us to solve the the previous state $V^*(s) = V^*(4,3)$ and $V^*(s) = V^*(3,4)$.
Solving those equations yield the optimal policy -

$$\pi^*(a^*|s) = 1, \forall s.$$

Note: for some state, we would yield multiple optimal action a^*, then randomly choose one. In this example, $\pi^*(\cdot|1,1)$ can be any distribution, for R and D are equally optimal.

4.2 Time Horizon Instead of Terminating State

Horizon of $t = H$ can be view as a special case of MDP with terminating state. **Time can be encoded into the states,** i.e., let $s = (\hat{s},t) \sim S = \hat{S} \times N$ where $\hat{s} \in S$ and $t \in N$.
In the previous grid world example, state incorporating time is not necessary.
Remind: state incorporating into time information is always by default in many papers, which use π^t to embrace such.
Note: in such cases, all states with $t = H$ are terminating states. Transitions become $P(\cdot|s_t,t,a_t)$.
Also, take $\gamma = 1$, then the cumulative reward function will be

$$V^\pi(\xi) = E_{\xi,\pi,P}[\Sigma_{t=0}^{H-1} r(s_t,a_t)].$$

Maximizing the function above would yield an optimal policy that **depends on time** $a_t \sim \pi(\cdot|s_t,t)$.
Note: whether taking the optimal policy or not,

$$V^\pi(\hat{s},H) = V^*(\hat{s},H) = 0, \forall \hat{s} \in \hat{S};$$

$$Q^\pi(\hat{s},H,a) = Q^*(\hat{s},H,a) = 0, \forall \hat{s} \in \hat{S}, a \in A.$$

Therefore, the Bellman Equations can be solved **level by level**, i.e., **at each time step**. Knowing

$$V^*(\hat{s},H-1) = max_{a\in A}\{r(\hat{s},a) + \gamma \Sigma_{\hat{s}'\in\hat{S}} P(\hat{s}'|\hat{s},H-1,a)V^*(\hat{s}',H)\}.$$

To conduct the **dynamic programming**, also start from the terminating state, i.e., level H - $V^*(\hat{s},H) = 0, \forall \hat{s} \in \hat{S}$. Then, assuming known reward function $r(\hat{s},a)$ and transition probability $P(\hat{s}'|\hat{s},H-1,a)$, as well as the solved $V^*(\hat{s}',H)$, we can solve the previous level $H-1$ states.

4.3 Example: Inventory Management - Time Horizon

- State $s = (n,t)$ as the inventory **at end of** the month t. Assuming no inventory at the beginning of month 1, i.e., $s_0 = (0,0)$ as the initial state. Therefore, $S = \{(n,t) : n \in \{0,\ldots,4\}, t \in \{1,2,3,4\}\} \cup \{(0,0)\}$.

- Action $a \in \{0,\ldots,5\}$ as the number of productions **at the beginning of next moth**. To satisfy the **demand and capacity constraints**,

$$a+n \geq d_{t+1}, a+n-d_{t+1} \leq K = 4.$$

- Transition is also **deterministic**. Simply use $s' = trans(s,a)$ as the transition mapping for $t = 0,1,2,3$ (**not considering transition for** $t = 4$). Let

$$trans((n,t),a) = (proj_{[0,4]}(n+a-d_{t+1}), t+1)$$

- Reward $r((n,t),a)$ in terms of $g = n+a-d_{t+1}$.

$$r((n,t),a) = -0.5 \times Proj_{[o,4]}(g) - 1 \times a - 3 \times \mathbf{1}_{a>0} - \infty \times \mathbf{1}_{g \notin [0,4]}.$$

  to **minimize the total cost**, no revenue is considered, but a penalty for over-capacity inventory is imposed.

- Discount factor $\gamma = 1$ also.

- Time horizon $H = 4$.

4.4 Example: Coin Picking Game - nondeterministic transitions

- State $s = (i,j)$ as the remaining coin from number i to number j. After each step two coins will be picked, therefore the number of remaining coins is even. $S = \{(1,2),(2,3),(3,4),(4,5),(5,6),(1,4),(2,5),(3,6),(1,6$ The terminating state is the null subset O.

- Action $a \in \{L,R\}$.

- Transition is **not deterministic**. Consider from the terminating state:

$$P(O|O,a) = 1, \forall a;$$

$$P(O|(i,i+1),a) = 1, \forall a;$$

$$P((i+2,j)|(i,j),L) = P((i+1,j-1)|(i,j),L) = 0.5,$$

$$P((i,j-2)|(i,j),R) = P((i+1,j-1)|(i,j),R) = 0.5.$$

- Reward is the value of coin picked.

$$r(O,L) = r(O,R) = 0;$$

$$r((i,j),L) = v_i, r((i,j),R) = v_j.$$

The Bellman Equations with $\gamma = 1$

$$V^*(s) = max_{a \in A}\{r(s,a) + \Sigma_{s' \in S} P(s'|s,a) V^*(s')\}.$$

To conduct dynamic programming, start from the terminating state -

$$V^*(O) = 0.$$

Then for the previous states, solve all the $V^*(i,i+1)$.

5 Infinite Horizon MDP: Value and Q Iterations

The infinite-horizon MDPs are without any terminating state **(transition dynamics are still known)**, and the corresponding approach is value iteration and Q iteration.
Value iteration and Q iteration are iterations using Bellman equations and Bellman operators for

$$V^{\pi}(\cdot),\ V^{*}(\cdot),\ Q^{\pi}(\cdot),\ \text{and}\ Q^{*}(\cdot).$$

Recall that solving V^* and Q^* would derive the optimal policy π^*.

5.1 Fixed-point problem of Bellman operators

Using Bellman operators, we try to solve fixed-point problems including

- Value iteration.

$$V^{\pi} = T^{\pi}V^{\pi}$$

$$V^{*} = T^{*}V^{*},$$

$$\text{where } T : R^{S} \to R^{S},$$

  and we can write the function for $[T^{\pi}V^{\pi}]_s$ and $[T^{*}V^{*}]_s$ as

$$[T^{\pi}V^{\pi}]_s := \Sigma_{a\in A}\pi(a|s)[r(s,a) + \gamma\Sigma_{s'\in S}P(s'|s,a)V^{\pi}_{s'}] = V^{\pi}_{s}$$
$$[T^{*}V^{*}]_s := \max_{a\in A}\{r(s,a) + \gamma\Sigma_{s'\in S}P(s'|s,a)V^{*}_{s'}\} = V^{*}_{s}.$$

- Q iteration.

$$Q^{\pi} = T^{\pi}Q^{\pi}$$

$$Q^{*} = T^{*}Q^{*},$$

$$\text{where } T : R^{S\times A} \to R^{S\times A},$$

  and we can write the function for $[T^{\pi}Q^{\pi}]_{s,a}$ and $[T^{*}Q^{*}]_{s,a}$ as

$$[T^{\pi}Q^{\pi}]_{s,a} := r(s,a) + \gamma\Sigma_{s'\in S}P(s'|s,a)\Sigma_{a'\in A}\pi(a'|s')Q^{\pi}_{s',a'} = Q^{\pi}_{s,a}$$
$$[T^{*}Q^{*}]_{s,a} := r(s,a) + \gamma\Sigma_{s'\in S}P(s'|s,a)\max_{a'\in A}\{Q^{*}_{s'a'}\} = Q^{*}_{s,a}.$$

5.2 Fixed-point Solution: Contraction Property

Let $0 < \gamma < 1$, the mapping $f(\cdot)$ is a $\gamma-$**contraction** under the norm $||\cdot||$ if

$$||f(x) - f(y)|| \leq \gamma||x-y||, \forall x, y.$$

Subsequently, the solution to the fixed-point problem $x = f(x)$ can be obtained by the **fixed-point iteration**.
Given the fixed-point problems of value and Q iterations, it can be proved that the Bellman operators are $\gamma-$**contraction** under the **infinity** norm, which is defined as $||x||_{\infty} := \max_i |x_i|$. Because they are $\gamma-$contraction, the iteration will converge to the true solutions.

$$||x^{t} - x^{*}|| = ||f(x^{t-1}) - f(x^{*})|| \leq \gamma||x^{t-1} - x^{*}|| \leq \gamma^{2}||x^{t-2} - x^{*}|| \cdots \leq \gamma^{t}||x^{0} - x^{*}||,$$
$$\Rightarrow \lim_{t\to\infty} x^{t} = x^{*}.$$

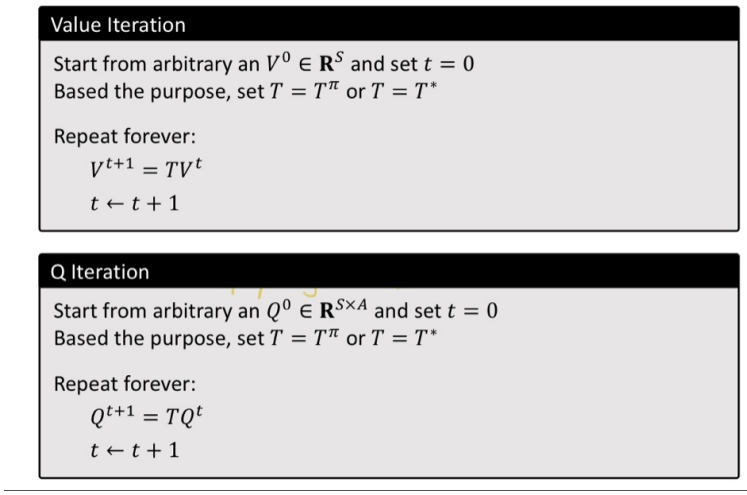

Figure 5: Value and Q Iteration

6 Searching Optimal Policy More Efficiently: Policy Iteration

When the state/action space is too large, using value or Q iteration to yield every Q^* before selecting the optimal policy is not that efficient. In this case, it is more efficient when using policy iteration. Note that the case here is still for known transition dynamics, so policy iteration is a model-based method.

It can be proved that in policy iteration, $\forall s$, the value function is monotonically increasing, i.e., $V^{\pi_k}(s) \leq V^{\pi_{k+1}}(s), \forall k \geq 0$.

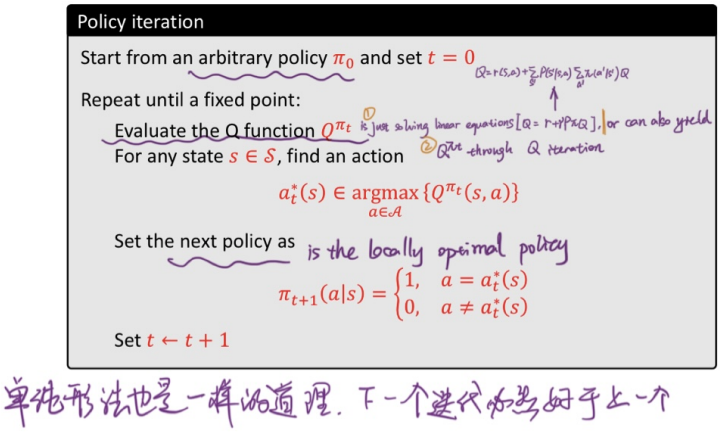

Figure 6: Policy Iteration

7 Monte Carlo Prediction and Control

When transition dynamics are unknown, model-free prediction and control will be utilized. They include Monte Carlo methods, TD methods and Fitted Q Iteration, whose **control methods are all action-value methods based on the Q function**.

- Monte Carlo methods are **tabular methods**.
- TD methods can be **tabular methods** for discrete state/action space, including SARSA and Q-learning, both using TD error to update Q functions, but with different way of updating policy.
- TD can also be combined with **function approximation** for continuous or large-scale state/action space.
- Fitted Q iterations is relevant to **function approximation** for continuous or large-scale state/action space.

To distinguish between prediction and control, Monte-Carlo methods for finding $Q \approx Q^{\pi}$ is prediction to evaluate a policy, Monte-Carlo methods for finding $\pi \approx \pi^{*}$ is control to improve the policy. Subsection 7.1 is about value **prediction** (policy evaluation), and subsection 7.2 and 7.3 are about policy **control**. Note that ε-greedy is a method for policy improvement, and is conducted on action choosing, so it is not related to the prediction part. In addition, subsection 7.4 introduce the off-policy Monte Carlo **estimation and control**, where control is just the estimation and improvement of the optimal policy.

7.1 Monte Carlo Prediction (Exploring Start)

Monte-Carlo sampling method can be applied to policy evaluation, namely, find the Q^{π} for a policy π. This algorithm will **converge** to the true Q-function.

Monte-Carlo Method (Exploring Start), for finding $Q \approx Q^{\pi}$

Initialize $Q(s,a) = N(s,a) = 0$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$.

Repeat forever:

Choose $(s_0, a_0) \in \mathcal{S} \times \mathcal{A}$ randomly s.t. all state-action pairs has non-zero probability to be observed. Generate an episode by policy π: $\{s_0, a_0, R_0, s_1, a_1, R_1, \cdots, s_T, a_T, R_T\}$. Set $G \leftarrow 0$.

Loop for $t = T, T-1, \cdots, 0$:

Set $G = \gamma G + R_t$

If (s_t, a_t) does not appear in $\{(s_k, a_k, R_k)\}_{k=0}^{t-1}$:

Update $Q(s_t, a_t) \leftarrow \frac{Q(s_t,a_t)N(s_t,a_t)+G}{N(s_t,a_t)+1}$

Update $N(s_t, a_t) = N(s_t, a_t) + 1$

Figure 7: Monte-Carlo (Exploring Start) for Evaluation

7.2 Monte Carlo Control (Exploring Start)

With A Terminating State

We should note that when updating $Q(s_t, a_t)$ at each $t = T, T-1, ..., 0$, T is not a fixed period, but if a terminating state appears at time $T' = T+1$, which means T is the state just happen before S_{end}, we select the trajectory with length $= T$.

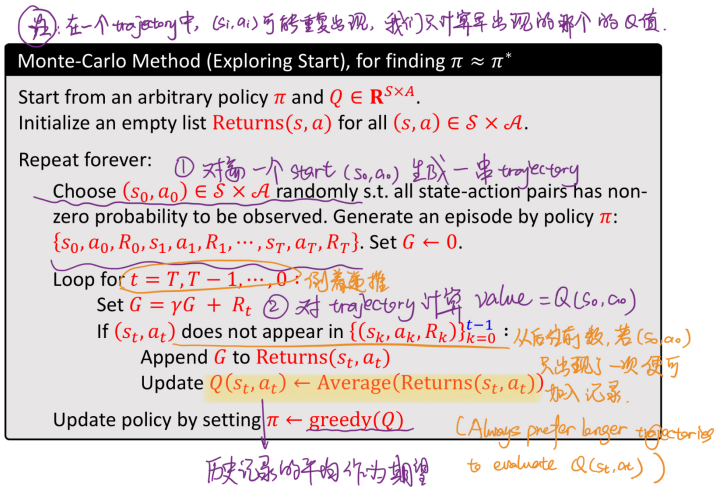

Figure 8: MCES w termination

Without A Terminating State

In this case, we set a horizon H. The bias brought by H

$$|Q^\pi(s,a) - E_\pi[\Sigma_{t=0}^{H}\gamma^t r_t]| = |E_\pi[\Sigma_{t=H+1}^{\infty}\gamma^t r_t]| \leq \Sigma_{t=H+1}^{\infty}\gamma^t c = \frac{\gamma^{H+1}c}{1-\gamma}$$

would vanish as H is set to be large. Usually, setting $H = O(\frac{1}{1-\gamma})$ is enough.

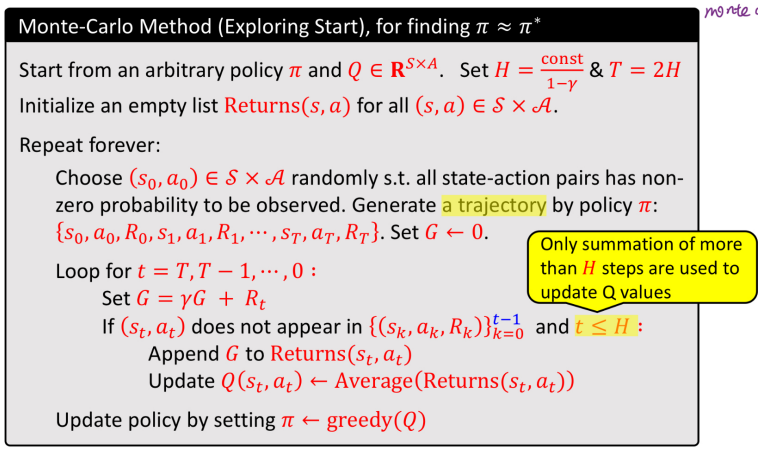

Figure 9: MCES w horizon

Estimation of $V^\pi(s)$

1. Start from the initial state s and sample N episodes, for which the discounted total rewards are computed and averaged as the final estimation. - if N is large enough, the estimation is accurate but much costly.

2. Directly use the training reward G as the estimation for each iteration. - this is not the estimator of $V^\pi(s)$, but the estimator of $V_\xi^\pi(s) = E_{s\sim\xi}V^\pi(s)$. This is **almost unbiased** for estimating V_ξ^π, but has **large variance**.

3. Use the running average of the value G obtained from iterations. - This has much smaller variance than method 2, but is biased before the algorithm converges.

Problems of MCES

- The algorithm may get stuck at a sub-optimality policy.
- MCES does not necessarily converge.
- The estimators are biased and noisy.
- May not be able to restart exactly from some arbitrary state action pairs.

7.3 Monte Carlo Control (ε-greedy)

With A Terminating State

To address the problems of MCES, it is not advisable to completely exclude exploration, because one will get trapped at some policy without chances of observing other state-action pairs. A more desirable method is Monte Carlo sampling with ε-greedy. In this approach, ε-greedy is used in choosing action and thus next state, but not used to initialize the system state.

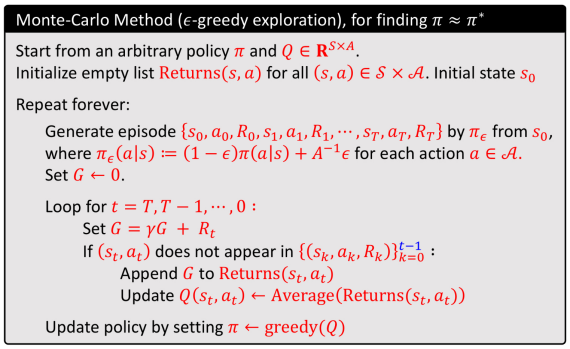

Figure 10: MC ε-greedy w termination

Without A Terminating State

Similarly, without a terminating state, a horizon should be set.

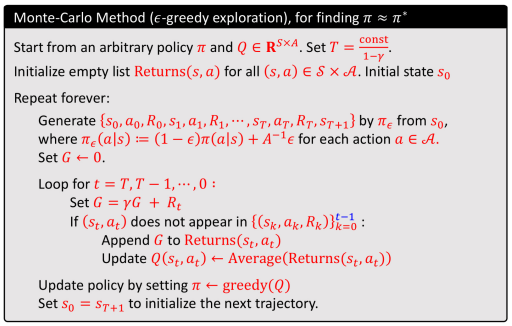

Figure 11: MC ε-greedy w horizon

7.4 Off-policy Monte Carlo Control

Terminology

- Target Policy: π which we want to evaluate, often an **exploiting policy**
- Behavioral Policy: π_b which we use to collect data, often taken as an **exploring policy**

Importance Sampling: a premise

Sampling a random variable X can be realized by sampling another random variable Y, as long as their probability functions are known. Similarly, evaluate the **target policy** can be realized by sampling from another policy named **behavioral policy**.
Define the **support** of a **probability density function** $u(x)$ (for discrete R.V., it is the **probability mass function**) as

$$supp(u) := \{x \in R^n : u(x) \neq 0\}.$$

Then the estimation of $f(X)$ can be determined by that of $f(Y)$ because

$$\begin{aligned} E_p[f(X)] : &= \int_{R^n} f(x)p(x)dx = \int_{R_n} f(y)p(y)dy \\ &= \int_{R^n} \frac{f(y)p(y)}{q(y)} q(y)dy \\ &= E_q[\frac{f(Y)p(Y)}{q(Y)}], \end{aligned}$$

where p and q are the pdf of X and Y respectively.
Therefore, the estimation of $f(X)$ can be yielded by the estimation of $\frac{f(Y)p(Y)}{q(Y)}$, which is computed as follow

$$\hat{S}_N = \frac{1}{N}\Sigma_{i=1}^N \frac{f(Y_i)p(Y_i)}{q(Y_i)},$$

and it is clear that

$$E_q[\hat{S}_N] = E_p[f(X)].$$

Off-policy Estimation

Directly estimating the target policy with data from behavior policy is rough and biased.
To conduct the off-policy estimation, the premise is a key requirement on the behavioral policy - reasoning just like ε-greedy that increase the exploration on unobserved (s,a),
The support of π_b must cover all state-action pairs, i.e.,

$$supp(\pi) = \mathscr{S} \times \mathscr{A}$$

$$\pi_b(a|s) > 0, \forall (s,a) \in \mathscr{S} \times \mathscr{A}.$$

- With this premise, there is no need to conduct ε-greedy because the requirement adds to explorations.
- With this premise, we aim to calculate the importance ratio for π to π_b.

Suppose a trajectory $\tau := \{s_t, a_t, r_t\}$. The distribution of τ collected by π_b is

$$P(\tau|s_0, a_0, \pi_b) = \Pi_{t=1}^{T-1} P(s_t|s_{t-1}, a_{t-1})\pi_b(a_t|s_t).$$

The distribution of τ collected by π is

$$P(\tau|s_0,a_0,\pi) = \Pi_{t=1}^{T-1} P(s_t|s_{t-1},a_{t-1})\pi(a_t|s_t).$$

Let $f(\tau) = \Sigma_{t=0}^{T-1}\gamma^t r_t$, then by the theorem of importance sampling,

$$\begin{aligned} Q^\pi(s_0,a_0) &\approx E[f(\tau)|s_0,a_0,\tau\sim\pi] \\ &= E[\frac{f(\tau)P(\tau|s_0,a_0,\pi)}{P(\tau|s_0,a_0,\pi_b)}|s_0,a_0,\tau\sim\pi_b], \end{aligned}$$

where the importance ratio is the term

$$\frac{P(\tau|s_0,a_0,\pi)}{P(\tau|s_0,a_0,\pi_b)} = \Pi_{t=1}^{T-1}\frac{\pi(a_t|s_t)}{\pi_b(a_t|s_t)}.$$

Note that when estimating the π in each iteration in this algorithm, we multiply the reward G_t with the importance ratio C before taking the average of the rewards to get the final estimation.

Off-policy Monte-Carlo Method, for finding $\pi \approx \pi^*$

Start from an arbitrary policy π and $Q \in \mathbf{R}^{S\times A}$ and behavioral policy π_b
Initialize empty list $\text{Returns}(s,a)$ for all $(s,a) \in \mathcal{S}\times\mathcal{A}$. Initial state s_0

Repeat forever:

Generate episode $\{s_0,a_0,R_0,s_1,a_1,R_1,\cdots,s_T,a_T,R_T\}$ by π_b from s_0.
Set $G \leftarrow 0$ and $C \leftarrow 1$

Loop for $t = T, T-1, \cdots, 0$:

Set $G = \gamma G + R_t$

If $t \le T-1$: set $C = \frac{\pi(a_{t+1}|s_{t+1})}{\pi_b(a_{t+1}|s_{t+1})} \times C$

If (s_t,a_t) does not appear in $\{(s_k,a_k,R_k)\}_{k=0}^{t-1}$:

Append $G \times C$ to $\text{Returns}(s_t,a_t)$

Update $Q(s_t,a_t) \leftarrow \text{Average}(\text{Returns}(s_t,a_t))$

Update policy by setting $\pi \leftarrow \text{greedy}(Q)$

Figure 12: Off-policy MC w termination

Potential Problems

1. Note that the target policy here is greedy(Q), an optimal policy.
2. Since the optimal policy is $\pi(a^*|s_t) = 1$ and $\pi(a|s_t) = 0$ for $a \neq a^*$, then in a trajectory τ, if $\pi(a_k|s_k) = 0$ for some k (i.e., explore an action that does not follow the target policy π), the importance ratio would be zero. This means one shall never learn $Q^\pi(s_t,a_t)$ for $t \le k$ and the algorithm only learns from the tail data (that occurs after k).
3. Due to the product of C, the estimator $G\cdot C$ has an extremely large variance. It can be proved that the variance of such an estimator is **exponentially high**, which is a typical issue of **almost all off-policy methods that is based on the importance sampling**.

8 Temporal Difference Prediction and Control

Just like Monte Carlo methods, TD can be used either in prediction (value estimation/policy evaluation) or in control (policy improvement).

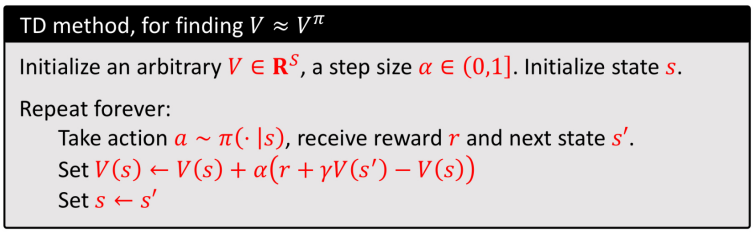

Figure 13: TD Estimation

When using TD error to improve the policy and find the optimal policy, we have two algorithms including SARSA and Q-learning, as shown in subsection 8.2 and 8.3 respectively. The difference between SARSA and Q-learning is that SARSA applies the ε-greedy policy to sample and evaluate it by **one step of TD update**; and that Q-learning adopts the greedy estimation of the Q function, which does not follow the sampling policy (ε-greedy).

8.1 TD Estimation

Temporal difference learning method is a **hybrid** of both fixed-point iteration (i.e., value and Q iteration) and the weighted average version of Monte-Carlo sampling method. Distinguish between TD estimation and weighted average of MC estimation:

- WM-MC: $V^{t+1}(s_t) = (1-\alpha)V^t(s_t) + \alpha G_t$, where $G_t = R_t + \Sigma_{i=1}^{h} \gamma^i R_{t+i}$ is the true discounted reward from t on.
- TD: $V^{t+1}(s_t) = (1-\alpha)V^t(s_t) + \alpha(R_t + \gamma V^t(S_{t+1}))$, where $V^t(s_{t+1})$ is the previous estimation for next state.

In the algorithm of TD estimation, if there is some terminating state s', we default $V(s') = 0$ and replace the step $s \leftarrow s'$ with "reinitialize s".

Convergence of TD Estimation

TD estimation, as one of the stochastic methods, does not converge exactly to V^{π} as value iteration, but converges to a **neighborhood of** V^{π}. To compare between them:

- Value iteration: update $V(s)$ with $V(s) + \alpha(E_{a\sim\pi,s'\sim P}[r + \gamma V(s')] - V(s))$, where the first term inside the brackets after α is the estimation of $r + \gamma V(s')$.
- TD: update $V(s)$ with $V(s) + \alpha(r + \gamma V(s') - V(s))$, where the first term inside the brackets after α is the observed experimental value of $r + \gamma V(s')$.

Therefore, the random fluctuation of the convergence comes from the error of

$$E_{a\sim\pi,s'\sim P}[r + \gamma V(s')] - (r + \gamma V(s'))$$

.

Note 1: it can be proved that the error would **shrink a little bit for the observed state** s_t but **remain unchanged for the unobserved states**. This mean there exists an upper bound of the error for those unobserved states.
Note 2: after τ iterations when every state is visited, the **squared error for each state** would

converges to an $O(\alpha)$ neighborhood of 0 in $\tilde{O}(\frac{\tau}{\alpha(1-\gamma)})$.
Note 3: the error/gap between $V^{t+1} - V^{\pi}$ is $O(\sqrt{\alpha})$. If the rewards are bounded by $O(1)$, then the error neighborhood would be $O(\sqrt{\frac{\alpha}{(1-\gamma)^3}}$, which means the step size α should be small. It is better to let α diminish to 0.
Note 4: the sample complexity will be $\tilde{O}(\frac{|S|\cdot|A|}{(1-\gamma)^4\varepsilon^2}$, which is the typical sample complexity for solving RL with this type of algorithm.
Example: random walk. This is an MRP without choosing any action. Experimental results show that (1) TD needs a large number of episodes to approach the true policy estimated by value iteration. (2) Besides, TD's empirical RMSE averaged over states would drop more quickly and be smaller than that of MC algorithm.

8.2 SARSA Control: on-policy learning

Remember that control is to learn the optimal value or Q functions. SARSA is an **on-policy** control, where it sample data based on the target policy, and it estimates the **Q-functions**.
SARSA applies the ε-greedy policy:

$$\pi(a|s) = \begin{cases} (1-\varepsilon) + \varepsilon/A & \text{if } a = a^*(s) \\ \varepsilon/A & \text{if } a \neq a^*(s) \end{cases}$$

where $a^*(s) = \arg\max_{a \in A} Q(s,a)$.
Here, $1/A = Unif(A)$, i.e., choosing each action with the same probability.
In the SARSA algorithm, when there is a terminating state s'', default $Q(s'',a) = 0, \forall a$ and update

SARSA method, for finding $\pi \approx \pi^*$

Initialize an arbitrary $Q \in \mathbf{R}^S$, a stepsize $\alpha \in (0,1]$. Set $\pi \leftarrow \epsilon$-greedy(Q)
Initialize state s, take $a \sim \pi(a|s)$, receive reward r and observe s'.

Repeat forever:
Take action $a' \sim \pi(\cdot\,|s')$, receive reward r' and next state s''.
Set $Q(s,a) \leftarrow Q(s,a) + \alpha\big(r + \gamma Q(s',a') - Q(s,a)\big)$
Set $\pi \leftarrow \epsilon$-greedy(Q)
Set $s \leftarrow s', a \leftarrow a', r \leftarrow r', s' \leftarrow s''$

Figure 14: SARSA

the (s,a,r,s') with reinitializing s, taking $a \sim \pi(a|s)$, receiving reward r and observing s'.
Note:

1. Diminishing espsilon would improve the policy learned by SARSA.
2. If we want to plot the reward history to evaluate the performance of the algorithm, use an additional evaluation function $evaluate()$ to **estimate $V^{\pi}(s)$ based on Q values**.
3. For the function $evaluate()$: In this algorithm, the training reward is very noisy due to huge variance. We have to report the progress by simulating the **expected reward** with a lot of repeated sampling.
4. Example: Cliff walking. (1) Using a fixed ε, SARSA yields the safetest but conservative path. (2) Using a diminishing ε, SARSA yields a middle path between the optimal (most aggressive) and the safest (most conservative) path.

8.3 Q-learning: off-policy learning

Q-learning is an off-policy learning, and adopt the **greedy estimation** of the Q function.

- Sampling (behavioral) policy: ε-greedy. This policy only act the role of exploration.
- Estimation (target) policy: $Greedy(Q)$

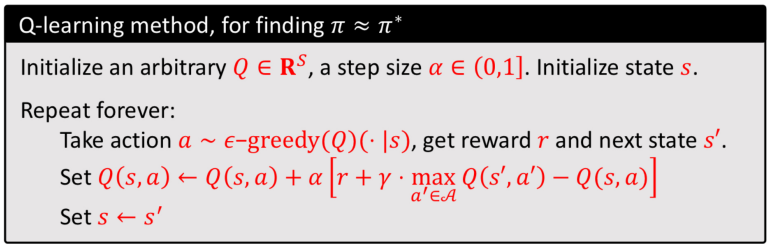

Figure 15: Q-learning

In the Q-learning algorithm, when there is a terminating state s', default $Q(s',a) = 0, \forall a$, and update the state with initializing state s. **Notes:**

1. Example: Cliff walking. Q-learning yield the most aggressive and optimal trajectory.
2. When evaluate the performance of the training reward, it can be seen that Q-learning has worse reward history than SARSA, because Q-learning as an aggressive method would always fail in the training process, while SARSA is more conservative.
3. A diminishing ε could be a possible fix to this issue.
4. SARSA has better training reward but has worse final policy. Q-learning has worse training reward but has better final policy.

9 Policy Gradient: Parameter θ For Policies

Methods in the previous sections are all based on Q functions. When the state/action space is to large or continuous, we may consider policy gradient, which learns a parameterized policy that can select actions **without consulting the Q functions**.
Parameterize the policy with π_θ, then the expected total discounted reward is a function of θ:

$$J(\theta) = V^{\pi_\theta}(\xi) = E_{\pi_\theta,\xi}[\Sigma_{t=0}^{\infty}\gamma^t r(s_t,a_t)].$$

To maximize the policy evaluation, we can apply stochastic gradient **ascent** to θ.

9.1 Softmax Policy Parameterization: discrete space

Take ϕ as a feature map from (s,a) and h is the linear reward function incorporating θ, then

$$h(s,a;\theta) = \phi(s,a)^T\theta,$$

$$\pi_\theta(a|s) = \frac{\exp(h(s,a;\theta)}{\Sigma_{a' \in A}\exp(h(a,a';\theta)}.$$

Note that h can also take the form of $f(\phi(s,a);\theta)$ which is a neural network.
For finite state/action space, we can have the **tabular softmax parameterization**, i.e., $h(s,a;\theta)$ directly substituted by the (s,a)-th element in the table θ.
Note: any deterministic poicy can be approximated by some tabular softmax policy.

9.2 Gaussian Policy Parameterization: continuous space

For continuous cases, we should utilize the mean and covariance of the action choice, which is $h(s;\theta)$ and Σ respectively. In such policies, $a \sim N(h(s;\theta),\Sigma)$ at state s.

$$\pi_\theta(a|s) = \frac{1}{\det(\Sigma)^{d/2}} \exp(-\frac{(a-h(s;\theta))^T\Sigma^{-1}(a-h(s;\theta))}{2}).$$

Note that the action space can be presented by vectors in R^d, because of which we can yield the vector-form mean $h(s;\theta)$ and the matrix-form covariance Σ.

9.3 REINFORCE: policy gradient via SGD

To compute the gradient of policy π_θ,

$$\nabla_\theta J(\theta) = \Sigma_{s\in S}\Sigma_{a\in A} d_\xi^{\pi_\theta}(s)\cdot Q^{\pi_\theta}(s,a)\cdot\nabla_\theta\pi_\theta(a|s),$$

where $d_\xi^{\pi_\theta}$ is the **state visitation measure** of the state s, or state occupancy measure.

$$d_\xi^{\pi_\theta}(s) = \Sigma_{t=0}^{\infty}\gamma^t P(s_t = s|s_0\sim\xi,\pi_\theta).$$

Because the summation of $d_\xi^{\pi_\theta}(s)$

$$\begin{aligned}\Sigma_{s\in S} d_\xi^{\pi_\theta}(s) &= \Sigma_{t=0}^{\infty}\gamma^t\Sigma_{s\in S}P(s_t=s|s_0\sim\xi,\pi_\theta)\\ &= \Sigma_{t=0}^{\infty}\gamma^t = \frac{1}{1-\gamma}.\end{aligned}$$

By this, we can have a normalized state visitation measure $(1-\gamma)d_\xi^{\pi_\theta}$, and this is also a probability distribution over the state space, called the **state visitation distribution**.
Note: It can be proved that the policy gradient has a equivalent form that enables us to easily obtain a **stochastic estimator** of the gradient by Monte Carlo Sampling.
When we are going to use stochastic gradient descent to update θ, we need a stochastic estimator of the gradient, and that can be yielded by Monte Carlo.
The stochastic gradient is

$$\nabla_\theta J(\theta) = E_{\pi_\theta,\xi}[\Sigma_{t=0}^{\infty}\gamma^t Q^{\pi_\theta}(s_t,a_t)\nabla_\theta\ln\pi_\theta(a_t|s_t)].$$

When there is a terminating state s_{T+1}, the stochastic gradient is

$$\}(\tau,\theta) = \Sigma_{t=0}^{T}\gamma^t G_t\nabla_\theta\ln\pi_\theta(a_t|s_t),$$

where $G_t = \Sigma_{\tau=0}^{T-t}\gamma^\tau r_{t+\tau}$ is an unbiased estimator of $Q^{\pi_\theta}(s_t,a_t)$.
The algorithm applying stochastic gradient of the policy is called REINFORCE.
Note: REINFORCE is a standard stochastic gradient method. It is provably **convergent and stable**.
Note: there is a variant of REINFORCE, which constantly changes the point θ when the gradient is computed. This efficient in practice but theoretically not correct.
Note: when there is no termination, the stochastic estimator is **biased** but the omitted terms will be negligibly small if the constant in the **horizon term** is reasonably large

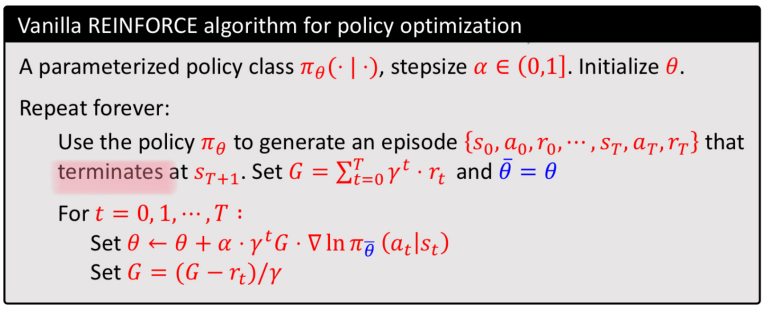

Figure 16: Vanilla REINFORCE

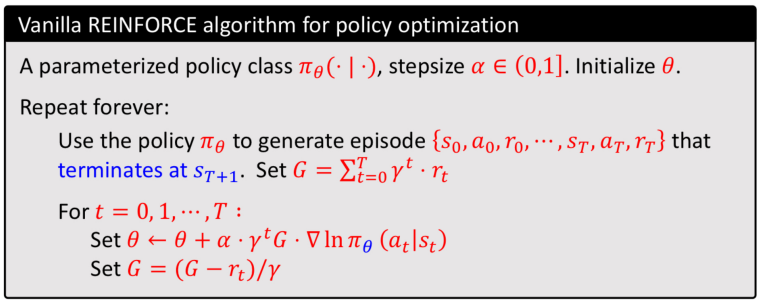

Figure 17: Variant of REINFORCE, efficient but incorrect

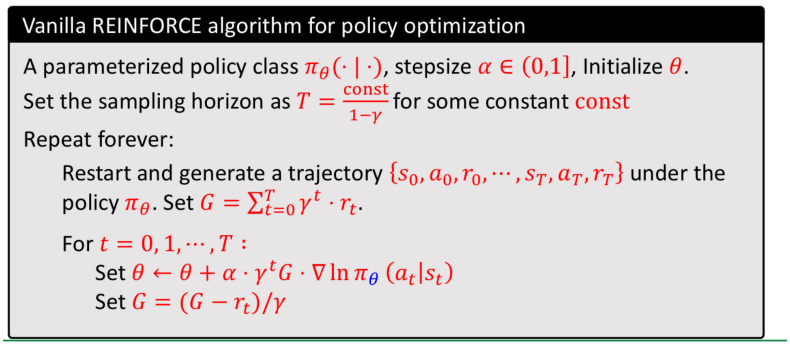

Figure 18: REINFORCE wo termination

1. α is the step size when updating θ
2. Example: Cliff walking. The algorithm yields the middle trajectory, but if it runs long enough time, it will converge to the optimal path.
3. There exist cases where $\Sigma_{a \in A} \pi_\theta(a|s) \neq 1$. In such cases, we should use a baseline, as in the next subsection.

9.4 REINFORCE with Baseline: policy gradient with baseline via SGD

$$\nabla_\theta J(\theta) = E_{\pi_\theta, \xi}[\Sigma_{t=0}^{\infty} \gamma^t (Q^{\pi_\theta}(s_t, a_t) - b(s_t)) \cdot \nabla_\theta \ln \pi_\theta(a_t|s_t)].$$

1. The typical choice of $b(s)$ is $V^{\pi_\theta}(s)$ or any approximation of it.

2. the baseline can reduce the variance of the policy gradient estimator.
3. Updating the approximate value of $V^{\pi_\theta}(s)$ for tabular MDP: TD method/MC sampling.
4. Updating the approximate value of $V^{\pi_\theta}(s)$ for MDP with function approximation: gradient MC/semi-gradient TD.
5. A good baseline can often (but not always) improve the performance of REINFORCE. An improperly selected baseline may slow down the training. In practice, finding a proper baseline is not always easy.

Here is the example where the baseline is estimated by gradient MC.
Here is the example where the baseline is estimated by TD method.

REINFORCE algorithm with baseline for policy optimization

A parameterized policy class $\pi_\theta(\cdot \mid \cdot)$ and value function $\hat{V}(\cdot; w)$ stepsize $\alpha_\theta, \alpha_w \in (0,1]$. Initialize θ and w

Repeat forever:

Use the policy π_θ to generate an episode $\{s_0, a_0, r_0, \cdots, s_T, a_T, r_T\}$ that terminates at s_{T+1}. Set $G = \sum_{t=0}^{T} \gamma^t \cdot r_t$ and $\bar{\theta} = \theta$

For $t = 0, 1, \cdots, T$:

Set $\delta = G - \hat{V}(s_t; w)$
Set $w \leftarrow w + \alpha_w \cdot \delta \cdot \nabla \hat{V}(s_t; w)$
Set $\theta \leftarrow \theta + \alpha_\theta \cdot \gamma^t \cdot \delta \cdot \nabla \ln \pi_{\bar{\theta}}(a_t|s_t)$
Set $G = (G - r_t)/\gamma$

Figure 19: REINFORCE w baseline estimated by MC

REINFORCE algorithm with baseline for policy optimization

A parameterized policy class $\pi_\theta(\cdot \mid \cdot)$ and stepsize $\alpha_\theta, \alpha_v \in (0,1]$. Initialize θ and $V = \mathbf{0} \in \mathbf{R}^S$

Repeat forever:

Use the policy π_θ to generate an episode $\{s_0, a_0, r_0, \cdots, s_T, a_T, r_T\}$ that terminates at s_{T+1}. Set $G = \sum_{t=0}^{T} \gamma^t \cdot r_t$ and $\bar{\theta} = \theta$

For $t = 0, 1, \cdots, T$:

Set $V(s_t) \leftarrow V(s_t) + \alpha_v \cdot \big(r_t + \gamma \cdot V(s_{t+1}) - V(s_t)\big)$
Set $\theta \leftarrow \theta + \alpha_\theta \cdot \gamma^t \big(G - V(s_t)\big) \cdot \nabla \ln \pi_{\bar{\theta}}(a_t|s_t)$
Set $G = (G - r_t)/\gamma$

Figure 20: REINFORCE w baseline estimated by TD

9.5 One-step Actor-Critic

In the update of the policy, the term G_t can have large variance, and always, $E[\hat{G}_t|s_t, a_t] \neq Q^{\pi_\theta}(s_t, a_t)$. Therefore, we can update the value G_t, too:

$$\hat{G}_t = r_t + \gamma \hat{V}(s_{t+1}; w).$$

This is the one-step actor-critic method. A baseline can also be incorporate.

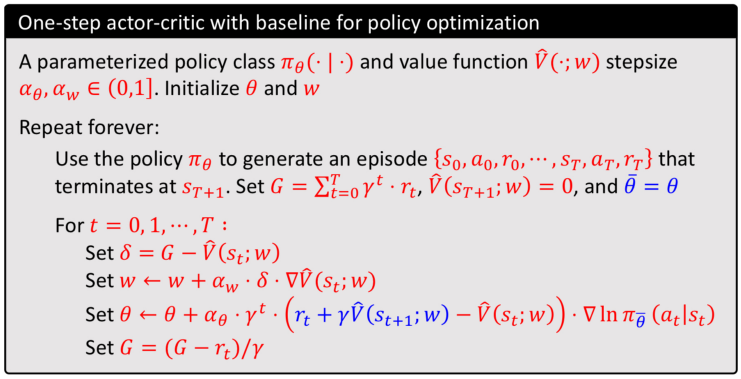

Figure 21: Actor Critic

1. Smaller variance allows much large learning rate α, which gives faster convergence
2. Example: Cliff walking. Moving towards the optimal trajectory.
3. Example: Short corridor: need much fewer episodes than raw REINFORCE.