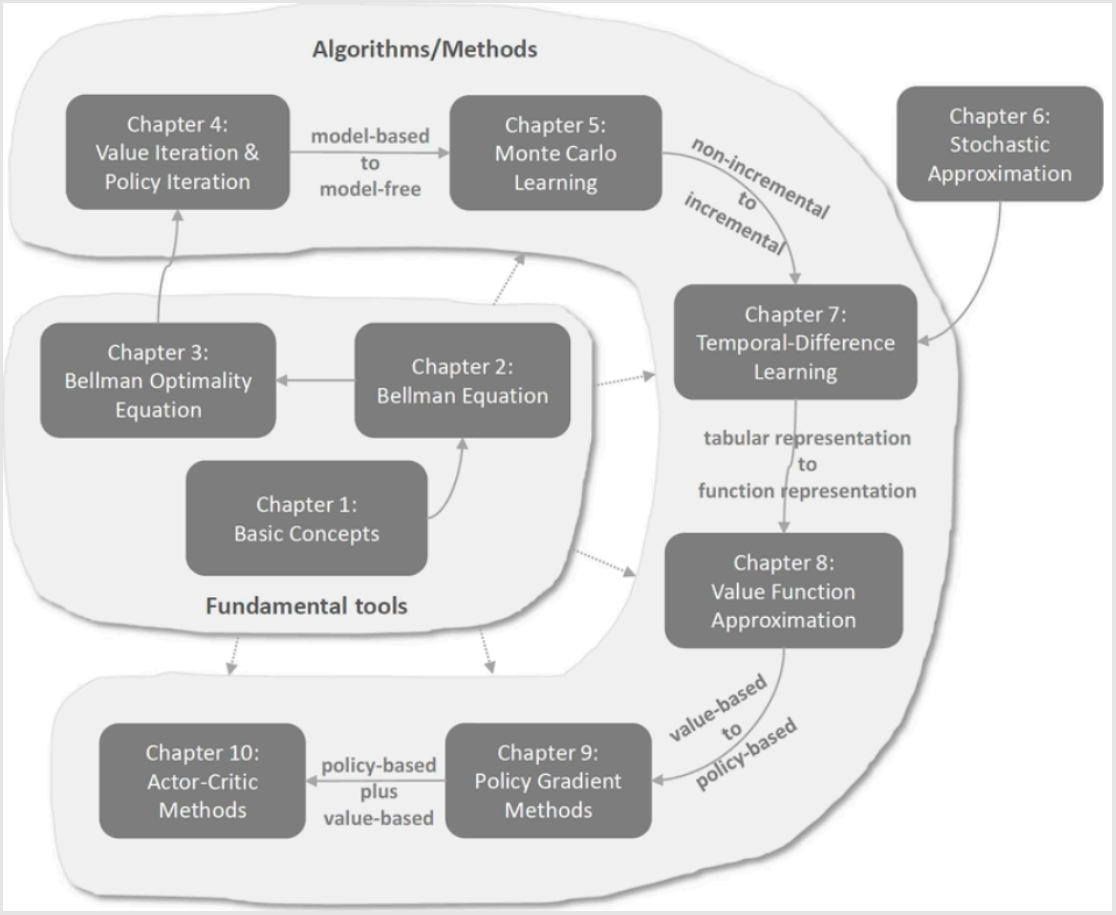
The Mathematics of Reinforcement Learning

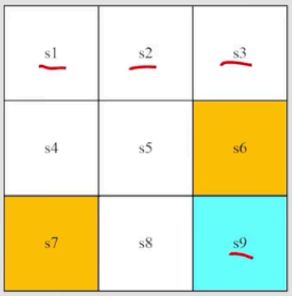
Outline



Chapter 1: Basic Concepts

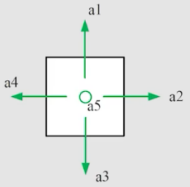
*A grid world example

1. **State, State space**



State space: the set of all states $\mathcal{S} = \{s_i\}_{i=1}^{9}$.

2. **Action, Action space <u>of a state</u>**



A is the function of s_i. $\mathcal{A}(s_i) = \{a_i\}_{i=1}^{5}$

3. **State transition**

At state s_1, if we choose action a_2, then what is the next state?

$$s_1 \xrightarrow{a_2} s_2$$

a. State transition defines the interaction with the environment.
(Can be defined in other ways)

b. Consider the "forbidden area": **accessible but with penalty**.
(May lead to "risk-taking for optimal policy")

c. **Tabular representation**: (Only for deterministic cases)

	a_1 (upwards)	a_2 (rightwards)	a_3 (downwards)	a_4 (leftwards)	a_5 (unchanged)
s_1	s_1	s_2	s_4	s_1	s_1
s_2	s_2	s_3	s_5	s_1	s_2
s_3	s_3	s_3	s_6	s_2	s_3
s_4	s_1	s_5	s_7	s_4	s_4
s_5	s_2	s_6	s_8	s_4	s_5
s_6	s_3	s_6	s_9	s_5	s_6
s_7	s_4	s_8	s_7	s_7	s_7
s_8	s_5	s_9	s_8	s_7	s_8
s_9	s_6	s_9	s_9	s_8	s_9

d. **State transition probability**: (More general, using conditional probability)

$$p(s_2|s_1, a_2) = 1$$

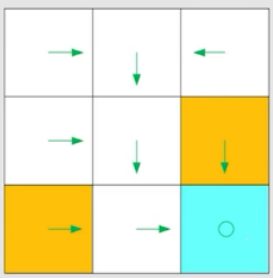
$$p(s_i|s_1, a_2) = 0 \quad \forall i \neq 2$$

Or:

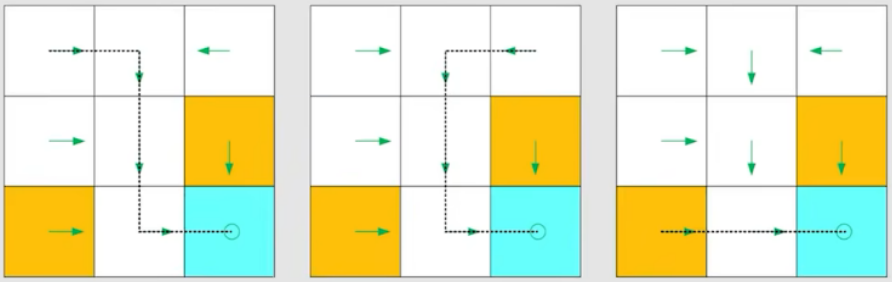
$$\begin{cases} p(s_2|s_1, a_2) = 0.5 \\ p(s_5|s_1, a_2) = 0.5 \end{cases}$$

Not just a function, but a **distribution.**

4. **Policy**: tell the agent what actions to take at a state.



a. Based on one policy, different **trajectories** can be obtained.
(with different starting points)

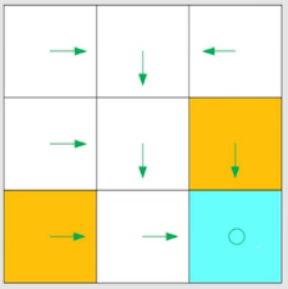


b. Mathematical representation: also conditional prob.

*__Deterministic__ policy:

For example, for state s_1:

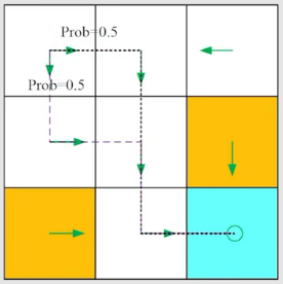
$$\pi(a_1|s_1) = 0$$
$$\pi(a_2|s_1) = 1$$
$$\pi(a_3|s_1) = 0$$
$$\pi(a_4|s_1) = 0$$
$$\pi(a_5|s_1) = 0$$



*__Stochastic__ policy:

In this policy, for s_1:

$$\pi(a_1|s_1) = 0$$
$$\pi(a_2|s_1) = 0.5$$
$$\pi(a_3|s_1) = 0.5$$
$$\pi(a_4|s_1) = 0$$
$$\pi(a_5|s_1) = 0$$



c. **Tabular representation** of a policy: (General)

	a_1 (upwards)	a_2 (rightwards)	a_3 (downwards)	a_4 (leftwards)	a_5 (unchanged)
s_1	0	0.5	0.5	0	0
s_2	0	0	1	0	0
s_3	0	0	0	1	0
s_4	0	1	0	0	0
s_5	0	0	1	0	0
s_6	0	0	1	0	0
s_7	0	1	0	0	0
s_8	0	1	0	0	0
s_9	0	0	0	0	1

We use matrix to save policy in coding.

*How to represent **“stochastic” in coding?**

Do **uniform sampling** of 0~1, decide the action based on the **interval** x falls into.

5. Reward

Reward: a real number we get after taking an action.

- A positive reward represents encouragement to take such actions.
- A negative reward represents punishment to take such actions.

a. Reward can be interpreted as **a human-machine interface.**

b. **Tabular representation**: (Only for deterministic cases)

	a_1 (upwards)	a_2 (rightwards)	a_3 (downwards)	a_4 (leftwards)	a_5 (unchanged)
s_1	r_{bound}	0	0	r_{bound}	0
s_2	r_{bound}	0	0	0	0
s_3	r_{bound}	r_{bound}	r_{forbid}	0	0
s_4	0	0	r_{forbid}	r_{bound}	0
s_5	0	r_{forbid}	0	0	0
s_6	0	r_{bound}	r_{target}	0	r_{forbid}
s_7	0	0	r_{bound}	r_{bound}	r_{forbid}
s_8	0	r_{target}	r_{bound}	r_{forbid}	0
s_9	r_{forbid}	r_{bound}	r_{bound}	0	r_{target}

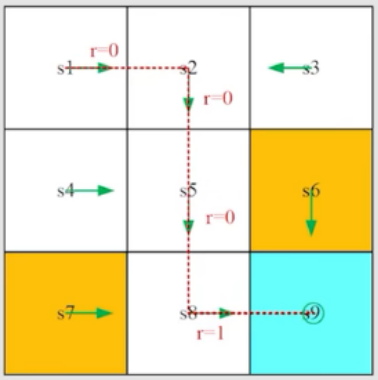
c. **Mathematical description**: (More general, using conditional probability)

- Intuition: At state s_1, if we choose action a_1, the reward is -1.
- Math: $p(r = -1|s_1, a_1) = 1$ and $p(r \neq -1|s_1, a_1) = 0$

d. Reminder: The reward depends only on the state and action, but **not the next state**. (Consider s1, a1 and s1, a5)

6. Trajectory, Return

a. A **trajectory** is a **state-action-reward chain**.



$$s_1 \xrightarrow[r=0]{a_2} s_2 \xrightarrow[r=0]{a_3} s_5 \xrightarrow[r=0]{a_3} s_8 \xrightarrow[r=1]{a_2} s_9$$

The **return** of a trajectory is the **sum of all the rewards** collected along.

$$\text{return} = 0 + 0 + 0 + 1 = 1$$

b. Return could be used to **evaluate** whether a **policy** is good or not.

c. **Discounted return**: (to handle invalid repetition and avoid divergence)

A trajectory may be infinite:

$$s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_3} s_8 \xrightarrow{a_2} s_9 \xrightarrow{a_5} s_9 \xrightarrow{a_5} s_9 \ldots$$

The return is

$$\text{return} = 0 + 0 + 0 + 1 + 1 + 1 + \cdots = \infty$$

Introducing **discount rate γ ∈ [0,1]**:

$$\begin{aligned}\text{discounted return} &= 0 + \gamma 0 + \gamma^2 0 + \gamma^3 1 + \gamma^4 1 + \gamma^5 1 + \ldots \\ &= \gamma^3(1 + \gamma + \gamma^2 + \ldots) = \gamma^3 \frac{1}{1-\gamma}.\end{aligned}$$

*Roles: make the sum finite; **balance the far and near future rewards**.
(γ closer to 0, more “short-sighted” ; γ closer to 1, more “long-term vision”)

7. **episode**:

a. An episode is a trajectory stopping at some **terminal states**.
*Tasks with episodes have finite trajectories and are called **episodic tasks**.
*Tasks without terminal states are called **continuing tasks**.

In fact, we can treat episodic and continuing tasks in a unified mathematical way by converting episodic tasks to continuing tasks.

- Option 1: Treat the target state as a special absorbing state. Once the agent reaches an absorbing state, it will never leave. The consequent rewards $r = 0$.
- Option 2: Treat the target state as a normal state with a policy. The agent can still leave the target state and gain $r = +1$ when entering the target state.

We consider option 2 in this course so that we don't need to distinguish the

b. target state from the others and can treat it as a normal state.

8. The perspective of **Markov decision process (MDP) :*****

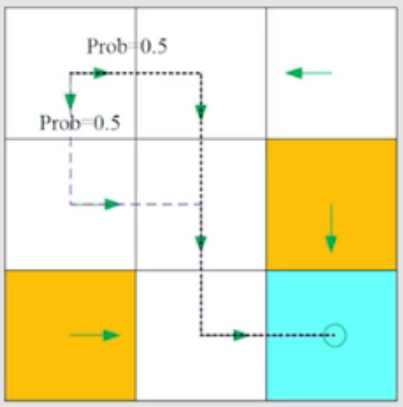
Key elements of MDP:

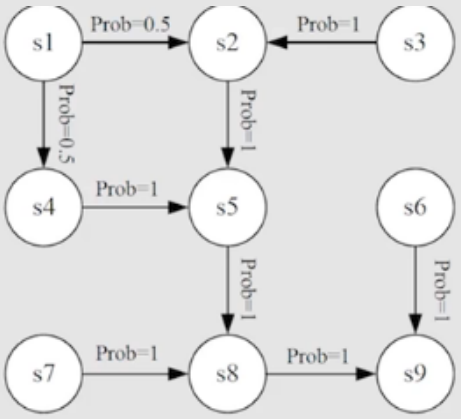
- Sets:
  - State: the set of states $\mathcal{S}$
  - Action: the set of actions $\mathcal{A}(s)$ is associated for state $s \in \mathcal{S}$.
  - Reward: the set of rewards $\mathcal{R}(s, a)$.
- Probability distribution:
  - State transition probability: at state s, taking action a, the probability to transit to state s' is $p(s'|s, a)$
  - Reward probability: at state s, taking action a, the probability to get reward r is $p(r|s, a)$
- Policy: at state s, the probability to choose action a is $\pi(a|s)$
- *Markov property*: memoryless property

$$p(s_{t+1}|a_{t+1}, s_t, \ldots, a_1, s_0) = p(s_{t+1}|a_{t+1}, s_t),$$
$$p(r_{t+1}|a_{t+1}, s_t, \ldots, a_1, s_0) = p(r_{t+1}|a_{t+1}, s_t).$$

a.

b. The grid can be abstracted as a more general model, **Markov process.**



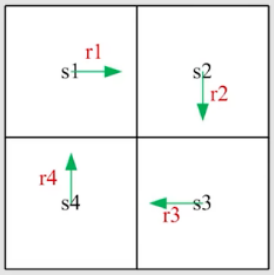


MDP becomes MP once the policy is given.

Chapter 2: Bellman Equation

1. Motivating examples:
   a. The importance of return to evaluate policies.
   b. How to calculate return?

   (notation: Let v_i denote the return obtained starting from s_i)



Method 1: By definition

$$v_1 = r_1 + \gamma r_2 + \gamma^2 r_3 + \dots$$

$$v_2 = r_2 + \gamma r_3 + \gamma^2 r_4 + \dots$$

$$v_3 = r_3 + \gamma r_4 + \gamma^2 r_1 + \dots$$

$$v_4 = r_4 + \gamma r_1 + \gamma^2 r_2 + \dots$$

Method 2: **Bootstrapping (自助法)**

$$v_1 = r_1 + \gamma(r_2 + \gamma r_3 + \dots) = r_1 + \gamma v_2$$

$$v_2 = r_2 + \gamma(r_3 + \gamma r_4 + \dots) = r_2 + \gamma v_3$$

$$v_3 = r_3 + \gamma(r_4 + \gamma r_1 + \dots) = r_3 + \gamma v_4$$

$$v_4 = r_4 + \gamma(r_1 + \gamma r_2 + \dots) = r_4 + \gamma v_1$$

*Write in matrix-vector form:

$$\underbrace{\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix}}_{\mathbf{v}} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{bmatrix} + \begin{bmatrix} \gamma v_2 \\ \gamma v_3 \\ \gamma v_4 \\ \gamma v_1 \end{bmatrix} = \underbrace{\begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{bmatrix}}_{\mathbf{r}} + \gamma \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{P}} \underbrace{\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix}}_{\mathbf{v}}$$

Namely, $\mathbf{v} = \mathbf{r} + \gamma \mathbf{P} \mathbf{v}$ (This is the Bellman equation for this problem)

Core idea: the value of one state relies on the values of other states.

2. **State value**
   a. **Some Notations**

$$S_t \xrightarrow{A_t} R_{t+1}, S_{t+1}$$

- $t, t+1$: discrete time instances
- S_t: state at time t
- A_t: the action taken at state S_t
- R_{t+1}: the reward obtained after taking A_t
- S_{t+1}: the state transited to after taking A_t

*Capitalized letters represent **random variables.**

This step is governed by the following probability distributions:

- $S_t \to A_t$ is governed by $\pi(A_t = a|S_t = s)$
- $S_t, A_t \to R_{t+1}$ is governed by $p(R_{t+1} = r|S_t = s, A_t = a)$
- $S_t, A_t \to S_{t+1}$ is governed by $p(S_{t+1} = s'|S_t = s, A_t = a)$

(Policy, reward prob and state transition prob, respectively.)

$$S_t \xrightarrow{A_t} R_{t+1}, S_{t+1} \xrightarrow{A_{t+1}} R_{t+2}, S_{t+2} \xrightarrow{A_{t+2}} R_{t+3}, \dots$$

The discounted return is

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

- $\gamma \in [0, 1)$ is a discount rate.
- G_t is also a random variable since $R_{t+1}, R_{t+2}, \dots$ are random variables.

b. **State value function**:

*Def: $v_\pi(s) = \mathbb{E}[G_t|S_t = s]$, the **expectation of** G_t (given S_t=s).

*Remarks:

- It is a function of s. It is a conditional expectation with the condition that the state starts from s.
- It is based on the policy π. For a different policy, the state value may be different.
- It represents the "value" of a state. If the state value is greater, then the policy is better because greater cumulative rewards can be obtained.

3. **Bellman equation**:
   a. Derivation:

The return G_t can be written as

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots, \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \dots), \\ &= R_{t+1} + \gamma G_{t+1}, \end{aligned}$$

Then, it follows from the definition of the state value that

$$\begin{aligned} v_\pi(s) &= \mathbb{E}[G_t|S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \mathbb{E}[R_{t+1}|S_t = s] + \gamma\mathbb{E}[G_{t+1}|S_t = s] \end{aligned}$$

Then calculate the two terms respectively:

$$\begin{aligned} \mathbb{E}[R_{t+1}|S_t = s] &= \sum_a \pi(a|s)\mathbb{E}[R_{t+1}|S_t = s, A_t = a] \\ &= \sum_a \pi(a|s) \sum_r p(r|s,a)r \end{aligned}$$

(The mean of **immediate rewards**)

$$\begin{aligned} \mathbb{E}[G_{t+1}|S_t = s] &= \sum_{s'} \mathbb{E}[G_{t+1}|S_t = s, S_{t+1} = s']p(s'|s) \\ &= \sum_{s'} \mathbb{E}[G_{t+1}|S_{t+1} = s']p(s'|s) \\ &= \sum_{s'} v_\pi(s')p(s'|s) \\ &= \sum_{s'} v_\pi(s') \sum_a p(s'|s,a)\pi(a|s) \end{aligned}$$

(The mean of **future rewards**)

Therefore, we have the **Bellman equation** (elementwise form):

$$\begin{aligned} v_\pi(s) &= \mathbb{E}[R_{t+1}|S_t = s] + \gamma\mathbb{E}[G_{t+1}|S_t = s], \\ &= \underbrace{\sum_a \pi(a|s) \sum_r p(r|s,a)r}_{\text{mean of immediate rewards}} + \underbrace{\gamma \sum_a \pi(a|s) \sum_{s'} p(s'|s,a)v_\pi(s')}_{\text{mean of future rewards}}, \\ &= \sum_a \pi(a|s) \left[\sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a)v_\pi(s') \right], \quad \forall s \in \mathcal{S}. \end{aligned}$$

*Highlights:

It characterizes the relationship among the state-value functions of different states;

It is **a set of equations**. (for each state)

*Related variables:

$v_\pi(s)$ and $v_\pi(s')$ are state values to be calculated. Bootstrapping!

$\pi(a|s)$ is a given policy. Solving the equation is called policy evaluation.

$p(r|s,a)$ and $p(s'|s,a)$ represent the dynamic model. What if the model is known or unknown?

b. Matrix-vector form ***

Rewrite the Bellman equation as

$$v_\pi(s) = r_\pi(s) + \gamma \sum_{s'} p_\pi(s'|s) v_\pi(s')$$

where

$$r_\pi(s) \triangleq \sum_a \pi(a|s) \sum_r p(r|s,a) r, \qquad p_\pi(s'|s) \triangleq \sum_a \pi(a|s) p(s'|s,a)$$

For state s_i, the Bellman equation is

$$v_\pi(s_i) = r_\pi(s_i) + \gamma \sum_{s_j} p_\pi(s_j|s_i) v_\pi(s_j)$$

Put all these equations for all the states together and rewrite to a matrix-vector form

$$v_\pi = r_\pi + \gamma P_\pi v_\pi$$

- $v_\pi = [v_\pi(s_1), \ldots, v_\pi(s_n)]^T \in \mathbb{R}^n$
- $r_\pi = [r_\pi(s_1), \ldots, r_\pi(s_n)]^T \in \mathbb{R}^n$
- $P_\pi \in \mathbb{R}^{n \times n}$, where $[P_\pi]_{ij} = p_\pi(s_j|s_i)$, is the *state transition matrix*

*A four-state example:

$$\underbrace{\begin{bmatrix} v_\pi(s_1) \\ v_\pi(s_2) \\ v_\pi(s_3) \\ v_\pi(s_4) \end{bmatrix}}_{v_\pi} = \underbrace{\begin{bmatrix} r_\pi(s_1) \\ r_\pi(s_2) \\ r_\pi(s_3) \\ r_\pi(s_4) \end{bmatrix}}_{r_\pi} + \gamma \underbrace{\begin{bmatrix} p_\pi(s_1|s_1) & p_\pi(s_2|s_1) & p_\pi(s_3|s_1) & p_\pi(s_4|s_1) \\ p_\pi(s_1|s_2) & p_\pi(s_2|s_2) & p_\pi(s_3|s_2) & p_\pi(s_4|s_2) \\ p_\pi(s_1|s_3) & p_\pi(s_2|s_3) & p_\pi(s_3|s_3) & p_\pi(s_4|s_3) \\ p_\pi(s_1|s_4) & p_\pi(s_2|s_4) & p_\pi(s_3|s_4) & p_\pi(s_4|s_4) \end{bmatrix}}_{P_\pi} \underbrace{\begin{bmatrix} v_\pi(s_1) \\ v_\pi(s_2) \\ v_\pi(s_3) \\ v_\pi(s_4) \end{bmatrix}}_{v_\pi}.$$

c. Solve state values

*Solving the Bellman equation and finding out the corresponding state values for a given policy is called **policy evaluation**.

*Methods:

*Closed-form solution: $v_\pi = (I - \gamma P_\pi)^{-1} r_\pi$

***Iteration solution**:

$$v_{k+1} = r_\pi + \gamma P_\pi v_k$$

This algorithm leads to a sequence $\{v_0, v_1, v_2, \dots\}$. We can show that

$$v_k \to v_\pi = (I - \gamma P_\pi)^{-1} r_\pi, \quad k \to \infty$$

4. **Action Value**:
   - State value: the average return the agent can get *starting from a state*.
   - Action value: the average return the agent can get *starting from a state* and *taking an action*.

   a.
   b. Def:
      - $q_\pi(s, a)$ is a function of the state-action pair (s, a)
      - $q_\pi(s, a)$ depends on π

      By the Total Expectation Formula,

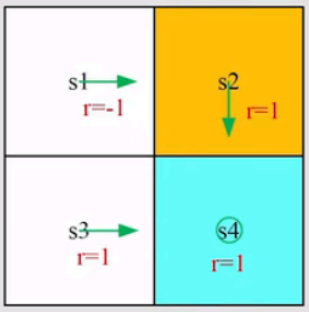
      $$v_\pi(s) = \sum_a \pi(a|s) q_\pi(s, a)$$

      (How to obtain state values from action values)
      And by the Bellman equation,

      $$q_\pi(s, a) = \sum_r p(r|s, a) r + \gamma \sum_{s'} p(s'|s, a) v_\pi(s')$$

      (How to obtain action values from state values)

*A confusing example:



The action value for state s1?

$q_\pi(s_1, a_2) = -1 + \gamma v_\pi(s_2)$,but

$q_\pi(s_1, a_1), q_\pi(s_1, a_3), q_\pi(s_1, a_4), q_\pi(s_1, a_5) =?$ Be careful!

Action value is not restricted by specific policies.

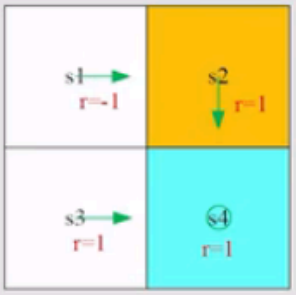
*The calculation of action value:

- We can first calculate all the state values and then calculate the action values.
- We can also directly calculate the action values with or without models.

(which leads to different algorithms.)

Chapter 3: Optimal Policy and Bellman Optimality Equation

1. Motivating examples:
    a. While the policy is not good, how can we improve it? By using action values.



Observe the action values that we obtained just now:

$$q_\pi(s_1, a_1) = 6.2, q_\pi(s_1, a_2) = 8, q_\pi(s_1, a_3) = 9,$$
$$q_\pi(s_1, a_4) = 6.2, q_\pi(s_1, a_5) = 7.2.$$

What if we select the greatest action value? Then, a new policy is obtained:

$$\pi_{\text{new}}(a|s_1) = \begin{cases} 1 & a = a^* \\ 0 & a \neq a^* \end{cases}$$

where $a^* = \arg\max_a q_\pi(s_1, a) = a_3$.

By iteration of this process, we obtain the optimal policy.

2. Optimal policy:
    a. Def:

The state value could be used to evaluate if a policy is good or not: if

$$v_{\pi_1}(s) \geq v_{\pi_2}(s) \quad \text{for all } s \in \mathcal{S}$$

then π_1 is "better" than π_2.

Definition

A policy π^* is optimal if $v_{\pi^*}(s) \geq v_\pi(s)$ for all s and for any other policy π.

3. Bellman Optimality Equation (BOE):
    a. Def:

$$v(s) = \max_{\pi} \sum_{a} \pi(a|s) \left(\sum_{r} p(r|s,a) r + \gamma \sum_{s'} p(s'|s,a) v(s') \right), \quad \forall s \in \mathcal{S}$$
$$= \max_{\pi} \sum_{a} \pi(a|s) q(s,a) \quad s \in \mathcal{S}$$

Remarks:

- $p(r|s,a), p(s'|s,a)$ are known.
- $v(s), v(s')$ are unknown and to be calculated.
- Is $\pi(s)$ known or unknown?

We need to solve the policy.

b. Matrix-vector:

$$v = \max_{\pi}(r_\pi + \gamma P_\pi v)$$

We notice that there are two unknowns, π and v.

Example (How to solve two unknowns from one equation)

Consider two variables $x, a \in \mathbb{R}$. Suppose they satisfy

$$x = \max_{a}(2x - 1 - a^2).$$

This equation has two unknowns. To solve them, first consider the right hand side. Regardless the value of x, $\max_a(2x - 1 - a^2) = 2x - 1$ where the maximization is achieved when $a = 0$. Second, when $a = 0$, the equation becomes $x = 2x - 1$, which leads to $x = 1$. Therefore, $a = 0$ and $x = 1$ are the solution of the equation.

(Fix one and solve the other first)

Fix $v'(s)$ first and solve π:

$$v(s) = \max_{\pi} \sum_a \pi(a|s) \left(\sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a)v(s') \right), \quad \forall s \in \mathcal{S}$$
$$= \max_{\pi} \sum_a \pi(a|s) q(s,a)$$

Example (How to solve $\max_\pi \sum_a \pi(a|s)q(s,a)$)

Suppose $q_1, q_2, q_3 \in \mathbb{R}$ are given. Find c_1^*, c_2^*, c_3^* solving

$$\max_{c_1,c_2,c_3} c_1q_1 + c_2q_2 + c_3q_3.$$

where $c_1 + c_2 + c_3 = 1$ and $c_1, c_2, c_3 \geq 0$.

Without loss of generality, suppose $q_3 \geq q_1, q_2$. Then, the optimal solution is $c_3^* = 1$ and $c_1^* = c_2^* = 0$. That is because for any c_1, c_2, c_3

$$q_3 = (c_1 + c_2 + c_3)q_3 = c_1q_3 + c_2q_3 + c_3q_3 \geq c_1q_1 + c_2q_2 + c_3q_3.$$

Inspired by the above example, considering that $\sum_a \pi(a|s) = 1$, we have

$$\max_{\pi} \sum_a \pi(a|s)q(s,a) = \max_{a \in \mathcal{A}(s)} q(s,a),$$

where the optimality is achieved when

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

where $a^* = \arg\max_a q(s,a)$.

Let $f(v) := \max_{\pi}(r_\pi + \gamma P_\pi v)$, then $v = f(v)$

c. Preliminaries: Contraction mapping theorem

*Fixed point **(**不动点**)**

$x \in X$ is a fixed point of $f : X \to X$ if

$$f(x) = x$$

*Contraction mapping (or contractive function):

$$\|f(x_1) - f(x_2)\| \leq \gamma \|x_1 - x_2\|$$

where $\gamma \in (0, 1)$.

- γ must be strictly less than 1 so that many limits such as $\gamma^k \to 0$ as $k \to 0$ hold.
- Here $\|\cdot\|$ can be any vector norm.

*Example:

- $x = f(x) = Ax$, where $x \in \mathbb{R}^n, A \in \mathbb{R}^{n \times n}$ and $\|A\| \leq \gamma < 1$.
  It is easy to verify that $x = 0$ is a fixed point since $0 = A0$. To see the contraction property,
  $\|Ax_1 - Ax_2\| = \|A(x_1 - x_2)\| \leq \|A\|\|x_1 - x_2\| \leq \gamma\|x_1 - x_2\|$.
  Therefore, $f(x) = Ax$ is a contraction mapping.

***

Theorem (Contraction Mapping Theorem)

For any equation that has the form of $x = f(x)$, if f is a contraction mapping, then

- *Existence: there exists a fixed point x^* satisfying $f(x^*) = x^*$.*
- *Uniqueness: The fixed point x^* is unique.*
- *Algorithm: Consider a sequence $\{x_k\}$ where $x_{k+1} = f(x_k)$, then $x_k \to x^*$ as $k \to \infty$. Moreover, the convergence rate is exponentially fast.*

d. Contraction property of BOE:

$$v = f(v) = \max_{\pi}(r_\pi + \gamma P_\pi v)$$

Theorem (Contraction Property)

$f(v)$ is a contraction mapping satisfying

$$\|f(v_1) - f(v_2)\| \leq \gamma \|v_1 - v_2\|$$

where γ is the discount rate!

By the contraction mapping theorem, we have:

Theorem (Existence, Uniqueness, and Algorithm)

For the BOE $v = f(v) = \max_\pi(r_\pi + \gamma P_\pi v)$, there always **exists** *a solution v^* and the solution is* **unique**. *The solution could be solved iteratively by*

$$v_{k+1} = f(v_k) = \max_\pi(r_\pi + \gamma P_\pi v_k)$$

This sequence $\{v_k\}$ converges to v^* **exponentially fast** *given any initial guess v_0. The convergence rate is determined by γ.*

*The solution for v is unique, while the solution for π is not necessarily unique.

e. Optimality:

$$v^* = \max_\pi(r_\pi + \gamma P_\pi v^*) \quad , \quad \pi^* = \arg\max_\pi(r_\pi + \gamma P_\pi v^*)$$

$v^* = r_{\pi^*} + \gamma P_{\pi^*} v^*$ (A special case of Bellman equation, where π=π*)

Theorem (Policy Optimality)

Suppose that v^* is the unique solution to $v = \max_\pi(r_\pi + \gamma P_\pi v)$, and v_π is the state value function satisfying $v_\pi = r_\pi + \gamma P_\pi v_\pi$ for any given policy π, then

$$v^* \geq v_\pi, \quad \forall \pi$$

Greedy Optimal Policy:

For any $s \in \mathcal{S}$, the deterministic greedy policy

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

is an optimal policy solving the BOE. Here,

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

where $q^*(s,a) := \sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a)v^*(s')$.

4. Analyzing optimal policies:
   a. What **factors determine the optimal policy**?

$$v(s) = \max_\pi \sum_a \pi(a|s) \left(\sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a)v(s') \right)$$

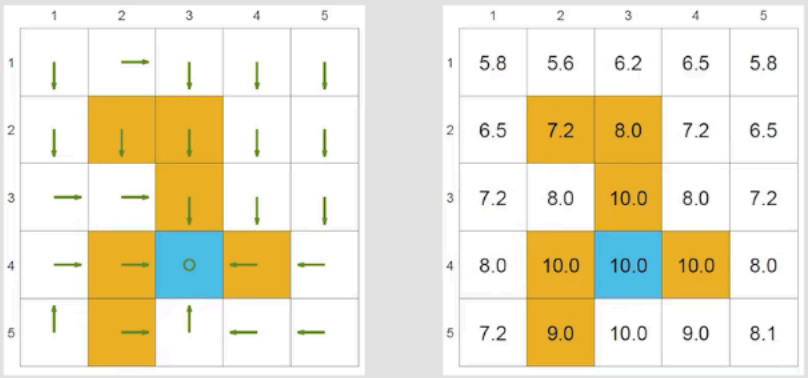
It can be clearly seen that there are three factors:

- Reward design: r
- System model: $p(s'|s,a)$, $p(r|s,a)$
- Discount rate: γ

(• $v(s), v(s'), \pi(a|s)$ are unknowns to be calculated)

Since the system model cannot be easily changed, we focus on the choice of **r and γ**.

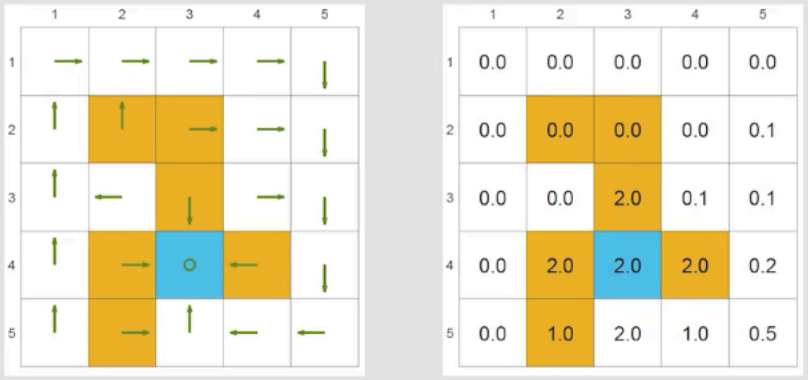
When γ = 0.9:



(a) $r_{\text{boundary}} = r_{\text{forbidden}} = -1$, $r_{\text{target}} = 1$, $\gamma = 0.9$

The optimal policy dares to take risks: entering forbidden areas!!

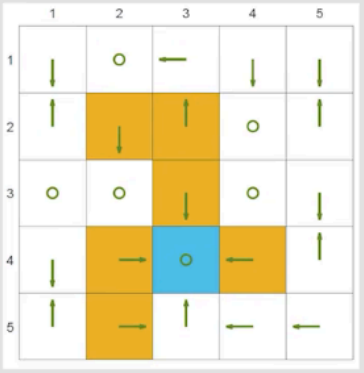
When γ = 0.5:



(b) The discount rate is $\gamma = 0.5$. Others are the same as (a).

The optimal policy becomes shorted-sighted! Avoid all the forbidden areas!

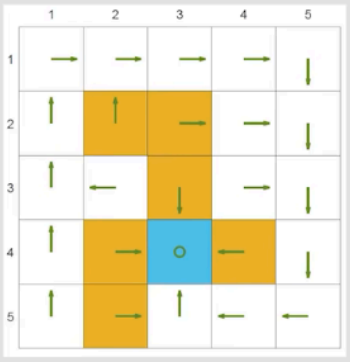
When γ = 0:



(c) The discount rate is $\gamma = 0$. Others are the same as (a).

The optimal policy becomes extremely short-sighted! Also, choose the action that has the greatest *immediate reward*! Cannot reach the target!

Changing the punishment (r) when entering forbidden areas will also make the optimal policy to avoid the forbidden areas:



(d) $r_{\text{forbidden}} = -10$. Others are the same are (a).

The optimal policy would also avoid the forbidden areas.

b. **Optimal Policy Invariance**:

Theorem (Optimal Policy Invariance)

Consider a Markov decision process with $v^* \in \mathbb{R}^{|\mathcal{S}|}$ as the optimal state value satisfying $v^* = \max_\pi(r_\pi + \gamma P_\pi v^*)$. If every reward r is changed by an affine transformation to $ar + b$, where $a, b \in \mathbb{R}$ and $a \neq 0$, then the corresponding optimal state value v' is also an affine transformation of v^*:

$$v' = av^* + \frac{b}{1-\gamma}\mathbf{1},$$

where $\gamma \in (0, 1)$ is the discount rate and $\mathbf{1} = [1, \ldots, 1]^T$. Consequently, the optimal policies are invariant to the affine transformation of the reward signals.

c. Avoiding Meaningless detour:

We do **not need punishment for taking detours**, since **the discount rate** will make policies with detours lead to later rewards and huger discounts.

(a) Optimal policy (b) Not optimal

Policy (a): return $= 1 + \gamma 1 + \gamma^2 1 + \cdots = 1/(1-\gamma) = 10.$

Policy (b): return $= 0 + \gamma 0 + \gamma^2 1 + \gamma^3 1 + \cdots = \gamma^2/(1-\gamma) = 8.1$

Chapter 4: Value Iteration and Policy Iteration

- Matrix-vector form is useful for theoretical analysis.
- Elementwise form is useful for implementation.

1. **Value iteration algorithm**
   a. Step1: Policy update (PU) (v_k *to* π_{k+1})

The elementwise form of

$$\pi_{k+1} = \arg\max_{\pi}(r_\pi + \gamma P_\pi v_k)$$

is

$$\pi_{k+1}(s) = \arg\max_{\pi} \sum_a \pi(a|s) \underbrace{\left(\sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a)v_k(s') \right)}_{q_k(s,a)}, \quad s \in \mathcal{S}$$

The optimal policy solving the above optimization problem is

$$\pi_{k+1}(a|s) = \begin{cases} 1 & a = a_k^*(s) \\ 0 & a \neq a_k^*(s) \end{cases}$$

where $a_k^*(s) = \arg\max_a q_k(a,s)$. π_{k+1} is called a *greedy* policy, since it simply selects the greatest q-value.

   b. Step2: Value update (VU) (π_{k+1} *to* v_{k+1})

The elementwise form of

$$v_{k+1} = r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_k$$

is

$$v_{k+1}(s) = \sum_a \pi_{k+1}(a|s) \underbrace{\left(\sum_r p(r|s,a) r + \gamma \sum_{s'} p(s'|s,a) v_k(s') \right)}_{q_k(s,a)}, \quad s \in \mathcal{S}$$

Since π_{k+1} is greedy, the above equation is simply

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

c. Procedure and Pseudocode***

▷ Procedure summary:

$v_k(s) \rightarrow q_k(s,a) \rightarrow$ greedy policy $\pi_{k+1}(a|s) \rightarrow$ new value $v_{k+1} = \max_a q_k(s,a)$

Pseudocode: Value iteration algorithm

Initialization: The probability model $p(r|s,a)$ and $p(s'|s,a)$ for all (s,a) are known. Initial guess v_0.

Aim: Search the optimal state value and an optimal policy solving the Bellman optimality equation.

While v_k has not converged in the sense that $\|v_k - v_{k-1}\|$ is greater than a predefined small threshold, for the kth iteration, do

For every state $s \in \mathcal{S}$, do

For every action $a \in \mathcal{A}(s)$, do

q-value: $q_k(s,a) = \sum_r p(r|s,a) r + \gamma \sum_{s'} p(s'|s,a) v_k(s')$

Maximum action value: $a_k^*(s) = \arg\max_a q_k(a,s)$

Policy update: $\pi_{k+1}(a|s) = 1$ if $a = a_k^*$, and $\pi_{k+1}(a|s) = 0$ otherwise

Value update: $v_{k+1}(s) = \max_a q_k(a,s)$

q-value is the key.

2. Policy iteration algorithm

a. Step1: Policy evaluation (PE) (π_k *to* v_{π_k})

$$v_{\pi_k} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}$$

*How to get v_{π_k}? Iterative solution for the Bellman equation.

$$v_{\pi_k}^{(j+1)} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}^{(j)}, \quad j = 0, 1, 2, \ldots$$

▷ Policy iteration is an iterative algorithm with another iterative algorithm embedded in the policy evaluation step!

*Elementwise form:

$$v_{\pi_k}^{(j+1)}(s) = \sum_a \pi_k(a|s) \left(\sum_r p(r|s,a) r + \gamma \sum_{s'} p(s'|s,a) v_{\pi_k}^{(j)}(s') \right), \quad s \in \mathcal{S},$$

Stop when $j \to \infty$ or j is sufficiently large or $\|v_{\pi_k}^{(j+1)} - v_{\pi_k}^{(j)}\|$ is sufficiently small.

b. Step2: Policy improvement (PI) (v_{π_k} *to* π_{k+1})

$$\pi_{k+1} = \arg\max_{\pi} (r_\pi + \gamma P_\pi v_{\pi_k})$$

*Why is π_{k+1} better than π_k?

Lemma (Policy Improvement)

If $\pi_{k+1} = \arg\max_\pi (r_\pi + \gamma P_\pi v_{\pi_k})$, then $v_{\pi_{k+1}} \geq v_{\pi_k}$ for any k.

*Elementwise form:

$$\pi_{k+1}(s) = \arg\max_{\pi} \sum_a \pi(a|s) \underbrace{\left(\sum_r p(r|s,a) r + \gamma \sum_{s'} p(s'|s,a) v_{\pi_k}(s') \right)}_{q_{\pi_k}(s,a)}, \quad s \in \mathcal{S}.$$

Here, $q_{\pi_k}(s,a)$ is the action value under policy π_k. Let

$$a_k^*(s) = \arg\max_a q_{\pi_k}(a, s)$$

Then, the greedy policy is

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

*$\{v_k\}$ finally converges to the optimal state value;

$\{\pi_k\}$ finally converges to an optimal policy.

Theorem (Convergence of Policy Iteration)

The state value sequence $\{v_{\pi_k}\}_{k=0}^{\infty}$ generated by the policy iteration algorithm converges to the optimal state value v^*. As a result, the policy sequence $\{\pi_k\}_{k=0}^{\infty}$ converges to an optimal policy.

c. Procedure and Pseudocode***

$$\pi_0 \xrightarrow{PE} v_{\pi_0} \xrightarrow{PI} \pi_1 \xrightarrow{PE} v_{\pi_1} \xrightarrow{PI} \pi_2 \xrightarrow{PE} v_{\pi_2} \xrightarrow{PI} \dots$$

Pseudocode: Policy iteration algorithm

Initialization: The probability model $p(r|s,a)$ and $p(s'|s,a)$ for all (s,a) are known. Initial guess π_0.

Aim: Search for the optimal state value and an optimal policy.

While the policy has not converged, for the kth iteration, do

Policy evaluation:

Initialization: an arbitrary initial guess $v_{\pi_k}^{(0)}$

While $v_{\pi_k}^{(j)}$ has not converged, for the jth iteration, do

For every state $s \in \mathcal{S}$, do

$$v_{\pi_k}^{(j+1)}(s) = \sum_a \pi_k(a|s) \left[\sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a) v_{\pi_k}^{(j)}(s')\right]$$

Policy improvement:

For every state $s \in \mathcal{S}$, do

For every action $a \in \mathcal{A}(s)$, do

$$q_{\pi_k}(s,a) = \sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a) v_{\pi_k}(s')$$

$$a_k^*(s) = \arg\max_a q_{\pi_k}(s,a)$$

$$\pi_{k+1}(a|s) = 1 \text{ if } a = a_k^*, \text{ and } \pi_{k+1}(a|s) = 0 \text{ otherwise}$$

*Closer states to the terminal states are modified earlier in policy iteration.

3. Truncated policy iteration algorithm
   a. Comparison for two kinds of iteration:

Policy iteration: start from π_0

- Policy evaluation (PE):

$$v_{\pi_k} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}$$

- Policy improvement (PI):

$$\pi_{k+1} = \arg\max_{\pi}(r_\pi + \gamma P_\pi v_{\pi_k})$$

Value iteration: start from v_0

- Policy update (PU):

$$\pi_{k+1} = \arg\max_{\pi}(r_\pi + \gamma P_\pi v_k)$$

- Value update (VU):

$$v_{k+1} = r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_k$$

Policy iteration: $\pi_0 \xrightarrow{PE} v_{\pi_0} \xrightarrow{PI} \pi_1 \xrightarrow{PE} v_{\pi_1} \xrightarrow{PI} \pi_2 \xrightarrow{PE} v_{\pi_2} \xrightarrow{PI} \dots$

Value iteration: $u_0 \xrightarrow{PU} \pi_1' \xrightarrow{VU} u_1 \xrightarrow{PU} \pi_2' \xrightarrow{VU} u_2 \xrightarrow{PU} \dots$

	Policy iteration algorithm	Value iteration algorithm	Comments
1) Policy:	π_0	N/A	
2) Value:	$v_{\pi_0} = r_{\pi_0} + \gamma P_{\pi_0} v_{\pi_0}$	$v_0 := v_{\pi_0}$	
3) Policy:	$\pi_1 = \arg\max_\pi(r_\pi + \gamma P_\pi v_{\pi_0})$	$\pi_1 = \arg\max_\pi(r_\pi + \gamma P_\pi v_0)$	The two policies are the same
4) Value:	$v_{\pi_1} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}$	$v_1 = r_{\pi_1} + \gamma P_{\pi_1} v_0$	$v_{\pi_1} \geq v_1$ since $v_{\pi_1} \geq v_{\pi_0}$
5) Policy:	$\pi_2 = \arg\max_\pi(r_\pi + \gamma P_\pi v_{\pi_1})$	$\pi_2' = \arg\max_\pi(r_\pi + \gamma P_\pi v_1)$	
⋮	⋮	⋮	⋮

The difference appears in the fourth step.
(Policy iteration solve the Bellman equation to get concise state value for π_k, while value iteration directly iterate v_k (not state value before it converges))

b. Truncated policy iteration (more general)

Consider the step of solving $v_{\pi_1} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}$:

$$
\begin{aligned}
& & v_{\pi_1}^{(0)} &= v_0 \\
\text{value iteration} \leftarrow v_1 \longleftarrow & & v_{\pi_1}^{(1)} &= r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(0)} \\
& & v_{\pi_1}^{(2)} &= r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(1)} \\
& & &\vdots \\
\text{truncated policy iteration} \leftarrow \bar{v}_1 \longleftarrow & & v_{\pi_1}^{(j)} &= r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(j-1)} \\
& & &\vdots \\
\text{policy iteration} \leftarrow v_{\pi_1} \longleftarrow & & v_{\pi_1}^{(\infty)} &= r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(\infty)}
\end{aligned}
$$

c. Pseudocode

Pseudocode: Truncated policy iteration algorithm

Initialization: The probability model $p(r|s,a)$ and $p(s'|s,a)$ for all (s,a) are known. Initial guess π_0.

Aim: Search for the optimal state value and an optimal policy.

While the policy has not converged, for the kth iteration, do

Policy evaluation:

Initialization: select the initial guess as $v_k^{(0)} = v_{k-1}$. The maximum iteration is set to be j_{truncate}.

While $j < j_{\text{truncate}}$, do

For every state $s \in \mathcal{S}$, do

$v_k^{(j+1)}(s) = \sum_a \pi_k(a|s) \left[\sum_r p(r|s,a) r + \gamma \sum_{s'} p(s'|s,a) v_k^{(j)}(s') \right]$

Set $v_k = v_k^{(j_{\text{truncate}})}$

Policy improvement:

For every state $s \in \mathcal{S}$, do

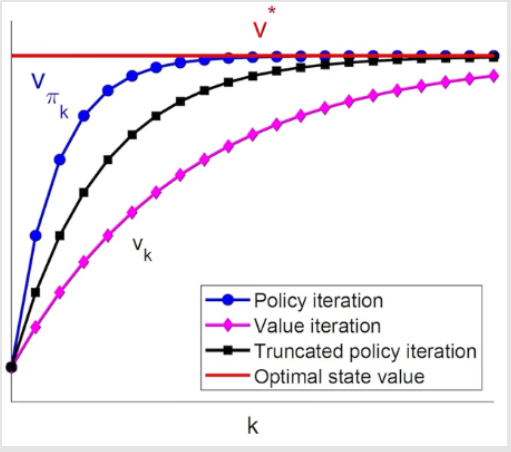
For every action $a \in \mathcal{A}(s)$, do

$q_k(s,a) = \sum_r p(r|s,a) r + \gamma \sum_{s'} p(s'|s,a) v_k(s')$

$a_k^*(s) = \arg\max_a q_k(s,a)$

$\pi_{k+1}(a|s) = 1$ if $a = a_k^*$, and $\pi_{k+1}(a|s) = 0$ otherwise

*Will the truncation determine convergence? No.



Chapter 5: Monte Carlo Learning

(Model-based to model-free)

1. Motivating example
   In some cases we cannot simply get precise distribution of the model.
   *The idea of **Monte Carlo mean estimation**:

   Suppose we get a sample sequence: $\{x_1, x_2, \ldots, x_N\}$.

   Then, the mean can be approximated as

   $$\mathbb{E}[X] \approx \bar{x} = \frac{1}{N}\sum_{j=1}^{N} x_j.$$

   *Theoretical basis: Law of Large Numbers (for i.i.d. sampling)
2. MC Basic algorithm

   a.

   The procedure of Monte Carlo estimation of action values:

   - Starting from (s, a), following policy π_k, generate an episode.
   - The return of this episode is $g(s, a)$
   - $g(s, a)$ is a sample of G_t in

     $$q_{\pi_k}(s, a) = \mathbb{E}[G_t|S_t = s, A_t = a]$$

   - Suppose we have a set of episodes and hence $\{g^{(j)}(s, a)\}$. Then,

     $$q_{\pi_k}(s, a) = \mathbb{E}[G_t|S_t = s, A_t = a] \approx \frac{1}{N}\sum_{i=1}^{N} g^{(i)}(s, a).$$

   *__Fundamental idea:__ When the model is unavailable, we use (and need) data.
   In RL, the data is also called **experience**.

   b. Algorithm:

   Given an initial policy π_0, there are two steps at the kth iteration.

   - **Step 1: policy evaluation.** This step is to obtain $q_{\pi_k}(s, a)$ for all (s, a). Specifically, for each action-state pair (s, a), run an infinite number of (or sufficiently many) episodes. The average of their returns is used to approximate $q_{\pi_k}(s, a)$.
   - **Step 2: policy improvement.** This step is to solve $\pi_{k+1}(s) = \arg\max_\pi \sum_a \pi(a|s) q_{\pi_k}(s, a)$ for all $s \in \mathcal{S}$. The greedy optimal policy is $\pi_{k+1}(a_k^*|s) = 1$ where $a_k^* = \arg\max_a q_{\pi_k}(s, a)$.

   Exactly the same as the policy iteration algorithm, except

   - Estimate $q_{\pi_k}(s, a)$ directly, instead of solving $v_{\pi_k}(s)$.

   c. Pseudocode:

Pseudocode: MC Basic algorithm (a model-free variant of policy iteration)

Initialization: Initial guess π_0.

Aim: Search for an optimal policy.

While the value estimate has not converged, for the kth iteration, do

  For every state $s \in \mathcal{S}$, do

    For every action $a \in \mathcal{A}(s)$, do

      Collect sufficiently many episodes starting from (s, a) following π_k

      MC-based policy evaluation step:

      $q_{\pi_k}(s, a)$ = average return of all the episodes starting from (s, a)

    Policy improvement step:

    $a_k^*(s) = \arg\max_a q_{\pi_k}(s, a)$

    $\pi_{k+1}(a|s) = 1$ if $a = a_k^*$, and $\pi_{k+1}(a|s) = 0$ otherwise

MC Basic is a variant of the policy iteration algorithm.
*If the current policy and model are deterministic, one episode would be sufficient to get the action value.

d. Examine the impact of **episode length**: (compared to "search radius")
The episode length should be sufficiently long to ensure each state can reach the terminal, but it does not have to be infinitely long.

3. MC Exploring Starts algorithm
(Use data more efficiently)
   a. **Visit**: every time a state-action pair appears in the episode, it is called a visit.
   MC Basic is an **initial-visit** method.
   b. Q1: **How to utilize the data**?

▷ The episode also visits other state-action pairs.

$$s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_4} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \dots \quad \text{[original episode]}$$

$$s_2 \xrightarrow{a_4} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \dots \quad \text{[episode starting from } (s_2, a_4)\text{]}$$

$$s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \dots \quad \text{[episode starting from } (s_1, a_2)\text{]}$$

$$s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \dots \quad \text{[episode starting from } (s_2, a_3)\text{]}$$

$$s_5 \xrightarrow{a_1} \dots \quad \text{[episode starting from } (s_5, a_1)\text{]}$$

Can estimate $q_\pi(s_1, a_2)$, $q_\pi(s_2, a_4)$, $q_\pi(s_2, a_3)$, $q_\pi(s_5, a_1)$,...

first-visit method: only use the first visit of a pair in an episode to estimate q.
every-visit method: update the estimation every visit of a pair in an episode.

   c. Q2: **When to update the policy**?
   *The first method (adopted in MC Basic): wait until all episodes are collected.

*The second method: use the return of a single episode to approximate the action value, **improving the policy episode-by-episode**. (Compared to Truncated PI)

▷ Generalized policy iteration:

- Not a specific algorithm.
- It refers to the general idea or framework of switching between policy-evaluation and policy-improvement processes.
- Many model-based and model-free RL algorithms fall into this framework.

d. Pseudocode of MC Exploring Starts:

Pseudocode: MC Exploring Starts (a sample-efficient variant of MC Basic)

Initialization: Initial guess π_0.
Aim: Search for an optimal policy.

For each episode, do

Episode generation: Randomly select a starting state-action pair (s_0, a_0) and ensure that all pairs can be possibly selected. Following the current policy, generate an episode of length T: $s_0, a_0, r_1, \ldots, s_{T-1}, a_{T-1}, r_T$.

Policy evaluation and policy improvement:

Initialization: $g \leftarrow 0$

For each step of the episode, $t = T-1, T-2, \ldots, 0$, do

$g \leftarrow \gamma g + r_{t+1}$

Use the first-visit strategy:

If (s_t, a_t) does not appear in $(s_0, a_0, s_1, a_1, \ldots, s_{t-1}, a_{t-1})$, then

$Returns(s_t, a_t) \leftarrow Returns(s_t, a_t) + g$

$q(s_t, a_t) = \text{average}(Returns(s_t, a_t))$

$\pi(a|s_t) = 1$ if $a = \arg\max_a q(s_t, a)$

t counts backward (T-1 →0) to improve computing efficiency.

e. Why do we need to consider **exploring starts**? To ensure traversal. (Exploring starts means starting an episode from every possible (s,a). In fact, what matters is not "starting", but "visiting", which is hard to guarantee.)
*The gap between theory and practice:

In theory, only if every action value for every state is well explored, can we select the optimal actions correctly.

In practice, exploring starts is difficult to achieve. For many applications, especially those involving physical interactions with environments, it is difficult to collect episodes starting from every state-action pair.

We can remove the requirement of exploring starts by using **soft policies**.

4. MC ε-Greedy algorithm
   a. **Soft policies**:
   A policy is called soft if the probability to **take any action** is positive.

(Greedy means deterministic, while soft means stochastic, thus exploratory)

b. **ε-Greedy policies**:

$$\pi(a|s) = \begin{cases} 1 - \dfrac{\varepsilon}{|\mathcal{A}(s)|}(|\mathcal{A}(s)| - 1), & \text{for the greedy action,} \\ \dfrac{\varepsilon}{|\mathcal{A}(s)|}, & \text{for the other } |\mathcal{A}(s)| - 1 \text{ actions.} \end{cases}$$

where $\varepsilon \in [0, 1]$ and $|\mathcal{A}(s)|$ is the number of actions for s.

* The chance to choose the greedy action is always greater than other actions, because $1 - \frac{\varepsilon}{|\mathcal{A}(s)|}(|\mathcal{A}(s)| - 1) = 1 - \varepsilon + \frac{\varepsilon}{|\mathcal{A}(s)|} \geq \frac{\varepsilon}{|\mathcal{A}(s)|}$.

*Why use ε-Greedy? Balance between **exploitation and exploration**!

- When $\varepsilon = 0$, it becomes greedy! Less exploration but more exploitation!
- When $\varepsilon = 1$, it becomes a uniform distribution. More exploration but less exploitation.

c. How to embed ε-Greedy into the MC-based RL algorithms? Modifying PI!

Now, the policy improvement step is changed to solve

$$\pi_{k+1}(s) = \arg\max_{\pi \in \Pi_\varepsilon} \sum_a \pi(a|s) q_{\pi_k}(s, a),$$

where Π_ε denotes the set of all ε-greedy policies with a fixed value of ε.

The optimal policy here is

$$\pi_{k+1}(a|s) = \begin{cases} 1 - \frac{|\mathcal{A}(s)|-1}{|\mathcal{A}(s)|}\varepsilon, & a = a_k^*, \\ \frac{1}{|\mathcal{A}(s)|}\varepsilon, & a \neq a_k^*. \end{cases}$$

The only difference lies in the form of policy improvement.

*MC ε-Greedy does not require exploring starts. (But still needs "visits")

d. Pseudocode: (Here we use every-visit method)

Pseudocode: MC ϵ-Greedy (a variant of MC Exploring Starts)

Initialization: Initial guess π_0 and the value of $\epsilon \in [0, 1]$

Aim: Search for an optimal policy.

For each episode, do

Episode generation: Randomly select a starting state-action pair (s_0, a_0). Following the current policy, generate an episode of length T: $s_0, a_0, r_1, \ldots, s_{T-1}, a_{T-1}, r_T$.

Policy evaluation and policy improvement:

Initialization: $g \leftarrow 0$

For each step of the episode, $t = T-1, T-2, \ldots, 0$, do

$g \leftarrow \gamma g + r_{t+1}$

Use the every-visit method:

If (s_t, a_t) does not appear in $(s_0, a_0, s_1, a_1, \ldots, s_{t-1}, a_{t-1})$, then

$Returns(s_t, a_t) \leftarrow Returns(s_t, a_t) + g$

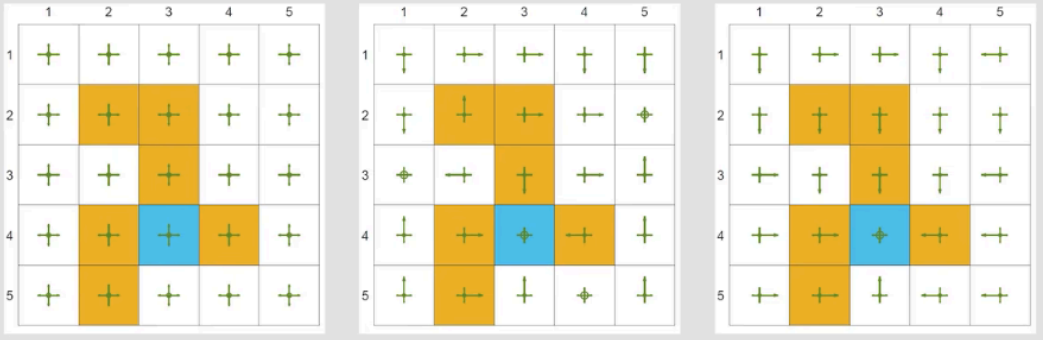
$q(s_t, a_t) = \text{average}(Returns(s_t, a_t))$

Let $a^* = \arg\max_a q(s_t, a)$ and

$$\pi(a|s_t) = \begin{cases} 1 - \frac{|\mathcal{A}(s_t)|-1}{|\mathcal{A}(s_t)|}\epsilon, & a = a^* \\ \frac{1}{|\mathcal{A}(s_t)|}\epsilon, & a \neq a^* \end{cases}$$

e. Example: (Here we only generate **one episode** with huge length, which ensure exploration without exploring starts.)

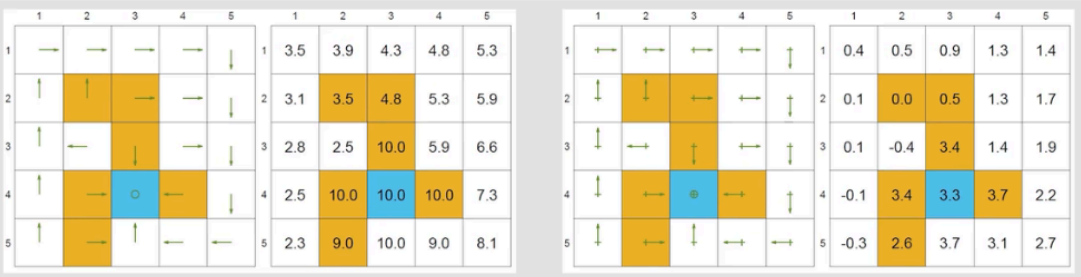
- Two iterations can lead to the optimal ε-greedy policy.



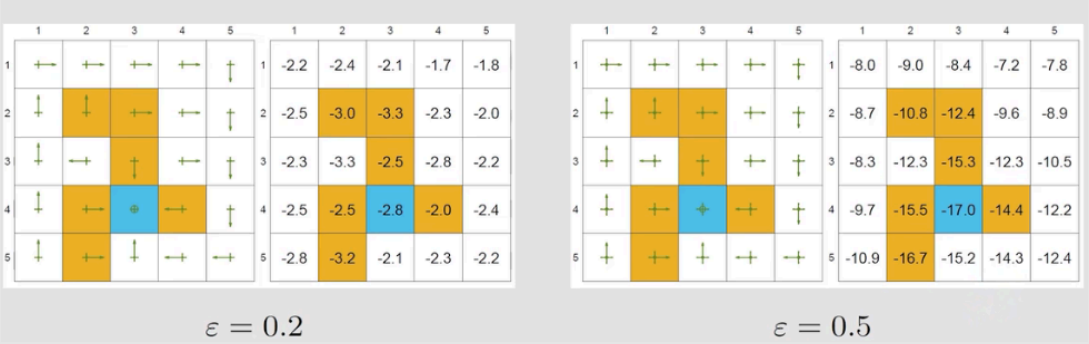
(a) Initial policy (b) After the first iteration (c) After the second iteration

Here, $r_{\text{boundary}} = -1$, $r_{\text{forbidden}} = -10$, $r_{\text{target}} = 1$, $\gamma = 0.9$

In practice we set ε to a value closer to 0; we can also use **annealing** (退火) strategy, namely initialize ε as a big value and let it gradually decrease to 0.



$\varepsilon = 0$ $\varepsilon = 0.1$



$\varepsilon = 0.2$ $\varepsilon = 0.5$

*Optimality: When ε increases, the optimality (state value) becomes worse.
***Consistency**: When ε is relatively large, the optimal ε-greedy policies are <u>not consistent</u> with the greedy optimal one.

Chapter 6: Stochastic Approximation and Stochastic Gradient Descent (随机近似理论和随机梯度下降)

1. Motivating examples
   *How to calculate the sample mean $\bar{x}$?
   Non-incremental way: collect all the samples then calculate the average.
   (Drawback: have to **wait**)
   We can avoid waiting by calculating in an **incremental and iterative** manner:

In particular, suppose

$$w_{k+1} = \frac{1}{k}\sum_{i=1}^{k} x_i, \quad k = 1, 2, \ldots$$

and hence

$$w_k = \frac{1}{k-1}\sum_{i=1}^{k-1} x_i, \quad k = 2, 3, \ldots$$

Then, w_{k+1} can be expressed in terms of w_k as

$$\begin{aligned} w_{k+1} = \frac{1}{k}\sum_{i=1}^{k} x_i &= \frac{1}{k}\left(\sum_{i=1}^{k-1} x_i + x_k\right) \\ &= \frac{1}{k}((k-1)w_k + x_k) = w_k - \frac{1}{k}(w_k - x_k). \end{aligned}$$

Therefore, we obtain the following iterative algorithm:

$$w_{k+1} = w_k - \frac{1}{k}(w_k - x_k).$$

   The mean estimate is not accurate in the beginning, but **it is better than nothing**.
   *A more general expression:

$$w_{k+1} = w_k - \alpha_k(w_k - x_k),$$

where $1/k$ is replaced by $\alpha_k > 0$.

α_k: Learning rate

   (still converges)
2. **Robbins-Monro (RM)** algorithm
   a. **Stochastic approximation (SA)**:

SA refers to a broad class of stochastic iterative algorithms solving root finding or optimization problems.

SA is powerful in the sense that it does **not require to know the expression** of the objective function nor its derivative.

b. Problem statement for RM:

Problem statement: Suppose we would like to find the root of the equation

$$g(w) = 0,$$

where $w \in \mathbb{R}$ is the variable to be solved and $g : \mathbb{R} \to \mathbb{R}$ is a function.

In the case that the expression of function g is **unknown**:
(e.g. an artificial neuron network, $y = g(w)$)

c. RM algorithm:

$$w_{k+1} = w_k - a_k \tilde{g}(w_k, \eta_k), \qquad k = 1, 2, 3, \ldots$$

where

- w_k is the kth estimate of the root
- $\tilde{g}(w_k, \eta_k) = g(w_k) + \eta_k$ is the kth noisy observation
- a_k is a positive coefficient.

η_k is random noise. (can be omitted)

The function $g(w)$ is a **black box**.

This algorithm relies on data:

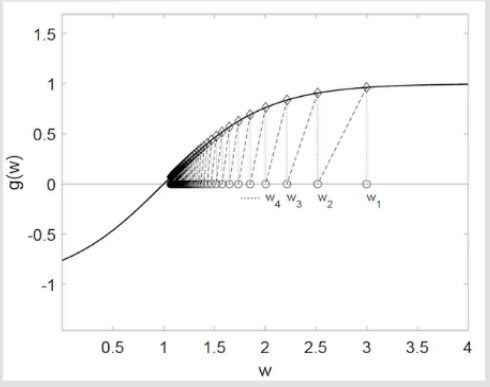
- Input sequence: $\{w_k\}$
- Noisy output sequence: $\{\tilde{g}(w_k, \eta_k)\}$

d. Convergence analysis:

*An illustrative example:

- $g(w) = \tanh(w - 1)$
- The true root of $g(w) = 0$ is $w^* = 1$.
- Parameters: $w_1 = 3$, $a_k = 1/k$, $\eta_k \equiv 0$ (no noise for the sake of simplicity)

Simulation result: w_k converges to the true root $w^* = 1$.



Intuition: w_{k+1} is closer to w^* than w_k.

RM is essentially a form of negative feedback.

e. RM Theorem:

Theorem (Robbins-Monro Theorem)

In the Robbins-Monro algorithm, if

1) $0 < c_1 \leq \nabla_w g(w) \leq c_2$ *for all* w;

2) $\sum_{k=1}^{\infty} a_k = \infty$ *and* $\sum_{k=1}^{\infty} a_k^2 < \infty$;

3) $\mathbb{E}[\eta_k | \mathcal{H}_k] = 0$ *and* $\mathbb{E}[\eta_k^2 | \mathcal{H}_k] < \infty$;

where $\mathcal{H}_k = \{w_k, w_{k-1}, \dots\}$, *then* w_k *converges with probability 1 (w.p.1) to the root* w^* *satisfying* $g(w^*) = 0$.

(w.p.1 means in the sense of probability)

*Explanation of the three conditions:

- $0 < c_1 \leq \nabla_w g(w) \leq c_2$ for all w

  This condition indicates
  - g to be monotonically increasing,which ensures that the root of $g(w) = 0$ exists and is unique
  - The gradient is bounded from the above.
- $\sum_{k=1}^{\infty} a_k = \infty$ and $\sum_{k=1}^{\infty} a_k^2 < \infty$
  - The condition of $\sum_{k=1}^{\infty} a_k^2 < \infty$ ensures that a_k converges to zero as $k \to \infty$.
  - The condition of $\sum_{k=1}^{\infty} a_k = \infty$ ensures that a_k do not converge to zero too fast.
- $\mathbb{E}[\eta_k | \mathcal{H}_k] = 0$ and $\mathbb{E}[\eta_k^2 | \mathcal{H}_k] < \infty$
  - A special yet common case is that $\{\eta_k\}$ is an iid stochastic sequence satisfying $\mathbb{E}[\eta_k] = 0$ and $\mathbb{E}[\eta_k^2] < \infty$. The observation error η_k is not required to be Gaussian.

What $\{a_k\}$ satisfies the two conditions? $\sum_{k=1}^{\infty} a_k^2 < \infty, \sum_{k=1}^{\infty} a_k = \infty$

One typical sequence is

$$a_k = \frac{1}{k}$$

f. Apply to mean estimation:

Recall that:

$$w_{k+1} = w_k + \alpha_k (x_k - w_k).$$

is the mean estimation algorithm.

This algorithm is a special form of the RM algorithm:

1) Consider a function:

$$g(w) \doteq w - \mathbb{E}[X].$$

Our aim is to solve $g(w) = 0$. If we can do that, then we can obtain $\mathbb{E}[X]$.

2) The observation we can get is

$$\tilde{g}(w, x) \doteq w - x,$$

because we can only obtain samples of X. Note that

$$\begin{aligned}\tilde{g}(w, \eta) = w - x &= w - x + \mathbb{E}[X] - \mathbb{E}[X] \\ &= (w - \mathbb{E}[X]) + (\mathbb{E}[X] - x) \doteq g(w) + \eta,\end{aligned}$$

3) The RM algorithm for solving $g(x) = 0$ is

$$w_{k+1} = w_k - \alpha_k \tilde{g}(w_k, \eta_k) = w_k - \alpha_k(w_k - x_k),$$

which is exactly the mean estimation algorithm.

*Dvoretzky's Theorem (Optional):

Theorem (Dvoretzky's Theorem)

Consider a stochastic process

$$w_{k+1} = (1 - \alpha_k)w_k + \beta_k \eta_k,$$

where $\{\alpha_k\}_{k=1}^{\infty}, \{\beta_k\}_{k=1}^{\infty}, \{\eta_k\}_{k=1}^{\infty}$ *are stochastic sequences. Here* $\alpha_k \geq 0, \beta_k \geq 0$ *for all* k*. Then,* w_k *would converge to zero with probability 1 if the following conditions are satisfied:*

1) $\sum_{k=1}^{\infty} \alpha_k = \infty$, $\sum_{k=1}^{\infty} \alpha_k^2 < \infty$; $\sum_{k=1}^{\infty} \beta_k^2 < \infty$ *uniformly w.p.1;*
2) $\mathbb{E}[\eta_k | \mathcal{H}_k] = 0$ *and* $\mathbb{E}[\eta_k^2 | \mathcal{H}_k] \leq C$ *w.p.1;*

where $\mathcal{H}_k = \{w_k, w_{k-1}, \ldots, \eta_{k-1}, \ldots, \alpha_{k-1}, \ldots, \beta_{k-1}, \ldots\}$.

3. Stochastic gradient descent

*Relationship between 3 algorithms: *Mean estimation* ∈ *SGD* ∈ *RM*

a. Problem statement:

Suppose we aim to solve the following optimization problem:

$$\min_w \quad J(w) = \mathbb{E}[f(w, X)]$$

- w is the parameter to be optimized.
- X is a random variable. The expectation is with respect to X.
- w and X can be either scalars or vectors. The function $f(\cdot)$ is a scalar.

*Method 1: gradient descent (GD)

$$w_{k+1} = w_k - \alpha_k \nabla_w \mathbb{E}[f(w_k, X)] = w_k - \alpha_k \mathbb{E}[\nabla_w f(w_k, X)]$$

Drawbacks: difficult to obtain the mean.
*Method 2: batch gradient descent (BGD)

$$\mathbb{E}[\nabla_w f(w_k, X)] \approx \frac{1}{n}\sum_{i=1}^{n} \nabla_w f(w_k, x_i).$$

$$w_{k+1} = w_k - \alpha_k \frac{1}{n}\sum_{i=1}^{n} \nabla_w f(w_k, x_i).$$

Drawbacks: it requires many samples in each iteration for each w_k.

*Method 3: **stochastic gradient descent (SGD)**

b. Def of SGD:

$$w_{k+1} = w_k - \alpha_k \nabla_w f(w_k, x_k)$$

*Comparison:

- Compared to the gradient descent method: Replace the true gradient $\mathbb{E}[\nabla_w f(w_k, X)]$ by the stochastic gradient $\nabla_w f(w_k, x_k)$.
- Compared to the batch gradient descent method: let $n = 1$.

c. Example:

$$\min_w \quad J(w) = \mathbb{E}[f(w, X)] = \mathbb{E}\left[\frac{1}{2}\|w - X\|^2\right],$$

where

$$f(w, X) = \|w - X\|^2/2 \qquad \nabla_w f(w, X) = w - X$$

The SGD algorithm for solving the above problem is

$$w_{k+1} = w_k - \alpha_k \nabla_w f(w_k, x_k) = w_k - \alpha_k (w_k - x_k)$$

It shows the mean estimation algorithm is a special SGD.

d. Convergence:
(prove that SGD is a special RM)

The aim of SGD is to minimize

$$J(w) = \mathbb{E}[f(w, X)]$$

This problem can be converted to a root-finding problem:

$$\nabla_w J(w) = \mathbb{E}[\nabla_w f(w, X)] = 0$$

Let

$$g(w) = \nabla_w J(w) = \mathbb{E}[\nabla_w f(w, X)].$$

Then, the aim of SGD is to find the root of $g(w) = 0$.

What we can measure is

$$\tilde{g}(w,\eta) = \nabla_w f(w,x)$$
$$= \underbrace{\mathbb{E}[\nabla_w f(w,X)]}_{g(w)} + \underbrace{\nabla_w f(w,x) - \mathbb{E}[\nabla_w f(w,X)]}_{\eta}.$$

Then, the RM algorithm for solving $g(w) = 0$ is

$$w_{k+1} = w_k - a_k \tilde{g}(w_k, \eta_k) = w_k - a_k \nabla_w f(w_k, x_k).$$

It is exactly the SGD algorithm.

Theorem (Convergence of SGD)

In the SGD algorithm, if

1) $0 < c_1 \leq \nabla_w^2 f(w,X) \leq c_2$;
2) $\sum_{k=1}^{\infty} a_k = \infty$ *and* $\sum_{k=1}^{\infty} a_k^2 < \infty$;
3) $\{x_k\}_{k=1}^{\infty}$ *is iid;*

then w_k *converges to the root of* $\nabla_w \mathbb{E}[f(w,X)] = 0$ *with probability 1.*

(The first condition includes the convexity of f)

e. Convergence pattern:
*Whether the convergence of SGD is slow or random?
Introducing the **relative error** between the stochastic and true gradient:

$$\delta_k \doteq \frac{|\nabla_w f(w_k,x_k) - \mathbb{E}[\nabla_w f(w_k,X)]|}{|\mathbb{E}[\nabla_w f(w_k,X)]|}.$$

$$\delta_k = \frac{|\nabla_w f(w_k,x_k) - \mathbb{E}[\nabla_w f(w_k,X)]|}{|\mathbb{E}[\nabla_w f(w_k,X)] - \mathbb{E}[\nabla_w f(w^*,X)]|} = \frac{|\nabla_w f(w_k,x_k) - \mathbb{E}[\nabla_w f(w_k,X)]|}{|\mathbb{E}[\nabla_w^2 f(\tilde{w}_k,X)(w_k - w^*)]|}.$$

where the last equality is due to the mean value theorem and $\tilde{w}_k \in [w_k, w^*]$.

Suppose f is strictly convex such that

$$\nabla_w^2 f \geq c > 0$$

for all w, X, where c is a positive bound.

Then, the denominator of δ_k becomes

$$\left|\mathbb{E}[\nabla_w^2 f(\tilde{w}_k,X)(w_k - w^*)]\right| = \left|\mathbb{E}[\nabla_w^2 f(\tilde{w}_k,X)](w_k - w^*)\right|$$
$$= \left|\mathbb{E}[\nabla_w^2 f(\tilde{w}_k,X)]\right|\left|(w_k - w^*)\right| \geq c|w_k - w^*|.$$

Then,

$$\delta_k \leq \frac{\left| \overbrace{\nabla_w f(w_k, x_k)}^{\text{stochastic gradient}} - \overbrace{\mathbb{E}[\nabla_w f(w_k, X)]}^{\text{true gradient}} \right|}{\underbrace{c|w_k - w^*|}_{\text{distance to the optimal solution}}}.$$

The relative error δ_k is inversely proportional to $|w_k - w^*|$.

- When $|w_k - w^*|$ is large, δ_k is small and SGD behaves like GD.
- When w_k is close to w^*, the relative error may be large and the convergence exhibits more randomness in the neighborhood of w^*.

f. A deterministic formulation

Consider the optimization problem:

$$\min_w \quad J(w) = \frac{1}{n} \sum_{i=1}^{n} f(w, x_i),$$

Here $\{x_i\}_{i=1}^n$ is just a set of real numbers, where x_i does not have to be a sample of any random variable.

Q1: Is this algorithm SGD?

Q2: How should we use $\{x_i\}_{i=1}^n$? (the order?)

In fact, we can introduce a random variable manually, converting the deterministic formulation to SGD.

In particular, suppose X is a random variable defined on the set $\{x_i\}_{i=1}^n$. Suppose its probability distribution is uniform such that

$$p(X = x_i) = 1/n$$

Then, the deterministic optimization problem becomes a stochastic one:

$$\min_w \quad J(w) = \frac{1}{n} \sum_{i=1}^{n} f(w, x_i) = \mathbb{E}[f(w, X)].$$

Therefore, we should use $\{x_i\}_{i=1}^n$ by random sampling.

4. The comparison between **BGD, MBGD and SGD**:

$$w_{k+1} = w_k - \alpha_k \frac{1}{n} \sum_{i=1}^{n} \nabla_w f(w_k, x_i), \qquad \text{(BGD)}$$

$$w_{k+1} = w_k - \alpha_k \frac{1}{m} \sum_{j \in \mathcal{I}_k} \nabla_w f(w_k, x_j), \qquad \text{(MBGD)}$$

$$w_{k+1} = w_k - \alpha_k \nabla_w f(w_k, x_k). \qquad \text{(SGD)}$$

- Compared to SGD, MBGD has less randomness because it uses more samples instead of just one as in SGD.
- Compared to BGD, MBGD does not require to use all the samples in every iteration, making it more flexible and efficient.
- If $m = 1$, MBGD becomes SGD.
- If $m = n$, MBGD does NOT become BGD strictly speaking because MBGD uses randomly fetched n samples whereas BGD uses all n numbers. In particular, MBGD may use a value in $\{x_i\}_{i=1}^n$ multiple times whereas BGD uses each number once.

*Example:

Given some numbers $\{x_i\}_{i=1}^n$, our aim is to calculate the mean $\bar{x} = \sum_{i=1}^n x_i/n$. This problem can be equivalently stated as the following optimization problem:

$$\min_w \quad J(w) = \frac{1}{2n}\sum_{i=1}^n \|w - x_i\|^2$$

The three algorithms for solving this problem are, respectively,

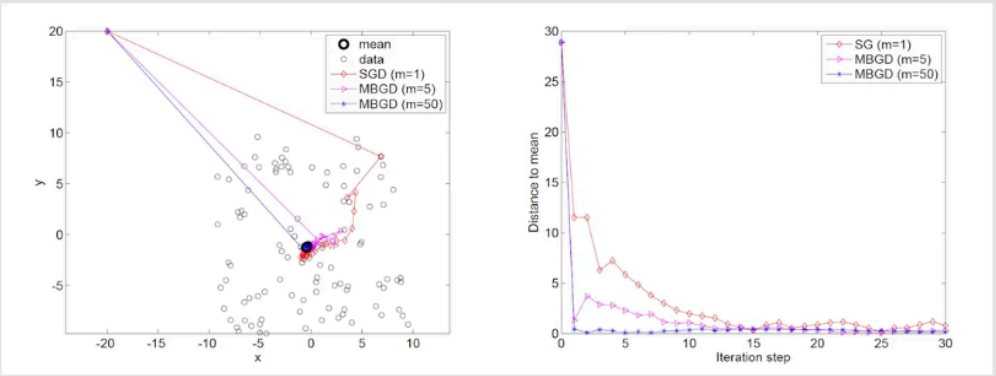
$$w_{k+1} = w_k - \alpha_k \frac{1}{n}\sum_{i=1}^n (w_k - x_i) = w_k - \alpha_k(w_k - \bar{x}), \qquad \text{(BGD)}$$

$$w_{k+1} = w_k - \alpha_k \frac{1}{m}\sum_{j\in\mathcal{I}_k} (w_k - x_j) = w_k - \alpha_k\left(w_k - \bar{x}_k^{(m)}\right), \qquad \text{(MBGD)}$$

$$w_{k+1} = w_k - \alpha_k(w_k - x_k), \qquad \text{(SGD)}$$

where $\bar{x}_k^{(m)} = \sum_{j\in\mathcal{I}_k} x_j/m$.

Let $\alpha_k = 1/k$. Given 100 points, using different mini-batch sizes leads to different convergence speed.



Bigger batch-sizes lead to faster convergence speed.

*Summary:

- Mean estimation: compute $\mathbb{E}[X]$ using $\{x_k\}$

$$w_{k+1} = w_k - \frac{1}{k}(w_k - x_k).$$

- RM algorithm: solve $g(w) = 0$ using $\{\tilde{g}(w_k, \eta_k)\}$

$$w_{k+1} = w_k - a_k \tilde{g}(w_k, \eta_k)$$

- SGD algorithm: minimize $J(w) = \mathbb{E}[f(w, X)]$ using $\{\nabla_w f(w_k, x_k)\}$

$$w_{k+1} = w_k - \alpha_k \nabla_w f(w_k, x_k),$$

Chapter 7: Temporal-Difference Learning

(Non-incremental to incremental)

1. Motivating examples
   Evolution of the mean estimation problem:

$$w = \mathbb{E}[X] \underset{\rightarrow}{} w = \mathbb{E}[v(X)] \underset{\rightarrow}{} w = \mathbb{E}[R + \gamma v(X)]$$

- Suppose we can obtain samples $\{x\}$ and $\{r\}$ of X and R. we define

$$\begin{aligned} g(w) &= w - \mathbb{E}[R + \gamma v(X)], \\ \tilde{g}(w, \eta) &= w - [r + \gamma v(x)] \\ &= (w - \mathbb{E}[R + \gamma v(X)]) + (\mathbb{E}[R + \gamma v(X)] - [r + \gamma v(x)]) \\ &\doteq g(w) + \eta. \end{aligned}$$

- Then, the problem becomes a root-finding problem: $g(w) = 0$. The corresponding RM algorithm is

$$w_{k+1} = w_k - \alpha_k \tilde{g}(w_k, \eta_k) = w_k - \alpha_k [w_k - (r_k + \gamma v(x_k))]$$

2. TD learning of state values (the most typical **TD**)

The data/experience required by the algorithm:

- $(s_0, r_1, s_1, \ldots, s_t, r_{t+1}, s_{t+1}, \ldots)$ or $\{(s_t, r_{t+1}, s_{t+1})\}_t$ generated following the given policy π.

The TD learning algorithm is

$$v_{t+1}(s_t) = v_t(s_t) - \alpha_t(s_t)\Big[v_t(s_t) - [r_{t+1} + \gamma v_t(s_{t+1})]\Big], \quad (1)$$

$$v_{t+1}(s) = v_t(s), \quad \forall s \neq s_t, \quad (2)$$

where $t = 0, 1, 2, \ldots$. Here, $v_t(s_t)$ is the estimated state value of $v_\pi(s_t)$; $\alpha_t(s_t)$ is the learning rate of s_t at time t.

a.

$$\underbrace{v_{t+1}(s_t)}_{\text{new estimate}} = \underbrace{v_t(s_t)}_{\text{current estimate}} - \alpha_t(s_t)\Big[\overbrace{v_t(s_t) - \underbrace{[r_{t+1} + \gamma v_t(s_{t+1})]}_{\text{TD target } \bar{v}_t}}^{\text{TD error } \delta_t}\Big]$$

*__TD target__: $\bar{v}_t \doteq r_{t+1} + \gamma v(s_{t+1})$

"Target" means $v(s_t)$ **is driven towards** $\bar{v}_t$**.**

$$\begin{aligned}
& v_{t+1}(s_t) = v_t(s_t) - \alpha_t(s_t)\big[v_t(s_t) - \bar{v}_t\big] \\
\Longrightarrow \quad & v_{t+1}(s_t) - \bar{v}_t = v_t(s_t) - \bar{v}_t - \alpha_t(s_t)\big[v_t(s_t) - \bar{v}_t\big] \\
\Longrightarrow \quad & v_{t+1}(s_t) - \bar{v}_t = [1 - \alpha_t(s_t)][v_t(s_t) - \bar{v}_t] \\
\Longrightarrow \quad & |v_{t+1}(s_t) - \bar{v}_t| = |1 - \alpha_t(s_t)||v_t(s_t) - \bar{v}_t|
\end{aligned}$$

Since $\alpha_t(s_t)$ is a small positive number, we have

$$0 < 1 - \alpha_t(s_t) < 1$$

Therefore,

$$|v_{t+1}(s_t) - \bar{v}_t| \leq |v_t(s_t) - \bar{v}_t|$$

which means $v(s_t)$ is driven towards $\bar{v}_t$!

*__TD error__: $\delta_t \doteq v(s_t) - [r_{t+1} + \gamma v(s_{t+1})] = v(s_t) - \bar{v}_t$

How to interpret the "TD error"?

- It is a difference between two consequent time steps.
- It reflects the deficiency between v_t and v_π. To see that, denote

$$\delta_{\pi,t} \doteq v_\pi(s_t) - [r_{t+1} + \gamma v_\pi(s_{t+1})]$$

Note that

$$\mathbb{E}[\delta_{\pi,t}|S_t = s_t] = v_\pi(s_t) - \mathbb{E}\big[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s_t\big] = 0.$$

  - If $v_t = v_\pi$, then δ_t should be zero (in the expectation sense).
  - Hence, if δ_t is not zero, then v_t is not equal to v_π.
- The TD error can be interpreted as innovation, which means new information obtained from the experience (s_t, r_{t+1}, s_{t+1}).

b. The TD algorithm only estimates the **state value** of a **given policy**.
(does not estimate the action values; does not search for optimal policies)
Mathematically, the TD algorithm **solves the Bellman equation** of a given policy π in a **model-free** manner.

*Bellman expectation equation (another expression):

$$v_\pi(s) = \mathbb{E}\big[R + \gamma v_\pi(S')|S = s\big], \quad s \in \mathcal{S}.$$

(since $\mathbb{E}[G|S=s] = \sum_a \pi(a|s) \sum_{s'} p(s'|s,a) v_\pi(s') = \mathbb{E}[v_\pi(S')|S=s]$,

S' represents the next state of S.)

c. TD is an RM to solve the Bellman equation:

$$g(v(s)) = v(s) - \mathbb{E}\big[R + \gamma v_\pi(S')|s\big] = 0$$

(View $v(s)$ as w, $v_\pi(s)$ as w^*)

$$\begin{aligned}\tilde{g}(v(s)) &= v(s) - \big[r + \gamma v_\pi(s')\big] \\ &= \underbrace{\Big(v(s) - \mathbb{E}\big[R + \gamma v_\pi(S')|s\big]\Big)}_{g(v(s))} + \underbrace{\Big(\mathbb{E}\big[R + \gamma v_\pi(S')|s\big] - \big[r + \gamma v_\pi(s')\big]\Big)}_{\eta}\end{aligned}$$

The RM algorithm for solving $g(v(s)) = 0$:

$$\begin{aligned}v_{k+1}(s) &= v_k(s) - \alpha_k \tilde{g}(v_k(s)) \\ &= v_k(s) - \alpha_k\Big(v_k(s) - \big[r_k + \gamma v_\pi(s'_k)\big]\Big), \quad k = 1, 2, 3, \ldots\end{aligned}$$

where $v_k(s)$ is the estimate of $v_\pi(s)$ at the kth step; r_k, s'_k are the samples of R, S' obtained at the kth step.

*Two modifications:

- One modification is that $\{(s, r, s')\}$ is changed to $\{(s_t, r_{t+1}, s_{t+1})\}$ so that the algorithm can utilize the sequential samples in an episode.

(Only the states that are **visited (s_t) in an episode** are updated with the algorithm, those are not visited remain as they were.)

- Another modification is that $v_\pi(s')$ is replaced by an estimate of it because we don't know it in advance.

Replace $v_\pi(s'_k)$ by $v_k(s'_k)$.

Then the RM becomes TD.

d. Convergence:

Theorem (Convergence of TD Learning)

By the TD algorithm (1), $v_t(s)$ converges with probability 1 to $v_\pi(s)$ for all $s \in \mathcal{S}$ as $t \to \infty$ if $\sum_t \alpha_t(s) = \infty$ and $\sum_t \alpha_t^2(s) < \infty$ for all $s \in \mathcal{S}$.

*Remarks:

- $\sum_t \alpha_t(s) = \infty$ and $\sum_t \alpha_t^2(s) < \infty$ must be valid for all $s \in \mathcal{S}$. At time step t, if $s = s_t$ which means that s is visited at time t, then $\alpha_t(s) > 0$; otherwise, $\alpha_t(s) = 0$ for all the other $s \neq s_t$. That requires every state must be visited an infinite (or sufficiently many) number of times.
- The learning rate α is often selected as a small constant. In this case, the condition that $\sum_t \alpha_t^2(s) < \infty$ is invalid anymore. When α is constant, it can still be shown that the algorithm converges in the sense of expectation sense.

e. Comparison between TD and MC:

TD/Sarsa learning	MC learning
Online: TD learning is online. It can update the state/action values immediately after receiving a reward.	**Offline:** MC learning is offline. It has to wait until an episode has been completely collected.
Continuing tasks: Since TD learning is online, it can handle both episodic and continuing tasks.	**Episodic tasks:** Since MC learning is offline, it can only handle episodic tasks that has terminate states.
Bootstrapping: TD bootstraps because the update of a value relies on the previous estimate of this value. Hence, it requires initial guesses.	**Non-bootstrapping:** MC is not bootstrapping, because it can directly estimate state/action values without any initial guess.
Low estimation variance: TD has lower than MC because there are fewer random variables. For instance, Sarsa requires $R_{t+1}, S_{t+1}, A_{t+1}$.	**High estimation variance:** To estimate $q_\pi(s_t, a_t)$, we need samples of $R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots$. Suppose the length of each episode is L. There are $\lvert\mathcal{A}\rvert^L$ possible episodes.

However, MC learning is unbiased, while TD is likely to include bias at the beginning.

3. TD learning of action values: **Sarsa**
   a. Algorithm: (replace $s \to v_\pi(s)$ by $(s,a) \to q_\pi(s,a)$)

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)\Big[q_t(s_t, a_t) - [r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})]\Big],$$

$$q_{t+1}(s, a) = q_t(s, a), \quad \forall (s, a) \neq (s_t, a_t),$$

where $t = 0, 1, 2, \ldots$.

- $q_t(s_t, a_t)$ is an estimate of $q_\pi(s_t, a_t)$;
- $\alpha_t(s_t, a_t)$ is the learning rate depending on s_t, a_t.

Sarsa is the abbreviation of state-action-reward-state-action.

b. Mathematically, Sarsa is a stochastic approximation algorithm solving the following equation:

$$q_\pi(s,a) = \mathbb{E}\left[R + \gamma q_\pi(S', A')|s, a\right], \quad \forall s, a.$$

(another expression of Bellman equation expressed in terms of action values)
The algorithm is used for policy evaluation.

c. Convergence:

Theorem (Convergence of Sarsa learning)

By the Sarsa algorithm, $q_t(s,a)$ converges with probability 1 to the action value $q_\pi(s,a)$ as $t \to \infty$ for all (s,a) if $\sum_t \alpha_t(s,a) = \infty$ and $\sum_t \alpha_t^2(s,a) < \infty$ for all (s,a).

d. Pseudocode (added an ϵ-greedy policy improvement step):

Pseudocode: Policy searching by Sarsa

For each episode, do

If the current s_t is not the target state, do

Collect the experience $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$: In particular, take action a_t following $\pi_t(s_t)$, generate r_{t+1}, s_{t+1}, and then take action a_{t+1} following $\pi_t(s_{t+1})$.

Update q-value:

$$q_{t+1}(s_t,a_t) = q_t(s_t,a_t) - \alpha_t(s_t,a_t)\Big[q_t(s_t,a_t) - [r_{t+1} + \gamma q_t(s_{t+1},a_{t+1})]\Big]$$

Update policy:

$$\pi_{t+1}(a|s_t) = 1 - \frac{\epsilon}{|\mathcal{A}|}(|\mathcal{A}| - 1) \text{ if } a = \arg\max_a q_{t+1}(s_t,a)$$
$$\pi_{t+1}(a|s_t) = \frac{\epsilon}{|\mathcal{A}|} \text{ otherwise}$$

- The policy of s_t is updated immediately after $q(s_t,a_t)$ is updated.
  This is based on the idea of generalized policy iteration.

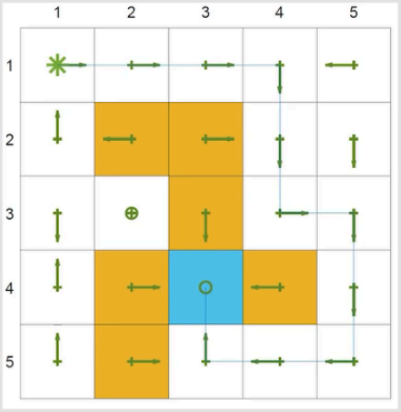
e. Sarsa-Examples:

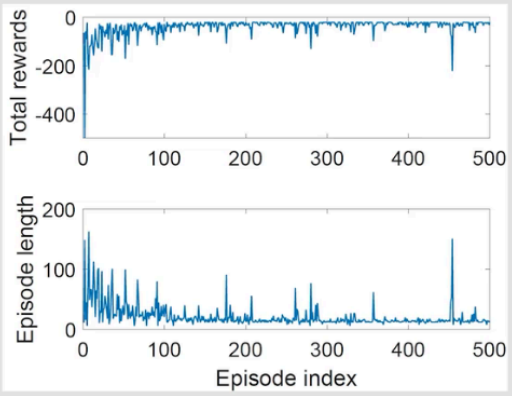
- The task is to find a good path from a specific starting state to the target state.
- Each episode starts from the top-left state and end in the target state.

(Not need to find out the optimal policy for every state.)

- $r_{\text{target}} = 0$, $r_{\text{forbidden}} = r_{\text{boundary}} = -10$, and $r_{\text{other}} = -1$. The learning rate is $\alpha = 0.1$ and the value of ϵ is 0.1.

- The left figures above show the final policy obtained by Sarsa.
  - Not all states have the optimal policy.
- The right figures show the total reward and length of every episode.
  - The metric of total reward per episode will be frequently used.





f. Variant: **Expected Sarsa**

$$q_{t+1}(s_t,a_t) = q_t(s_t,a_t) - \alpha_t(s_t,a_t)\Big[q_t(s_t,a_t) - (r_{t+1} + \gamma \mathbb{E}[q_t(s_{t+1},A)])\Big],$$

$$q_{t+1}(s,a) = q_t(s,a), \quad \forall (s,a) \neq (s_t,a_t),$$

where

$$\mathbb{E}[q_t(s_{t+1},A)]) = \sum_a \pi_t(a|s_{t+1}) q_t(s_{t+1},a) \doteq v_t(s_{t+1})$$

is the expected value of $q_t(s_{t+1},a)$ under policy π_t.

- The TD target is changed from $r_{t+1} + \gamma q_t(s_{t+1},a_{t+1})$ as in Sarsa to $r_{t+1} + \gamma \mathbb{E}[q_t(s_{t+1},A)]$ as in Expected Sarsa.
- Need more computation. But it is beneficial in the sense that it reduces the estimation variances because it reduces random variables in Sarsa from $\{s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}\}$ to $\{s_t, a_t, r_{t+1}, s_{t+1}\}$.

*Mathematically:

What does the algorithm do mathematically? Expected Sarsa is a stochastic approximation algorithm for solving the following equation:

$$q_\pi(s,a) = \mathbb{E}\Big[R_{t+1} + \gamma \mathbb{E}_{A_{t+1}\sim\pi(S_{t+1})}[q_\pi(S_{t+1},A_{t+1})]\Big|S_t = s, A_t = a\Big], \quad \forall s,a.$$

The above equation is another expression of the Bellman equation:

$$q_\pi(s,a) = \mathbb{E}\Big[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s, A_t = a\Big],$$

g. Variant: **n-step Sarsa** (can unify Sarsa and MC learning)

The definition of action value is

$$q_\pi(s,a) = \mathbb{E}[G_t | S_t = s, A_t = a].$$

The discounted return G_t can be written in different forms as

$$\begin{aligned}
\text{Sarsa} \longleftarrow \quad & G_t^{(1)} = R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}), \\
& G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2}, A_{t+2}), \\
& \vdots \\
n\text{-step Sarsa} \longleftarrow \quad & G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^n q_\pi(S_{t+n}, A_{t+n}), \\
& \vdots \\
\text{MC} \longleftarrow \quad & G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots
\end{aligned}$$

- Sarsa aims to solve

$$q_\pi(s,a) = \mathbb{E}[G_t^{(1)} | s, a] = \mathbb{E}[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | s, a].$$

- MC learning aims to solve

$$q_\pi(s,a) = \mathbb{E}[G_t^{(\infty)} | s, a] = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | s, a].$$

- An intermediate algorithm called *n-step Sarsa* aims to solve

$$q_\pi(s,a) = \mathbb{E}[G_t^{(n)} | s, a] = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^n q_\pi(S_{t+n}, A_{t+n}) | s, a].$$

The only difference lies in TD target.

- Since n-step Sarsa includes Sarsa and MC learning as two extreme cases, its performance is a blend of Sarsa and MC learning:
  - If n is large, its performance is close to MC learning and hence has a large variance but a small bias.
  - If n is small, its performance is close to Sarsa and hence has a relatively large bias due to the initial guess and relatively low variance.
- Finally, n-step Sarsa is also for policy evaluation. It can be combined with the policy improvement step to search for optimal policies.

4. TD learning of optimal action values: **Q-learning**
   a. Algorithm:

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t) \left[q_t(s_t, a_t) - [r_{t+1} + \gamma \max_{a \in \mathcal{A}} q_t(s_{t+1}, a)] \right],$$

$$q_{t+1}(s,a) = q_t(s,a), \quad \forall (s,a) \neq (s_t, a_t),$$

The only difference lies in TD target.

- The TD target in Q-learning is $r_{t+1} + \gamma \max_{a \in \mathcal{A}} q_t(s_{t+1}, a)$
- The TD target in Sarsa is $r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})$.

b. Mathematically,

It aims to solve

$$q(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_a q(S_{t+1}, a) \middle| S_t = s, A_t = a\right], \quad \forall s, a.$$

which is the Bellman optimality equation expressed in terms of action value.

c. **Off-policy** vs **on-policy** ***(a classification method for **all RL** algorithms)

There exist two policies in a TD learning task:

- The behavior policy is used to generate experience samples.
- The target policy is constantly updated toward an optimal policy.

On-policy vs off-policy:

- When the behavior policy is the same as the target policy, such kind of learning is called on-policy.
- When they are different, the learning is called off-policy.

Advantages of off-policy learning:

- It can search for optimal policies based on the experience samples generated by any other policies.
  - As an important special case, the behavior policy can be selected to be exploratory. For example, if we would like to estimate the action values of all state-action pairs, we can use a exploratory policy to generate episodes visiting every state-action pair sufficiently many times.

*How to judge if a TD algorithm is on-policy or off-policy?

- First, check what the algorithm does mathematically.
- Second, check what things are required to implement the algorithm.

*Analyze three following algorithms:

Sarsa: on-policy

- First, Sarsa aims to solve the Bellman equation of a given policy π:

$$q_\pi(s,a) = \mathbb{E}\left[R + \gamma q_\pi(S', A') | s, a\right], \quad \forall s, a.$$

where $R \sim p(R|s,a)$, $S' \sim p(S'|s,a)$, $A' \sim \pi(A'|S')$.

- Second, the algorithm is

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)\Big[q_t(s_t, a_t) - [r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})]\Big],$$

  which requires $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$:
  - If (s_t, a_t) is given, then r_{t+1} and s_{t+1} do not depend on any policy!
  - a_{t+1} is generated following $\pi_t(s_{t+1})$!
- π_t is both the target and behavior policy.

MC learning: on-policy

- First, the MC method aims to solve

$$q_\pi(s, a) = \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \ldots | S_t = s, A_t = a\right], \quad \forall s, a.$$

  where the sample is generated following a given policy π.
- Second, the implementation of the MC method is

$$q(s, a) \approx r_{t+1} + \gamma r_{t+2} + \ldots$$

- A policy is used to generate samples, which is further used to estimate the action values of the policy. Based on the action values, we can improve the policy.

Q-learning: <u>off-policy</u>

- First, Q-learning aims to solve the Bellman optimality equation

$$q(s, a) = \mathbb{E}\left[R_{t+1} + \gamma \max_a q(S_{t+1}, a) \Big| S_t = s, A_t = a\right], \quad \forall s, a.$$

- Second, the algorithm is

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)\left[q_t(s_t, a_t) - [r_{t+1} + \gamma \max_{a \in \mathcal{A}} q_t(s_{t+1}, a)]\right]$$

  which requires $(s_t, a_t, r_{t+1}, s_{t+1})$.
  - If (s_t, a_t) is given, then r_{t+1} and s_{t+1} do not depend on any policy!
- The behavior policy to generate a_t from s_t can be anything. The target policy is the optimal policy.

d. Pseudocode:
Q-learning can be implemented in an either <u>off-policy or on-policy</u> way.

Pseudocode: Policy searching by Q-learning (on-policy version)

For each episode, do

If the current s_t is not the target state, do

Collect the experience $(s_t, a_t, r_{t+1}, s_{t+1})$: In particular, take action a_t following $\pi_t(s_t)$, generate r_{t+1}, s_{t+1}.

Update q-value:

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)\Big[q_t(s_t, a_t) - [r_{t+1} + \gamma \max_a q_t(s_{t+1}, a)]\Big]$$

Update policy:

$$\pi_{t+1}(a|s_t) = 1 - \frac{\epsilon}{|\mathcal{A}|}(|\mathcal{A}| - 1) \text{ if } a = \arg\max_a q_{t+1}(s_t, a)$$

$$\pi_{t+1}(a|s_t) = \frac{\epsilon}{|\mathcal{A}|} \text{ otherwise}$$

Pseudocode: Optimal policy search by Q-learning (off-policy version)

For each episode $\{s_0, a_0, r_1, s_1, a_1, r_2, \dots\}$ generated by π_b, do

For each step $t = 0, 1, 2, \dots$ of the episode, do

Update q-value:

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)\Big[q(s_t, a_t) - [r_{t+1} + \gamma \max_a q_t(s_{t+1}, a)]\Big]$$

Update target policy:

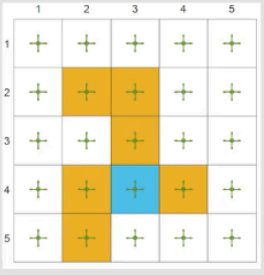
$$\pi_{T,t+1}(a|s_t) = 1 \text{ if } a = \arg\max_a q_{t+1}(s_t, a)$$

$$\pi_{T,t+1}(a|s_t) = 0 \text{ otherwise}$$

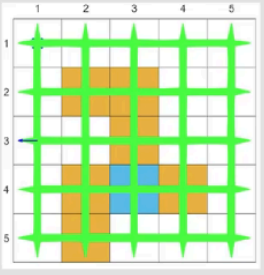
We do not have to make π_T ϵ-greedy, since π_b is in charge of generating exploratory experience.

e. Example:

- The task in these examples is to find an optimal policy for all the states.
- The reward setting is $r_{\text{boundary}} = r_{\text{forbidden}} = -1$, and $r_{\text{target}} = 1$. The discount rate is $\gamma = 0.9$. The learning rate is $\alpha = 0.1$.

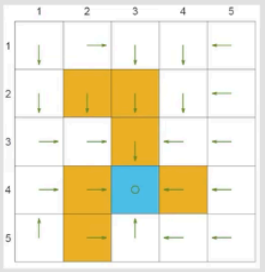


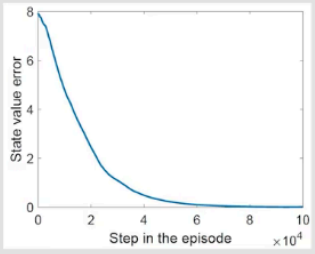
(a) Behavior policy



(b) Generated episode

The policy found by off-policy Q-learning:

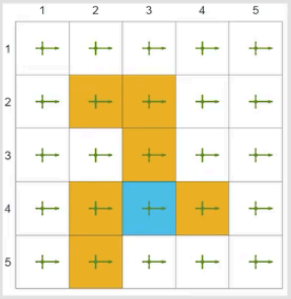


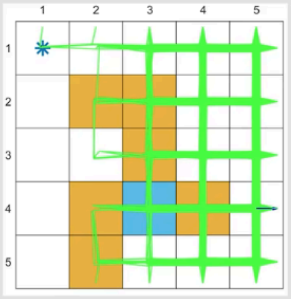


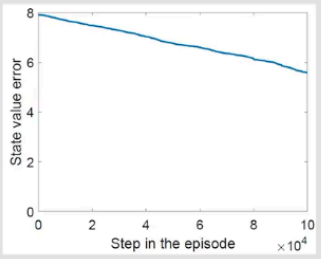
(a) Estimated policy (b) State value error

ao

If the behaviour policy is not sufficiently exploratory:







(a) Behavior policy (b) Generated episode (c) Q-learning result

$\epsilon = 0.5$

5. A unified point of view
   a. Expression:

All the algorithms we introduced in this lecture can be expressed in a unified expression:

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)[q_t(s_t, a_t) - \bar{q}_t],$$

where $\bar{q}_t$ is the *TD target*.

Different TD algorithms have different $\bar{q}_t$.

Algorithm	**Expression of $\bar{q}_t$**
Sarsa	$\bar{q}_t = r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})$
n-step Sarsa	$\bar{q}_t = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^n q_t(s_{t+n}, a_{t+n})$
Expected Sarsa	$\bar{q}_t = r_{t+1} + \gamma \sum_a \pi_t(a\|s_{t+1}) q_t(s_{t+1}, a)$
Q-learning	$\bar{q}_t = r_{t+1} + \gamma \max_a q_t(s_{t+1}, a)$
Monte Carlo	$\bar{q}_t = r_{t+1} + \gamma r_{t+2} + \ldots$

The MC method can also be expressed in this unified expression by setting $\alpha_t(s_t, a_t) = 1$ and hence $q_{t+1}(s_t, a_t) = \bar{q}_t$.

   b. Mathematical essence:

All the algorithms can be viewed as stochastic approximation algorithms solving the Bellman equation or Bellman optimality equation:

Algorithm	Equation aimed to solve
Sarsa	BE: $q_\pi(s,a) = \mathbb{E}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \middle| S_t = s, A_t = a\right]$
n-step Sarsa	BE: $q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^n q_\pi(s_{t+n}, a_{t+n}) \middle| S_t = s, A_t = a]$
Expected Sarsa	BE: $q_\pi(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \mathbb{E}_{A_{t+1}}[q_\pi(S_{t+1}, A_{t+1})] \middle| S_t = s, A_t = a\right]$
Q-learning	BOE: $q(s,a) = \mathbb{E}\left[R_{t+1} + \max_a q(S_{t+1}, a) \middle| S_t = s, A_t = a\right]$
Monte Carlo	BE: $q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \ldots \middle| S_t = s, A_t = a]$

Chapter 8: Value Function Approximation

(tabular representation to function representation; the Introduction of neural networks)

1. Motivating examples: curve fitting

*So far, state and action values are represented by **tables**:

- For example, action value:

	a_1	a_2	a_3	a_4	a_5
s_1	$q_\pi(s_1, a_1)$	$q_\pi(s_1, a_2)$	$q_\pi(s_1, a_3)$	$q_\pi(s_1, a_4)$	$q_\pi(s_1, a_5)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
s_9	$q_\pi(s_9, a_1)$	$q_\pi(s_9, a_2)$	$q_\pi(s_9, a_3)$	$q_\pi(s_9, a_4)$	$q_\pi(s_9, a_5)$

- Advantage: intuitive and easy to analyze
- Disadvantage: difficult to handle large or continuous state or action spaces. Two aspects: 1) storage; 2) generalization ability

Function fitting:

- Suppose there are one-dimensional states $s_1, \ldots, s_{|\mathcal{S}|}$.
- Their state values are $v_\pi(s_1), \ldots, v_\pi(s_{|\mathcal{S}|})$, where π is a given policy.
- Suppose $|\mathcal{S}|$ is very large and we hope to use a simple curve to approximate these dots to save storage.

*Use the simplest **straight line**:

$$\hat{v}(s,w) = as + b = \underbrace{[s,1]}_{\phi^T(s)} \underbrace{\begin{bmatrix} a \\ b \end{bmatrix}}_{w} = \phi^T(s)w$$

where

- w is the parameter vector
- $\phi(s)$ the feature vector of s
- $\hat{v}(s,w)$ is linear in w

Benefits:

- The tabular representation needs to store $|\mathcal{S}|$ state values. Now, we need to only store two parameters a and b.
- Every time we would like to use the value of s, we can calculate $\phi^T(s)w$.

Cost: the state values can not be represented accurately. (value "approximation")
*We can increase the order of the curve to improve accuracy:
Second-order curve:

$$\hat{v}(s,w) = as^2 + bs + c = \underbrace{[s^2, s, 1]}_{\phi^T(s)} \underbrace{\begin{bmatrix} a \\ b \\ c \end{bmatrix}}_{w} = \phi^T(s)w.$$

In this case,

- The dimensions of w and $\phi(s)$ increase, but the values may be fitted more accurately.
- Although $\hat{v}(s,w)$ is nonlinear in s, it is linear in w. The nonlinearity is contained in $\phi(s)$.

*Idea: Approximate the state and action values using **parameterized functions:**
$\hat{v}(s,w) \approx v_\pi(s)$ where $w \in \mathbb{R}^m$ is the parameter vector.

- **Advantage:**
  1) **Storage:** The dimension of w may be much less than $|\mathcal{S}|$.
  2) **Generalization:** When a state s is visited, the parameter w is updated so that the values of some other unvisited states can also be updated. In this way, the learned values can generalize to unvisited states.

2. Algorithm for state value function approximation
   a. Objective function (scalar):

- To find the optimal w, we need two steps.
  - The first step is to define an objective function.
  - The second step is to derive algorithms optimizing the objective function.

The objective function:

$$J(w) = \mathbb{E}[(v_\pi(S) - \hat{v}(S, w))^2].$$

Our goal is to find the best *w* that can minimize *J*(*w*).
*How should we consider the **probability distribution of state value**?
Uniform? No, because the importance (or the frequency to be visited) of states varies. (Greater weight means **smaller error for a specific state**)
***Stationary distribution/Steady-state distribution**:
This distribution describes the **long-run behaviour** of a Markov process, in other words, **the probability that the agent is at any state** after it runs a long time following a policy.

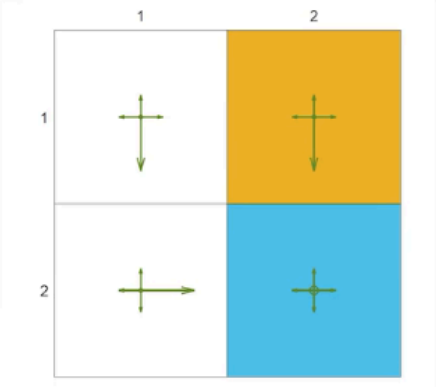
- Let $\{d_\pi(s)\}_{s\in\mathcal{S}}$ denote the stationary distribution of the Markov process under policy π. By definition, $d_\pi(s) \geq 0$ and $\sum_{s\in\mathcal{S}} d_\pi(s) = 1$.
- The objective function can be rewritten as

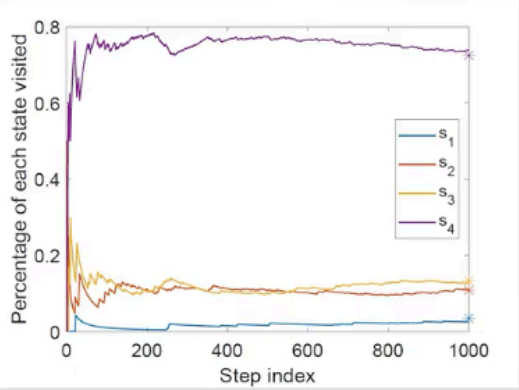
$$J(w) = \mathbb{E}[(v_\pi(S) - \hat{v}(S, w))^2] = \sum_{s\in\mathcal{S}} d_\pi(s)(v_\pi(s) - \hat{v}(s, w))^2.$$

  This function is a weighted squared error.
- Since more frequently visited states have higher values of $d_\pi(s)$, their weights in the objective function are also higher than those rarely visited states.

Illustrative example:

$$d_\pi(s) \approx \frac{n_\pi(s)}{\sum_{s'\in\mathcal{S}} n_\pi(s')}$$





How to predict stationary distribution in theory?

By calculating the eigenvector of the state transition matrix.

$$d_\pi^T = d_\pi^T P_\pi$$

For this example, we have P_π as

$$P_\pi = \begin{bmatrix} 0.3 & 0.1 & 0.6 & 0 \\ 0.1 & 0.3 & 0 & 0.6 \\ 0.1 & 0 & 0.3 & 0.6 \\ 0 & 0.1 & 0.1 & 0.8 \end{bmatrix}.$$

It can be calculated that the left eigenvector for the eigenvalue of one is

$$d_\pi = \begin{bmatrix} 0.0345, 0.1084, 0.1330, 0.7241 \end{bmatrix}^T$$

b. Optimization algorithms:

While we have the objective function, the next step is to optimize it.

- To minimize the objective function $J(w)$, we can use the gradient-descent algorithm:

$$w_{k+1} = w_k - \alpha_k \nabla_w J(w_k)$$

The true gradient is

$$\begin{aligned} \nabla_w J(w) &= \nabla_w \mathbb{E}[(v_\pi(S) - \hat{v}(S, w))^2] \\ &= \mathbb{E}[\nabla_w (v_\pi(S) - \hat{v}(S, w))^2] \\ &= 2\mathbb{E}[(v_\pi(S) - \hat{v}(S, w))(-\nabla_w \hat{v}(S, w))] \\ &= -2\mathbb{E}[(v_\pi(S) - \hat{v}(S, w))\nabla_w \hat{v}(S, w)] \end{aligned}$$

How to avoid the calculation of expectation? Use **SGD**!

$$w_{t+1} = w_t + \alpha_t (v_\pi(s_t) - \hat{v}(s_t, w_t))\nabla_w \hat{v}(s_t, w_t),$$

where s_t is a sample of S. Here, $2\alpha_k$ is merged to α_k.

- This algorithm is not implementable because it requires the true state value v_π, which is the unknown to be estimated.
- We can replace $v_\pi(s_t)$ with an approximation so that the algorithm is implementable.

- First, **Monte Carlo learning with function approximation**
  Let g_t be the discounted return starting from s_t in the episode. Then, g_t can be used to approximate $v_\pi(s_t)$. The algorithm becomes

$$w_{t+1} = w_t + \alpha_t(g_t - \hat{v}(s_t, w_t))\nabla_w \hat{v}(s_t, w_t).$$

- Second, **TD learning with function approximation**
  By the spirit of TD learning, $r_{t+1} + \gamma\hat{v}(s_{t+1}, w_t)$ can be viewed as an approximation of $v_\pi(s_t)$. Then, the algorithm becomes

$$w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma\hat{v}(s_{t+1}, w_t) - \hat{v}(s_t, w_t)\right] \nabla_w \hat{v}(s_t, w_t).$$

*Pseudocode:

Pseudocode: TD learning with function approximation

Initialization: A function $\hat{v}(s, w)$ that is a differentiable in w. Initial parameter w_0.

Aim: Approximate the true state values of a given policy π.

For each episode generated following the policy π, do

For each step (s_t, r_{t+1}, s_{t+1}), do

In the general case,

$$w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma\hat{v}(s_{t+1}, w_t) - \hat{v}(s_t, w_t)\right] \nabla_w \hat{v}(s_t, w_t)$$

In the linear case,

$$w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma\phi^T(s_{t+1})w_t - \phi^T(s_t)w_t\right]\phi(s_t)$$

(Can only estimate state value so far)

c. Selection of **function approximators**:

*Linear function (TD-Linear): $\nabla_w \hat{v}(s, w) = \phi(s).$

$$w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma\phi^T(s_{t+1})w_t - \phi^T(s_t)w_t\right]\phi(s_t),$$

The tabular representation is a special case of linear function approximation:

Recall that the TD-Linear algorithm is

$$w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma\phi^T(s_{t+1})w_t - \phi^T(s_t)w_t\right]\phi(s_t),$$

- When $\phi(s_t) = e_s$, the above algorithm becomes

$$w_{t+1} = w_t + \alpha_t \left(r_{t+1} + \gamma w_t(s_{t+1}) - w_t(s_t)\right) e_{s_t}.$$

This is a vector equation that merely updates the s_tth entry of w_t.

- Multiplying $e_{s_t}^T$ on both sides of the equation gives

$$w_{t+1}(s_t) = w_t(s_t) + \alpha_t \left(r_{t+1} + \gamma w_t(s_{t+1}) - w_t(s_t)\right),$$

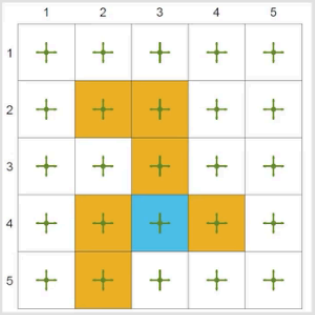
which is exactly the tabular TD algorithm.

*Neuron network: (as a nonlinear function approximator)

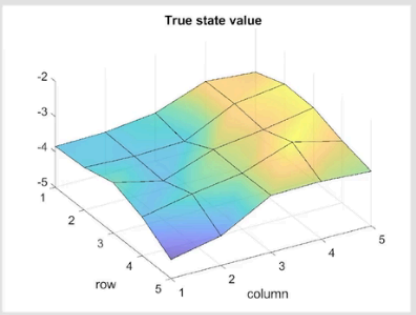
The input of the NN is the state, the output is $\hat{v}(s, w)$, and the network parameter is w.

d. Illustrative examples:

- The true state values and the 3D visualization



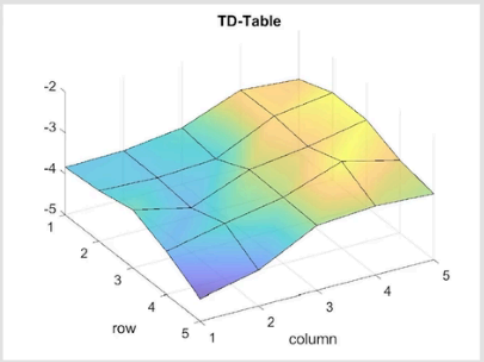
	1	2	3	4	5
1	-3.8	-3.8	-3.6	-3.1	-3.2
2	-3.8	-3.8	-3.8	-3.1	-2.9
3	-3.6	-3.9	-3.4	-3.2	-2.9
4	-3.9	-3.6	-3.4	-2.9	-3.2
5	-4.5	-4.2	-3.4	-3.4	-3.5

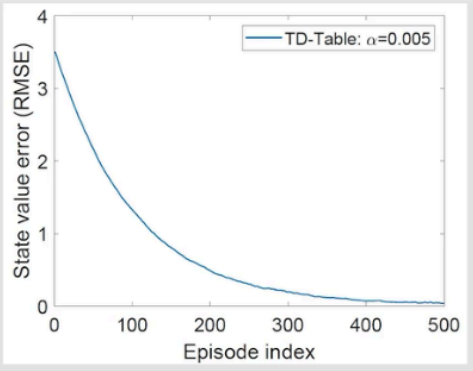


Experience samples:

- 500 episodes were generated following the given policy.
- Each episode has 500 steps and starts from a randomly selected state-action pair following a uniform distribution.

Using TD-Table:





Using TD-Linear:

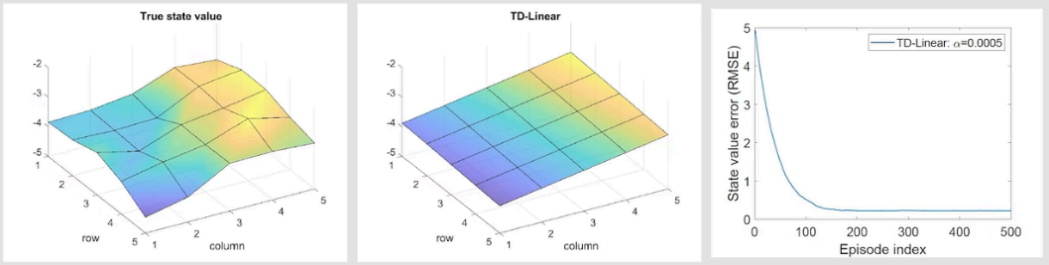
Feature vector selection:

$$\phi(s) = \begin{bmatrix} 1 \\ x \\ y \end{bmatrix} \in \mathbb{R}^3.$$

In this case, the approximated state value is

$$\hat{v}(s, w) = \phi^T(s)w = [1, x, y] \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = w_1 + w_2 x + w_3 y.$$

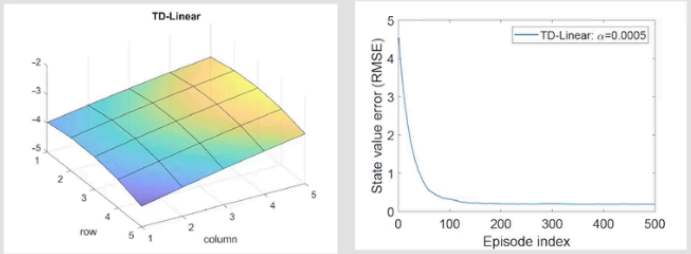
Results by the TD-Linear algorithm:



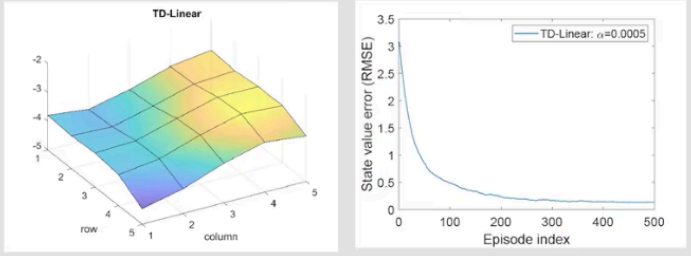
To enhance the approximation ability, we can use high-order feature vectors and hence more parameters.

$$\phi(s) = [1, x, y, x^2, y^2, xy]^T \in \mathbb{R}^6.$$

$$\phi(s) = [1, x, y, x^2, y^2, xy, x^3, y^3, x^2y, xy^2]^T \in \mathbb{R}^{10}.$$



The above figure: $\phi(s) \in \mathbb{R}^6$



The above figure: $\phi(s) \in \mathbb{R}^{10}$

(High-order is still TD-linear!)

3. Sarsa + function approximation
   a. Algorithm:

$$w_{t+1} = w_t + \alpha_t \Big[r_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}, w_t) - \hat{q}(s_t, a_t, w_t) \Big] \nabla_w \hat{q}(s_t, a_t, w_t).$$

   b. Pseudocode (combine PE and PI):

Pseudocode: Sarsa with function approximation

Aim: Search a policy that can lead the agent to the target from an initial state-action pair (s_0, a_0).

For each episode, do

If the current s_t is not the target state, do

Take action a_t following $\pi_t(s_t)$, generate r_{t+1}, s_{t+1}, and then take action a_{t+1} following $\pi_t(s_{t+1})$

Value update (parameter update):

$$w_{t+1} = w_t + \alpha_t \Big[r_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}, w_t) - \hat{q}(s_t, a_t, w_t) \Big] \nabla_w \hat{q}(s_t, a_t, w_t)$$

Policy update:

$$\pi_{t+1}(a|s_t) = 1 - \frac{\varepsilon}{|\mathcal{A}(s)|}(|\mathcal{A}(s)| - 1) \quad \text{if } a = \arg\max_{a \in \mathcal{A}(s_t)} \hat{q}(s_t, a, w_{t+1})$$

$$\pi_{t+1}(a|s_t) = \frac{\varepsilon}{|\mathcal{A}(s)|} \quad \text{otherwise}$$

4. Q-learning + function approximation
   a. Algorithm:

$$w_{t+1} = w_t + \alpha_t \Big[r_{t+1} + \gamma \max_{a \in \mathcal{A}(s_{t+1})} \hat{q}(s_{t+1}, a, w_t) - \hat{q}(s_t, a_t, w_t) \Big] \nabla_w \hat{q}(s_t, a_t, w_t)$$

   b. Pseudocode:
   Here we use an on-policy version.

Pseudocode: Q-learning with function approximation (on-policy version)

Initialization: Initial parameter vector w_0. Initial policy π_0. Small $\varepsilon > 0$.
Aim: Search a good policy that can lead the agent to the target from an initial state-action pair (s_0, a_0).

For each episode, do

If the current s_t is not the target state, do

Take action a_t following $\pi_t(s_t)$, and generate r_{t+1}, s_{t+1}

Value update (parameter update):

$$w_{t+1} = w_t + \alpha_t \Big[r_{t+1} + \gamma \max_{a \in \mathcal{A}(s_{t+1})} \hat{q}(s_{t+1}, a, w_t) - \hat{q}(s_t, a_t, w_t) \Big] \nabla_w \hat{q}(s_t, a_t, w_t)$$

Policy update:

$\pi_{t+1}(a|s_t) = 1 - \frac{\varepsilon}{|\mathcal{A}(s)|}(|\mathcal{A}(s)| - 1)$ if $a = \arg\max_{a \in \mathcal{A}(s_t)} \hat{q}(s_t, a, w_{t+1})$

$\pi_{t+1}(a|s_t) = \frac{\varepsilon}{|\mathcal{A}(s)|}$ otherwise

5. Deep Q-learning (DQN)

a. Loss function (TD error):

Deep Q-learning aims to minimize the objective function/loss function:

$$J(w) = \mathbb{E}\left[\left(R + \gamma \max_{a \in \mathcal{A}(S')} \hat{q}(S', a, w) - \hat{q}(S, A, w)\right)^2\right],$$

where (S, A, R, S') are random variables.

- This is actually the Bellman optimality error. That is because

$$q(s, a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a \in \mathcal{A}(S_{t+1})} q(S_{t+1}, a) \middle| S_t = s, A_t = a\right], \quad \forall s, a$$

The value of $R + \gamma \max_{a \in \mathcal{A}(S')} \hat{q}(S', a, w) - \hat{q}(S, A, w)$ should be zero in the expectation sense

*We want to minimize the objective function using gradient-descent.
But how to **calculate the gradient**?

$$J(w) = \mathbb{E}\left[\left(R + \gamma \max_{a \in \mathcal{A}(S')} \hat{q}(S', a, w) - \hat{q}(S, A, w)\right)^2\right],$$

the parameter w not only appears in $\hat{q}(S, A, w)$ but also in

$$y \doteq R + \gamma \max_{a \in \mathcal{A}(S')} \hat{q}(S', a, w)$$

- For the sake of simplicity, we can assume that w in y is fixed (at least for a while) when we calculate the gradient.

To do that, we can introduce two networks.

- One is a main network representing $\hat{q}(s, a, w)$
- The other is a target network $\hat{q}(s, a, w_T)$.

The objective function in this case degenerates to

$$J = \mathbb{E}\left[\left(R + \gamma \max_{a \in \mathcal{A}(S')} \hat{q}(S', a, w_T) - \hat{q}(S, A, w)\right)^2\right],$$

where w_T is the target network parameter.

When w_T is fixed, the gradient of J can be easily obtained as

$$\nabla_w J = \mathbb{E}\left[\left(R + \gamma \max_{a \in \mathcal{A}(S')} \hat{q}(S', a, w_T) - \hat{q}(S, A, w)\right) \nabla_w \hat{q}(S, A, w)\right].$$

We first fix w_T and using the gradient above to update w, after a while we assign the new value of w to w_T, such a cycle.

b. Techniques:

*__Two networks__, a main network and a target network.

- Let w and w_T denote the parameters of the main and target networks, respectively. They are set to be the same initially.
- In every iteration, we draw a mini-batch of samples $\{(s, a, r, s')\}$ from the replay buffer (will be explained later).
- The inputs of the networks include state s and action a. The target output is $y_T \doteq r + \gamma \max_{a \in \mathcal{A}(s')} \hat{q}(s', a, w_T)$. Then, we directly minimize the TD error or called loss function $(y_T - \hat{q}(s, a, w))^2$ over the mini-batch $\{(s, a, y_T)\}$.

*__Experience replay (经验回放)__

- After we have collected some experience samples, we do NOT use these samples in the order they were collected.
- Instead, we store them in a set, called replay buffer $\mathcal{B} \doteq \{(s, a, r, s')\}$
- Every time we train the neural network, we can draw a mini-batch of random samples from the replay buffer.
- The draw of samples, or called experience replay, should follow a uniform distribution (why?).

Let's get back to the objective function:

$$J = \mathbb{E}\left[\left(R + \gamma \max_{a \in \mathcal{A}(S')} \hat{q}(S', a, w) - \hat{q}(S, A, w)\right)^2\right]$$

- $(S, A) \sim d$: (S, A) is an index and treated as a single random variable
- $R \sim p(R|S, A), S' \sim p(S'|S, A)$: R and S are determined by the system model.
- The distribution of the state-action pair (S, A) is assumed to be uniform.

- However, the samples are not uniformly collected because they are generated consequently by certain policies.
- To break the correlation between consequent samples, we can use the experience replay technique by uniformly drawing samples from the replay buffer.

That's why we need a replay buffer.
Experience replay is also more sample efficient. (can be used repeatedly)

c. Pseudocode:

Pseudocode: Deep Q-learning (off-policy version)

Aim: Learn an optimal target network to approximate the optimal action values from the experience samples generated by a behavior policy π_b.

Store the experience samples generated by π_b in a replay buffer $\mathcal{B} = \{(s, a, r, s')\}$

For each iteration, do

Uniformly draw a mini-batch of samples from $\mathcal{B}$

For each sample (s, a, r, s'), calculate the target value as $y_T = r + \gamma \max_{a \in \mathcal{A}(s')} \hat{q}(s', a, w_T)$, where w_T is the parameter of the target network

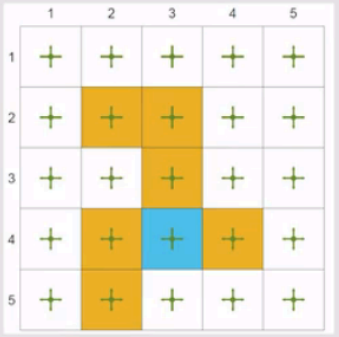
Update the main network to minimize $(y_T - \hat{q}(s, a, w))^2$ using the mini-batch $\{(s, a, y_T)\}$

Set $w_T = w$ every C iterations

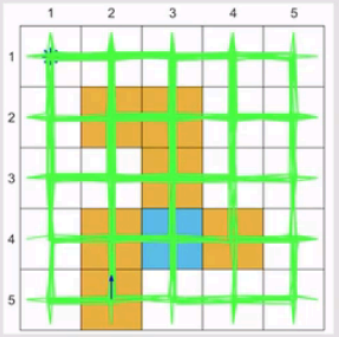
d. Example:

- This example aims to learn optimal action values for every state-action pair.
- Once the optimal action values are obtained, the optimal greedy policy can be obtained immediately.
- One single episode is used to train the network.
- This episode is generated by an exploratory behavior policy shown in Figure (a).
- The episode only has 1,000 steps! The tabular Q-learning requires 100,000 steps.
- A shallow neural network with one single hidden layer is used as a nonlinear approximator of $\hat{q}(s, a, w)$. The hidden layer has 100 neurons.

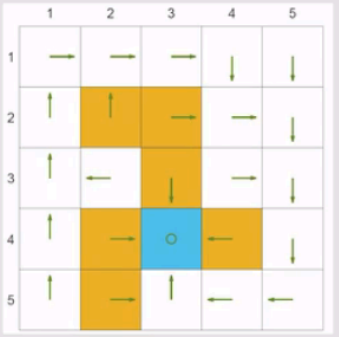
We can get optimal policies with very little data.



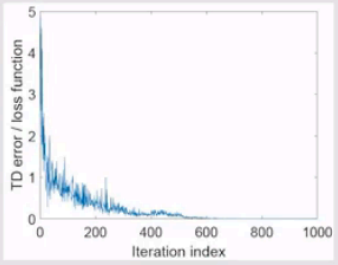
The behavior policy.



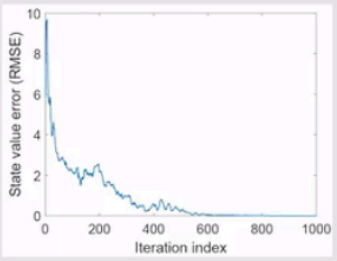
An episode of 1,000 steps.



The obtained policy.

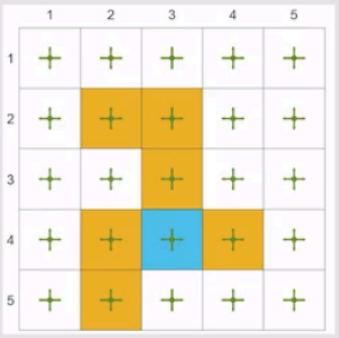


The TD error converges to zero.

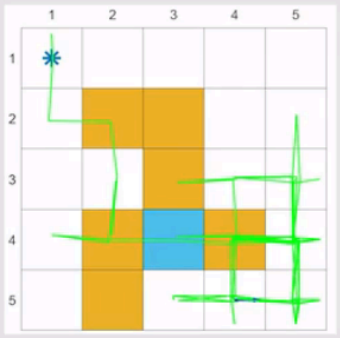


The state estimation error converges to zero.

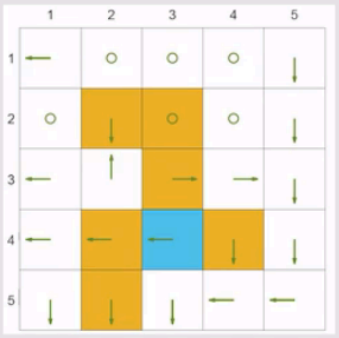
Change the episode length to 100: Insufficient data.



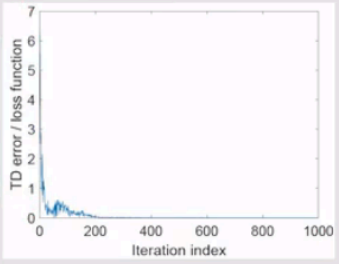
The behavior policy.



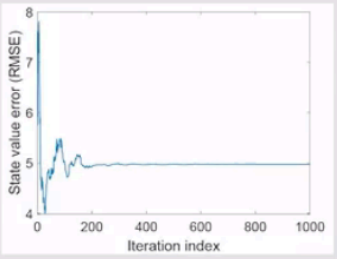
An episode of 100 steps.



The final policy.



The TD error converges to zero.



The state error does not converge to zero.

Chapter 9: Policy Function Approximation / Policy Gradient (PG) Methods

(value-based to policy-based)

1. Basic idea of PG
   *So far, policies are also represented by **tables**:

- The action probabilities of all states are stored in a table $\pi(a|s)$. Each entry of the table is indexed by a state and an action.

	a_1	a_2	a_3	a_4	a_5
s_1	$\pi(a_1\|s_1)$	$\pi(a_2\|s_1)$	$\pi(a_3\|s_1)$	$\pi(a_4\|s_1)$	$\pi(a_5\|s_1)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
s_9	$\pi(a_1\|s_9)$	$\pi(a_2\|s_9)$	$\pi(a_3\|s_9)$	$\pi(a_4\|s_9)$	$\pi(a_5\|s_9)$

- We can directly access or change a value in the table.

From tabular to function:

Now, policies can be represented by parameterized functions:

$$\pi(a|s,\theta)$$

where $\theta \in \mathbb{R}^m$ is a parameter vector.

- The function can be, for example, a neural network, whose input is s, output is the probability to take each action, and parameter is θ.
- **Advantage:** when the state space is large, the tabular representation will be of low efficiency in terms of storage and generalization.
- The function representation is also sometimes written as $\pi(a, s, \theta)$, $\pi_\theta(a|s)$, or $\pi_\theta(a, s)$.

*Differences between tabular and function:

- First, how to define optimal policies?
  - When represented as a table, a policy π is optimal if it can maximize *every state value*.
  - When represented by a function, a policy π is optimal if it can maximize certain *scalar metrics*.
- Second, how to access the probability of an action?
  - In the tabular case, the probability of taking a at s can be directly accessed by looking up the tabular policy.
  - In the case of function representation, we need to calculate the value of $\pi(a|s,\theta)$ given the function structure and the parameter.

- Third, how to update policies?
  - When represented by a table, a policy π can be updated by directly changing the entries in the table.
  - When represented by a parameterized function, a policy π cannot be updated in this way anymore. Instead, it can only be updated by changing *the parameter* θ.

*Basic idea of policy gradient:

- First, metrics (or objective functions) to define optimal policies: $J(\theta)$, which can define optimal policies.
- Second, gradient-based optimization algorithms to search for optimal policies:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\theta_t)$$

Although the idea is simple, the complication emerges when we try to answer the following questions.

- What appropriate metrics should be used?
- How to calculate the gradients of the metrics?

2. Metrics to define optimal policies (Objective function)
   a. The average value:

The first metric is the average state value or simply called average value. In particular, the metric is defined as

$$\bar{v}_\pi = \sum_{s \in \mathcal{S}} d(s) v_\pi(s)$$

- $\bar{v}_\pi$ is a weighted average of the state values.
- $d(s) \geq 0$ is the weight for state s.
- Since $\sum_{s \in \mathcal{S}} d(s) = 1$, we can interpret $d(s)$ as a probability distribution. Then, the metric can be written as

$$\bar{v}_\pi = \mathbb{E}[v_\pi(S)]$$

where $S \sim d$.

*Vector-product form:

$$\bar{v}_\pi = \sum_{s \in \mathcal{S}} d(s) v_\pi(s) = d^T v_\pi$$

where

$$v_\pi = [\dots, v_\pi(s), \dots]^T \in \mathbb{R}^{|\mathcal{S}|}$$
$$d = [\dots, d(s), \dots]^T \in \mathbb{R}^{|\mathcal{S}|}.$$

This expression is particularly useful when we analyze its gradient.

*How to select the distribution d?

The first case is that d is **independent** of the policy π.

- This case is relatively simple because the gradient of the metric is easier to calculate.
- In this case, we specifically denote d as d_0 and $\bar{v}_\pi$ as $\bar{v}_\pi^0$.
- How to select d_0?
  - One trivial way is to treat all the states equally important and hence select $d_0(s) = 1/|\mathcal{S}|$.
  - Another important case is that we are only interested in a specific state s_0. For example, the episodes in some tasks always start from the same state s_0. Then, we only care about the long-term return starting from s_0. In this case,

$$d_0(s_0) = 1, \quad d_0(s \neq s_0) = 0.$$

The second case is that d **depends** on the policy π.

- A common way to select d as $d_\pi(s)$, which is the stationary distribution under π. Details of stationary distribution can be found in the last lecture and the book.
  - One basic property of d_π is that it satisfies

$$d_\pi^T P_\pi = d_\pi^T,$$

    where P_π is the state transition probability matrix.
  - The interpretation of selecting d_π is as follows.
    - If one state is frequently visited in the long run, it is more important and deserves more weight.
    - If a state is hardly visited, then we give it less weight.

b. The average reward:

The second metric is average one-step reward or simply average reward. In particular, the metric is

$$\bar{r}_\pi \doteq \sum_{s\in\mathcal{S}} d_\pi(s) r_\pi(s) = \mathbb{E}[r_\pi(S)],$$

where $S \sim d_\pi$. Here,

$$r_\pi(s) \doteq \sum_{a\in\mathcal{A}} \pi(a|s) r(s,a)$$

is the average of the one-step immediate reward that can be obtained starting from state s, and

$$r(s,a) = \mathbb{E}[R|s,a] = \sum_r r p(r|s,a)$$

- The weight d_π is the stationary distribution.
- As its name suggests, $\bar{r}_\pi$ is simply a weighted average of the one-step immediate rewards.

*An equivalent definition:

- Suppose an agent follows a given policy and generate a trajectory with the rewards as $(R_{t+1}, R_{t+2}, \dots)$.
- The average single-step reward along this trajectory is

$$\lim_{n\to\infty} \frac{1}{n}\mathbb{E}\Big[R_{t+1} + R_{t+2} + \cdots + R_{t+n}|S_t = s_0\Big]$$
$$= \lim_{n\to\infty} \frac{1}{n}\mathbb{E}\left[\sum_{k=1}^{n} R_{t+k}|S_t = s_0\right]$$

  where s_0 is the starting state of the trajectory.

An important property: (The starting state does not matter)

$$\lim_{n\to\infty} \frac{1}{n}\mathbb{E}\left[\sum_{k=1}^{n} R_{t+k}|S_t = s_0\right] = \lim_{n\to\infty} \frac{1}{n}\mathbb{E}\left[\sum_{k=1}^{n} R_{t+k}\right]$$
$$= \sum_s d_\pi(s) r_\pi(s)$$

This form is common in the literature.

Remarks:

1:

- All these metrics are functions of π.
- Since π is parameterized by θ, these metrics are functions of θ.
- In other words, different values of θ can generate different metric values.
- Therefore, we can search for the optimal values of θ to maximize these metrics.

2:

- Intuitively, $\bar{r}_\pi$ is more short-sighted because it merely considers the immediate rewards, whereas $\bar{v}_\pi$ considers the total reward overall steps.
- However, the two metrics are equivalent to each other.
  In the discounted case where $\gamma < 1$, it holds that

$$\bar{r}_\pi = (1-\gamma)\bar{v}_\pi.$$

3: (An equivalent definition of v_π)

You will see the following metric often in the literature:

$$J(\theta) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1}\right]$$

(By total expectation formula)

3. Gradients of the metrics

Summary of the results about the gradients:

$$\nabla_\theta J(\theta) = \sum_{s\in\mathcal{S}} \eta(s) \sum_{a\in\mathcal{A}} \nabla_\theta \pi(a|s,\theta) q_\pi(s,a)$$

where

- $J(\theta)$ can be $\bar{v}_\pi$, $\bar{r}_\pi$, or $\bar{v}^0_\pi$.
- "$=$" may denote strict equality, approximation, or proportional to.
- η is a distribution or weight of the states.

Some specific results:

$$\nabla_\theta \bar{r}_\pi \simeq \sum_s d_\pi(s) \sum_a \nabla_\theta \pi(a|s,\theta) q_\pi(s,a),$$

$$\nabla_\theta \bar{v}_\pi = \frac{1}{1-\gamma} \nabla_\theta \bar{r}_\pi$$

$$\nabla_\theta \bar{v}_\pi^0 = \sum_{s \in \mathcal{S}} \rho_\pi(s) \sum_{a \in \mathcal{A}} \nabla_\theta \pi(a|s,\theta) q_\pi(s,a)$$

*

A compact and useful form of the gradient:

$$\begin{aligned} \nabla_\theta J(\theta) &= \sum_{s \in \mathcal{S}} \eta(s) \sum_{a \in \mathcal{A}} \nabla_\theta \pi(a|s,\theta) q_\pi(s,a) \\ &= \mathbb{E}\big[\nabla_\theta \ln \pi(A|S,\theta) q_\pi(S,A)\big] \end{aligned}$$

Proof:

$$\begin{aligned} \nabla_\theta J &= \sum_s d(s) \sum_a \nabla_\theta \pi(a|s,\theta) q_\pi(s,a) \\ &= \sum_s d(s) \sum_a \pi(a|s,\theta) \nabla_\theta \ln \pi(a|s,\theta) q_\pi(s,a) \\ &= \mathbb{E}_{S \sim d} \left[\sum_a \pi(a|S,\theta) \nabla_\theta \ln \pi(a|S,\theta) q_\pi(S,a) \right] \\ &= \mathbb{E}_{S \sim d, A \sim \pi} \big[\nabla_\theta \ln \pi(A|S,\theta) q_\pi(S,A)\big] \\ &\doteq \mathbb{E}\big[\nabla_\theta \ln \pi(A|S,\theta) q_\pi(S,A)\big] \end{aligned}$$

Why is this expression useful?

- Because we can use samples to approximate the gradient!

$$\nabla_\theta J \approx \nabla_\theta \ln \pi(a|s,\theta) q_\pi(s,a) \qquad \text{(SGD)}$$

Some remarks: Because we need to calculate $\ln \pi(a|s,\theta)$, we must ensure that for all s, a, θ

$$\pi(a|s,\theta) > 0$$

- This can be archived by using softmax functions that can normalize the entries in a vector from $(-\infty, +\infty)$ to $(0, 1)$.

- For example, for any vector $x = [x_1, \ldots, x_n]^T$,

$$z_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

  where $z_i \in (0, 1)$ and $\sum_{i=1}^{n} z_i = 1$.
- Then, the policy function has the form of

$$\pi(a|s, \theta) = \frac{e^{h(s,a,\theta)}}{\sum_{a' \in \mathcal{A}} e^{h(s,a',\theta)}},$$

  where $h(s, a, \theta)$ is another function.
- Such a form based on the softmax function can be realized by a neural network whose input is s and parameter is θ. The network has $|\mathcal{A}|$ outputs, each of which corresponds to $\pi(a|s, \theta)$ for an action a. The activation function of the output layer should be softmax.
- Since $\pi(a|s, \theta) > 0$ for all a, the parameterized policy is stochastic and hence exploratory.
- There also exist deterministic policy gradient (DPG) methods.

4. Gradient-ascent algorithm (**REINFORCE**)
   - The gradient-ascent algorithm maximizing $J(\theta)$ is

$$\begin{aligned}\theta_{t+1} &= \theta_t + \alpha \nabla_\theta J(\theta) \\ &= \theta_t + \alpha \mathbb{E}\Big[\nabla_\theta \ln \pi(A|S, \theta_t) q_\pi(S, A)\Big]\end{aligned}$$

   - The true gradient can be replaced by a stochastic one:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta \ln \pi(a_t|s_t, \theta_t) q_\pi(s_t, a_t)$$

   - Furthermore, since q_π is unknown, it can be approximated:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta \ln \pi(a_t|s_t, \theta_t) q_t(s_t, a_t)$$

     There are different methods to approximate $q_\pi(s_t, a_t)$
     - In this lecture, Monte-Carlo based method, *REINFORCE*
     - In the next lecture, TD method and more

*Remarks:

Remark 1: How to do sampling?

$$\mathbb{E}_{S\sim d, A\sim\pi}\Big[\nabla_\theta \ln \pi(A|S,\theta_t) q_\pi(S,A)\Big] \longrightarrow \nabla_\theta \ln \pi(a|s,\theta_t) q_\pi(s,a)$$

- How to sample S?
  - $S \sim d$, where the distribution d is a long-run behavior under π.
- How to sample A?
  - $A \sim \pi(A|S,\theta)$. Hence, a_t should be sampled following $\pi(\theta_t)$ at s_t.
  - Therefore, the policy gradient method is on-policy.

Remark 2: How to interpret this algorithm?

Since

$$\nabla_\theta \ln \pi(a_t|s_t,\theta_t) = \frac{\nabla_\theta \pi(a_t|s_t,\theta_t)}{\pi(a_t|s_t,\theta_t)}$$

the algorithm can be rewritten as

$$\begin{aligned}\theta_{t+1} &= \theta_t + \alpha \nabla_\theta \ln \pi(a_t|s_t,\theta_t) q_t(s_t,a_t) \\ &= \theta_t + \alpha \underbrace{\left(\frac{q_t(s_t,a_t)}{\pi(a_t|s_t,\theta_t)}\right)}_{\beta_t} \nabla_\theta \pi(a_t|s_t,\theta_t).\end{aligned}$$

Therefore, we have the important expression of the algorithm:

$$\theta_{t+1} = \theta_t + \alpha\beta_t \nabla_\theta \pi(a_t|s_t,\theta_t)$$

Intuition: When $\alpha\beta_t$ is sufficiently small

- If $\beta_t > 0$, the probability of choosing (s_t, a_t) is enhanced:

  $$\pi(a_t|s_t,\theta_{t+1}) > \pi(a_t|s_t,\theta_t)$$

  The greater β_t is, the stronger the enhancement is.
- If $\beta_t < 0$, then $\pi(a_t|s_t,\theta_{t+1}) < \pi(a_t|s_t,\theta_t)$.

Math: When $\theta_{t+1} - \theta_t$ is sufficiently small, we have

$$\begin{aligned}\pi(a_t|s_t,\theta_{t+1}) &\approx \pi(a_t|s_t,\theta_t) + (\nabla_\theta \pi(a_t|s_t,\theta_t))^T (\theta_{t+1} - \theta_t) \\ &= \pi(a_t|s_t,\theta_t) + \alpha\beta_t (\nabla_\theta \pi(a_t|s_t,\theta_t))^T (\nabla_\theta \pi(a_t|s_t,\theta_t)) \\ &= \pi(a_t|s_t,\theta_t) + \alpha\beta_t \|\nabla_\theta \pi(a_t|s_t,\theta_t)\|^2\end{aligned}$$

Moreover:

The coefficient β_t can well balance exploration and exploitation.

- First, β_t is proportional to $q_t(s_t, a_t)$.
  - If $q_t(s_t, a_t)$ is great, then β_t is great.
  - Therefore, the algorithm intends to enhance actions with greater values.
- Second, β_t is inversely proportional to $\pi(a_t|s_t, \theta_t)$.
  - If $\pi(a_t|s_t, \theta_t)$ is small, then β_t is large.
  - Therefore, the algorithm intends to explore actions that have low probabilities.

*Pseudocode (Monte Carlo, offline):

Pseudocode: Policy Gradient by Monte Carlo (REINFORCE)

Initialization: A parameterized function $\pi(a|s, \theta)$, $\gamma \in (0, 1)$, and $\alpha > 0$.
Aim: Search for an optimal policy maximizing $J(\theta)$.

For the kth iteration, do

Select s_0 and generate an episode following $\pi(\theta_k)$. Suppose the episode is $\{s_0, a_0, r_1, \ldots, s_{T-1}, a_{T-1}, r_T\}$.

For $t = 0, 1, \ldots, T-1$, do

Value update: $q_t(s_t, a_t) = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$

Policy update: $\theta_{t+1} = \theta_t + \alpha \nabla_\theta \ln \pi(a_t|s_t, \theta_t) q_t(s_t, a_t)$

$\theta_k = \theta_T$

Chapter 10: Actor-Critic Methods

What are "actor" and "critic"?

- Here, "actor" refers to policy update. It is called *actor* is because the policies will be applied to take actions.
- Here, "critic" refers to policy evaluation or value estimation. It is called *critic* because it criticizes the policy by evaluating it.

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta \ln \pi(a_t|s_t, \theta_t) q_t(s_t, a_t)$$

We can see "actor" and "critic" from this algorithm:

- This algorithm corresponds to actor!
- The algorithm estimating $q_t(s, a)$ corresponds to critic!

How to get $q_t(s_t, a_t)$?

So far, we have studied two ways to estimate action values:

- **Monte Carlo learning:** If MC is used, the corresponding algorithm is called REINFORCE or Monte Carlo policy gradient.
  - We introduced in the last lecture.
- **Temporal-difference learning:** If TD is used, such kind of algorithms are usually called actor-critic.

1. The simplest actor-critic (QAC):

*Pseudocode:

The simplest actor-critic algorithm (QAC)

Aim: Search for an optimal policy by maximizing $J(\theta)$.

At time step t in each episode, do

Generate a_t following $\pi(a|s_t, \theta_t)$, observe r_{t+1}, s_{t+1}, and then generate a_{t+1} following $\pi(a|s_{t+1}, \theta_t)$.

Critic (value update):

$$w_{t+1} = w_t + \alpha_w \big[r_{t+1} + \gamma q(s_{t+1}, a_{t+1}, w_t) - q(s_t, a_t, w_t)\big] \nabla_w q(s_t, a_t, w_t)$$

Actor (policy update):

$$\theta_{t+1} = \theta_t + \alpha_\theta \nabla_\theta \ln \pi(a_t|s_t, \theta_t) q(s_t, a_t, w_{t+1})$$

*Remarks:

- The critic corresponds to "SARSA+value function approximation".
- The actor corresponds to the policy update algorithm.
- The algorithm is on-policy (why is PG on-policy?).
  - Since the policy is stochastic, no need to use techniques like ε-greedy.

2. Advantage actor-critic (A2C):
*The core idea is to **introduce a baseline to reduce variance**.
   a. Baseline invariance:

Property: the policy gradient is invariant to an additional baseline:

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{S\sim\eta, A\sim\pi}\Big[\nabla_\theta \ln \pi(A|S,\theta_t) q_\pi(S,A)\Big] \\ &= \mathbb{E}_{S\sim\eta, A\sim\pi}\Big[\nabla_\theta \ln \pi(A|S,\theta_t)(q_\pi(S,A) - b(S))\Big]\end{aligned}$$

Here, the additional baseline $b(S)$ is a scalar function of S.

First, why is it valid?

That is because

$$\mathbb{E}_{S\sim\eta, A\sim\pi}\Big[\nabla_\theta \ln \pi(A|S,\theta_t) b(S)\Big] = 0$$

The details:

$$\begin{aligned}\mathbb{E}_{S\sim\eta, A\sim\pi}\Big[\nabla_\theta \ln \pi(A|S,\theta_t) b(S)\Big] &= \sum_{s\in\mathcal{S}} \eta(s) \sum_{a\in\mathcal{A}} \pi(a|s,\theta_t) \nabla_\theta \ln \pi(a|s,\theta_t) b(s) \\ &= \sum_{s\in\mathcal{S}} \eta(s) \sum_{a\in\mathcal{A}} \nabla_\theta \pi(a|s,\theta_t) b(s) \\ &= \sum_{s\in\mathcal{S}} \eta(s) b(s) \sum_{a\in\mathcal{A}} \nabla_\theta \pi(a|s,\theta_t) \\ &= \sum_{s\in\mathcal{S}} \eta(s) b(s) \nabla_\theta \sum_{a\in\mathcal{A}} \pi(a|s,\theta_t) \\ &= \sum_{s\in\mathcal{S}} \eta(s) b(s) \nabla_\theta 1 = 0\end{aligned}$$

Second, why is the baseline useful?

The gradient is $\nabla_\theta J(\theta) = \mathbb{E}[X]$ where

$$X(S,A) \doteq \nabla_\theta \ln \pi(A|S,\theta_t)[q(S,A) - b(S)]$$

We have

- $\mathbb{E}[X]$ is invariant to $b(S)$.
- $\mathrm{var}(X)$ is NOT invariant to $b(S)$.

Our goal: Select an optimal baseline b to minimize $\mathrm{var}(X)$

- Benefit: when we use a random sample to approximate $\mathbb{E}[X]$, the estimation variance would also be small.

- The optimal baseline that can minimize $\text{var}(X)$ is, for any $s \in \mathcal{S}$,

$$b^*(s) = \frac{\mathbb{E}_{A\sim\pi}[\|\nabla_\theta \ln \pi(A|s,\theta_t)\|^2 q(s,A)]}{\mathbb{E}_{A\sim\pi}[\|\nabla_\theta \ln \pi(A|s,\theta_t)\|^2]}.$$

- We can remove the weight $\|\nabla_\theta \ln \pi(A|s,\theta_t)\|^2$ and select the suboptimal baseline:

$$b(s) = \mathbb{E}_{A\sim\pi}[q(s,A)] = v_\pi(s)$$

b. Algorithm:

$$\begin{aligned}\theta_{t+1} &= \theta_t + \alpha\mathbb{E}\Big[\nabla_\theta \ln \pi(A|S,\theta_t)[q_\pi(S,A) - v_\pi(S)]\Big] \\ &\doteq \theta_t + \alpha\mathbb{E}\Big[\nabla_\theta \ln \pi(A|S,\theta_t)\delta_\pi(S,A)\Big]\end{aligned}$$

where

$$\delta_\pi(S,A) \doteq q_\pi(S,A) - v_\pi(S)$$

is called the advantage function (why called advantage?).

- the stochastic version of this algorithm is

$$\begin{aligned}\theta_{t+1} &= \theta_t + \alpha\nabla_\theta \ln \pi(a_t|s_t,\theta_t)[q_t(s_t,a_t) - v_t(s_t)] \\ &= \theta_t + \alpha\nabla_\theta \ln \pi(a_t|s_t,\theta_t)\delta_t(s_t,a_t)\end{aligned}$$

Moreover, the algorithm can be reexpressed as

$$\begin{aligned}\theta_{t+1} &= \theta_t + \alpha\nabla_\theta \ln \pi(a_t|s_t,\theta_t)\delta_t(s_t,a_t) \\ &= \theta_t + \alpha\frac{\nabla_\theta \pi(a_t|s_t,\theta_t)}{\pi(a_t|s_t,\theta_t)}\delta_t(s_t,a_t) \\ &= \theta_t + \alpha\underbrace{\left(\frac{\delta_t(s_t,a_t)}{\pi(a_t|s_t,\theta_t)}\right)}_{\text{step size}}\nabla_\theta \pi(a_t|s_t,\theta_t)\end{aligned}$$

- The step size is proportional to the relative value δ_t rather than the absolute value q_t, which is more reasonable.
- It can still well balance exploration and exploitation.

Furthermore, the advantage function is approximated by the TD error:

$$\delta_t = q_t(s_t, a_t) - v_t(s_t) \to r_{t+1} + \gamma v_t(s_{t+1}) - v_t(s_t)$$

- This approximation is reasonable because

$$\mathbb{E}[q_\pi(S, A) - v_\pi(S)|S = s_t, A = a_t] = \mathbb{E}\Big[R + \gamma v_\pi(S') - v_\pi(S)|S = s_t, A = a_t\Big]$$

- Benefit: only need one network to approximate $v_\pi(s)$ rather than two networks for $q_\pi(s, a)$ and $v_\pi(s)$.

c. Pseudocode:

Advantage actor-critic (A2C) or TD actor-critic

Aim: Search for an optimal policy by maximizing $J(\theta)$.

At time step t in each episode, do

Generate a_t following $\pi(a|s_t, \theta_t)$ and then observe r_{t+1}, s_{t+1}.

TD error (advantage function):

$$\delta_t = r_{t+1} + \gamma v(s_{t+1}, w_t) - v(s_t, w_t)$$

Critic (value update):

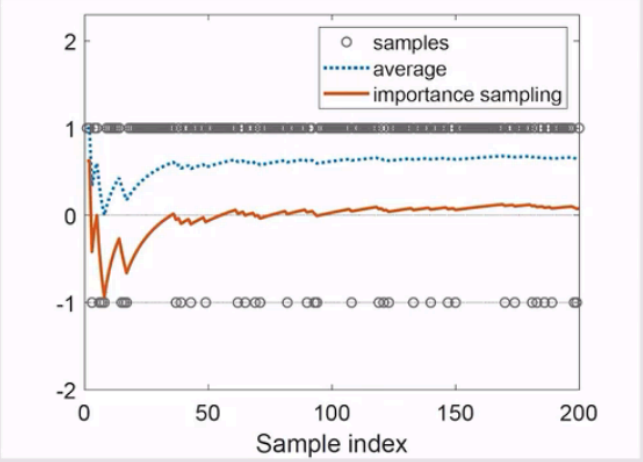
$$w_{t+1} = w_t + \alpha_w \delta_t \nabla_w v(s_t, w_t)$$

Actor (policy update):

$$\theta_{t+1} = \theta_t + \alpha_\theta \delta_t \nabla_\theta \ln \pi(a_t|s_t, \theta_t)$$

3. Off-policy actor-critic:
   - Policy gradient is on-policy.
     - Why? because the gradient is $\nabla_\theta J(\theta) = \mathbb{E}_{S\sim\eta, A\sim\pi}[*]$
   - Can we convert it to off-policy?
     - Yes, by importance sampling
     - The importance sampling technique is not limited to AC, but also to any algorithm that aims to estimate an expectation.

   a. Importance sampling:
   (Estimate p_0 using data sampled from distribution p_1)



Note that

$$\mathbb{E}_{X\sim p_0}[X] = \sum_x p_0(x)x = \sum_x p_1(x) \underbrace{\frac{p_0(x)}{p_1(x)}x}_{f(x)} = \mathbb{E}_{X\sim p_1}[f(X)]$$

- Thus, we can estimate $\mathbb{E}_{X\sim p_1}[f(X)]$ in order to estimate $\mathbb{E}_{X\sim p_0}[X]$.
- How to estimate $\mathbb{E}_{X\sim p_1}[f(X)]$? Easy. Let

$$\bar{f} \doteq \frac{1}{n}\sum_{i=1}^{n} f(x_i), \qquad \text{where } x_i \sim p_1$$

Therefore, $\bar{f}$ is a good approximation for $\mathbb{E}_{X\sim p_1}[f(X)] = \mathbb{E}_{X\sim p_0}[X]$

$$\mathbb{E}_{X\sim p_0}[X] \approx \bar{f} = \frac{1}{n}\sum_{i=1}^{n} f(x_i) = \frac{1}{n}\sum_{i=1}^{n} \frac{p_0(x_i)}{p_1(x_i)}x_i$$

- $\frac{p_0(x_i)}{p_1(x_i)}$ is called the *importance weight*.
  - If $p_1(x_i) = p_0(x_i)$, the importance weight is one and $\bar{f}$ becomes $\bar{x}$.
  - If $p_0(x_i) \geq p_1(x_i)$, x_i can be more often sampled by p_0 than p_1. The importance weight (> 1) can emphasize the importance of this sample.

You may ask: While $\bar{f} = \frac{1}{n}\sum_{i=1}^{n} \frac{p_0(x_i)}{p_1(x_i)}x_i$ requires $p_0(x)$, if I know $p_0(x)$, why not directly calculate the expectation?

Answer: It is applicable to the case where it is easy to calculate $p_0(x)$ given an x, but difficult to calculate the expectation.

- For example, continuous case, complex expression of p_0, or no expression of p_0 (e.g., p_0 represented by a neural network).

b. Off-policy policy gradient:

Theorem (Off-policy policy gradient theorem)

In the discounted case where $\gamma \in (0,1)$, the gradient of $J(\theta)$ is

$$\nabla_\theta J(\theta) = \mathbb{E}_{S\sim\rho, A\sim\beta}\left[\frac{\pi(A|S,\theta)}{\beta(A|S)}\nabla_\theta \ln \pi(A|S,\theta) q_\pi(S,A)\right]$$

where β is the behavior policy and ρ is a state distribution.
See the details and the proof in my book.

Two differences from on-policy:
*$A\sim\beta$ (behaviour policy) rather than $A\sim\pi$.
*$\frac{\pi(A|S,\theta)}{\beta(A|S)}$ is the importance weight.

c. Algorithm:

The corresponding stochastic gradient-ascent algorithm is

$$\theta_{t+1} = \theta_t + \alpha_\theta \frac{\pi(a_t|s_t,\theta_t)}{\beta(a_t|s_t)}\nabla_\theta \ln \pi(a_t|s_t,\theta_t)\big(q_t(s_t,a_t) - v_t(s_t)\big)$$

Similar to the on-policy case,

$$q_t(s_t,a_t) - v_t(s_t) \approx r_{t+1} + \gamma v_t(s_{t+1}) - v_t(s_t) \doteq \delta_t(s_t,a_t)$$

Then, the algorithm becomes

$$\theta_{t+1} = \theta_t + \alpha_\theta \frac{\pi(a_t|s_t,\theta_t)}{\beta(a_t|s_t)}\nabla_\theta \ln \pi(a_t|s_t,\theta_t)\delta_t(s_t,a_t)$$

and hence

$$\theta_{t+1} = \theta_t + \alpha_\theta \left(\frac{\delta_t(s_t,a_t)}{\beta(a_t|s_t)}\right)\nabla_\theta \pi(a_t|s_t,\theta_t)$$

Note that here the step size $\frac{\delta_t(s_t,a_t)}{\beta(a_t|s_t)}$ can only guarantee exploitation, because the denominator is fixed.
*Pseudocode:

Off-policy actor-critic based on importance sampling

Initialization: A given behavior policy $\beta(a|s)$. A target policy $\pi(a|s,\theta_0)$ where θ_0 is the initial parameter vector. A value function $v(s,w_0)$ where w_0 is the initial parameter vector.

Aim: Search for an optimal policy by maximizing $J(\theta)$.

At time step t in each episode, do

Generate a_t following $\beta(s_t)$ and then observe r_{t+1}, s_{t+1}.

TD error (advantage function):

$$\delta_t = r_{t+1} + \gamma v(s_{t+1}, w_t) - v(s_t, w_t)$$

Critic (value update):

$$w_{t+1} = w_t + \alpha_w \frac{\pi(a_t|s_t,\theta_t)}{\beta(a_t|s_t)} \delta_t \nabla_w v(s_t, w_t)$$

Actor (policy update):

$$\theta_{t+1} = \theta_t + \alpha_\theta \frac{\pi(a_t|s_t,\theta_t)}{\beta(a_t|s_t)} \delta_t \nabla_\theta \ln \pi(a_t|s_t,\theta_t)$$

4. Deterministic actor-critic (DPG):
(Why we need to consider deterministic policies? To handle continuous action.)

- Up to now, a general policy is denoted as $\pi(a|s,\theta) \in [0,1]$, which can be either stochastic or deterministic.
- Now, the deterministic policy is specifically denoted as

$$a = \mu(s,\theta) \doteq \mu(s)$$

  - μ is a mapping from $\mathcal{S}$ to $\mathcal{A}$.
  - μ can be represented by, for example, a neural network with the input as s, the output as a, and the parameter as θ.
  - We may write $\mu(s,\theta)$ in short as $\mu(s)$.

a. Deterministic policy gradient:

Theorem (Deterministic policy gradient theorem in the discounted case)

In the discounted case where $\gamma \in (0,1)$, the gradient of $J(\theta)$ is

$$\begin{aligned}\nabla_\theta J(\theta) &= \sum_{s\in\mathcal{S}} \rho_\mu(s) \nabla_\theta \mu(s) \big(\nabla_a q_\mu(s,a)\big)|_{a=\mu(s)} \\ &= \mathbb{E}_{S\sim\rho_\mu}\left[\nabla_\theta \mu(S) \big(\nabla_a q_\mu(S,a)\big)|_{a=\mu(S)}\right]\end{aligned}$$

Here, ρ_μ is a state distribution.

One important difference from the stochastic case:

- The gradient does not involve the distribution of the action A (why?).
- As a result, the deterministic policy gradient method is off-policy.

b. Algorithm:

Based on the policy gradient, the gradient-ascent algorithm for maximizing $J(\theta)$ is:

$$\theta_{t+1} = \theta_t + \alpha_\theta \mathbb{E}_{S\sim\rho_\mu}\left[\nabla_\theta \mu(S)\big(\nabla_a q_\mu(S,a)\big)|_{a=\mu(S)}\right]$$

The corresponding stochastic gradient-ascent algorithm is

$$\theta_{t+1} = \theta_t + \alpha_\theta \nabla_\theta \mu(s_t)\big(\nabla_a q_\mu(s_t,a)\big)|_{a=\mu(s_t)}$$

*Pseudocode:

Deterministic actor-critic algorithm

Initialization: A given behavior policy $\beta(a|s)$. A deterministic target policy $\mu(s,\theta_0)$ where θ_0 is the initial parameter vector. A value function $v(s,w_0)$ where w_0 is the initial parameter vector.

Aim: Search for an optimal policy by maximizing $J(\theta)$.

At time step t in each episode, do

Generate a_t following β and then observe r_{t+1}, s_{t+1}.

TD error:

$$\delta_t = r_{t+1} + \gamma q(s_{t+1}, \mu(s_{t+1},\theta_t), w_t) - q(s_t,a_t,w_t)$$

Critic (value update):

$$w_{t+1} = w_t + \alpha_w \delta_t \nabla_w q(s_t,a_t,w_t)$$

Actor (policy update):

$$\theta_{t+1} = \theta_t + \alpha_\theta \nabla_\theta \mu(s_t,\theta_t)\big(\nabla_a q(s_t,a,w_{t+1})\big)|_{a=\mu(s_t)}$$

*Remarks:

- This is an off-policy implementation where the behavior policy β may be different from μ.
- β can also be replaced by μ+noise.
- How to select the function to represent $q(s,a,w)$?
  - Linear function: $q(s,a,w) = \phi^T(s,a)w$ where $\phi(s,a)$ is the feature vector. Details can be found in the DPG paper.
  - Neural networks: deep deterministic policy gradient (DDPG) method.

Replacing β by μ + *noise* makes the algorithm on-policy and exploratory.