

# Lecture 1: Basic Concepts in Reinforcement Learning

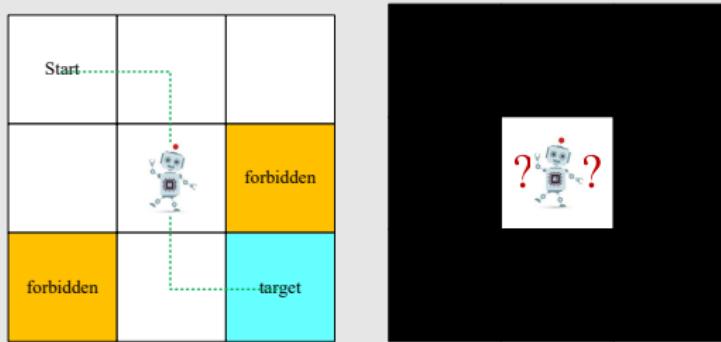
Shiyu Zhao

School of Engineering, Westlake University

# Contents

- First, introduce fundamental concepts in reinforcement learning (RL) by examples.
- Second, formalize the concepts in the context of Markov decision processes.

# A grid-world example



An illustrative example used throughout this course:

- Grid of cells: Accessible/forbidden/target cells, boundary.
- Very easy to understand and useful for illustration

Task:

- Given any starting area, find a “good” way to the target.
- How to define “good”? Avoid forbidden cells, detours, or boundary.

# State

*State*: The status of the agent with respect to the environment.

- For the grid-world example, the location of the agent is the state. There are nine possible locations and hence nine states:  $s_1, s_2, \dots, s_9$ .

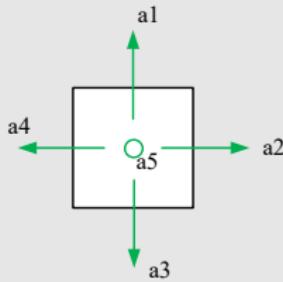
s1	s2	s3
s4	s5	s6
s7	s8	s9

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

# Action

**Action:** For each state, there are five possible actions:  $a_1, \dots, a_5$

- $a_1$ : move upwards;
- $a_2$ : move rightwards;
- $a_3$ : move downwards;
- $a_4$ : move leftwards;
- $a_5$ : stay unchanged;



s1	s2	s3
s4	s5	s6
s7	s8	s9

**Action space of a state:** the set of all possible actions of a state.

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

**Question:** can different states have different sets of actions?

## State transition

s1	s2	s3
s4	s5	s6
s7	s8	s9

When taking an action, the agent may move from one state to another. Such a process is called *state transition*.

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

$$s_1 \xrightarrow{a_2} s_2$$

- At state  $s_1$ , if we choose action  $a_1$ , then what is the next state?

$$s_1 \xrightarrow{a_1} s_1$$

- State transition defines the interaction with the environment.
- **Question:** Can we define the state transition in other ways? Yes.

## State transition



*Forbidden area:* At state  $s_5$ , if we choose action  $a_2$ , then what is the next state?

- Case 1: the forbidden area is accessible but with penalty. Then,

$$s_5 \xrightarrow{a_2} s_6$$

- Case 2: the forbidden area is inaccessible (e.g., surrounded by a wall)

$$s_5 \xrightarrow{a_2} s_5$$

We consider the first case, which is more general and challenging.

## State transition

s1	s2	s3
s4	s5	s6
s7	s8	s9

Tabular representation: We can use a table to describe the state transition:

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

Can only represent *deterministic cases*.

## State transition

s1	s2	s3
s4	s5	s6
s7	s8	s9

*State transition probability:* use probability to describe state transition!

- Intuition: At state  $s_1$ , if we choose action  $a_2$ , the next state is  $s_2$ .
- Math:

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

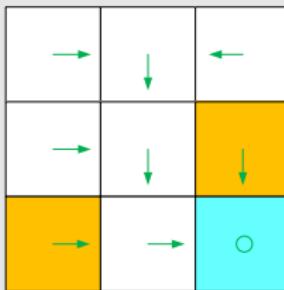
$$p(s_i|s_1, a_2) = 0 \quad \forall i \neq 2$$

Here it is a **deterministic** case. The state transition could be **stochastic** (for example, wind gust).

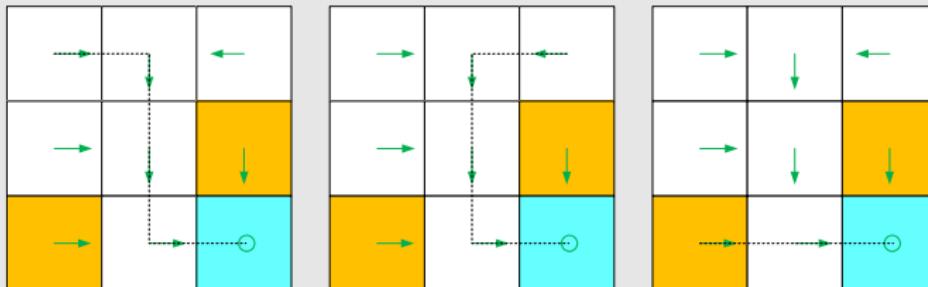
# Policy

*Policy* tells the agent what actions to take at a state.

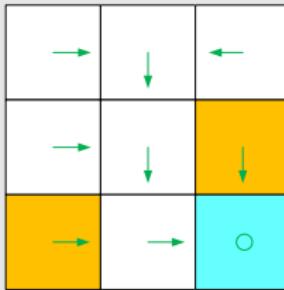
**Intuitive representation:** The arrows demonstrate a policy.



Based on this policy, we get the following paths with different starting points.



# Policy



**Mathematical representation:** using conditional probability

For example, for state  $s_1$ :

$$\pi(a_1|s_1) = 0$$

$$\pi(a_2|s_1) = 1$$

$$\pi(a_3|s_1) = 0$$

$$\pi(a_4|s_1) = 0$$

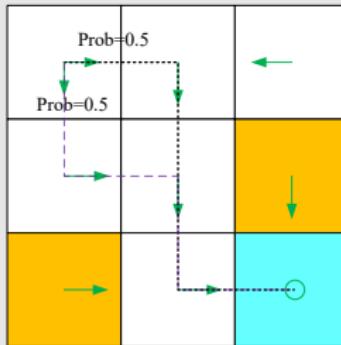
$$\pi(a_5|s_1) = 0$$

It is a **deterministic** policy.

# Policy

There are **stochastic** policies.

For example:



In this policy, for  $s_1$ :

$$\pi(a_1|s_1) = 0$$

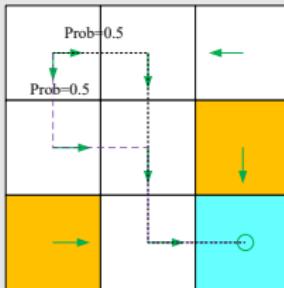
$$\pi(a_2|s_1) = 0.5$$

$$\pi(a_3|s_1) = 0.5$$

$$\pi(a_4|s_1) = 0$$

$$\pi(a_5|s_1) = 0$$

# Policy



**Tabular representation of a policy:** how to use this table.

	$a_1$ (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

Can represent either *deterministic* or *stochastic* cases.

# Reward

**Reward is one of the most unique concepts of RL.**

*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.

Questions:

- What about a zero reward? No punishment.
- Can positive mean punishment? Yes.

## Reward

s1	s2	s3
s4	s5	s6
s7	s8	s9

In the grid-world example, the rewards are designed as follows:

- If the agent attempts to get out of the boundary, let  $r_{\text{bound}} = -1$
- If the agent attempts to enter a forbidden cell, let  $r_{\text{forbid}} = -1$
- If the agent reaches the target cell, let  $r_{\text{target}} = +1$
- Otherwise, the agent gets a reward of  $r = 0$ .

Reward can be interpreted as a **human-machine interface**, with which we can guide the agent to behave as what we expect.

For example, with the above designed rewards, the agent will try to avoid getting out of the boundary or stepping into the forbidden cells.

# Reward

s1	s2	s3
s4	s5	s6
s7	s8	s9

**Tabular representation of reward transition:** how to use the table?

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

Can only represent *deterministic* cases.

# Reward

s1	s2	s3
s4	s5	s6
s7	s8	s9

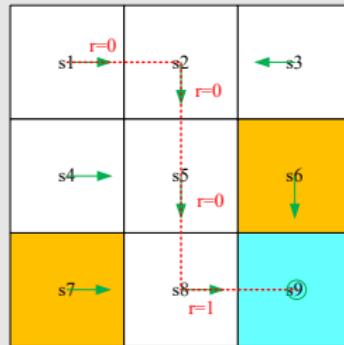
**Mathematical description:** conditional probability

- 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  $p(r \neq -1|s_1, a_1) = 0$

Remarks:

- Here it is a deterministic case. The reward transition could be stochastic.
- For example, if you study hard, you will get rewards. But how much is uncertain.
- The reward depends on the state and action, but not the next state (for example, consider  $s_1, a_1$  and  $s_1, a_5$ ).

## Trajectory and return



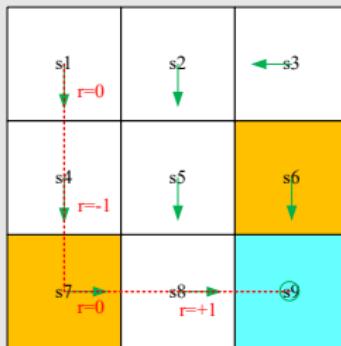
A *trajectory* is a state-action-reward chain:

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

The *return* of this trajectory is the sum of all the rewards collected along the trajectory:

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

## Trajectory and return



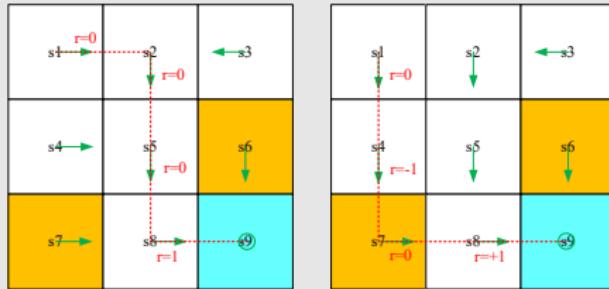
A different policy gives a different trajectory:

$$s_1 \xrightarrow[a_3]{r=0} s_4 \xrightarrow[a_3]{r=-1} s_7 \xrightarrow[a_2]{r=0} s_8 \xrightarrow[a_2]{r=+1} s_9$$

The return of this path is:

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

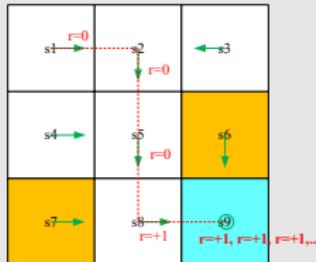
## Trajectory and return



Which policy is better?

- **Intuition:** the first is better, because it avoids the forbidden areas.
- **Mathematics:** the first one is better, since it has a greater return!
- Return could be used to evaluate whether a policy is good or not (see details in the next lecture)!

## Discounted return



A trajectory may be infinite:

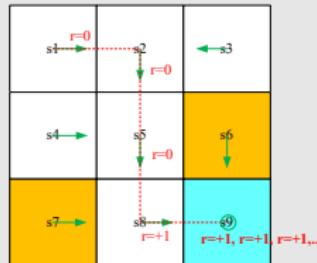
$$s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_3} s_8 \xrightarrow{a_2} s_9 \xrightarrow{a_5} s_9 \xrightarrow{a_5} s_9 \dots$$

The return is

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

The definition is invalid since the return diverges!

## Discounted return



Need to introduce a *discount rate*  $\gamma \in [0, 1)$

*Discounted return:*

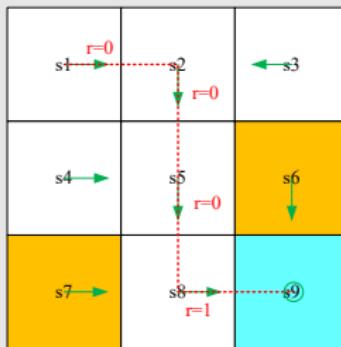
$$\begin{aligned}\text{discounted return} &= 0 + \gamma^0 + \gamma^2 0 + \gamma^3 1 + \gamma^4 1 + \gamma^5 1 + \dots \\ &= \gamma^3 (1 + \gamma + \gamma^2 + \dots) = \gamma^3 \frac{1}{1 - \gamma}.\end{aligned}$$

Roles: 1) the sum becomes finite; 2) balance the far and near future rewards:

- If  $\gamma$  is close to 0, the value of the discounted return is dominated by the rewards obtained in the near future.
- If  $\gamma$  is close to 1, the value of the discounted return is dominated by the rewards obtained in the far future.

## Episode

When interacting with the environment following a policy, the agent may stop at some *terminal states*. The resulting trajectory is called an *episode* (or a trial).



Example: episode

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

An episode is usually assumed to be a finite trajectory. Tasks with episodes are called *episodic tasks*.

## Episode

Some tasks may have no terminal states, meaning the interaction with the environment will never end. Such tasks are called *continuing tasks*.

In the grid-world example, should we stop after arriving the target?

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

# Markov decision process (MDP)

Key elements of MDP:

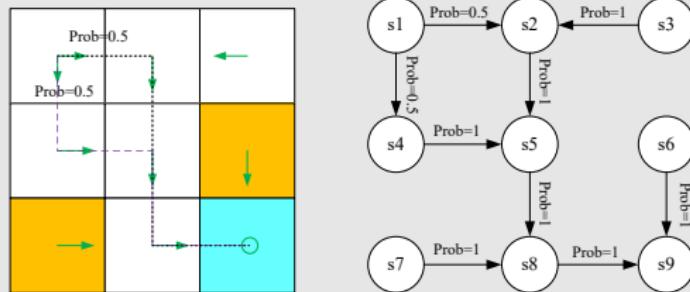
- Sets:
  - State: the set of states  $\mathcal{S}$
  - Action: the set of actions  $\mathcal{A}(s)$  is associated for state  $s \in \mathcal{S}$ .
  - Reward: the set of rewards  $\mathcal{R}(s, a)$ .
- Probability distribution:
  - State transition probability: at state  $s$ , taking action  $a$ , 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

$$p(s_{t+1}|a_t, s_t, \dots, a_0, s_0) = p(s_{t+1}|a_t, s_t),$$
$$p(r_{t+1}|a_t, s_t, \dots, a_0, s_0) = p(r_{t+1}|a_t, s_t).$$

All the concepts introduced in this lecture can be put in the framework in MDP.

# Markov decision process (MDP)

The grid world could be abstracted as a more general model, *Markov process*.



The circles represent states and the links with arrows represent the state transition.

Markov decision process becomes Markov process once the policy is given!

# Summary

By using grid-world examples, we demonstrated the following key concepts:

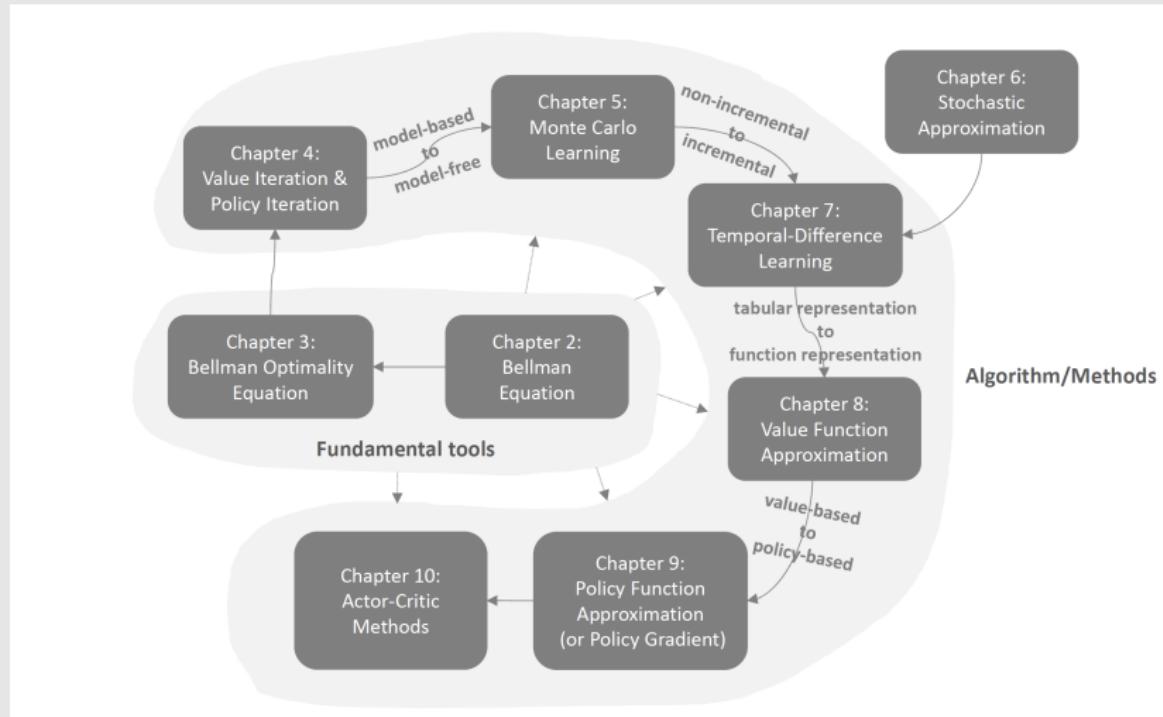
- State
- Action
- State transition, state transition probability  $p(s'|s, a)$
- Reward, reward probability  $p(r|s, a)$
- Trajectory, episode, return, discounted return
- Markov decision process

# Lecture 2: Bellman Equation

Shiyu Zhao

School of Engineering, Westlake University

# Outline



# Outline

In this lecture:

- A core concept: state value
- A fundamental tool: the Bellman equation

# Outline

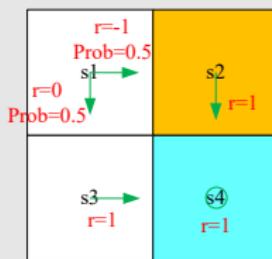
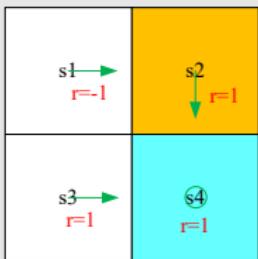
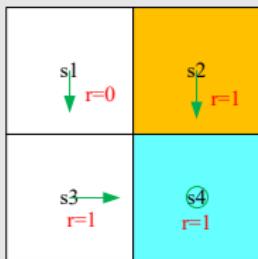
- 1 Motivating examples**
- 2 State value**
- 3 Bellman equation: Derivation**
- 4 Bellman equation: Matrix-vector form**
- 5 Bellman equation: Solve the state values**
- 6 Action value**
- 7 Summary**

# Outline

- 1 Motivating examples
- 2 State value
- 3 Bellman equation: Derivation
- 4 Bellman equation: Matrix-vector form
- 5 Bellman equation: Solve the state values
- 6 Action value
- 7 Summary

# Motivating example 1: Why return is important?

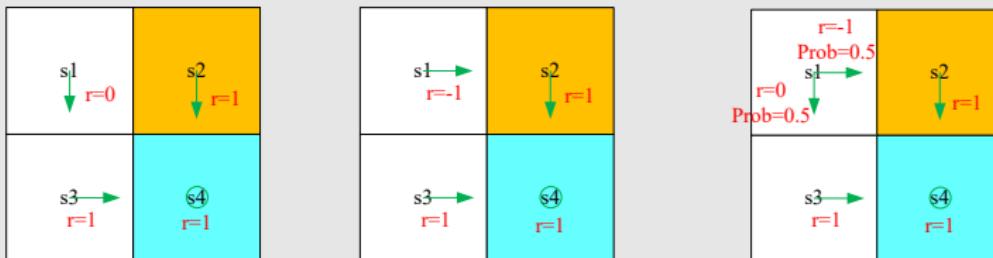
- What is return? The (discounted) sum of the rewards obtained along a trajectory.
- Why return is important? See the following examples.



- Question: From the starting point  $s_1$ , which policy is the “best”? Which is the “worst”?  
Intuition: the first is the best and the second is the worst, because of the forbidden area.
- Question: can we use mathematics to describe such an intuition?  
Answer: Return could be used to evaluate policies. See the following.

# Motivating example 1: Why return is important?

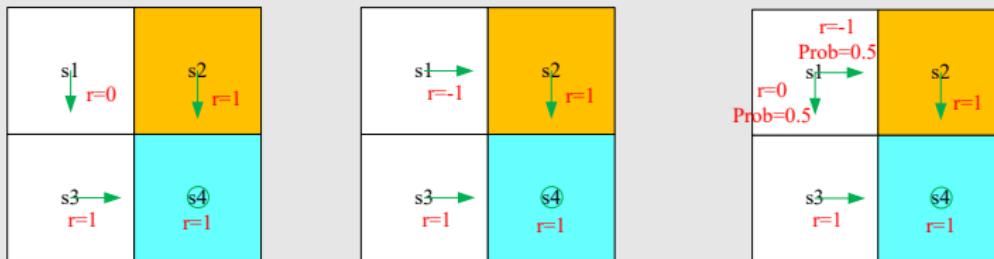
- What is return? The (discounted) sum of the rewards obtained along a trajectory.
- Why return is important? See the following examples.



- Question: From the starting point  $s_1$ , which policy is the “best”? Which is the “worst”?  
Intuition: the first is the best and the second is the worst, because of the forbidden area.
- Question: can we use mathematics to describe such an intuition?  
Answer: Return could be used to evaluate policies. See the following.

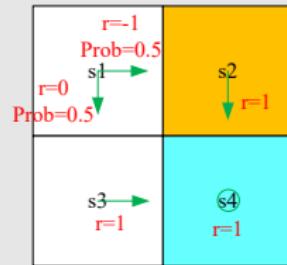
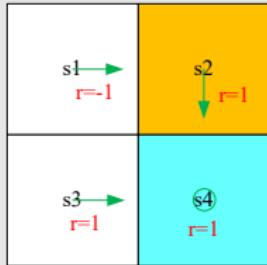
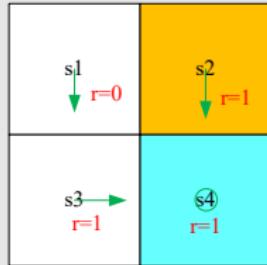
# Motivating example 1: Why return is important?

- What is return? The (discounted) sum of the rewards obtained along a trajectory.
- Why return is important? See the following examples.



- Question: From the starting point  $s_1$ , which policy is the “best”? Which is the “worst”?  
Intuition: the first is the best and the second is the worst, because of the forbidden area.
- Question: can we use mathematics to describe such an intuition?  
Answer: Return could be used to evaluate policies. See the following.

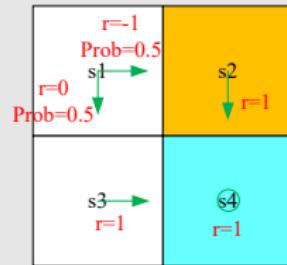
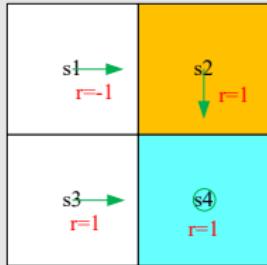
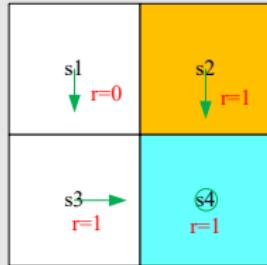
# Motivating example 1: Why return is important?



Based on policy 1 (left figure), starting from  $s_1$ , the discounted return is

$$\begin{aligned}\text{return}_1 &= 0 + \gamma 1 + \gamma^2 1 + \dots, \\ &= \gamma(1 + \gamma + \gamma^2 + \dots), \\ &= \frac{\gamma}{1 - \gamma}.\end{aligned}$$

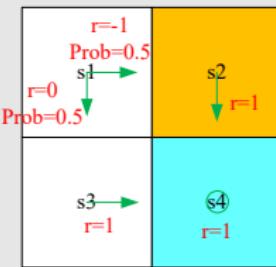
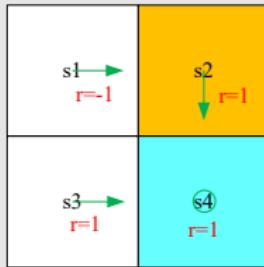
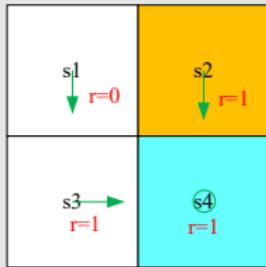
# Motivating example 1: Why return is important?



Based on policy 1 (left figure), starting from  $s_1$ , the discounted return is

$$\begin{aligned}\text{return}_1 &= 0 + \gamma 1 + \gamma^2 1 + \dots, \\ &= \gamma(1 + \gamma + \gamma^2 + \dots), \\ &= \frac{\gamma}{1 - \gamma}.\end{aligned}$$

# Motivating example 1: Why return is important?

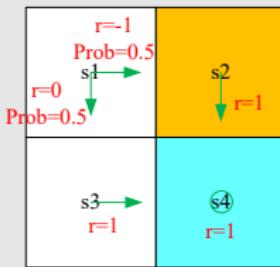
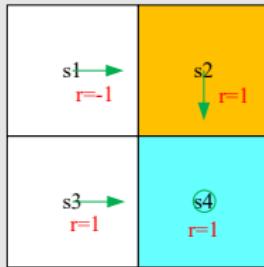
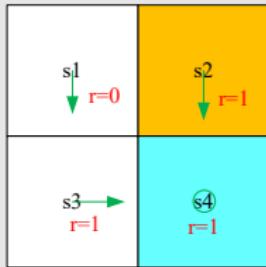


**Exercise:** Based on policy 2 (middle figure), starting from  $s_1$ , what is the discounted return?

Answer:

$$\begin{aligned}\text{return}_2 &= -1 + \gamma(1 + \gamma^2 + \dots), \\ &= -1 + \gamma(1 + \gamma + \gamma^2 + \dots), \\ &= -1 + \frac{\gamma}{1 - \gamma}.\end{aligned}$$

# Motivating example 1: Why return is important?

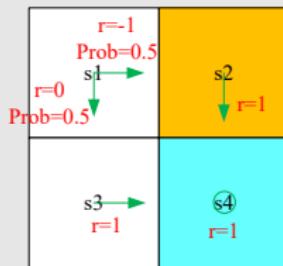
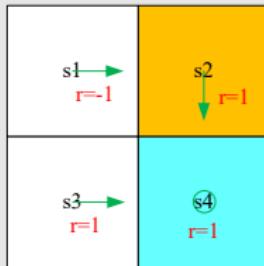
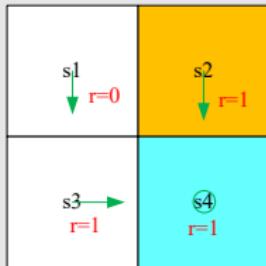


**Exercise:** Based on policy 2 (middle figure), starting from  $s_1$ , what is the discounted return?

Answer:

$$\begin{aligned}\text{return}_2 &= -1 + \gamma(1 + \gamma^2 + \dots), \\ &= -1 + \gamma(1 + \gamma + \gamma^2 + \dots), \\ &= -1 + \frac{\gamma}{1 - \gamma}.\end{aligned}$$

# Motivating example 1: Why return is important?



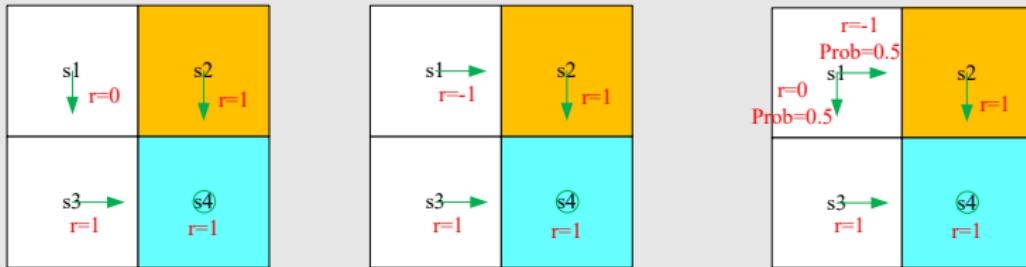
Policy 3 is stochastic!

**Exercise:** Based on policy 3 (right figure), starting from  $s_1$ , the discounted return is

Answer:

$$\begin{aligned}\text{return}_3 &= 0.5 \left( -1 + \frac{\gamma}{1-\gamma} \right) + 0.5 \left( \frac{\gamma}{1-\gamma} \right), \\ &= -0.5 + \frac{\gamma}{1-\gamma}.\end{aligned}$$

# Motivating example 1: Why return is important?



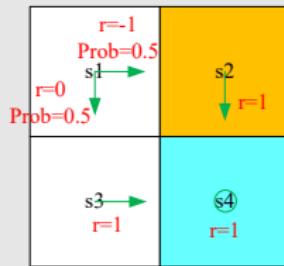
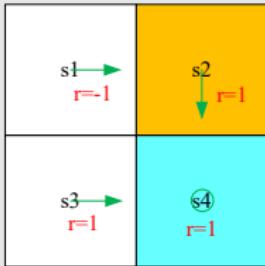
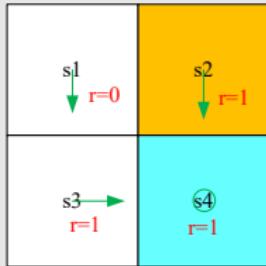
Policy 3 is stochastic!

**Exercise:** Based on policy 3 (right figure), starting from  $s_1$ , the discounted return is

Answer:

$$\begin{aligned}\text{return}_3 &= 0.5 \left( -1 + \frac{\gamma}{1-\gamma} \right) + 0.5 \left( \frac{\gamma}{1-\gamma} \right), \\ &= -0.5 + \frac{\gamma}{1-\gamma}.\end{aligned}$$

# Motivating example 1: Why return is important?



In summary, starting from  $s_1$ ,

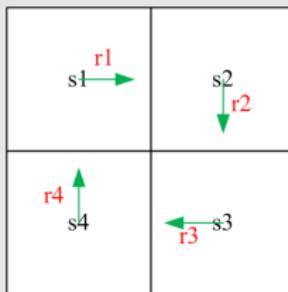
$$\text{return}_1 > \text{return}_3 > \text{return}_2$$

The above inequality suggests that the first policy is the best and the second policy is the worst, which is exactly the same as our intuition.

**Calculating return is important to evaluate a policy.**

## Motivating example 2: How to calculate return?

While return is important, how to calculate it?



Method 1: by definition

Let  $v_i$  denote the return obtained starting from  $s_i$  ( $i = 1, 2, 3, 4$ )

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

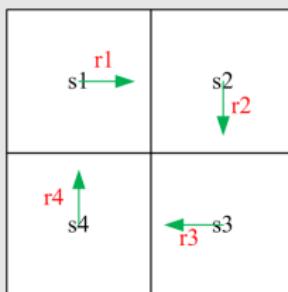
$$v_2 = r_2 + \gamma r_3 + \gamma^2 r_4 + \dots$$

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

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

## Motivating example 2: How to calculate return?

While return is important, how to calculate it?



Method 1: by definition

Let  $v_i$  denote the return obtained starting from  $s_i$  ( $i = 1, 2, 3, 4$ )

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

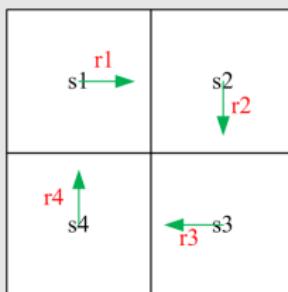
$$v_2 = r_2 + \gamma r_3 + \gamma^2 r_4 + \dots$$

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

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

## Motivating example 2: How to calculate return?

While return is important, how to calculate it?



Method 1: by definition

Let  $v_i$  denote the return obtained starting from  $s_i$  ( $i = 1, 2, 3, 4$ )

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

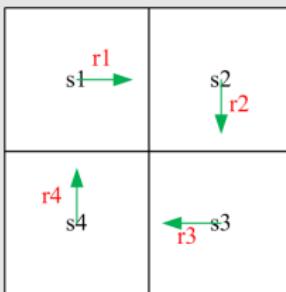
$$v_2 = r_2 + \gamma r_3 + \gamma^2 r_4 + \dots$$

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

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

# Motivating example 2: How to calculate return?

While return is important, how to calculate it?



Method 2:

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

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

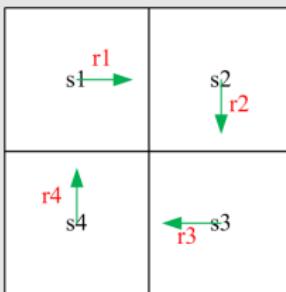
$$v_3 = r_3 + \gamma(r_4 + \gamma r_1 + \dots) = r_3 + \gamma v_4$$

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

- The returns rely on each other. *Bootstrapping!*

# Motivating example 2: How to calculate return?

While return is important, how to calculate it?



Method 2:

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

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

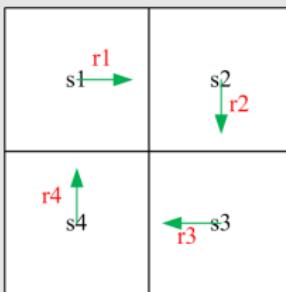
$$v_3 = r_3 + \gamma(r_4 + \gamma r_1 + \dots) = r_3 + \gamma v_4$$

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

- The returns rely on each other. *Bootstrapping!*

# Motivating example 2: How to calculate return?

While return is important, how to calculate it?



Method 2:

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

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

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

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

- The returns rely on each other. *Bootstrapping!*

## Motivating example 2: How to calculate return?

How to solve these equations? Write in the following matrix-vector form:

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

which can be rewritten as

$$\mathbf{v} = \mathbf{r} + \gamma \mathbf{Pv}$$

This is the Bellman equation (for this specific deterministic problem)!!

- Though simple, it demonstrates the core idea: the value of one state relies on the values of other states.
- A matrix-vector form is more clear to see how to solve the state values.

## Motivating example 2: How to calculate return?

How to solve these equations? Write in the following matrix-vector form:

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

which can be rewritten as

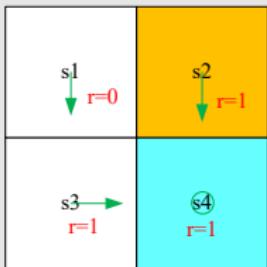
$$\mathbf{v} = \mathbf{r} + \gamma \mathbf{Pv}$$

This is the Bellman equation (for this specific deterministic problem)!!

- Though simple, it demonstrates the core idea: the value of one state relies on the values of other states.
- A matrix-vector form is more clear to see how to solve the state values.

## Motivating example 2: How to calculate return?

**Exercise:** Consider the policy shown in the figure. Please write out the relation among the returns (that is to write out the Bellman equation)



Answer:

$$v_1 = 0 + \gamma v_3$$

$$v_2 = 1 + \gamma v_4$$

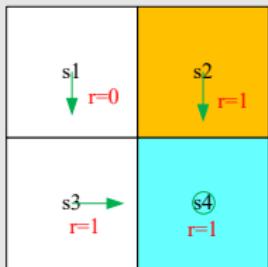
$$v_3 = 1 + \gamma v_4$$

$$v_4 = 1 + \gamma v_4$$

**Exercise:** How to solve them? We can first calculate  $v_4$ , and then

## Motivating example 2: How to calculate return?

**Exercise:** Consider the policy shown in the figure. Please write out the relation among the returns (that is to write out the Bellman equation)



Answer:

$$v_1 = 0 + \gamma v_3$$

$$v_2 = 1 + \gamma v_4$$

$$v_3 = 1 + \gamma v_4$$

$$v_4 = 1 + \gamma v_4$$

**Exercise:** How to solve them? We can first calculate  $v_4$ , and then

# Outline

- 1 Motivating examples
- 2 State value
- 3 Bellman equation: Derivation
- 4 Bellman equation: Matrix-vector form
- 5 Bellman equation: Solve the state values
- 6 Action value
- 7 Summary

# Some notations

Consider the following single-step process:

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

- $t, t + 1$ : 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  $A_t$

Note that  $S_t, A_t, R_{t+1}$  are all *random variables*.

This step is governed by the following probability distributions:

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

At this moment, we assume we know the model (i.e., the probability distributions)!

# Some notations

Consider the following single-step process:

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

- $t, t + 1$ : 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  $A_t$

Note that  $S_t, A_t, R_{t+1}$  are all *random variables*.

This step is governed by the following probability distributions:

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

At this moment, we assume we know the model (i.e., the probability distributions)!

# Some notations

Consider the following multi-step trajectory:

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

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.

# Some notations

Consider the following multi-step trajectory:

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

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.

# State value

The expectation (or called expected value or mean) of  $G_t$  is defined as the *state-value function* or simply *state value*:

$$v_{\pi}(s) = \mathbb{E}[G_t | S_t = s]$$

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  $\pi$ . 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.

Q: What is the relationship between return and state value?

A: The state value is the mean of all possible returns that can be obtained starting from a state. If everything -  $\pi(a|s)$ ,  $p(r|s, a)$ ,  $p(s'|s, a)$  - is deterministic, then state value is the same as return.

## State value

The expectation (or called expected value or mean) of  $G_t$  is defined as the *state-value function* or simply *state value*:

$$v_{\pi}(s) = \mathbb{E}[G_t | S_t = s]$$

Remarks:

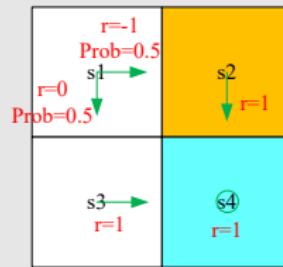
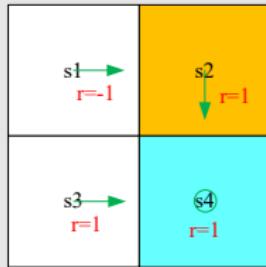
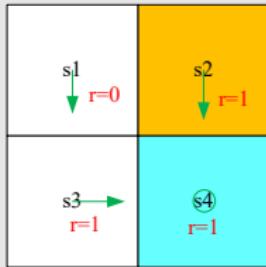
- It is a function of  $s$ . It is a conditional expectation with the condition that the state starts from  $s$ .
- It is based on the policy  $\pi$ . For a different policy, the state value may be different.
- It represents the “value” of a state. If the state value is greater, then the policy is better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

Q: What is the relationship between return and state value?

A: The state value is the mean of all possible returns that can be obtained starting from a state. If everything -  $\pi(a|s)$ ,  $p(r|s, a)$ ,  $p(s'|s, a)$  - is deterministic, then state value is the same as return.

# State value

Example:



Recall the returns obtained from  $s_1$  for the three examples:

$$v_{\pi_1}(s_1) = 0 + \gamma 1 + \gamma^2 1 + \dots = \gamma(1 + \gamma + \gamma^2 + \dots) = \frac{\gamma}{1 - \gamma}$$

$$v_{\pi_2}(s_1) = -1 + \gamma 1 + \gamma^2 1 + \dots = -1 + \gamma(1 + \gamma + \gamma^2 + \dots) = -1 + \frac{\gamma}{1 - \gamma}$$

$$v_{\pi_3}(s_1) = 0.5 \left( -1 + \frac{\gamma}{1 - \gamma} \right) + 0.5 \left( \frac{\gamma}{1 - \gamma} \right) = -0.5 + \frac{\gamma}{1 - \gamma}$$

# Outline

- 1 Motivating examples
- 2 State value
- 3 Bellman equation: Derivation
- 4 Bellman equation: Matrix-vector form
- 5 Bellman equation: Solve the state values
- 6 Action value
- 7 Summary

# Bellman equation

- While state value is important, how to calculate? The answer lies in the Bellman equation.
- In a word, the Bellman equation describes the relationship among the values of all states.
- Next, we derive the Bellman equation.
  - There is some math.
  - We already have the intuition.

# Deriving the Bellman equation

Consider a random trajectory:

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

The 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

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

Next, calculate the two terms, respectively.

# Deriving the Bellman equation

Consider a random trajectory:

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

The 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

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

Next, calculate the two terms, respectively.

# Deriving the Bellman equation

First, calculate the first term  $\mathbb{E}[R_{t+1}|S_t = s]$ :

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

Note that

- This is the mean of *immediate rewards*

# Deriving the Bellman equation

First, calculate the first term  $\mathbb{E}[R_{t+1}|S_t = s]$ :

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

Note that

- This is the mean of *immediate rewards*

# Deriving the Bellman equation

Second, calculate the second term  $\mathbb{E}[G_{t+1}|S_t = s]$ :

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

Note that

- This is the mean of *future rewards*
- $\mathbb{E}[G_{t+1}|S_t = s, S_{t+1} = s'] = \mathbb{E}[G_{t+1}|S_{t+1} = s']$  due to the memoryless Markov property.

# Deriving the Bellman equation

Second, calculate the second term  $\mathbb{E}[G_{t+1}|S_t = s]$ :

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

Note that

- This is the mean of *future rewards*
- $\mathbb{E}[G_{t+1}|S_t = s, S_{t+1} = s'] = \mathbb{E}[G_{t+1}|S_{t+1} = s']$  due to the memoryless Markov property.

# Deriving the Bellman equation

Therefore, we have

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

Highlights:

- The above equation is called the *Bellman equation*, which characterizes the relationship among the state-value functions of different states.
- It consists of two terms: the immediate reward term and the future reward term.
- A set of equations: every state has an equation like this!!!

# Deriving the Bellman equation

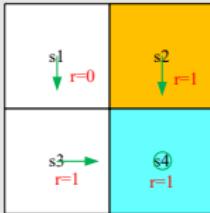
Therefore, we have

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

Highlights: symbols in this equation

- $v_{\pi}(s)$  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?

# An illustrative example



Write out the Bellman equation according to the general expression:

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

This example is simple because the policy is deterministic.

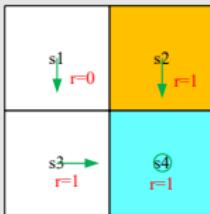
First, consider the state value of  $s_1$ :

- $\pi(a = a_3|s_1) = 1$  and  $\pi(a \neq a_3|s_1) = 0$ .
- $p(s' = s_3|s_1, a_3) = 1$  and  $p(s' \neq s_3|s_1, a_3) = 0$ .
- $p(r = 0|s_1, a_3) = 1$  and  $p(r \neq 0|s_1, a_3) = 0$ .

Substituting them into the Bellman equation gives

$$v_{\pi}(s_1) = 0 + \gamma v_{\pi}(s_3)$$

# An illustrative example



Write out the Bellman equation according to the general expression:

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

This example is simple because the policy is deterministic.

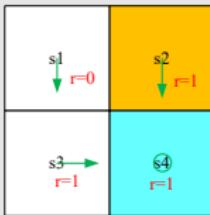
First, consider the state value of  $s_1$ :

- $\pi(a = a_3|s_1) = 1$  and  $\pi(a \neq a_3|s_1) = 0$ .
- $p(s' = s_3|s_1, a_3) = 1$  and  $p(s' \neq s_3|s_1, a_3) = 0$ .
- $p(r = 0|s_1, a_3) = 1$  and  $p(r \neq 0|s_1, a_3) = 0$ .

Substituting them into the Bellman equation gives

$$v_{\pi}(s_1) = 0 + \gamma v_{\pi}(s_3)$$

# An illustrative example



Write out the Bellman equation according to the general expression:

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

This example is simple because the policy is deterministic.

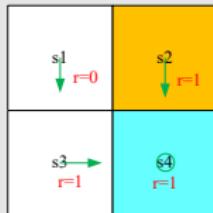
First, consider the state value of  $s_1$ :

- $\pi(a = a_3|s_1) = 1$  and  $\pi(a \neq a_3|s_1) = 0$ .
- $p(s' = s_3|s_1, a_3) = 1$  and  $p(s' \neq s_3|s_1, a_3) = 0$ .
- $p(r = 0|s_1, a_3) = 1$  and  $p(r \neq 0|s_1, a_3) = 0$ .

Substituting them into the Bellman equation gives

$$v_{\pi}(s_1) = 0 + \gamma v_{\pi}(s_3)$$

# An illustrative example



Write out the Bellman equation according to the general expression.

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

Similarly, it can be obtained that

$$v_\pi(s_1) = 0 + \gamma v_\pi(s_3),$$

$$v_\pi(s_2) = 1 + \gamma v_\pi(s_4),$$

$$v_\pi(s_3) = 1 + \gamma v_\pi(s_4),$$

$$v_\pi(s_4) = 1 + \gamma v_\pi(s_4).$$

# An illustrative example

How to solve them?

$$v_{\pi}(s_1) = 0 + \gamma v_{\pi}(s_3),$$

$$v_{\pi}(s_2) = 1 + \gamma v_{\pi}(s_4),$$

$$v_{\pi}(s_3) = 1 + \gamma v_{\pi}(s_4),$$

$$v_{\pi}(s_4) = 1 + \gamma v_{\pi}(s_4).$$

Solve the above equations one by one from the last to the first:

$$v_{\pi}(s_4) = \frac{1}{1 - \gamma},$$

$$v_{\pi}(s_3) = \frac{1}{1 - \gamma},$$

$$v_{\pi}(s_2) = \frac{1}{1 - \gamma},$$

$$v_{\pi}(s_1) = \frac{\gamma}{1 - \gamma}.$$

# An illustrative example

How to solve them?

$$v_\pi(s_1) = 0 + \gamma v_\pi(s_3),$$

$$v_\pi(s_2) = 1 + \gamma v_\pi(s_4),$$

$$v_\pi(s_3) = 1 + \gamma v_\pi(s_4),$$

$$v_\pi(s_4) = 1 + \gamma v_\pi(s_4).$$

Solve the above equations one by one from the last to the first:

$$v_\pi(s_4) = \frac{1}{1 - \gamma},$$

$$v_\pi(s_3) = \frac{1}{1 - \gamma},$$

$$v_\pi(s_2) = \frac{1}{1 - \gamma},$$

$$v_\pi(s_1) = \frac{\gamma}{1 - \gamma}.$$

## An illustrative example

If  $\gamma = 0.9$ , then

$$v_{\pi}(s_4) = \frac{1}{1 - 0.9} = 10,$$

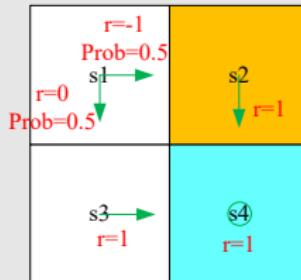
$$v_{\pi}(s_3) = \frac{1}{1 - 0.9} = 10,$$

$$v_{\pi}(s_2) = \frac{1}{1 - 0.9} = 10,$$

$$v_{\pi}(s_1) = \frac{0.9}{1 - 0.9} = 9.$$

What to do after we have calculated state values? Be patient  
(calculating action value and improve policy)

# Exercise



**Exercise:**

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

- write out the Bellman equations for each state.
- solve the state values from the Bellman equations.
- compare with the policy in the last example.

# Exercise

Answer:

$$v_{\pi}(s_1) = 0.5[0 + \gamma v_{\pi}(s_3)] + 0.5[-1 + \gamma v_{\pi}(s_2)],$$

$$v_{\pi}(s_2) = 1 + \gamma v_{\pi}(s_4),$$

$$v_{\pi}(s_3) = 1 + \gamma v_{\pi}(s_4),$$

$$v_{\pi}(s_4) = 1 + \gamma v_{\pi}(s_4).$$

Solve the above equations one by one from the last to the first.

$$v_{\pi}(s_4) = \frac{1}{1-\gamma}, \quad v_{\pi}(s_3) = \frac{1}{1-\gamma}, \quad v_{\pi}(s_2) = \frac{1}{1-\gamma},$$

$$v_{\pi}(s_1) = 0.5[0 + \gamma v_{\pi}(s_3)] + 0.5[-1 + \gamma v_{\pi}(s_2)],$$

$$= -0.5 + \frac{\gamma}{1-\gamma}.$$

Substituting  $\gamma = 0.9$  yields

$$v_{\pi}(s_4) = 10, \quad v_{\pi}(s_3) = 10, \quad v_{\pi}(s_2) = 10, \quad v_{\pi}(s_1) = -0.5 + 9 = 8.5.$$

Compare with the previous policy. This one is worse.

# Outline

- 1 Motivating examples
- 2 State value
- 3 Bellman equation: Derivation
- 4 Bellman equation: Matrix-vector form
- 5 Bellman equation: Solve the state values
- 6 Action value
- 7 Summary

# Matrix-vector form of the Bellman equation

Why consider the matrix-vector form?

- How to solve the Bellman equation?

One unknown relies on another unknown.

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

- The above *elementwise form* is valid for every state  $s \in \mathcal{S}$ . That means there are  $|\mathcal{S}|$  equations like this!
- If we put all the equations together, we have a set of linear equations, which can be concisely written in a *matrix-vector form*.
- The matrix-vector form is very elegant and important.

# Matrix-vector form of the Bellman equation

Recall that:

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

Rewrite the Bellman equation as

$$v_\pi(s) = r_\pi(s) + \gamma \sum_{s'} p_\pi(s'|s)v_\pi(s') \quad (1)$$

where

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

# Matrix-vector form of the Bellman equation

Suppose the states could be indexed as  $s_i$  ( $i = 1, \dots, n$ ).

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

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

where

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

# Illustrative examples

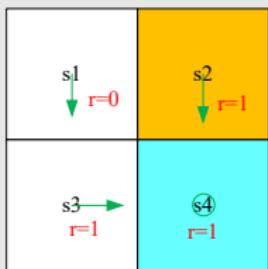
If there are four states,  $v_\pi = r_\pi + \gamma P_\pi v_\pi$  can be written out as

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

# Illustrative examples

If there are four states,  $v_\pi = r_\pi + \gamma P_\pi v_\pi$  can be written out as

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



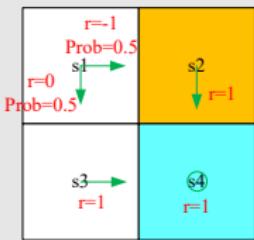
For this specific example:

$$\begin{bmatrix} v_\pi(s_1) \\ v_\pi(s_2) \\ v_\pi(s_3) \\ v_\pi(s_4) \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} + \gamma \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_\pi(s_1) \\ v_\pi(s_2) \\ v_\pi(s_3) \\ v_\pi(s_4) \end{bmatrix}$$

# Illustrative examples

If there are four states,  $v_\pi = r_\pi + \gamma P_\pi v_\pi$  can be written out as

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



For this specific example:

$$\begin{bmatrix} v_\pi(s_1) \\ v_\pi(s_2) \\ v_\pi(s_3) \\ v_\pi(s_4) \end{bmatrix} = \begin{bmatrix} 0.5(0) + 0.5(-1) \\ 1 \\ 1 \\ 1 \end{bmatrix} + \gamma \begin{bmatrix} 0 & 0.5 & 0.5 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_\pi(s_1) \\ v_\pi(s_2) \\ v_\pi(s_3) \\ v_\pi(s_4) \end{bmatrix}.$$

# Outline

- 1 Motivating examples
- 2 State value
- 3 Bellman equation: Derivation
- 4 Bellman equation: Matrix-vector form
- 5 Bellman equation: Solve the state values
- 6 Action value
- 7 Summary

# Solve state values

Why to solve state values?

- Given a policy, finding out the corresponding state values is called *policy evaluation*! It is a fundamental problem in RL. It is the foundation to find better policies.
- It is important to understand how to solve the Bellman equation.

# Solve state values

The Bellman equation in matrix-vector form is

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

- The *closed-form solution* is:

$$v_\pi = (I - \gamma P_\pi)^{-1} r_\pi$$

In practice, we still need to use numerical tools to calculate the matrix inverse.

Can we avoid the matrix inverse operation? Yes, by iterative algorithms.

- An *iterative solution* is:

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

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

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

# Solve state values

The Bellman equation in matrix-vector form is

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

- The *closed-form solution* is:

$$v_\pi = (I - \gamma P_\pi)^{-1} r_\pi$$

In practice, we still need to use numerical tools to calculate the matrix inverse.

Can we avoid the matrix inverse operation? Yes, by iterative algorithms.

- An *iterative solution* is:

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

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

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

## Solve state values (optional)

Proof.

Define the error as  $\delta_k = v_k - v_\pi$ . We only need to show  $\delta_k \rightarrow 0$ . Substituting  $v_{k+1} = \delta_{k+1} + v_\pi$  and  $v_k = \delta_k + v_\pi$  into  $v_{k+1} = r_\pi + \gamma P_\pi v_k$  gives

$$\delta_{k+1} + v_\pi = r_\pi + \gamma P_\pi(\delta_k + v_\pi),$$

which can be rewritten as

$$\delta_{k+1} = -v_\pi + r_\pi + \gamma P_\pi \delta_k + \gamma P_\pi v_\pi = \gamma P_\pi \delta_k.$$

As a result,

$$\delta_{k+1} = \gamma P_\pi \delta_k = \gamma^2 P_\pi^2 \delta_{k-1} = \dots = \gamma^{k+1} P_\pi^{k+1} \delta_0.$$

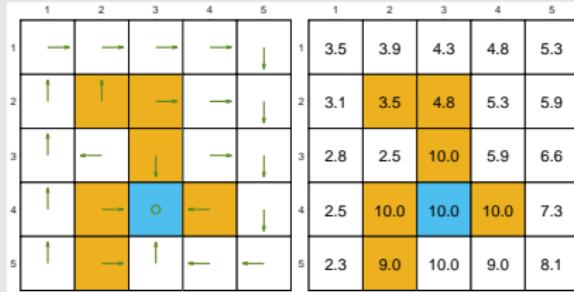
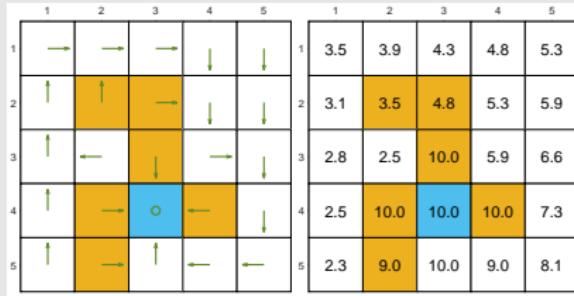
Note that  $0 \leq P_\pi^k \leq 1$ , which means every entry of  $P_\pi^k$  is no greater than 1 for any  $k = 0, 1, 2, \dots$ . That is because  $P_\pi^k \mathbf{1} = \mathbf{1}$ , where  $\mathbf{1} = [1, \dots, 1]^T$ . On the other hand, since  $\gamma < 1$ , we know  $\gamma^k \rightarrow 0$  and hence  $\delta_{k+1} = \gamma^{k+1} P_\pi^{k+1} \delta_0 \rightarrow 0$  as  $k \rightarrow \infty$ .

□

# Solve state values

Examples:  $r_{\text{boundary}} = r_{\text{forbidden}} = -1$ ,  $r_{\text{target}} = +1$ ,  $\gamma = 0.9$

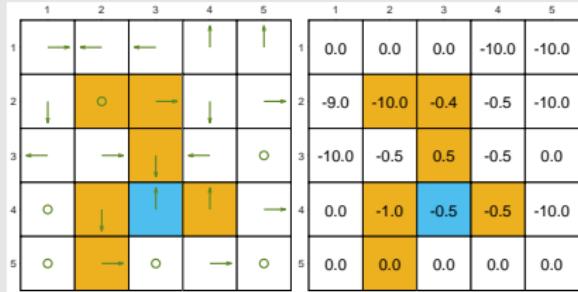
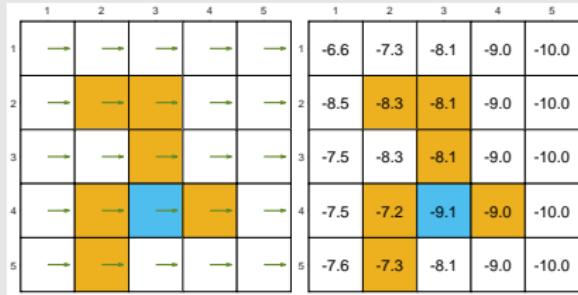
The following are two “good” policies and the state values. The two policies are different for the top two states in the forth column.



# Solve state values

Examples:  $r_{\text{boundary}} = r_{\text{forbidden}} = -1$ ,  $r_{\text{target}} = +1$ ,  $\gamma = 0.9$

The following are two “bad” policies and the state values. The state values are less than those of the good policies.



# Outline

- 1 Motivating examples
- 2 State value
- 3 Bellman equation: Derivation
- 4 Bellman equation: Matrix-vector form
- 5 Bellman equation: Solve the state values
- 6 Action value
- 7 Summary

# Action value

From state value to 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*.

Why do we care action value? Because we want to know which action is better. This point will be clearer in the following lectures.

We will frequently use action values.

# Action value

Definition:

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

- $q_\pi(s, a)$  is a function of the state-action pair  $(s, a)$
- $q_\pi(s, a)$  depends on  $\pi$

It follows from the properties of conditional expectation that

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

Hence,

$$v_\pi(s) = \sum_a \pi(a|s) q_\pi(s, a) \tag{2}$$

# Action value

Definition:

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

- $q_\pi(s, a)$  is a function of the state-action pair  $(s, a)$
- $q_\pi(s, a)$  depends on  $\pi$

It follows from the properties of conditional expectation that

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

Hence,

$$v_\pi(s) = \sum_a \pi(a|s) q_\pi(s, a) \tag{2}$$

# Action value

Recall that the state value is given by

$$v_{\pi}(s) = \sum_a \pi(a|s) \left[ \underbrace{\sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a)v_{\pi}(s')}_{q_{\pi}(s,a)} \right] \quad (3)$$

By comparing (2) and (3), we have the **action-value function** as

$$q_{\pi}(s,a) = \sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a)v_{\pi}(s') \quad (4)$$

(2) and (4) are the two sides of the same coin:

- (2) shows how to obtain state values from action values.
- (4) shows how to obtain action values from state values.

# Action value

Recall that the state value is given by

$$v_{\pi}(s) = \sum_a \pi(a|s) \left[ \underbrace{\sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a)v_{\pi}(s')}_{q_{\pi}(s,a)} \right] \quad (3)$$

By comparing (2) and (3), we have the **action-value function** as

$$q_{\pi}(s,a) = \sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a)v_{\pi}(s') \quad (4)$$

(2) and (4) are the two sides of the same coin:

- (2) shows how to obtain state values from action values.
- (4) shows how to obtain action values from state values.

# Action value

Recall that the state value is given by

$$v_{\pi}(s) = \sum_a \pi(a|s) \left[ \underbrace{\sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a)v_{\pi}(s')}_{q_{\pi}(s,a)} \right] \quad (3)$$

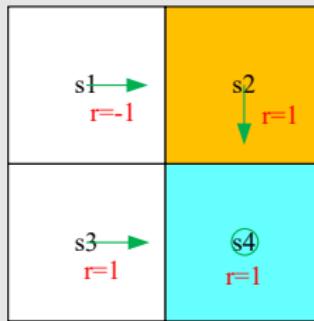
By comparing (2) and (3), we have the **action-value function** as

$$q_{\pi}(s,a) = \sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a)v_{\pi}(s') \quad (4)$$

(2) and (4) are the two sides of the same coin:

- (2) shows how to obtain state values from action values.
- (4) shows how to obtain action values from state values.

# Illustrative example for action value



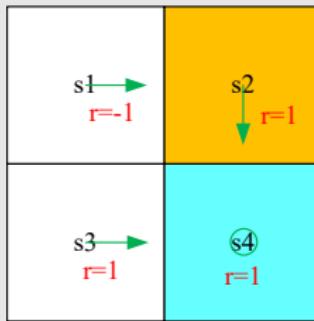
Write out the action values for state  $s_1$ .

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

Questions:

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

# Illustrative example for action value



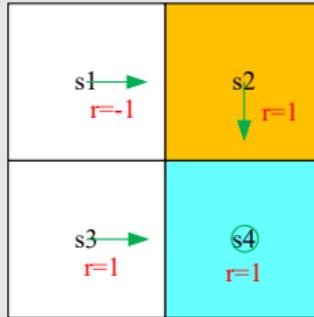
Write out the action values for state  $s_1$ .

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

Questions:

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

## Illustrative example for action value



For the other actions:

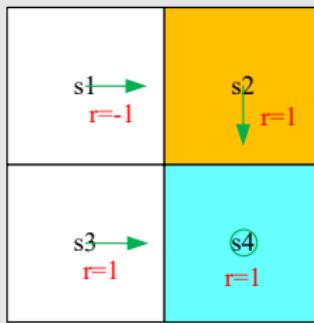
$$q_{\pi}(s_1, a_1) = -1 + \gamma v_{\pi}(s_1),$$

$$q_{\pi}(s_1, a_3) = 0 + \gamma v_{\pi}(s_3),$$

$$q_{\pi}(s_1, a_4) = -1 + \gamma v_{\pi}(s_1),$$

$$q_{\pi}(s_1, a_5) = 0 + \gamma v_{\pi}(s_1).$$

# Illustrative example for action value



Highlights:

- Action value is important since we care about which action to take.
- 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.

# Outline

- 1 Motivating examples
- 2 State value
- 3 Bellman equation: Derivation
- 4 Bellman equation: Matrix-vector form
- 5 Bellman equation: Solve the state values
- 6 Action value
- 7 Summary

# Summary

Key concepts and results:

- State value:  $v_\pi(s) = \mathbb{E}[G_t | S_t = s]$
- Action value:  $q_\pi(s, a) = \mathbb{E}[G_t | S_t = s, A_t = a]$
- The Bellman equation (elementwise form):

$$\begin{aligned} v_\pi(s) &= \sum_a \pi(a|s) \left[ \underbrace{\sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v_\pi(s')}_{q_\pi(s, a)} \right] \\ &= \sum_a \pi(a|s) q_\pi(s, a) \end{aligned}$$

- The Bellman equation (matrix-vector form):

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

- How to solve the Bellman equation: closed-form solution, iterative solution

# Summary

Key concepts and results:

- State value:  $v_\pi(s) = \mathbb{E}[G_t | S_t = s]$
- Action value:  $q_\pi(s, a) = \mathbb{E}[G_t | S_t = s, A_t = a]$
- The Bellman equation (elementwise form):

$$\begin{aligned} v_\pi(s) &= \sum_a \pi(a|s) \left[ \underbrace{\sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v_\pi(s')}_{q_\pi(s, a)} \right] \\ &= \sum_a \pi(a|s) q_\pi(s, a) \end{aligned}$$

- The Bellman equation (matrix-vector form):

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

- How to solve the Bellman equation: closed-form solution, iterative solution

# Summary

Key concepts and results:

- State value:  $v_\pi(s) = \mathbb{E}[G_t | S_t = s]$
- Action value:  $q_\pi(s, a) = \mathbb{E}[G_t | S_t = s, A_t = a]$
- The Bellman equation (elementwise form):

$$\begin{aligned} v_\pi(s) &= \sum_a \pi(a|s) \left[ \underbrace{\sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v_\pi(s')}_{q_\pi(s, a)} \right] \\ &= \sum_a \pi(a|s) q_\pi(s, a) \end{aligned}$$

- The Bellman equation (matrix-vector form):

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

- How to solve the Bellman equation: closed-form solution, iterative solution

# Summary

Key concepts and results:

- State value:  $v_\pi(s) = \mathbb{E}[G_t | S_t = s]$
- Action value:  $q_\pi(s, a) = \mathbb{E}[G_t | S_t = s, A_t = a]$
- The Bellman equation (elementwise form):

$$\begin{aligned} v_\pi(s) &= \sum_a \pi(a|s) \left[ \underbrace{\sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v_\pi(s')}_{q_\pi(s, a)} \right] \\ &= \sum_a \pi(a|s) q_\pi(s, a) \end{aligned}$$

- The Bellman equation (matrix-vector form):

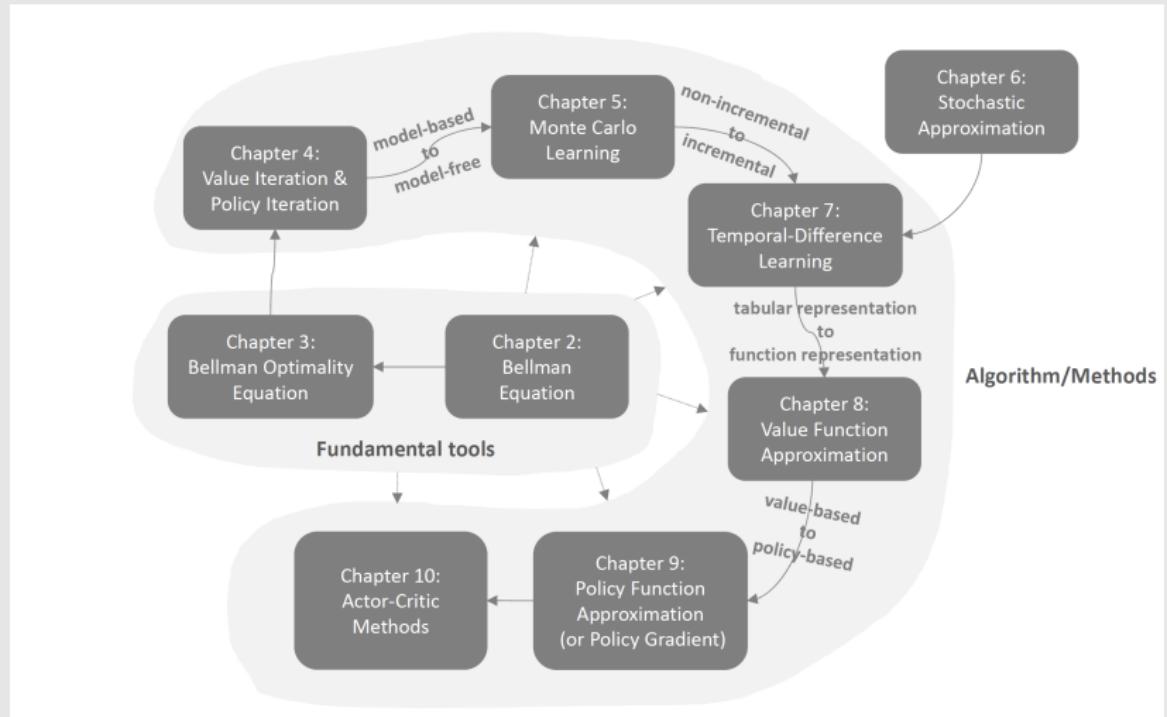
$$v_\pi = r_\pi + \gamma P_\pi v_\pi$$

- How to solve the Bellman equation: closed-form solution, iterative solution

# Optimal Policy and Bellman Optimality Equation

Shiyu Zhao

# Outline



# Outline

In this lecture:

- Core concepts: optimal state value and optimal policy
- A fundamental tool: the Bellman optimality equation (BOE)

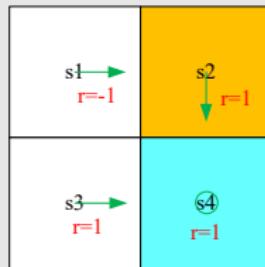
# Outline

- 1 Motivating examples
- 2 Definition of optimal policy
- 3 BOE: Introduction
- 4 BOE: Maximization on the right-hand side
- 5 BOE: Rewrite as  $v = f(v)$
- 6 Contraction mapping theorem
- 7 BOE: Solution
- 8 BOE: Optimality
- 9 Analyzing optimal policies

# Outline

- 1 Motivating examples
- 2 Definition of optimal policy
- 3 BOE: Introduction
- 4 BOE: Maximization on the right-hand side
- 5 BOE: Rewrite as  $v = f(v)$
- 6 Contraction mapping theorem
- 7 BOE: Solution
- 8 BOE: Optimality
- 9 Analyzing optimal policies

# Motivating examples



Bellman equation:

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

$$v_\pi(s_2) = +1 + \gamma v_\pi(s_4),$$

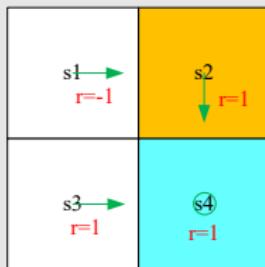
$$v_\pi(s_3) = +1 + \gamma v_\pi(s_4),$$

$$v_\pi(s_4) = +1 + \gamma v_\pi(s_4).$$

State value: Let  $\gamma = 0.9$ . Then, it can be calculated that

$$v_\pi(s_4) = v_\pi(s_3) = v_\pi(s_2) = 10, \quad v_\pi(s_1) = 8.$$

# Motivating examples



Action value: consider  $s_1$

$$q_{\pi}(s_1, a_1) = -1 + \gamma v_{\pi}(s_1) = 6.2,$$

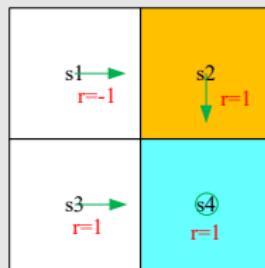
$$q_{\pi}(s_1, a_2) = -1 + \gamma v_{\pi}(s_2) = 8,$$

$$q_{\pi}(s_1, a_3) = 0 + \gamma v_{\pi}(s_3) = 9,$$

$$q_{\pi}(s_1, a_4) = -1 + \gamma v_{\pi}(s_1) = 6.2,$$

$$q_{\pi}(s_1, a_5) = 0 + \gamma v_{\pi}(s_1) = 7.2.$$

# Motivating examples



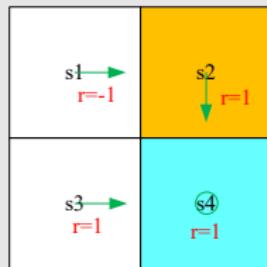
Question: While the policy is not good, how can we improve it?

Answer: by using action values.

The current policy  $\pi(a|s_1)$  is

$$\pi(a|s_1) = \begin{cases} 1 & a = a_2 \\ 0 & a \neq a_2 \end{cases}$$

# Motivating examples



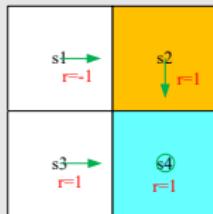
Question: While the policy is not good, how can we improve it?

Answer: by using action values.

The current policy  $\pi(a|s_1)$  is

$$\pi(a|s_1) = \begin{cases} 1 & a = a_2 \\ 0 & a \neq a_2 \end{cases}$$

# Motivating examples



Observe the action values that we obtained just now:

$$q_{\pi}(s_1, a_1) = 6.2, q_{\pi}(s_1, a_2) = 8, q_{\pi}(s_1, \textcolor{blue}{a}_3) = 9,$$

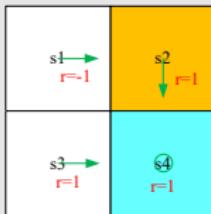
$$q_{\pi}(s_1, a_4) = 6.2, q_{\pi}(s_1, a_5) = 7.2.$$

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

# Motivating examples



Observe the action values that we obtained just now:

$$q_{\pi}(s_1, a_1) = 6.2, q_{\pi}(s_1, a_2) = 8, q_{\pi}(s_1, a_3) = 9,$$

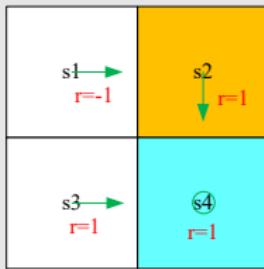
$$q_{\pi}(s_1, a_4) = 6.2, q_{\pi}(s_1, a_5) = 7.2.$$

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

# Motivating examples



Question: why doing this can improve the policy?

- Intuition: action values can be used to evaluate actions.
- Math: nontrivial and will be introduced in this lecture.

# Outline

- 1 Motivating examples
- 2 Definition of optimal policy
- 3 BOE: Introduction
- 4 BOE: Maximization on the right-hand side
- 5 BOE: Rewrite as  $v = f(v)$
- 6 Contraction mapping theorem
- 7 BOE: Solution
- 8 BOE: Optimality
- 9 Analyzing optimal policies

# Optimal policy

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  $\pi_1$  is “better” than  $\pi_2$ .

The definition leads to many questions:

- Does the optimal policy exist?
- Is the optimal policy unique?
- Is the optimal policy stochastic or deterministic?
- How to obtain the optimal policy?

To answer these questions, we study the *Bellman optimality equation*.

# Optimal policy

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  $\pi_1$  is “better” than  $\pi_2$ .

## Definition

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

The definition leads to many questions:

- Does the optimal policy exist?
- Is the optimal policy unique?
- Is the optimal policy stochastic or deterministic?
- How to obtain the optimal policy?

To answer these questions, we study the *Bellman optimality equation*.

# Optimal policy

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  $\pi_1$  is “better” than  $\pi_2$ .

## Definition

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

The definition leads to many questions:

- Does the optimal policy exist?
- Is the optimal policy unique?
- Is the optimal policy stochastic or deterministic?
- How to obtain the optimal policy?

To answer these questions, we study the *Bellman optimality equation*.

# Optimal policy

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  $\pi_1$  is “better” than  $\pi_2$ .

## Definition

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

The definition leads to many questions:

- Does the optimal policy exist?
- Is the optimal policy unique?
- Is the optimal policy stochastic or deterministic?
- How to obtain the optimal policy?

To answer these questions, we study the *Bellman optimality equation*.

# Outline

- 1 Motivating examples
- 2 Definition of optimal policy
- 3 BOE: Introduction**
- 4 BOE: Maximization on the right-hand side
- 5 BOE: Rewrite as  $v = f(v)$
- 6 Contraction mapping theorem
- 7 BOE: Solution
- 8 BOE: Optimality
- 9 Analyzing optimal policies

# Bellman optimality equation (BOE)

**Bellman optimality equation (elementwise form):**

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

# Bellman optimality equation (BOE)

**Bellman optimality equation (elementwise form):**

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

# Bellman optimality equation (BOE)

**Bellman optimality equation (elementwise form):**

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

# Bellman optimality equation (BOE)

**Bellman optimality equation (elementwise form):**

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

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?

# Bellman optimality equation (BOE)

**Bellman optimality equation (matrix-vector form):**

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

where the elements corresponding to  $s$  or  $s'$  are

$$[r_{\pi}]_s \triangleq \sum_a \pi(a|s) \sum_r p(r|s, a)r,$$

$$[P_{\pi}]_{s,s'} = p(s'|s) \triangleq \sum_a \pi(a|s) \sum_{s'} p(s'|s, a)$$

Here  $\max_{\pi}$  is performed elementwise.

# Bellman optimality equation (BOE)

**Bellman optimality equation (matrix-vector form):**

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

BOE is **tricky** yet **elegant**!

- Why elegant? It describes the optimal policy and optimal state value in an elegant way.
- Why tricky? There is a maximization on the right-hand side, which may not be straightforward to see how to compute.
- Many questions to answer:
  - Algorithm: how to solve this equation?
  - Existence: does this equation have solutions?
  - Uniqueness: is the solution to this equation unique?
  - Optimality: how is it related to optimal policy?

# Bellman optimality equation (BOE)

**Bellman optimality equation (matrix-vector form):**

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

BOE is **tricky** yet **elegant**!

- Why elegant? It describes the optimal policy and optimal state value in an elegant way.
- Why tricky? There is a maximization on the right-hand side, which may not be straightforward to see how to compute.
- Many questions to answer:
  - Algorithm: how to solve this equation?
  - Existence: does this equation have solutions?
  - Uniqueness: is the solution to this equation unique?
  - Optimality: how is it related to optimal policy?

# Outline

- 1 Motivating examples
- 2 Definition of optimal policy
- 3 BOE: Introduction
- 4 BOE: Maximization on the right-hand side
- 5 BOE: Rewrite as  $v = f(v)$
- 6 Contraction mapping theorem
- 7 BOE: Solution
- 8 BOE: Optimality
- 9 Analyzing optimal policies

# Maximization on the right-hand side of BOE

BOE: elementwise form

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

BOE: matrix-vector form  $v = \max_{\pi} (r_{\pi} + \gamma P_{\pi} v)$

# Maximization on the right-hand side of BOE

BOE: elementwise form

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

BOE: matrix-vector form  $v = \max_{\pi} (r_{\pi} + \gamma P_{\pi} v)$

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.

# Maximization on the right-hand side of BOE

Fix  $v'(s)$  first and solve  $\pi$ :

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

# Maximization on the right-hand side of BOE

Fix  $v'(s)$  first and solve  $\pi$ :

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

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

$$\max_{c_1, c_2, c_3} c_1 q_1 + c_2 q_2 + c_3 q_3.$$

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  $c_1, c_2, c_3$

$$q_3 = (c_1 + c_2 + c_3)q_3 = c_1 q_3 + c_2 q_3 + c_3 q_3 \geq c_1 q_1 + c_2 q_2 + c_3 q_3.$$

# Maximization on the right-hand side of BOE

Fix  $v'(s)$  first and solve  $\pi$ :

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

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

# Outline

- 1 Motivating examples
- 2 Definition of optimal policy
- 3 BOE: Introduction
- 4 BOE: Maximization on the right-hand side
- 5 BOE: Rewrite as  $v = f(v)$
- 6 Contraction mapping theorem
- 7 BOE: Solution
- 8 BOE: Optimality
- 9 Analyzing optimal policies

# Solve the Bellman optimality equation

The BOE is  $v = \max_{\pi} (r_{\pi} + \gamma P_{\pi} v)$ . Let

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

Then, the Bellman optimality equation becomes

$$v = f(v)$$

where

$$[f(v)]_s = \max_{\pi} \sum_a \pi(a|s) q(s, a), \quad s \in \mathcal{S}$$

Next, how to solve the equation?

# Solve the Bellman optimality equation

The BOE is  $v = \max_{\pi} (r_{\pi} + \gamma P_{\pi} v)$ . Let

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

Then, the Bellman optimality equation becomes

$$v = f(v)$$

where

$$[f(v)]_s = \max_{\pi} \sum_a \pi(a|s) q(s, a), \quad s \in \mathcal{S}$$

Next, how to solve the equation?

# Outline

- 1 Motivating examples
- 2 Definition of optimal policy
- 3 BOE: Introduction
- 4 BOE: Maximization on the right-hand side
- 5 BOE: Rewrite as  $v = f(v)$
- 6 Contraction mapping theorem
- 7 BOE: Solution
- 8 BOE: Optimality
- 9 Analyzing optimal policies

# Preliminaries: Contraction mapping theorem

Some concepts:

- **Fixed point:**  $x \in X$  is a fixed point of  $f : X \rightarrow X$  if

$$f(x) = x$$

- Contraction mapping (or contractive function):  $f$  is a contraction mapping if

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

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

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

# Preliminaries: Contraction mapping theorem

Some concepts:

- **Fixed point:**  $x \in X$  is a fixed point of  $f : X \rightarrow X$  if

$$f(x) = x$$

- **Contraction mapping (or contractive function):**  $f$  is a contraction mapping if

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

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

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

# Preliminaries: Contraction mapping theorem

Examples to demonstrate the concepts.

## Example

- $x = f(x) = 0.5x, x \in \mathbb{R}$ .

It is easy to verify that  $x = 0$  is a fixed point since  $0 = 0.5 \times 0$ .

Moreover,  $f(x) = 0.5x$  is a contraction mapping because

$$\|0.5x_1 - 0.5x_2\| = 0.5\|x_1 - x_2\| \leq \gamma\|x_1 - x_2\| \text{ for any } \gamma \in [0.5, 1).$$

# Preliminaries: Contraction mapping theorem

Examples to demonstrate the concepts.

## Example

- $x = f(x) = 0.5x, x \in \mathbb{R}$ .

It is easy to verify that  $x = 0$  is a fixed point since  $0 = 0.5 \times 0$ .

Moreover,  $f(x) = 0.5x$  is a contraction mapping because

$$\|0.5x_1 - 0.5x_2\| = 0.5\|x_1 - x_2\| \leq \gamma\|x_1 - x_2\| \text{ for any } \gamma \in [0.5, 1).$$

- $x = f(x) = Ax$ , 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.

# Preliminaries: Contraction mapping theorem

## 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  $x_k \rightarrow x^*$  as  $k \rightarrow \infty$ . Moreover, the convergence rate is exponentially fast.*

For the proof of this theorem, see the book.

# Preliminaries: Contraction mapping theorem

Examples:

- $x = 0.5x$ , where  $f(x) = 0.5x$  and  $x \in \mathbb{R}$   
 $x^* = 0$  is the unique fixed point. It can be solved iteratively by

$$x_{k+1} = 0.5x_k$$

- $x = Ax$ , where  $f(x) = Ax$  and  $x \in \mathbb{R}^n$ ,  $A \in \mathbb{R}^{n \times n}$  and  $\|A\| < 1$   
 $x^* = 0$  is the unique fixed point. It can be solved iteratively by

$$x_{k+1} = Ax_k$$

# Preliminaries: Contraction mapping theorem

Examples:

- $x = 0.5x$ , where  $f(x) = 0.5x$  and  $x \in \mathbb{R}$   
 $x^* = 0$  is the unique fixed point. It can be solved iteratively by

$$x_{k+1} = 0.5x_k$$

- $x = Ax$ , where  $f(x) = Ax$  and  $x \in \mathbb{R}^n, A \in \mathbb{R}^{n \times n}$  and  $\|A\| < 1$   
 $x^* = 0$  is the unique fixed point. It can be solved iteratively by

$$x_{k+1} = Ax_k$$

# Outline

- 1 Motivating examples
- 2 Definition of optimal policy
- 3 BOE: Introduction
- 4 BOE: Maximization on the right-hand side
- 5 BOE: Rewrite as  $v = f(v)$
- 6 Contraction mapping theorem
- 7 BOE: Solution
- 8 BOE: Optimality
- 9 Analyzing optimal policies

# Contraction property of BOE

Let's come back to the Bellman optimality equation:

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

For the proof of this lemma, see our book.

# Contraction property of BOE

Let's come back to the Bellman optimality equation:

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

Theorem (Contraction Property)

$f(v)$  is a contraction mapping satisfying

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

where  $\gamma$  is the discount rate!

For the proof of this lemma, see our book.

# Solve the Bellman optimality equation

Applying the contraction mapping theorem gives the following results.

## 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  $\gamma$ .

# Solve the Bellman optimality equation

The iterative algorithm:

Matrix-vector form:

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

Elementwise form:

$$\begin{aligned} v_{k+1}(s) &= \max_{\pi} \sum_a \pi(a|s) \left( \sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v_k(s') \right) \\ &= \max_{\pi} \sum_a \pi(a|s) q_k(s, a) \\ &= \max_a q_k(s, a) \end{aligned}$$

# Solve the Bellman optimality equation

The iterative algorithm:

Matrix-vector form:

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

Elementwise form:

$$\begin{aligned} v_{k+1}(s) &= \max_{\pi} \sum_a \pi(a|s) \left( \sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a) v_k(s') \right) \\ &= \max_{\pi} \sum_a \pi(a|s) q_k(s, a) \\ &= \max_a q_k(s, a) \end{aligned}$$

# Solve the Bellman optimality equation

Procedure summary:

- For any  $s$ , current estimated value  $v_k(s)$
- For any  $a \in \mathcal{A}(s)$ , calculate

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

- Calculate the greedy policy  $\pi_{k+1}$  for  $s$  as

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

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

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

The above algorithm is actually the value iteration algorithm as discussed in the next lecture.

# Solve the Bellman optimality equation

Procedure summary:

- For any  $s$ , current estimated value  $v_k(s)$
- For any  $a \in \mathcal{A}(s)$ , calculate

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

- Calculate the greedy policy  $\pi_{k+1}$  for  $s$  as

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

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

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

The above algorithm is actually the value iteration algorithm as discussed in the next lecture.

# Solve the Bellman optimality equation

Procedure summary:

- For any  $s$ , current estimated value  $v_k(s)$
- For any  $a \in \mathcal{A}(s)$ , calculate

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

- Calculate the greedy policy  $\pi_{k+1}$  for  $s$  as

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

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

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

The above algorithm is actually the value iteration algorithm as discussed in the next lecture.

# Solve the Bellman optimality equation

Procedure summary:

- For any  $s$ , current estimated value  $v_k(s)$
- For any  $a \in \mathcal{A}(s)$ , calculate

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

- Calculate the greedy policy  $\pi_{k+1}$  for  $s$  as

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

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

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

The above algorithm is actually the value iteration algorithm as discussed in the next lecture.

# Solve the Bellman optimality equation

Procedure summary:

- For any  $s$ , current estimated value  $v_k(s)$
- For any  $a \in \mathcal{A}(s)$ , calculate

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

- Calculate the greedy policy  $\pi_{k+1}$  for  $s$  as

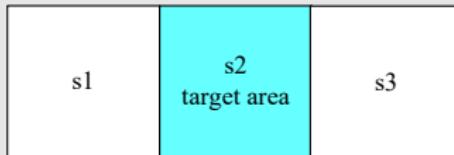
$$\pi_{k+1}(a|s) = \begin{cases} 1 & a = a_k^*(s) \\ 0 & a \neq a_k^*(s) \end{cases}$$

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

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

The above algorithm is actually the **value iteration algorithm** as discussed in the next lecture.

# Example



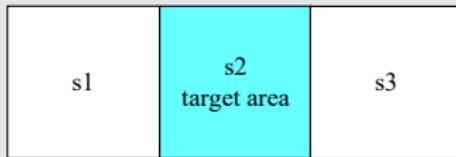
Example: Manually solve the BOE.

- Why manually? Can understand better.
- Why so simple example? Can be calculated manually.

Actions:  $a_\ell, a_0, a_r$  represent go left, stay unchanged, and go right.

Reward: entering the target area: +1; try to go out of boundary -1.

# Example

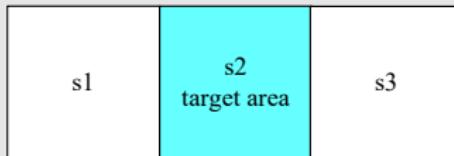


The values of  $q(s, a)$

q-value table	$a_\ell$	$a_0$	$a_r$
$s_1$	$-1 + \gamma v(s_1)$	$0 + \gamma v(s_1)$	$1 + \gamma v(s_2)$
$s_2$	$0 + \gamma v(s_1)$	$1 + \gamma v(s_2)$	$0 + \gamma v(s_3)$
$s_3$	$1 + \gamma v(s_2)$	$0 + \gamma v(s_3)$	$-1 + \gamma v(s_3)$

Consider  $\gamma = 0.9$

# Example



The values of  $q(s, a)$

q-value table	$a_\ell$	$a_0$	$a_r$
$s_1$	$-1 + \gamma v(s_1)$	$0 + \gamma v(s_1)$	$1 + \gamma v(s_2)$
$s_2$	$0 + \gamma v(s_1)$	$1 + \gamma v(s_2)$	$0 + \gamma v(s_3)$
$s_3$	$1 + \gamma v(s_2)$	$0 + \gamma v(s_3)$	$-1 + \gamma v(s_3)$

Consider  $\gamma = 0.9$

# Example

Our objective is to find  $v^*(s_i)$  and  $\pi^*$

$k = 0$ :

v-value: select  $v_0(s_1) = v_0(s_2) = v_0(s_3) = 0$

q-value (using the previous table):

	$a_\ell$	$a_0$	$a_r$
$s_1$	-1	0	1
$s_2$	0	1	0
$s_3$	1	0	-1

Greedy policy (select the greatest q-value)

$$\pi(a_r|s_1) = 1, \quad \pi(a_0|s_2) = 1, \quad \pi(a_\ell|s_3) = 1$$

v-value:  $v_1(s) = \max_a q_0(s, a)$

$$v_1(s_1) = v_1(s_2) = v_1(s_3) = 1$$

This this policy good? Yes!

# Example

Our objective is to find  $v^*(s_i)$  and  $\pi^*$

$k = 0$ :

v-value: select  $v_0(s_1) = v_0(s_2) = v_0(s_3) = 0$

q-value (using the previous table):

	$a_\ell$	$a_0$	$a_r$
$s_1$	-1	0	1
$s_2$	0	1	0
$s_3$	1	0	-1

Greedy policy (select the greatest q-value)

$$\pi(a_r|s_1) = 1, \quad \pi(a_0|s_2) = 1, \quad \pi(a_\ell|s_3) = 1$$

v-value:  $v_1(s) = \max_a q_0(s, a)$

$$v_1(s_1) = v_1(s_2) = v_1(s_3) = 1$$

This this policy good? Yes!

# Example

Our objective is to find  $v^*(s_i)$  and  $\pi^*$

$k = 0$ :

v-value: select  $v_0(s_1) = v_0(s_2) = v_0(s_3) = 0$

q-value (using the previous table):

	$a_\ell$	$a_0$	$a_r$
$s_1$	-1	0	1
$s_2$	0	1	0
$s_3$	1	0	-1

Greedy policy (select the greatest q-value)

$$\pi(a_r|s_1) = 1, \quad \pi(a_0|s_2) = 1, \quad \pi(a_\ell|s_3) = 1$$

v-value:  $v_1(s) = \max_a q_0(s, a)$

$$v_1(s_1) = v_1(s_2) = v_1(s_3) = 1$$

This this policy good? Yes!

# Example

- $k = 1$ :

Excise: With  $v_1(s)$  calculated in the last step, calculate by yourself.

q-value:

	$a_\ell$	$a_0$	$a_r$
$s_1$	-0.1	0.9	1.9
$s_2$	0.9	1.9	0.9
$s_3$	1.9	0.9	-0.1

Greedy policy (select the greatest q-value):

$$\pi(a_r|s_1) = 1, \quad \pi(a_0|s_2) = 1, \quad \pi(a_\ell|s_3) = 1$$

The policy is the same as the previous one, which is already optimal.

v-value:  $v_2(s) = \dots$

- $k = 2, 3, \dots$

# Example

- $k = 1$ :

Excise: With  $v_1(s)$  calculated in the last step, calculate by yourself.

q-value:

	$a_\ell$	$a_0$	$a_r$
$s_1$	-0.1	0.9	1.9
$s_2$	0.9	1.9	0.9
$s_3$	1.9	0.9	-0.1

Greedy policy (select the greatest q-value):

$$\pi(a_r|s_1) = 1, \quad \pi(a_0|s_2) = 1, \quad \pi(a_\ell|s_3) = 1$$

The policy is the same as the previous one, which is already optimal.

v-value:  $v_2(s) = \dots$

- $k = 2, 3, \dots$

# Example

- $k = 1$ :

Excise: With  $v_1(s)$  calculated in the last step, calculate by yourself.

q-value:

	$a_\ell$	$a_0$	$a_r$
$s_1$	-0.1	0.9	1.9
$s_2$	0.9	1.9	0.9
$s_3$	1.9	0.9	-0.1

Greedy policy (select the greatest q-value):

$$\pi(a_r|s_1) = 1, \quad \pi(a_0|s_2) = 1, \quad \pi(a_\ell|s_3) = 1$$

The policy is the same as the previous one, which is already optimal.

v-value:  $v_2(s) = \dots$

- $k = 2, 3, \dots$

# Outline

- 1 Motivating examples
- 2 Definition of optimal policy
- 3 BOE: Introduction
- 4 BOE: Maximization on the right-hand side
- 5 BOE: Rewrite as  $v = f(v)$
- 6 Contraction mapping theorem
- 7 BOE: Solution
- 8 BOE: Optimality
- 9 Analyzing optimal policies

# Policy optimality

Suppose  $v^*$  is the solution to the Bellman optimality equation. It satisfies

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

Suppose

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

Then

$$v^* = r_{\pi^*} + \gamma P_{\pi^*} v^*$$

Therefore,  $\pi^*$  is a policy and  $v^* = v_{\pi^*}$  is the corresponding state value.

Is  $\pi^*$  the optimal policy? Is  $v^*$  the greatest state value can be achieved?

# Policy optimality

## Theorem (Policy Optimality)

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

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

For the proof, please see our book.

Now we understand why we study the BOE. That is because it describes the optimal state value and optimal policy.

# Optimal policy

What does an optimal policy  $\pi^*$  look like?

Theorem (Greedy Optimal Policy)

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

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

is an optimal policy solving the BOE. Here,

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

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

Proof: simple.  $\pi^*(s) = \arg \max_\pi \sum_a \pi(a|s) \underbrace{\left( \sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v^*(s') \right)}_{q^*(s, a)}$

# Outline

- 1 Motivating examples
- 2 Definition of optimal policy
- 3 BOE: Introduction
- 4 BOE: Maximization on the right-hand side
- 5 BOE: Rewrite as  $v = f(v)$
- 6 Contraction mapping theorem
- 7 BOE: Solution
- 8 BOE: Optimality
- 9 Analyzing optimal policies

# Analyzing optimal policies

What factors determine the optimal policy?

It can be clearly seen from the BOE

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

that there are three factors:

- Reward design:  $r$
- System model:  $p(s'|s, a)$ ,  $p(r|s, a)$
- Discount rate:  $\gamma$
- $v(s), v(s'), \pi(a|s)$  are unknowns to be calculated

Next, we use examples to show how changing  $r$  and  $\gamma$  can change the optimal policy.

# Analyzing optimal policies

What factors determine the optimal policy?

It can be clearly seen from the BOE

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

that there are three factors:

- Reward design:  $r$
- System model:  $p(s'|s, a)$ ,  $p(r|s, a)$
- Discount rate:  $\gamma$
- $v(s), v(s'), \pi(a|s)$  are unknowns to be calculated

Next, we use examples to show how changing  $r$  and  $\gamma$  can change the optimal policy.

# Analyzing optimal policies

What factors determine the optimal policy?

It can be clearly seen from the BOE

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

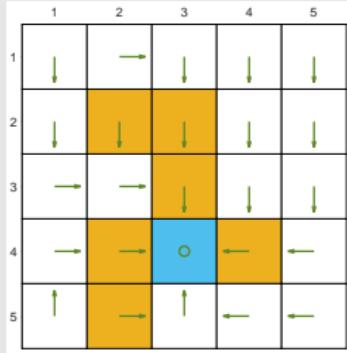
that there are three factors:

- Reward design:  $r$
- System model:  $p(s'|s, a)$ ,  $p(r|s, a)$
- Discount rate:  $\gamma$
- $v(s), v(s'), \pi(a|s)$  are unknowns to be calculated

Next, we use examples to show how changing  $r$  and  $\gamma$  can change the optimal policy.

# Analyzing optimal policies

The optimal policy and the corresponding optimal state value are obtained by solving the BOE.



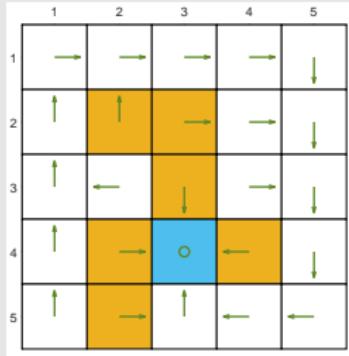
	1	2	3	4	5
1	5.8	5.6	6.2	6.5	5.8
2	6.5	7.2	8.0	7.2	6.5
3	7.2	8.0	10.0	8.0	7.2
4	8.0	10.0	10.0	10.0	8.0
5	7.2	9.0	10.0	9.0	8.1

(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!!

# Analyzing optimal policies

If we change  $\gamma = 0.9$  to  $\gamma = 0.5$



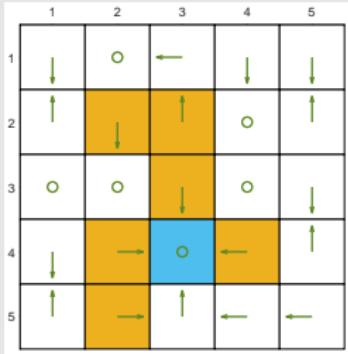
	1	2	3	4	5
1	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.1
3	0.0	0.0	2.0	0.1	0.1
4	0.0	2.0	2.0	2.0	0.2
5	0.0	1.0	2.0	1.0	0.5

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

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

# Analyzing optimal policies

If we change  $\gamma$  to 0



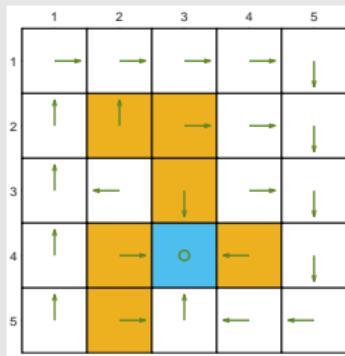
	1	2	3	4	5
1	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	1.0	0.0	0.0
4	0.0	1.0	1.0	1.0	0.0
5	0.0	0.0	1.0	0.0	0.0

(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!

# Analyzing optimal policies

If we increase the punishment when entering forbidden areas  
 $(r_{\text{forbidden}} = -1 \text{ to } r_{\text{forbidden}} = -10)$



	1	2	3	4	5
1	3.5	3.9	4.3	4.8	5.3
2	3.1	3.5	4.8	5.3	5.9
3	2.8	2.5	10.0	5.9	6.6
4	2.5	10.0	10.0	10.0	7.3
5	2.3	9.0	10.0	9.0	8.1

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

The optimal policy would also avoid the forbidden areas.

# Analyzing optimal policies

What if we change  $r \rightarrow ar + b$ ?

For example,

$$r_{\text{boundary}} = r_{\text{forbidden}} = -1, \quad r_{\text{target}} = 1$$

becomes

$$r_{\text{boundary}} = r_{\text{forbidden}} = 0, \quad r_{\text{target}} = 2, \quad r_{\text{otherstep}} = 1$$

The optimal policy remains the same!

What matters is not the absolute reward values! It is their relative values!

# Analyzing optimal policies

## Theorem (Optimal Policy Invariance)

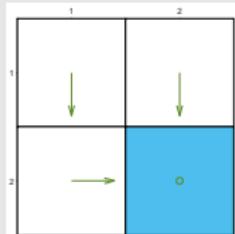
Consider a Markov decision process with  $v^* \in \mathbb{R}^{|\mathcal{S}|}$  as the optimal state value satisfying  $v^* = \max_{\pi} (r_{\pi} + \gamma P_{\pi} v^*)$ . If every reward  $r$  is changed by an affine transformation to  $ar + b$ , where  $a, b \in \mathbb{R}$  and  $a \neq 0$ , then the 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  $\mathbf{1} = [1, \dots, 1]^T$ . Consequently, the optimal policies are invariant to the affine transformation of the reward signals.

# Analyzing optimal policies

Meaningless detour?



(a) Optimal policy

A 2x2 grid representing a two-player game. Player 1 is the row player, and Player 2 is the column player. The payoffs are as follows:

		1	2
1		9.0	8.1
2		10.0	10.0

Arrows indicate the policy: Player 1's row 1 leads to column 2; Player 2's column 1 leads to row 1. A blue shaded area covers the bottom-right cell (10.0). A green dot is placed in the top-right cell (8.1).

(b) Not optimal

The policy in (a) is optimal, the policy in (b) is not.

Question: Why the optimal policy is not (b)? Why does the optimal policy not take meaningless detours? There is no punishment for taking detours!!

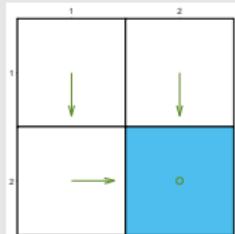
Due to the discount rate!

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

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

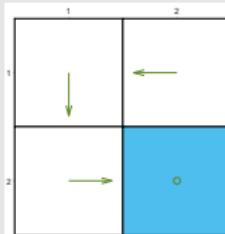
# Analyzing optimal policies

Meaningless detour?



(a) Optimal policy

1	9.0	10.0
2	10.0	10.0



(b) Not optimal

The policy in (a) is optimal, the policy in (b) is not.

Question: Why the optimal policy is not (b)? Why does the optimal policy not take meaningless detours? **There is no punishment for taking detours!!**

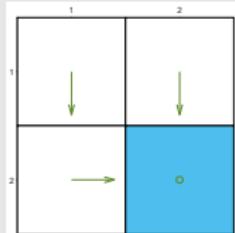
Due to the discount rate!

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

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

# Analyzing optimal policies

Meaningless detour?

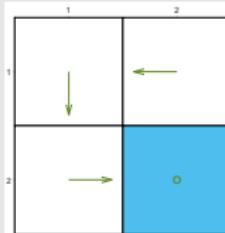


(a) Optimal policy

1	2
1	9.0
2	10.0

1	2
1	10.0
2	10.0



(b) Not optimal

The policy in (a) is optimal, the policy in (b) is not.

Question: Why the optimal policy is not (b)? Why does the optimal policy not take meaningless detours? **There is no punishment for taking detours!!**

Due to the discount rate!

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

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

# Summary

Bellman optimality equation:

- Elementwise form:

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

- Matrix-vector form:

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

# Summary

Questions about the Bellman optimality equation:

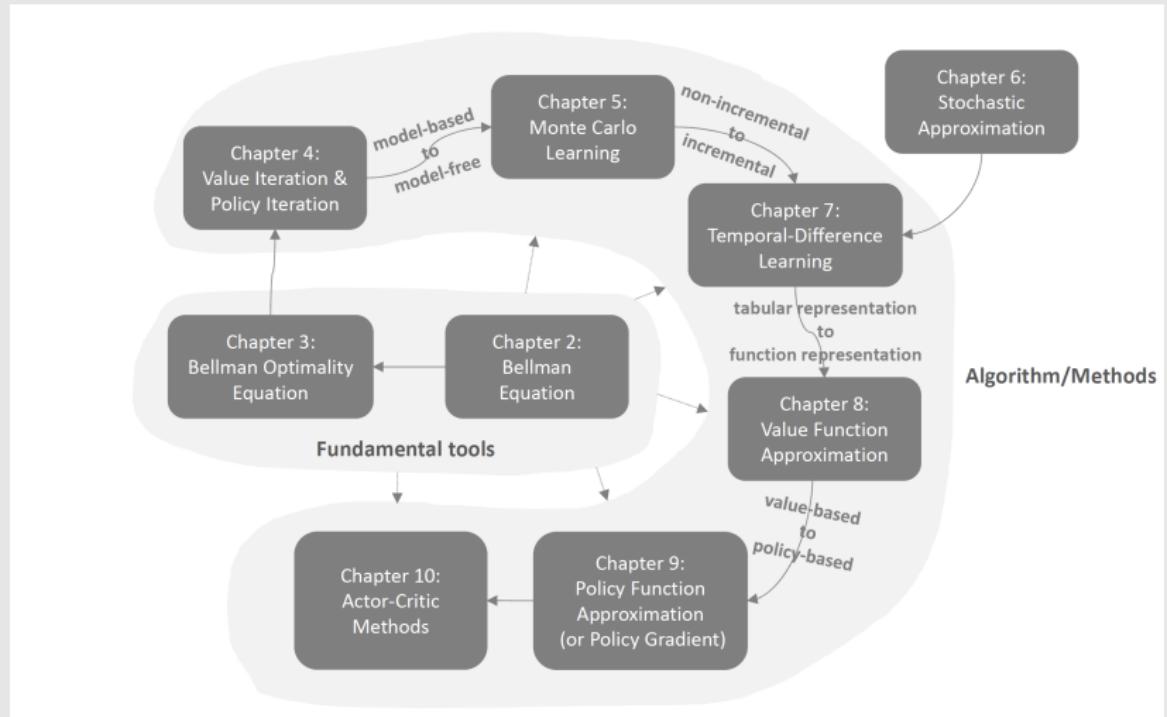
- Existence: does this equation have solutions?
  - Yes, by the contraction mapping Theorem
- Uniqueness: is the solution to this equation unique?
  - Yes, by the contraction mapping Theorem
- Algorithm: how to solve this equation?
  - Iterative algorithm suggested by the contraction mapping Theorem
- Optimality: why we study this equation
  - Because its solution corresponds to the optimal state value and optimal policy.

Finally, we understand why it is important to study the BOE!

# Lecture 4: Value Iteration and Policy Iteration

Shiyu Zhao

# Outline



# Outline

- 1** Value iteration algorithm
- 2** Policy iteration algorithm
- 3** Truncated policy iteration algorithm

# Outline

- 1** Value iteration algorithm
- 2** Policy iteration algorithm
- 3** Truncated policy iteration algorithm

# Value iteration algorithm

- ▷ How to solve the Bellman optimality equation?

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

- ▷ In the last lecture, we know that the contraction mapping theorem suggest an iterative algorithm:

$$v_{k+1} = f(v_k) = \max_{\pi} (r_{\pi} + \gamma P_{\pi} v_k), \quad k = 1, 2, 3 \dots$$

where  $v_0$  can be arbitrary.

- ▷ This algorithm can eventually find the optimal state value and an optimal policy.
- ▷ This algorithm is called *value iteration*!
- ▷ We will see that the math about the BOE that we have learned finally pays off!

# Value iteration algorithm

The algorithm

$$v_{k+1} = f(v_k) = \max_{\pi} (r_{\pi} + \gamma P_{\pi} v_k), \quad k = 1, 2, 3 \dots$$

can be decomposed to two steps.

- Step 1: policy update. This step is to solve

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

where  $v_k$  is given.

- Step 2: value update.

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

Question: is  $v_k$  a state value? No, because it is not ensured that  $v_k$  satisfies a Bellman equation.

# Value iteration algorithm

- ▷ Next, we need to study the elementwise form in order to implement the algorithm.
  - Matrix-vector form is useful for theoretical analysis.
  - Elementwise form is useful for implementation.

# Value iteration algorithm - Elementwise form

▷ Step 1: Policy update

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

# Value iteration algorithm - Elementwise form

▷ Step 2: Value update

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  $\pi_{k+1}$  is greedy, the above equation is simply

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

# Value iteration algorithm - Pseudocode

▷ Procedure summary:

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

## 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  $k$ th iteration, do

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

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

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

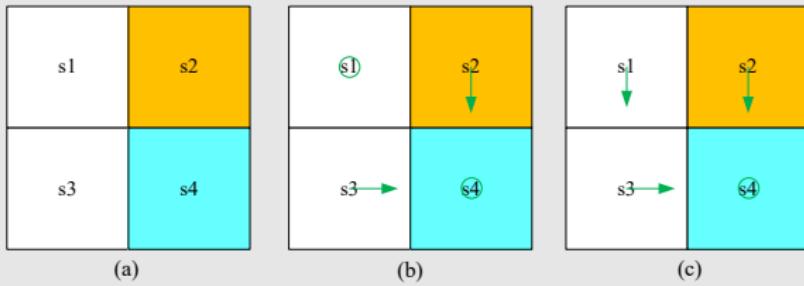
$$\text{Maximum action value: } a_k^*(s) = \arg \max_a q_k(a, s)$$

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

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

# Value iteration algorithm - Example

- ▷ The reward setting is  $r_{\text{boundary}} = r_{\text{forbidden}} = -1$ ,  $r_{\text{target}} = 1$ . The discount rate is  $\gamma = 0.9$ .



q-table: The expression of  $q(s, a)$ .

q-value	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$
$s_1$	$-1 + \gamma v(s_1)$	$-1 + \gamma v(s_2)$	$0 + \gamma v(s_3)$	$-1 + \gamma v(s_1)$	$0 + \gamma v(s_1)$
$s_2$	$-1 + \gamma v(s_2)$	$-1 + \gamma v(s_2)$	$1 + \gamma v(s_4)$	$0 + \gamma v(s_1)$	$-1 + \gamma v(s_2)$
$s_3$	$0 + \gamma v(s_1)$	$1 + \gamma v(s_4)$	$-1 + \gamma v(s_3)$	$-1 + \gamma v(s_3)$	$0 + \gamma v(s_3)$
$s_4$	$-1 + \gamma v(s_2)$	$-1 + \gamma v(s_4)$	$-1 + \gamma v(s_4)$	$0 + \gamma v(s_3)$	$1 + \gamma v(s_4)$

# Value iteration algorithm - Example

- $k = 0$ : let  $v_0(s_1) = v_0(s_2) = v_0(s_3) = v_0(s_4) = 0$

q-value	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$
$s_1$	-1	-1	0	-1	0
$s_2$	-1	-1	1	0	-1
$s_3$	0	1	-1	-1	0
$s_4$	-1	-1	-1	0	1

Step 1: Policy update:

$$\pi_1(a_5|s_1) = 1, \pi_1(a_3|s_2) = 1, \pi_1(a_2|s_3) = 1, \pi_1(a_5|s_4) = 1$$

This policy is visualized in Figure (b).

Step 2: Value update:

$$v_1(s_1) = 0, v_1(s_2) = 1, v_1(s_3) = 1, v_1(s_4) = 1.$$

# Value iteration algorithm - Example

- $k = 1$ : since  $v_1(s_1) = 0, v_1(s_2) = 1, v_1(s_3) = 1, v_1(s_4) = 1$ , we have

q-table	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$
$s_1$	$-1 + \gamma 0$	$-1 + \gamma 1$	$0 + \gamma 1$	$-1 + \gamma 0$	$0 + \gamma 0$
$s_2$	$-1 + \gamma 1$	$-1 + \gamma 1$	$1 + \gamma 1$	$0 + \gamma 0$	$-1 + \gamma 1$
$s_3$	$0 + \gamma 0$	$1 + \gamma 1$	$-1 + \gamma 1$	$-1 + \gamma 1$	$0 + \gamma 1$
$s_4$	$-1 + \gamma 1$	$-1 + \gamma 1$	$-1 + \gamma 1$	$0 + \gamma 1$	$1 + \gamma 1$

Step 1: Policy update:

$$\pi_2(a_3|s_1) = 1, \pi_2(a_3|s_2) = 1, \pi_2(a_2|s_3) = 1, \pi_2(a_5|s_4) = 1.$$

Step 2: Value update:

$$v_2(s_1) = \gamma 1, v_2(s_2) = 1 + \gamma 1, v_2(s_3) = 1 + \gamma 1, v_2(s_4) = 1 + \gamma 1.$$

This policy is visualized in Figure (c).

The policy is already optimal!!

- $k = 2, 3, \dots$ . Stop when  $\|v_k - v_{k+1}\|$  is smaller than a predefined threshold.

# Outline

- 1** Value iteration algorithm
- 2** Policy iteration algorithm
- 3** Truncated policy iteration algorithm

# Policy iteration algorithm

▷ Algorithm description:

Given a random initial policy  $\pi_0$ ,

- Step 1: policy evaluation (PE)

This step is to calculate the state value of  $\pi_k$ :

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

Note that  $v_{\pi_k}$  is a state value function.

- Step 2: policy improvement (PI)

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

The maximization is componentwise!

# Policy iteration algorithm

- ▷ The algorithm leads to a sequence

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

PE=policy evaluation, PI=policy improvement

- ▷ Questions:

- Q1: In the policy evaluation step, how to get the state value  $v_{\pi_k}$  by solving the Bellman equation?
- Q2: In the policy improvement step, why is the new policy  $\pi_{k+1}$  better than  $\pi_k$ ?
- Q3: Why such an iterative algorithm can finally reach an optimal policy?
- Q4: What is the relationship between this policy iteration algorithm and the previous value iteration algorithm?

# Policy iteration algorithm

- ▷ Q1: In the policy evaluation step, how to get the state value  $v_{\pi_k}$  by solving the Bellman equation?

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

- Closed-form solution:

$$v_{\pi_k} = (I - \gamma P_{\pi_k})^{-1} r_{\pi_k}$$

- Iterative solution:

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

- ▷ Already studied in the lecture on Bellman equation.
- ▷ Policy iteration is an iterative algorithm with another iterative algorithm embedded in the policy evaluation step!

# Policy iteration algorithm

- ▷ Q2: In the policy improvement step, why is the new policy  $\pi_{k+1}$  better than  $\pi_k$ ?

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  $k$ .

See the proof in the book.

# Policy iteration algorithm

- ▷ Q3: Why can such an iterative algorithm finally reach an optimal policy?

Since every iteration would improve the policy, we know

$$v_{\pi_0} \leq v_{\pi_1} \leq v_{\pi_2} \leq \cdots \leq v_{\pi_k} \leq \cdots \leq v^*.$$

As a result,  $v_{\pi_k}$  keeps increasing and will converge. Still need to prove it converges to  $v^*$ .

## 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.*

# Policy iteration algorithm

- ▷ **Q4: What is the relationship between policy iteration and value iteration?**

Related to the answer to Q3 and will be explained in detail later.

# Policy iteration algorithm - Elementwise form

## Step 1: Policy evaluation

- ▷ Matrix-vector form:  $v_{\pi_k}^{(j+1)} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}^{(j)}, \quad j = 0, 1, 2, \dots$
- ▷ 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  $j \rightarrow \infty$  or  $j$  is sufficiently large or  $\|v_{\pi_k}^{(j+1)} - v_{\pi_k}^{(j)}\|$  is sufficiently small.

# Policy iteration algorithm - Elementwise form

## Step 2: Policy improvement

- ▷ Matrix-vector form:  $\pi_{k+1} = \arg \max_{\pi} (r_{\pi} + \gamma P_{\pi} \mathbf{v}_{\pi_k})$
- ▷ Elementwise form

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

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

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

Then, the greedy policy is

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

# Policy iteration algorithm - Implementation

## 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  $\pi_0$ .

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

While the policy has not converged, for the  $k$ th iteration, do

*Policy evaluation:*

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

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

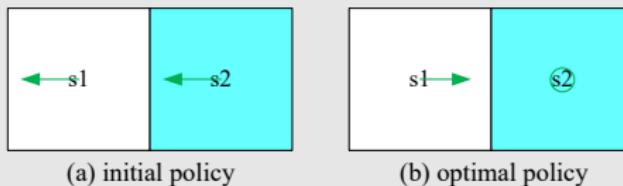
For every action  $a \in \mathcal{A}(s)$ , do

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

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

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

# Policy iteration algorithm - Simple example



- ▷ The reward setting is  $r_{\text{boundary}} = -1$  and  $r_{\text{target}} = 1$ . The discount rate is  $\gamma = 0.9$ .
- ▷ Actions:  $a_\ell, a_0, a_r$  represent go left, stay unchanged, and go right.
- ▷ Aim: use policy iteration to find out the optimal policy.

# Policy iteration algorithm - Simple example

▷ Iteration  $k = 0$ : Step 1: policy evaluation

$\pi_0$  is selected as the policy in Figure (a). The Bellman equation is

$$v_{\pi_0}(s_1) = -1 + \gamma v_{\pi_0}(s_1),$$

$$v_{\pi_0}(s_2) = 0 + \gamma v_{\pi_0}(s_1).$$

- Solve the equations directly:

$$v_{\pi_0}(s_1) = -10, \quad v_{\pi_0}(s_2) = -9.$$

- Solve the equations iteratively. Select the initial guess as

$$v_{\pi_0}^{(0)}(s_1) = v_{\pi_0}^{(0)}(s_2) = 0:$$

$$\begin{cases} v_{\pi_0}^{(1)}(s_1) = -1 + \gamma v_{\pi_0}^{(0)}(s_1) = -1, \\ v_{\pi_0}^{(1)}(s_2) = 0 + \gamma v_{\pi_0}^{(0)}(s_1) = 0, \end{cases}$$

$$\begin{cases} v_{\pi_0}^{(2)}(s_1) = -1 + \gamma v_{\pi_0}^{(1)}(s_1) = -1.9, \\ v_{\pi_0}^{(2)}(s_2) = 0 + \gamma v_{\pi_0}^{(1)}(s_1) = -0.9, \end{cases}$$

$$\begin{cases} v_{\pi_0}^{(3)}(s_1) = -1 + \gamma v_{\pi_0}^{(2)}(s_1) = -2.71, \\ v_{\pi_0}^{(3)}(s_2) = 0 + \gamma v_{\pi_0}^{(2)}(s_1) = -1.71, \end{cases}$$

...

# Policy iteration algorithm - Simple example

▷ Iteration  $k = 0$ : Step 2: policy improvement

The expression of  $q_{\pi_k}(s, a)$ :

$q_{\pi_k}(s, a)$	$a_\ell$	$a_0$	$a_r$
$s_1$	$-1 + \gamma v_{\pi_k}(s_1)$	$0 + \gamma v_{\pi_k}(s_1)$	$1 + \gamma v_{\pi_k}(s_2)$
$s_2$	$0 + \gamma v_{\pi_k}(s_1)$	$1 + \gamma v_{\pi_k}(s_2)$	$-1 + \gamma v_{\pi_k}(s_2)$

Substituting  $v_{\pi_0}(s_1) = -10$ ,  $v_{\pi_0}(s_2) = -9$  and  $\gamma = 0.9$  gives

$q_{\pi_0}(s, a)$	$a_\ell$	$a_0$	$a_r$
$s_1$	-10	-9	-7.1
$s_2$	-9	-7.1	-9.1

By seeking the greatest value of  $q_{\pi_0}$ , the improved policy is:

$$\pi_1(a_r|s_1) = 1, \quad \pi_1(a_0|s_2) = 1.$$

This policy is optimal after one iteration! In your programming, should continue until the stopping criterion is satisfied.

## Policy iteration algorithm - Simple example

Excise! Set the left cell as the target area.

Now you know another powerful algorithm searching for optimal policies!

Now let's apply it and see what we can find.

# Policy iteration algorithm - Complicated example

- ▷ Setting:  $r_{\text{boundary}} = -1$ ,  $r_{\text{forbidden}} = -10$ ,  $r_{\text{target}} = 1$ ,  $\gamma = 0.9$ .
- ▷ Let's check out the intermediate policies and state values.

	1	2	3	4	5
1	○	○	○	○	○
2	○	○	○	○	○
3	○	○	○	○	○
4	○	○	○	○	○
5	○	○	○	○	○

	1	2	3	4	5
1	0.0	0.0	0.0	0.0	0.0
2	0.0	-100.0	-100.0	0.0	0.0
3	0.0	0.0	-100.0	0.0	0.0
4	0.0	-100.0	10.0	-100.0	0.0
5	0.0	-100.0	0.0	0.0	0.0

$\pi_0$  and  $v_{\pi_0}$

	1	2	3	4	5
1	↓	→	→	→	→
2	↓	↓	↓	↓	↓
3	↓	→	↓	→	→
4	↓	→	→	→	↓
5	↓	→	↑	→	→

	1	2	3	4	5
1	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	10.0	0.0	0.0
4	0.0	10.0	10.0	10.0	0.0
5	0.0	9.0	10.0	0.0	0.0

$\pi_1$  and  $v_{\pi_1}$

	1	2	3	4	5
1	○	○	→	○	○
2	○	↑	↓	↓	←
3	↑	→	↓	→	→
4	↑	→	○	→	○
5	↑	→	↑	→	↓

	1	2	3	4	5
1	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	10.0	0.0	0.0
4	0.0	10.0	10.0	10.0	0.0
5	0.0	9.0	10.0	9.0	0.0

$\pi_2$  and  $v_{\pi_2}$

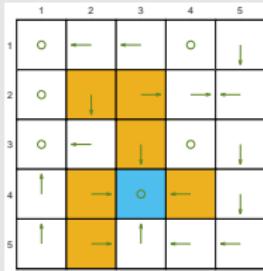
	1	2	3	4	5
1	○	→	→	→	○
2	↑	↑	↑	↑	↓
3	○	○	↓	→	↑
4	↓	→	→	→	↓
5	○	→	↑	→	→

	1	2	3	4	5
1	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	10.0	0.0	0.0
4	0.0	10.0	10.0	10.0	7.3
5	0.0	9.0	10.0	9.0	8.1

$\pi_3$  and  $v_{\pi_3}$

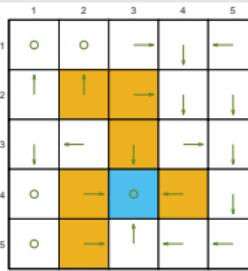
# Policy iteration algorithm - Complicated example

▷ Interesting pattern of the policies and state values



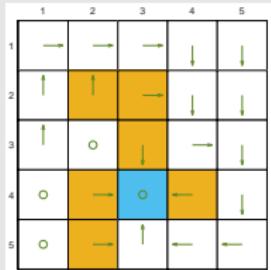
$\pi_4$  and  $v_{\pi_4}$

⋮

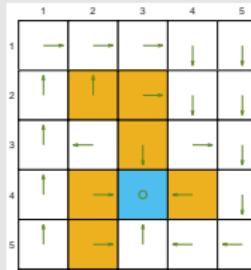


$\pi_5$  and  $v_{\pi_5}$

⋮



$\pi_9$  and  $v_{\pi_9}$



$\pi_{10}$  and  $v_{\pi_{10}}$

# Outline

- 1 Value iteration algorithm
- 2 Policy iteration algorithm
- 3 Truncated policy iteration algorithm

# Compare value iteration and policy iteration

Policy iteration: start from  $\pi_0$

- 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):

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

# Compare value iteration and policy iteration

- ▷ The two algorithms are very similar:

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:  $u_0 \xrightarrow{PU} \pi'_1 \xrightarrow{VU} u_1 \xrightarrow{PU} \pi'_2 \xrightarrow{VU} u_2 \xrightarrow{PU} \dots$

PE=policy evaluation. PI=policy improvement.

PU=policy update. VU=value update.

# Compare value iteration and policy iteration

▷ Let's compare the steps carefully:

	Policy iteration algorithm	Value iteration algorithm	Comments
1) Policy:	$\pi_0$	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:	$\pi_2 = \arg \max_{\pi} (r_{\pi} + \gamma P_{\pi} v_{\pi_1})$	$\pi'_2 = \arg \max_{\pi} (r_{\pi} + \gamma P_{\pi} v_1)$	
:	:	:	:

- They start from the same initial condition.
- The first three steps are the same.
- The fourth step becomes different:
  - In policy iteration, solving  $v_{\pi_1} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}$  requires an iterative algorithm (an infinite number of iterations)
  - In value iteration,  $v_1 = r_{\pi_1} + \gamma P_{\pi_1} v_0$  is a one-step iteration

# Compare value iteration and policy iteration

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

$$v_{\pi_1}^{(0)} = v_0$$

$$\text{value iteration} \leftarrow v_1 \leftarrow v_{\pi_1}^{(1)} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(0)}$$

$$v_{\pi_1}^{(2)} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(1)}$$

⋮

$$\text{truncated policy iteration} \leftarrow \bar{v}_1 \leftarrow v_{\pi_1}^{(j)} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(j-1)}$$

⋮

$$\text{policy iteration} \leftarrow v_{\pi_1} \leftarrow v_{\pi_1}^{(\infty)} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(\infty)}$$

- The value iteration algorithm computes *once*.
- The policy iteration algorithm computes *an infinite number of iterations*.
- The **truncated policy iteration algorithm** computes *a finite number of iterations* (say  $j$ ). The rest iterations from  $j$  to  $\infty$  are truncated.

# Truncated policy iteration - 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  $\pi_0$ .

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

While the policy has not converged, for the  $k$ th iteration, do

*Policy evaluation:*

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

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

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

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

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

# Truncated policy iteration - Convergence

- ▷ Will the truncation undermine convergence?

## Proposition (Value Improvement)

Consider the iterative algorithm for solving the policy evaluation step:

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

If the initial guess is selected as  $v_{\pi_k}^{(0)} = v_{\pi_{k-1}}$ , it holds that

$$v_{\pi_k}^{(j+1)} \geq v_{\pi_k}^{(j)}$$

for every  $j = 0, 1, 2, \dots$

For the proof, see the book.

# Truncated policy iteration - Convergence

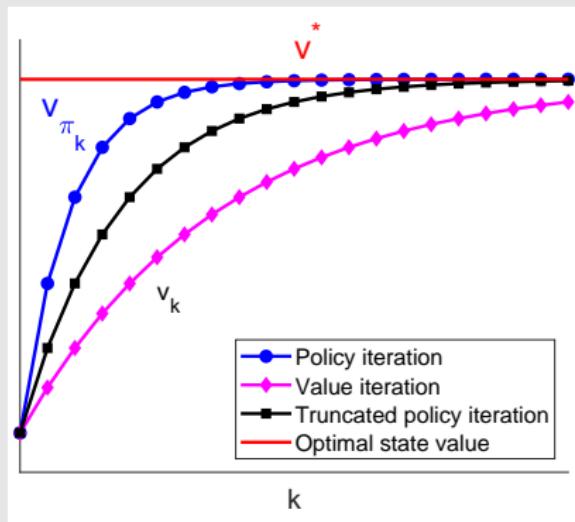


Figure: Illustration of the relationship among value iteration, policy iteration, and truncated policy iteration.

The convergence proof of PI is based on that of VI. Since VI converges, we know PI converges.

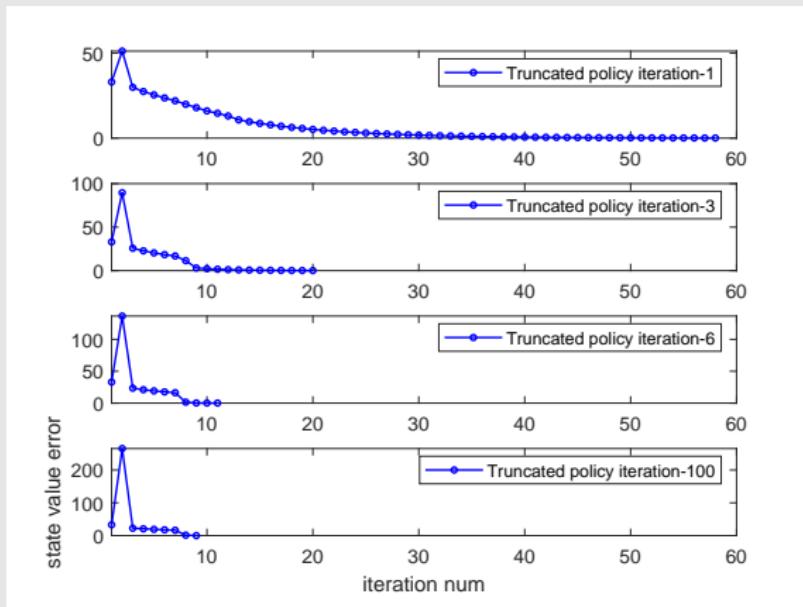
# Truncated policy iteration - Example

▷ Setup: The same as the previous example. Below is the initial policy.

	1	2	3	4	5
1	○	○	○	○	○
2	○	○	○	○	○
3	○	○	○	○	○
4	○	○	○	○	○
5	○	○	○	○	○

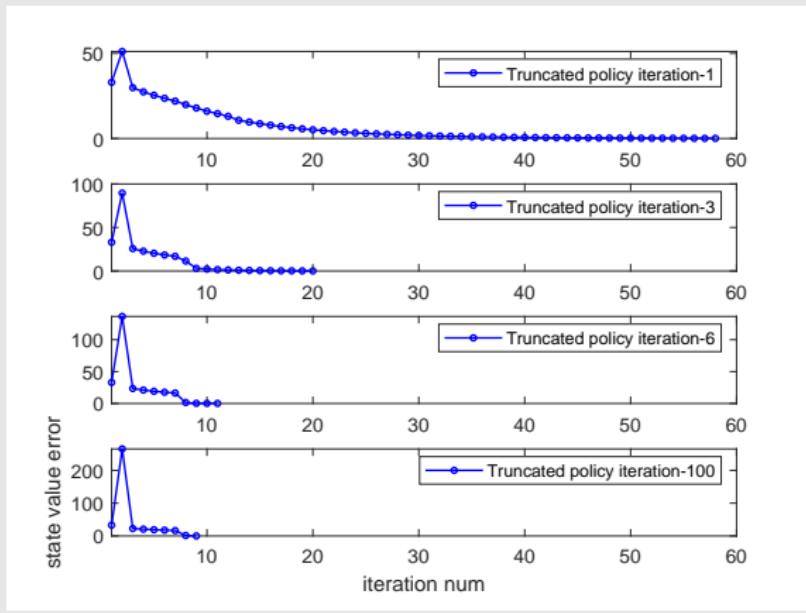
▷ Define  $\|v_k - v^*\|$  as the state value error at time  $k$ . The stop criterion is  $\|v_k - v^*\| < 0.01$ .

# Truncated policy iteration - Example



- ▷ “Truncated policy iteration- $x$ ” where  $x = 1, 3, 6, 100$  refers to a truncated policy iteration algorithm where the policy evaluation step runs  $x$  iterations.

# Truncated policy iteration - Example



- ▷ The greater the value of  $x$  is, the faster the value estimate converges.
- ▷ However, the benefit of increasing  $x$  drops quickly when  $x$  is large.
- ▷ In practice, run a few number of iterations in the policy evaluation step.

# Summary

- ▷ Value iteration: it is the iterative algorithm solving the Bellman optimality equation: given an initial value  $v_0$ ,

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

⇓

$$\begin{cases} \text{Policy update: } \pi_{k+1} = \arg \max_{\pi} (r_{\pi} + \gamma P_{\pi} v_k) \\ \text{Value update: } v_{k+1} = r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_k \end{cases}$$

- ▷ Policy iteration: given an initial policy  $\pi_0$ ,

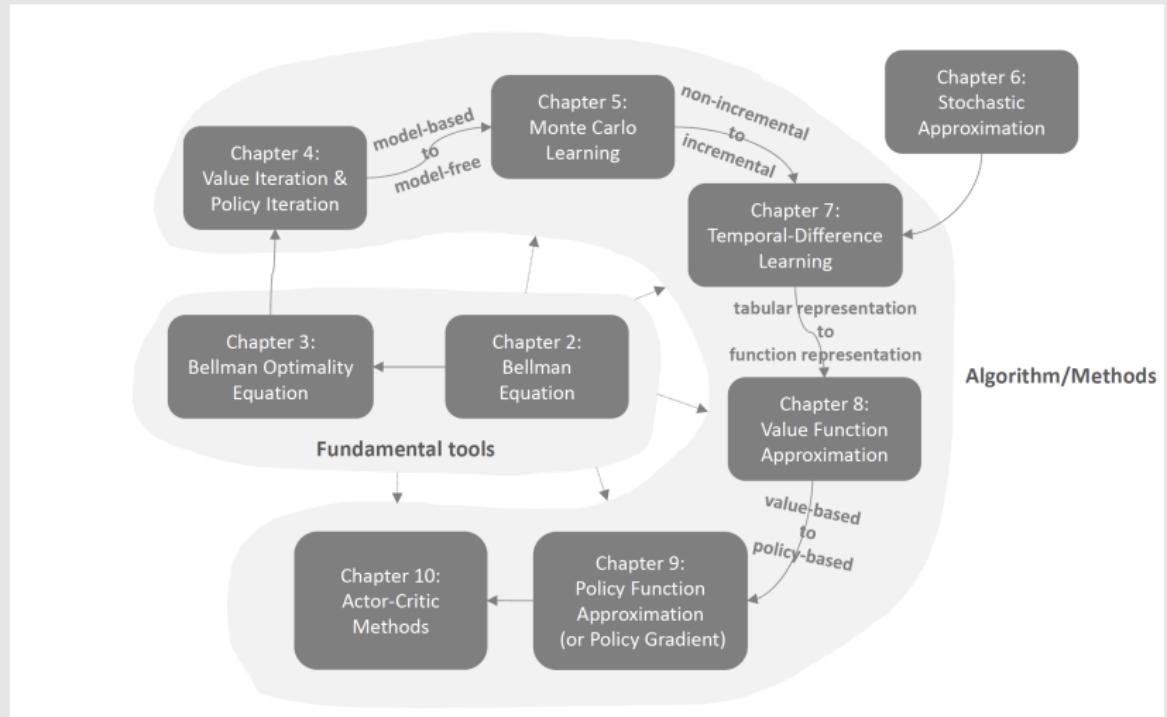
$$\begin{cases} \text{Policy evaluation: } v_{\pi_k} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k} \\ \text{Policy improvement: } \pi_{k+1} = \arg \max_{\pi} (r_{\pi} + \gamma P_{\pi} v_{\pi_k}) \end{cases}$$

- ▷ Truncated policy iteration

# Lecture 5: Monte Carlo Learning

Shiyu Zhao

# Outline



# Outline

- 1 Motivating example
- 2 The simplest MC-based RL algorithm
  - Algorithm: MC Basic
- 3 Use data more efficiently
  - Algorithm: MC Exploring Starts
- 4 MC without exploring starts
  - Algorithm: MC  $\varepsilon$ -Greedy

# Outline

- 1 Motivating example
- 2 The simplest MC-based RL algorithm
  - Algorithm: MC Basic
- 3 Use data more efficiently
  - Algorithm: MC Exploring Starts
- 4 MC without exploring starts
  - Algorithm: MC  $\epsilon$ -Greedy

# Motivating example: Monte Carlo estimation

- ▷ How can we estimate something **without models?**
  - The simplest idea: *Monte Carlo estimation.*
- ▷ **Example: Flip a coin**

The result (either head or tail) is denoted as a random variable  $X$

- if the result is head, then  $X = +1$
- if the result is tail, then  $X = -1$

The *aim* is to compute  $\mathbb{E}[X]$ .

# Motivating example: Monte Carlo estimation

## ▷ Method 1: Model-based

- Suppose the probabilistic model is known as

$$p(X = 1) = 0.5, \quad p(X = -1) = 0.5$$

Then by definition

$$\mathbb{E}[X] = \sum_x xp(x) = 1 \times 0.5 + (-1) \times 0.5 = 0$$

- Problem: it may be impossible to know the precise distribution!!

# Motivating example: Monte Carlo estimation

## ▷ Method 2: Model-free

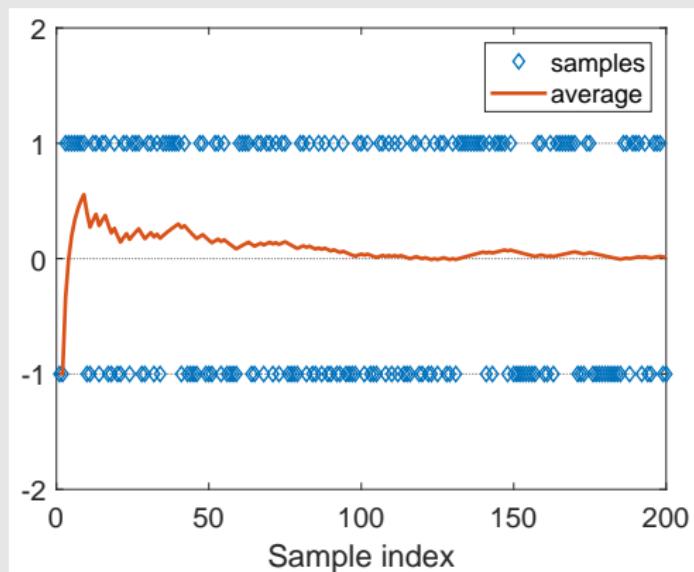
- *Idea:* Flip the coin many times, and then calculate the average of the outcomes.
- Suppose we get a sample sequence:  $\{x_1, x_2, \dots, x_N\}$ .  
Then, the mean can be approximated as

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

This is the idea of Monte Carlo estimation!

# Motivating example: Monte Carlo estimation

- ▷ **Question:** Is the Monte Carlo estimation accurate?
  - When  $N$  is *small*, the approximation is inaccurate.
  - As  $N$  *increases*, the approximation becomes more and more accurate.



# Motivating example: Monte Carlo estimation

## Law of Large Numbers

For a random variable  $X$ . Suppose  $\{x_j\}_{j=1}^N$  are some iid samples. Let  $\bar{x} = \frac{1}{N} \sum_{j=1}^N x_j$  be the average of the samples. Then,

$$\mathbb{E}[\bar{x}] = \mathbb{E}[X],$$

$$\text{Var}[\bar{x}] = \frac{1}{N} \text{Var}[X].$$

As a result,  $\bar{x}$  is an unbiased estimate of  $\mathbb{E}[X]$  and its variance decreases to zero as  $N$  increases to infinity.

- ▷ The samples must be iid (independent and identically distributed)
- ▷ For the proof, see the book.

# Motivating example: Monte Carlo estimation

## ▷ Summary:

- Monte Carlo estimation refers to a broad class of techniques that rely on repeated random sampling to solve approximation problems.
- Why we care about Monte Carlo estimation? Because it does not require the model!
- Why we care about mean estimation? Because state value and action value are defined as expectations of random variables!

# Outline

- 1 Motivating example
- 2 The simplest MC-based RL algorithm
  - Algorithm: MC Basic
- 3 Use data more efficiently
  - Algorithm: MC Exploring Starts
- 4 MC without exploring starts
  - Algorithm: MC  $\epsilon$ -Greedy

# Convert policy iteration to be model-free

The key to understand the algorithm is to understand [how to convert the policy iteration algorithm to be model-free.](#)

- Should understand policy iteration well.
- Should understand the idea of Monte Carlo mean estimation.

# Convert policy iteration to be model-free

Policy iteration has two steps in each iteration:

$$\left\{ \begin{array}{l} \textbf{Policy evaluation: } v_{\pi_k} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k} \\ \textbf{Policy improvement: } \pi_{k+1} = \arg \max_{\pi} (r_{\pi} + \gamma P_{\pi} v_{\pi_k}) \end{array} \right.$$

The elementwise form of the **policy improvement step** is:

$$\begin{aligned} \pi_{k+1}(s) &= \arg \max_{\pi} \sum_a \pi(a|s) \left[ \sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v_{\pi_k}(s') \right] \\ &= \arg \max_{\pi} \sum_a \pi(a|s) q_{\pi_k}(s, a), \quad s \in \mathcal{S} \end{aligned}$$

The key is  $q_{\pi_k}(s, a)$ !

# Convert policy iteration to be model-free

Two expressions of action value:

- **Expression 1 requires the model:**

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

- **Expression 2 does not require the model:**

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

Idea to achieve model-free RL: We can use expression 2 to calculate  $q_{\pi_k}(s, a)$  based on *data (samples or experiences)*!

# Convert policy iteration to be model-free

## The procedure of Monte Carlo estimation of action values:

- Starting from  $(s, a)$ , following policy  $\pi_k$ , 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 model is unavailable, we can use data.

# The MC Basic algorithm

▷ Description of the algorithm:

Given an initial policy  $\pi_0$ , there are two steps at the  $k$ th 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

$$\pi_{k+1}(s) = \arg \max_{\pi} \sum_a \pi(a|s) q_{\pi_k}(s, a) \text{ for all } s \in \mathcal{S}. \text{ The greedy optimal policy is } \pi_{k+1}(a_k^*|s) = 1 \text{ where } a_k^* = \arg \max_a q_{\pi_k}(s, a).$$

**Exactly the same as the policy iteration algorithm, except**

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

# The MC Basic algorithm

▷ Description of the algorithm:

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

**Initialization:** Initial guess  $\pi_0$ .

**Aim:** Search for an optimal policy.

While the value estimate has not converged, for the  $k$ th 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  $\pi_k$

*MC-based policy evaluation step:*

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

*Policy improvement step:*

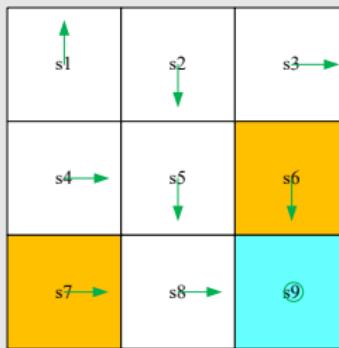
$$a_k^*(s) = \arg \max_a q_{\pi_k}(s, a)$$

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

# The MC Basic algorithm

- MC Basic is a variant of the policy iteration algorithm.
- The model-free algorithms are built up based on model-based ones. It is, therefore, necessary to understand model-based algorithms first before studying model-free algorithms.
- MC Basic is useful to reveal the core idea of MC-based model-free RL, but not practical due to low efficiency.
- Why does MC Basic estimate *action values* instead of state values? That is because state values cannot be used to improve policies directly. When models are not available, we should directly estimate action values.
- Since policy iteration is convergent, the convergence of MC Basic is also guaranteed to be convergent given sufficient episodes.

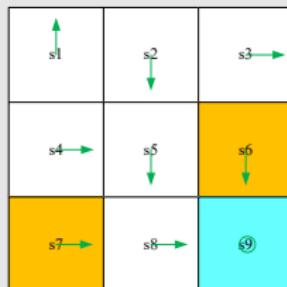
# Illustrative example 1: step by step



Task:

- An initial policy is shown in the figure.
- Use MC Basic to find the optimal policy.
- $r_{\text{boundary}} = -1$ ,  $r_{\text{forbidden}} = -1$ ,  $r_{\text{target}} = 1$ ,  $\gamma = 0.9$ .

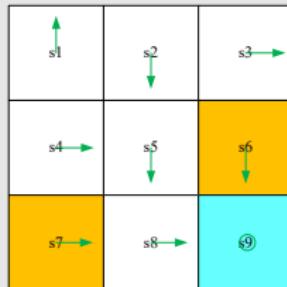
# Illustrative example 1: step by step



Outline: given the current policy  $\pi_k$

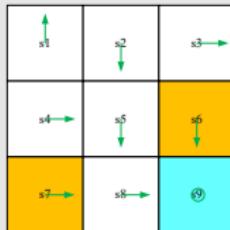
- Step 1 - policy evaluation: calculate  $q_{\pi_k}(s, a)$   
How many state-action pairs?  $9 \text{ states} \times 5 \text{ actions} = 45 \text{ state-action pairs!}$
- Step 2 - policy improvement: select the greedy action  
 $a^*(s) = \arg \max_{a_i} q_{\pi_k}(s, a)$

# Illustrative example 1: step by step



- ▷ Due to space limitation, we only show  $q_{\pi_k}(s_1, a)$
- ▷ **Step 1 - policy evaluation:**
  - Since the current policy is deterministic, one episode would be sufficient to get the action value!
  - If the current policy is stochastic, an infinite number of episodes (or at least many) are required!

# Illustrative example 1: step by step



- Starting from  $(s_1, a_1)$ , the episode is  $s_1 \xrightarrow{a_1} s_1 \xrightarrow{a_1} s_1 \xrightarrow{a_1} \dots$  Hence, the action value is

$$q_{\pi_0}(s_1, a_1) = -1 + \gamma(-1) + \gamma^2(-1) + \dots$$

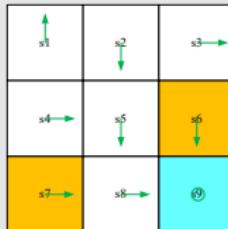
- Starting from  $(s_1, a_2)$ , the episode is  $s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_3} \dots$  Hence, the action value is

$$q_{\pi_0}(s_1, a_2) = 0 + \gamma 0 + \gamma^2 0 + \gamma^3(1) + \gamma^4(1) + \dots$$

- Starting from  $(s_1, a_3)$ , the episode is  $s_1 \xrightarrow{a_3} s_4 \xrightarrow{a_2} s_5 \xrightarrow{a_3} \dots$  Hence, the action value is

$$q_{\pi_0}(s_1, a_3) = 0 + \gamma 0 + \gamma^2 0 + \gamma^3(1) + \gamma^4(1) + \dots$$

# Illustrative example 1: step by step



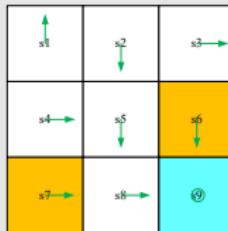
- Starting from  $(s_1, a_4)$ , the episode is  $s_1 \xrightarrow{a_4} s_1 \xrightarrow{a_1} s_1 \xrightarrow{a_1} \dots$ . Hence, the action value is

$$q_{\pi_0}(s_1, a_4) = -1 + \gamma(-1) + \gamma^2(-1) + \dots$$

- Starting from  $(s_1, a_5)$ , the episode is  $s_1 \xrightarrow{a_5} s_1 \xrightarrow{a_1} s_1 \xrightarrow{a_1} \dots$ . Hence, the action value is

$$q_{\pi_0}(s_1, a_5) = 0 + \gamma(-1) + \gamma^2(-1) + \dots$$

# Illustrative example 1: step by step



## ▷ Step 2 - policy improvement:

- By observing the action values, we see that

$$q_{\pi_0}(s_1, a_2) = q_{\pi_0}(s_1, a_3)$$

are the *maximum*.

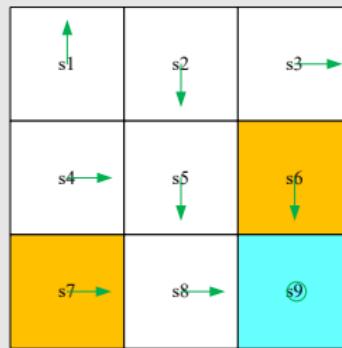
- As a result, the policy can be improved as

$$\pi_1(a_2|s_1) = 1 \text{ or } \pi_1(a_3|s_1) = 1.$$

In either way, the new policy for  $s_1$  becomes optimal.

One iteration is sufficient for this simple example!

# Illustrative example 1: step by step



Exercise: now update the policy for  $s_3$  using MC Basic!

## Illustrative example 2: Episode length

Examine [the impact of episode length](#):

- We need sample episodes, but the length of an episode cannot be infinitely long.
- How long should be the episodes?

Example setup:

- 5-by-5 grid world
- Reward setting:  $r_{\text{boundary}} = -1$ ,  $r_{\text{forbidden}} = -10$ ,  $r_{\text{target}} = 1$ ,  $\gamma = 0.9$

# Illustrative example 2: Episode length

▷ Use MC Basic to search optimal policies with different episode lengths.

Episode length=1						
1	2	3	4	5		
1	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	1.0	0.0	0.0	
4	0.0	1.0	1.0	1.0	0.0	
5	0.0	0.0	1.0	0.0	0.0	

Episode length=1						
1	2	3	4	5		
1	○	→	→	↓	○	
2	○	↓	↑	↓	↓	
3	↑	○	↓	↑	↓	
4	↑	→	○	→	↓	
5	○	→	↓	→	○	

Episode length=2						
1	2	3	4	5		
1	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	1.9	0.0	0.0	
4	0.0	1.9	1.9	1.9	0.0	
5	0.0	0.9	1.9	0.9	0.0	

Episode length=2						
1	2	3	4	5		
1	→	○	→	↓	○	
2	↓	→	→	↓	↓	
3	→	→	↓	↑	↓	
4	↓	→	○	→	○	
5	○	→	↓	→	○	

Estimated state value and policy with episode length=1

Estimated state value and policy with episode length=2

Episode length=3						
1	2	3	4	5		
1	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	2.7	0.0	0.0	
4	0.0	2.7	2.7	2.7	0.0	
5	0.0	1.7	2.7	1.7	0.8	

Episode length=3						
1	2	3	4	5		
1	↓	→	○	↓	→	
2	↓	→	↑	○	○	
3	↓	→	→	→	↓	
4	↑	→	○	→	↑	
5	○	→	↑	↓	→	

Episode length=4						
1	2	3	4	5		
1	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	3.4	0.0	0.0	
4	0.0	3.4	3.4	3.4	0.7	
5	0.0	2.4	3.4	2.4	1.5	

Episode length=4						
1	2	3	4	5		
1	→	→	→	↓		
2	↓	→	→	○	↑	
3	↑	→	→	↓	↑	
4	↓	→	○	→	↓	
5	○	→	↓	→	→	

Estimated state value and policy with episode length=3

Estimated state value and policy with episode length=4

## Illustrative example 2: Episode length

	Episode length=14				
1	1.2	1.6	2.0	2.5	3.0
2	0.9	1.2	2.5	3.0	3.6
3	0.5	0.3	7.7	3.6	4.3
4	0.3	7.7	7.7	7.7	5.0
5	0.0	6.7	7.7	6.7	5.8

	Episode length=14				
1	—	—	—	—	—
2	↑	—	—	—	↓
3	↑	—	—	—	↓
4	↑	—	○	—	↓
5	↑	—	—	—	—

	Episode length=15				
1	1.4	1.8	2.2	2.7	3.3
2	1.1	1.4	2.7	3.3	3.8
3	0.8	0.5	7.9	3.8	4.5
4	0.5	7.9	7.9	7.9	5.2
5	0.2	6.9	7.9	6.9	6.0

	Episode length=15				
1	—	—	—	—	—
2	—	—	—	—	—
3	—	—	—	—	—
4	—	—	○	—	—
5	—	—	—	—	—

Estimated state value and policy with episode length=14

Estimated state value and policy with episode length=15

	Episode length=30				
1	3.1	3.5	3.9	4.4	4.9
2	2.7	3.1	4.4	4.9	5.5
3	2.4	2.1	9.6	5.5	6.1
4	2.1	9.6	9.6	9.6	6.9
5	1.9	8.6	9.6	8.6	7.7

	Episode length=30				
1	—	—	—	—	—
2	↑	—	—	—	↓
3	↑	—	—	—	↓
4	↑	—	○	—	↓
5	↑	—	—	—	—

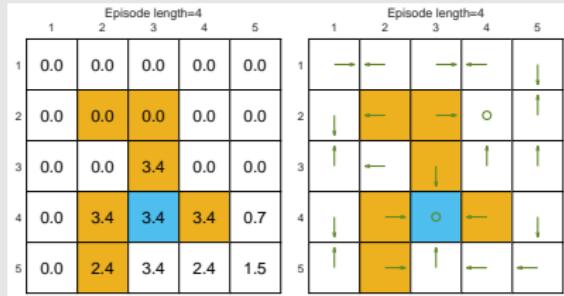
	Episode length=100				
1	3.5	3.9	4.3	4.8	5.3
2	3.1	3.5	4.8	5.3	5.9
3	2.8	2.5	10.0	5.9	6.6
4	2.5	10.0	10.0	10.0	7.3
5	2.3	9.0	10.0	9.0	8.1

	Episode length=100				
1	—	—	—	—	—
2	↑	—	—	—	—
3	↑	—	—	—	—
4	↑	—	○	—	—
5	↑	—	—	—	—

Estimated state value and policy with episode length=30

Estimated state value and policy with episode length=100

## Illustrative example 2: Episode length



### ▷ Findings:

- When the episode length is short, only the states that are close to the target have nonzero state values.
- As the episode length increases, the states that are closer to the target have nonzero values earlier than those farther away.
- The episode length should be sufficiently long.
- The episode length does not have to be infinitely long.

# Outline

- 1 Motivating example
- 2 The simplest MC-based RL algorithm
  - Algorithm: MC Basic
- 3 Use data more efficiently
  - Algorithm: MC Exploring Starts
- 4 MC without exploring starts
  - Algorithm: MC  $\epsilon$ -Greedy

# Use data more efficiently

The MC Basic algorithm:

- **Advantage:** reveal the core idea clearly!
- **Disadvantage:** too simple to be practical.

However, MC Basic can be extended to be more efficient.

# Use data more efficiently

- ▷ Consider a grid-world example, following a policy  $\pi$ , we can get an episode such as

$$s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_4} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \dots$$

- ▷ **Visit:** every time a state-action pair appears in the episode, it is called a *visit* of that state-action pair.
- ▷ Methods to use the data: **Initial-visit method**
  - Just calculate the return and approximate  $q_\pi(s_1, a_2)$ .
  - This is what the MC Basic algorithm does.
  - Disadvantage: Not fully utilize the data.

# Use data more efficiently

▷ The episode also visits other state-action pairs.

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

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

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

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

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

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

## Data-efficient methods:

- first-visit method
- every-visit method

# Update value estimate more efficiently

- ▷ Another aspect in MC-based RL is **when to update the policy**. There are two methods.
  - **The first method** is, in the policy evaluation step, to collect all the episodes starting from a state-action pair and then use the average return to approximate the action value.
    - This is the one adopted by the MC Basic algorithm.
    - The problem of this method is that the agent has to wait until all episodes have been collected.
  - **The second method** uses the return of a single episode to approximate the action value.
    - In this way, we can improve the policy episode-by-episode.

## Update value estimate more efficiently

- ▷ Will the second method cause problems?
  - One may say that the return of a single episode cannot accurately approximate the corresponding action value.
  - In fact, we have done that in the truncated policy iteration algorithm introduced in the last chapter!
- ▷ 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.

# MC Exploring Starts

- ▷ If we use data and update estimate more efficiently, we get a new algorithm called MC Exploring Starts:

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

**Initialization:** Initial guess  $\pi_0$ .

**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  $T$ :  $s_0, a_0, r_1, \dots, s_{T-1}, a_{T-1}, r_T$ .

*Policy evaluation and policy improvement:*

Initialization:  $g \leftarrow 0$

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

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

*Use the first-visit method:*

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

$$\text{Returns}(s_t, a_t) \leftarrow \text{Returns}(s_t, a_t) + g$$

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

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

# MC Exploring Starts

- ▷ **What is exploring starts?**
- Exploring starts means we need to generate sufficiently many episodes starting from *every* state-action pair.
- Both MC Basic and MC Exploring Starts need this assumption.

# MC Exploring Starts

## ▷ Why do we need to consider exploring starts?

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

On the contrary, if an action is not explored, this action may happen to be the optimal one and hence be missed.

- 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.

Therefore, there is a gap between theory and practice.

Can we remove the requirement of exploring starts? We next show that we can do that by using soft policies.

# Outline

- 1 Motivating example
- 2 The simplest MC-based RL algorithm
  - Algorithm: MC Basic
- 3 Use data more efficiently
  - Algorithm: MC Exploring Starts
- 4 MC without exploring starts
  - Algorithm: MC  $\epsilon$ -Greedy

# Soft policies

- ▷ A policy is called *soft* if the probability to take any action is positive.
- ▷ Why introduce soft policies?
  - With a soft policy, a few episodes that are sufficiently long can visit every state-action pair for sufficiently many times.
  - Then, we do not need to have a large number of episodes starting from every state-action pair. Hence, the requirement of exploring starts can thus be removed.

# $\varepsilon$ -greedy policies

▷ What soft policies will we use? Answer:  $\varepsilon$ -greedy policies

- **What is an  $\varepsilon$ -greedy policy?**

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

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  $\varepsilon$ -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.

# MC $\varepsilon$ -Greedy algorithm

- ▷ How to embed  $\varepsilon$ -greedy into the MC-based RL algorithms?

Originally, the policy improvement step in MC Basic and MC Exploring Starts is to solve

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

where  $\Pi$  denotes the set of all possible policies. The optimal policy here is

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

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

# MC $\varepsilon$ -Greedy algorithm

- ▷ How to embed  $\varepsilon$ -greedy into the MC-based RL algorithms?

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  $\Pi_\varepsilon$  denotes the set of all  $\varepsilon$ -greedy policies with a fixed value of  $\varepsilon$ .

The optimal policy here is

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

- MC  $\varepsilon$ -Greedy is the same as that of MC Exploring Starts except that the former uses  $\varepsilon$ -greedy policies.
- It does not require exploring starts, but still requires to visit all state-action pairs in a different form.

# MC $\varepsilon$ -Greedy algorithm

## Pseudocode: MC $\varepsilon$ -Greedy (a variant of MC Exploring Starts)

**Initialization:** Initial guess  $\pi_0$  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  $T$ :  $s_0, a_0, r_1, \dots, s_{T-1}, a_{T-1}, r_T$ .

*Policy evaluation and policy improvement:*

Initialization:  $g \leftarrow 0$

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

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

*Use the every-visit method:*

$$\text{Returns}(s_t, a_t) \leftarrow \text{Returns}(s_t, a_t) + g$$

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

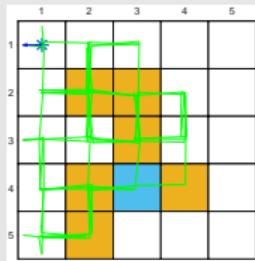
Let  $a^* = \arg \max_a q(s_t, a)$  and

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

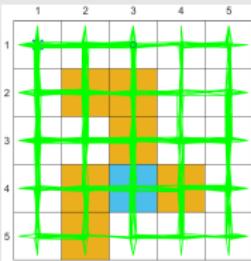
# Exploration ability

- ▷ Can a single episode visit all state-action pairs?

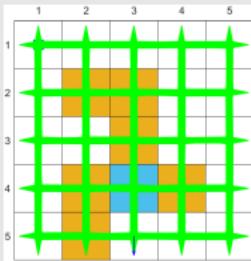
When  $\varepsilon = 1$ , the policy (uniform distribution) has the strongest exploration ability.



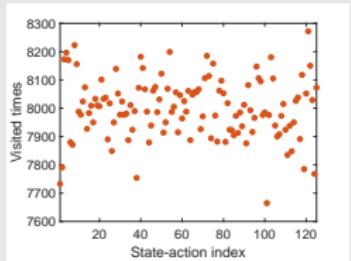
(a) 100 steps



(b) 1000 steps



(c) 10000 steps



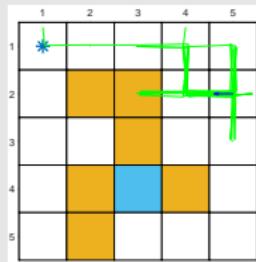
(d)

[Click here to play a video](#) (the video is only on my computer)

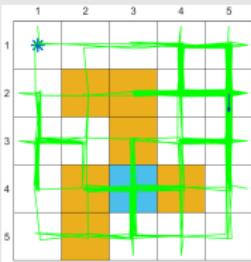
# Exploration ability

- ▷ Can a single episode visit all state-action pairs?

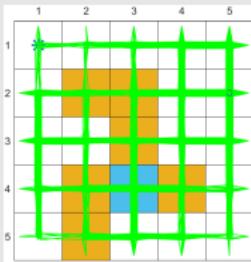
When  $\varepsilon$  is small, the exploration ability of the policy is also small.



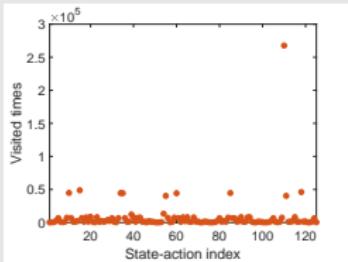
(a) 100 steps



(b) 1000 steps



(c) 10000 steps



(d)

# Estimate based on one episode

- ▷ Run the MC  $\varepsilon$ -Greedy algorithm as follows. In every iteration:
  - In the episode generation step, use the previous policy generates an episode of 1 million steps!
  - In the rest steps, use the single episode to update the policy.
  - Two iterations can lead to the optimal  $\varepsilon$ -greedy policy.

	1	2	3	4	5
1	+	+	+	+	+
2	+	+	+	+	+
3	+	+	+	+	+
4	+	+	+	+	+
5	+	+	+	+	+

(a) Initial policy

	1	2	3	4	5
1	+	+	+	+	+
2	+	+	+	+	+
3	*	+	+	+	+
4	+	+	*	+	+
5	+	+	*	+	+

(b) After the first iteration

	1	2	3	4	5
1	+	+	+	+	+
2	+	+	+	+	+
3	+	+	+	+	+
4	+	+	*	+	+
5	+	+	*	+	+

(c) After the second iteration

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

# Optimality vs exploration

- ▷ Compared to greedy policies,
  - The **advantage** of  $\varepsilon$ -greedy policies is that they have stronger exploration ability so that the exploring starts condition is not required.
  - The **disadvantage** is that  $\varepsilon$ -greedy policies are **not optimal** in general (we can only show that there always exist greedy policies that are optimal).
  - The final policy given by the MC  $\varepsilon$ -Greedy algorithm is only optimal in the set  $\Pi_\varepsilon$  of all  $\varepsilon$ -greedy policies.
  - $\varepsilon$  cannot be too large.
- ▷ Next, we use examples to demonstrate. The setup is  $r_{\text{boundary}} = -1$ ,  $r_{\text{forbidden}} = -10$ ,  $r_{\text{target}} = 1$ ,  $\gamma = 0.9$

# Optimality

- Given an  $\varepsilon$ -greedy policy, what is its state value?

	1	2	3	4	5
1	→	→	→	→	↓
2	↑	↑	↑	→	↓
3	↑	→	↑	→	↓
4	↑	→	↑	↑	↓
5	↑	→	↑	→	↓

	1	2	3	4	5
1	3.5	3.9	4.3	4.8	5.3
2	3.1	3.5	4.8	5.3	5.9
3	2.8	2.5	10.0	5.9	6.6
4	2.5	10.0	10.0	10.0	7.3
5	2.3	9.0	10.0	9.0	8.1

$$\varepsilon = 0$$

	1	2	3	4	5
1	↔	↔	↔	↔	↓
2	↓	↓	↓	↔	↓
3	↓	→	↑	→	↓
4	↓	→	↑	↑	↓
5	↓	→	↑	→	↓

	1	2	3	4	5
1	0.4	0.5	0.9	1.3	1.4
2	0.1	0.0	0.5	1.3	1.7
3	0.1	-0.4	3.4	1.4	1.9
4	-0.1	3.4	3.3	3.7	2.2
5	-0.3	2.6	3.7	3.1	2.7

$$\varepsilon = 0.1$$

	1	2	3	4	5
1	↔	↔	↔	↔	↓
2	↑	↑	↑	↔	↑
3	↓	→	↑	→	↑
4	↑	→	↑	↑	↑
5	↓	→	↑	→	↑

	1	2	3	4	5
1	-2.2	-2.4	-2.1	-1.7	-1.8
2	-2.5	-3.0	-3.3	-2.3	-2.0
3	-2.3	-3.3	-2.5	-2.8	-2.2
4	-2.5	-2.5	-2.8	-2.0	-2.4
5	-2.8	-3.2	-2.1	-2.3	-2.2

$$\varepsilon = 0.2$$

	1	2	3	4	5
1	+	+	+	+	+
2	+	+	+	+	+
3	+	+	+	+	+
4	+	+	+	+	+
5	+	+	+	+	+

	1	2	3	4	5
1	-8.0	-9.0	-8.4	-7.2	-7.8
2	-8.7	-10.8	-12.4	-9.6	-8.9
3	-8.3	-12.3	-15.3	-12.3	-10.5
4	-9.7	-15.5	-17.0	-14.4	-12.2
5	-10.9	-16.7	-15.2	-14.3	-12.4

$$\varepsilon = 0.5$$

- When  $\varepsilon$  increases, the optimality of the policy becomes worse!

- Why is the state value of the target state negative?

# Consistency

- ▷ Find the optimal  $\varepsilon$ -greedy policies and their state values?

	1	2	3	4	5	
1	—	→	→	—	↓	1
2	↑	↓	→	—	↓	2
3	↑	→	↓	—	↓	3
4	↑	→	○	—	↓	4
5	↑	→	↑	—	→	5

	1	2	3	4	5	
1	3.5	3.9	4.3	4.8	5.3	1
2	3.1	3.5	4.8	5.3	5.9	2
3	2.8	2.5	10.0	5.9	6.6	3
4	2.5	10.0	10.0	10.0	7.3	4
5	2.3	9.0	10.0	9.0	8.1	5

$$\varepsilon = 0$$

	1	2	3	4	5	
1	←	→	←	→	↓	1
2	↓	↑	↓	↑	↓	2
3	↓	→	↓	→	↓	3
4	↓	→	↓	→	↓	4
5	↓	→	↓	→	↓	5

	1	2	3	4	5	
1	0.4	0.5	0.9	1.3	1.4	1
2	0.1	0.0	0.5	1.3	1.7	2
3	0.1	-0.4	3.4	1.4	1.9	3
4	-0.1	3.4	3.3	3.7	2.2	4
5	-0.3	2.6	3.7	3.1	2.7	5

$$\varepsilon = 0.1$$

	1	2	3	4	5	
1	⊕	→	→	→	↑	1
2	+	↓	→	→	⊕	2
3	⊕	→	→	→	↓	3
4	+	→	⊕	→	↑	4
5	↓	↓	→	→	⊕	5

	1	2	3	4	5	
1	-1.1	-1.5	-1.1	-0.6	-0.6	1
2	-1.5	-2.2	-2.3	-1.0	-0.6	2
3	-1.2	-2.4	-2.2	-1.5	-0.6	3
4	-1.6	-2.3	-2.6	-1.4	-1.1	4
5	-2.0	-3.0	-1.8	-1.4	-1.0	5

$$\varepsilon = 0.2$$

	1	2	3	4	5	
1	-4.3	-5.5	-4.5	-2.6	-2.3	1
2	-5.6	-7.7	-7.7	-4.1	-2.4	2
3	-5.5	-9.0	-8.0	-5.6	-2.8	3
4	-6.8	-8.9	-9.4	-5.5	-4.2	4
5	-7.9	-10.1	-6.7	-5.1	-3.7	5

$$\varepsilon = 0.5$$

- ▷ The optimal  $\varepsilon$ -greedy policies are not *consistent* with the greedy optimal one! Why is that? Consider the target for example.

# Summary

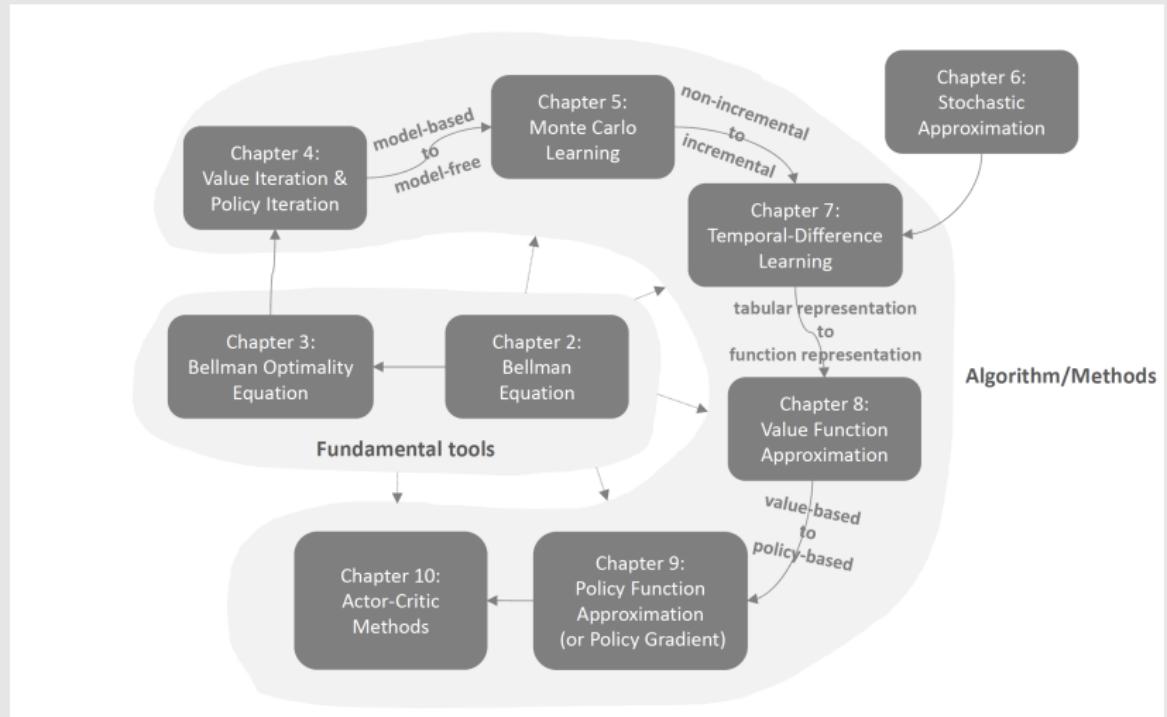
Key points:

- Mean estimation by the Monte Carlo methods
- Three algorithms:
  - MC Basic
  - MC Exploring Starts
  - MC  $\varepsilon$ -Greedy
- Relationship among the three algorithms
- Optimality vs exploration of  $\varepsilon$ -greedy policies

Lecture 6:  
Stochastic Approximation  
and  
Stochastic Gradient Descent

Shiyu Zhao

# Outline



# Introduction

- In the last lecture, we introduced Monte-Carlo learning.
- In the next lecture, we will introduce temporal-difference (TD) learning.
- In this lecture, we press the pause button to get us better prepared.

Why?

- The ideas and expressions of TD algorithms are very different from the algorithms we studied so far.
- Many students who see the TD algorithms the first time many wonder why these algorithms were designed in the first place and why they work effectively.
- There is a knowledge gap!

# Introduction

In this lecture,

- We fill the knowledge gap between the previous and upcoming lectures by introducing basic stochastic approximation (SA) algorithms.
- We will see in the next lecture that the temporal-difference algorithms are special SA algorithms. As a result, it will be much easier to understand these algorithms.

# Outline

- 1 Motivating examples**
- 2 Robbins-Monro algorithm**
  - Algorithm description
  - Illustrative examples
  - Convergence analysis
  - Application to mean estimation
- 3 Stochastic gradient descent**
  - Algorithm description
  - Examples and application
  - Convergence analysis
  - Convergence pattern
  - A deterministic formulation
- 4 BGD, MBGD, and SGD**
- 5 Summary**

# Outline

- 1 Motivating examples**
- 2 Robbins-Monro algorithm**
  - Algorithm description
  - Illustrative examples
  - Convergence analysis
  - Application to mean estimation
- 3 Stochastic gradient descent**
  - Algorithm description
  - Examples and application
  - Convergence analysis
  - Convergence pattern
  - A deterministic formulation
- 4 BGD, MBGD, and SGD**
- 5 Summary**

# Motivating example: mean estimation, again

## Revisit the mean estimation problem:

- Consider a random variable  $X$ .
- Our aim is to estimate  $\mathbb{E}[X]$ .
- Suppose that we collected a sequence of iid samples  $\{x_i\}_{i=1}^N$ .
- The expectation of  $X$  can be approximated by

$$\mathbb{E}[X] \approx \bar{x} := \frac{1}{N} \sum_{i=1}^N x_i.$$

## We already know from the last lecture:

- This approximation is the basic idea of Monte Carlo estimation.
- We know that  $\bar{x} \rightarrow \mathbb{E}[X]$  as  $N \rightarrow \infty$ .

## Why do we care about mean estimation so much?

- Many values in RL such as state/action values are defined as means.

# Motivating example: mean estimation

**New question:** how to calculate the mean  $\bar{x}$ ?

$$\mathbb{E}[X] \approx \bar{x} := \frac{1}{N} \sum_{i=1}^N x_i.$$

We have two ways.

- **The first way**, which is trivial, is to collect all the samples then calculate the average.
- The **drawback** of such way is that, if the samples are collected one by one over a period of time, we have to wait until all the samples to be collected.
- **The second way** can avoid this drawback because it calculates the average in an **incremental** and **iterative** manner.

# Motivating example: mean estimation

In particular, suppose

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

and hence

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

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:

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

## Motivating example: mean estimation

We can use

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

to calculate the mean  $\bar{x}$  incrementally:

$$w_1 = x_1,$$

$$w_2 = w_1 - \frac{1}{1}(w_1 - x_1) = x_1,$$

$$w_3 = w_2 - \frac{1}{2}(w_2 - x_2) = x_1 - \frac{1}{2}(x_1 - x_2) = \frac{1}{2}(x_1 + x_2),$$

$$w_4 = w_3 - \frac{1}{3}(w_3 - x_3) = \frac{1}{3}(x_1 + x_2 + x_3),$$

⋮

$$w_{k+1} = \frac{1}{k} \sum_{i=1}^k x_i.$$

# Motivating example: mean estimation

Remarks about this algorithm:

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

- An advantage of this algorithm is that a mean estimate can be obtained immediately once a sample is received. Then, the mean estimate can be used for other purposes immediately.
- The mean estimate is not accurate in the beginning due to insufficient samples (that is  $w_k \neq \mathbb{E}[X]$ ). However, it is better than nothing. As more samples are obtained, the estimate can be improved gradually (that is  $w_k \rightarrow \mathbb{E}[X]$  as  $k \rightarrow \infty$ ).

## Motivating example: mean estimation

Furthermore, consider an algorithm with a more general expression:

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

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

- Does this algorithm still converge to the mean  $\mathbb{E}[X]$ ? We will show that the answer is yes if  $\{\alpha_k\}$  satisfy some mild conditions.
- We will also show that this algorithm is a special **SA algorithm** and also a special **stochastic gradient descent algorithm**.
- In the next lecture, we will see that the temporal-difference algorithms have similar (but more complex) expressions.

# Outline

- 1 Motivating examples**
- 2 Robbins-Monro algorithm**
  - Algorithm description
  - Illustrative examples
  - Convergence analysis
  - Application to mean estimation
- 3 Stochastic gradient descent**
  - Algorithm description
  - Examples and application
  - Convergence analysis
  - Convergence pattern
  - A deterministic formulation
- 4 BGD, MBGD, and SGD**
- 5 Summary**

# Outline

- 1 Motivating examples**
- 2 Robbins-Monro algorithm**
  - Algorithm description
  - Illustrative examples
  - Convergence analysis
  - Application to mean estimation
- 3 Stochastic gradient descent**
  - Algorithm description
  - Examples and application
  - Convergence analysis
  - Convergence pattern
  - A deterministic formulation
- 4 BGD, MBGD, and SGD**
- 5 Summary**

# Robbins-Monro algorithm

Stochastic approximation (SA):

- SA refers to a broad class of stochastic iterative algorithms solving root finding or optimization problems.
- Compared to many other root-finding algorithms such as gradient-based methods, SA is powerful in the sense that it does not require to know the expression of the objective function nor its derivative.

Robbins-Monro (RM) algorithm:

- The is a pioneering work in the field of stochastic approximation.
- The famous stochastic gradient descent algorithm is a special form of the RM algorithm.
- It can be used to analyze the mean estimation algorithms introduced in the beginning.

## Robbins-Monro algorithm – Problem statement

**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  $g : \mathbb{R} \rightarrow \mathbb{R}$  is a function.

- Many problems can be eventually converted to this root finding problem. For example, suppose  $J(w)$  is an objective function to be minimized. Then, the optimization problem can be converged to

$$g(w) = \nabla_w J(w) = 0$$

- Note that an equation like  $g(w) = c$  with  $c$  as a constant can also be converted to the above equation by rewriting  $g(w) - c$  as a new function.

# Robbins-Monro algorithm – Problem statement

**How to calculate the root of  $g(w) = 0$ ?**

- If the expression of  $g$  or its derivative is known, there are many numerical algorithms that can solve this problem.
- What if the expression of the function  $g$  is unknown? For example, the function is represented by an artificial neuron network.

# Robbins-Monro algorithm – The algorithm

The Robbins-Monro (RM) algorithm can solve this problem:

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

where

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

The function  $g(w)$  is a **black box!** This algorithm relies on data:

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

**Philosophy:** without model, we need data!

- Here, the model refers to the expression of the function.

# Outline

- 1 Motivating examples**
- 2 Robbins-Monro algorithm**
  - Algorithm description
  - Illustrative examples
  - Convergence analysis
  - Application to mean estimation
- 3 Stochastic gradient descent**
  - Algorithm description
  - Examples and application
  - Convergence analysis
  - Convergence pattern
  - A deterministic formulation
- 4 BGD, MBGD, and SGD**
- 5 Summary**

# Robbins-Monro algorithm – Illustrative examples

**Excise:** manually solve  $g(w) = w - 10$  using the RM algorithm.

**Set:**  $w_1 = 20$ ,  $a_k \equiv 0.5$ ,  $\eta_k = 0$  (i.e., no observation error)

$$w_1 = 20 \implies g(w_1) = 10$$

$$w_2 = w_1 - a_1 g(w_1) = 20 - 0.5 * 10 = 15 \implies g(w_2) = 5$$

$$w_3 = w_2 - a_2 g(w_2) = 15 - 0.5 * 5 = 12.5 \implies g(w_3) = 2.5$$

⋮

$$w_k \rightarrow 10$$

Excises:

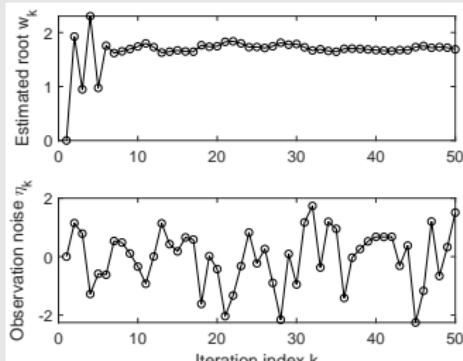
- What if  $a_k = 1$ ?
- What if  $a_k = 2$ ?

# Robbins-Monro algorithm – Illustrative examples

Another example: solve  $g(w) = w^3 - 5$  using the RM algorithm.

- The true root is  $5^{1/3} \approx 1.71$ .
- We only know is  $\tilde{g}(w) = g(w) + \eta$ .
- Suppose  $\eta_k$  is iid and obeys a standard normal distribution with a mean of zero and standard deviation of 1.
- The initial guess is  $w_1 = 0$  and  $a_k$  is selected to be  $a_k = 1/k$ .

The evolution of  $w_k$  is shown in the figure. As can be seen, the estimate  $w_k$  can converge to the true root.



# Outline

- 1 Motivating examples**
- 2 Robbins-Monro algorithm**
  - Algorithm description
  - Illustrative examples
  - Convergence analysis
  - Application to mean estimation
- 3 Stochastic gradient descent**
  - Algorithm description
  - Examples and application
  - Convergence analysis
  - Convergence pattern
  - A deterministic formulation
- 4 BGD, MBGD, and SGD**
- 5 Summary**

## Robbins-Monro algorithm – Convergence properties

Why can the RM algorithm find the root of  $g(w) = 0$ ?

- First present an illustrative example.
- Second give the rigorous convergence analysis.

# Robbins-Monro algorithm – Convergence properties

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)

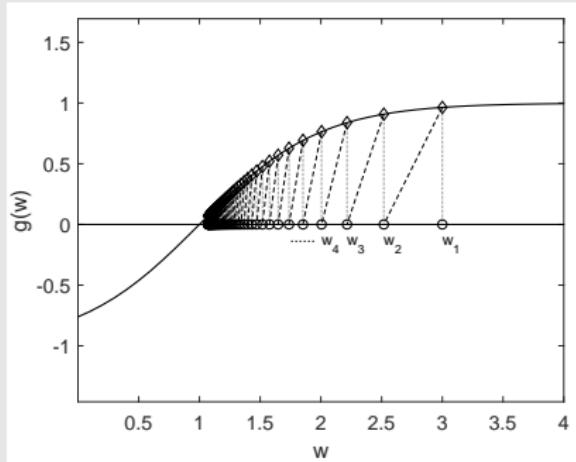
The RM algorithm in this case is

$$w_{k+1} = w_k - a_k g(w_k)$$

since  $\tilde{g}(w_k, \eta_k) = g(w_k)$  when  $\eta_k = 0$ .

# Robbins-Monro algorithm – Convergence properties

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



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

- When  $w_k > w^*$ , we have  $g(w_k) > 0$ . Then,

$$w_{k+1} = w_k - a_k g(w_k) < w_k \text{ and hence } w_{k+1} \text{ is closer to } w^* \text{ than } w_k.$$

- When  $w_k < w^*$ , we have  $g(w_k) < 0$ . Then,

$$w_{k+1} = w_k - a_k g(w_k) > w_k \text{ and } w_{k+1} \text{ is closer to } w^* \text{ than } w_k.$$

# Robbins-Monro algorithm – Convergence properties

The above analysis is intuitive, but not rigorous. A rigorous convergence result is given below.

## 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  $g(w^*) = 0$ .*

# Robbins-Monro algorithm – Convergence properties

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  $k \rightarrow \infty$ .
- 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  $\eta_k$  is not required to be Gaussian.

# Robbins-Monro algorithm – Convergence properties

Examine the second condition more closely:

$$\sum_{k=1}^{\infty} a_k^2 < \infty \quad \sum_{k=1}^{\infty} a_k = \infty$$

- First,  $\sum_{k=1}^{\infty} a_k^2 < \infty$  indicates that  $a_k \rightarrow 0$  as  $k \rightarrow \infty$ .
- **Why is this condition important?**

Since

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

- If  $a_k \rightarrow 0$ , then  $a_k \tilde{g}(w_k, \eta_k) \rightarrow 0$  and hence  $w_{k+1} - w_k \rightarrow 0$ .
- We need the fact that  $w_{k+1} - w_k \rightarrow 0$  if  $w_k$  converges eventually.
- If  $w_k \rightarrow w^*$ ,  $g(w_k) \rightarrow 0$  and  $\tilde{g}(w_k, \eta_k)$  is dominant by  $\eta_k$ .

# Robbins-Monro algorithm – Convergence properties

Examine the second condition more closely:

$$\sum_{k=1}^{\infty} a_k^2 < \infty \quad \sum_{k=1}^{\infty} a_k = \infty$$

- Second,  $\sum_{k=1}^{\infty} a_k = \infty$  indicates that  $a_k$  should not converge to zero too fast.
- **Why is this condition important?**

Summarizing  $w_2 = w_1 - a_1 \tilde{g}(w_1, \eta_1)$ ,  $w_3 = w_2 - a_2 \tilde{g}(w_2, \eta_2)$ ,  $\dots$ ,  
 $w_{k+1} = w_k - a_k \tilde{g}(w_k, \eta_k)$  leads to

$$w_1 - w_{\infty} = \sum_{k=1}^{\infty} a_k \tilde{g}(w_k, \eta_k).$$

Suppose  $w_{\infty} = w^*$ . If  $\sum_{k=1}^{\infty} a_k < \infty$ , then  $\sum_{k=1}^{\infty} a_k \tilde{g}(w_k, \eta_k)$  may be bounded. Then, if the initial guess  $w_1$  is chosen arbitrarily far away from  $w^*$ , then the above equality would be invalid.

# Robbins-Monro algorithm – Convergence properties

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

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

- It holds that

$$\lim_{n \rightarrow \infty} \left( \sum_{k=1}^n \frac{1}{k} - \ln n \right) = \kappa,$$

where  $\kappa \approx 0.577$  is called the Euler-Mascheroni constant (also called Euler's constant).

- It is notable that

$$\sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6} < \infty.$$

The limit  $\sum_{k=1}^{\infty} 1/k^2$  also has a specific name in the number theory:  
Basel problem.

## Robbins-Monro algorithm – Convergence properties

If the three conditions are not satisfied, the algorithm may not work.

- For example,  $g(w) = w^3 - 5$  does not satisfy the first condition on gradient boundedness. If the initial guess is good, the algorithm can converge (locally). Otherwise, it will diverge.

We will see that  $a_k$  is often selected as a **sufficiently small constant** in many RL algorithms. Although the second condition is not satisfied in this case, the algorithm can still work effectively.

# Outline

- 1 Motivating examples**
- 2 Robbins-Monro algorithm**
  - Algorithm description
  - Illustrative examples
  - Convergence analysis
  - Application to mean estimation
- 3 Stochastic gradient descent**
  - Algorithm description
  - Examples and application
  - Convergence analysis
  - Convergence pattern
  - A deterministic formulation
- 4 BGD, MBGD, and SGD**
- 5 Summary**

# Robbins-Monro algorithm – Apply to mean estimation

Recall that

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

is the mean estimation algorithm.

We know that

- If  $\alpha_k = 1/k$ , then  $w_{k+1} = 1/k \sum_{i=1}^k x_i$ .
- If  $\alpha_k$  is not  $1/k$ , the convergence was not analyzed.

Next, we show that this algorithm is a special case of the RM algorithm.

Then, its convergence naturally follows.

# Robbins-Monro algorithm – Apply to mean estimation

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.

The convergence naturally follows.

# Dvoretzky's convergence 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  $\mathcal{H}_k = \{w_k, w_{k-1}, \dots, \eta_{k-1}, \dots, \alpha_{k-1}, \dots, \beta_{k-1}, \dots\}$ .

- A more general result than the RM theorem. It can be used to prove the RM theorem
- It can also directly analyze the mean estimation problem.
- An extension of it can be used to analyze Q-learning and TD learning algorithms.

# Outline

- 1 Motivating examples**
- 2 Robbins-Monro algorithm**
  - Algorithm description
  - Illustrative examples
  - Convergence analysis
  - Application to mean estimation
- 3 Stochastic gradient descent**
  - Algorithm description
  - Examples and application
  - Convergence analysis
  - Convergence pattern
  - A deterministic formulation
- 4 BGD, MBGD, and SGD**
- 5 Summary**

# Outline

- 1 Motivating examples**
- 2 Robbins-Monro algorithm**
  - Algorithm description
  - Illustrative examples
  - Convergence analysis
  - Application to mean estimation
- 3 Stochastic gradient descent**
  - Algorithm description
  - Examples and application
  - Convergence analysis
  - Convergence pattern
  - A deterministic formulation
- 4 BGD, MBGD, and SGD**
- 5 Summary**

# Stochastic gradient descent

Next, we introduce stochastic gradient descent (SGD) algorithms:

- SGD is widely used in the field of machine learning and also in RL.
- SGD is a special RM algorithm.
- The mean estimation algorithm is a special SGD algorithm.

Suppose we aim to solve the following optimization problem:

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

- $w$  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.

# Stochastic gradient descent

## Method 1: gradient descent (GD)

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

Drawback: the expected value is difficult to obtain.

## Method 2: batch gradient descent (BGD)

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

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

Drawback: it requires many samples in each iteration for each  $w_k$ .

# Stochastic gradient descent – Algorithm

## Method 3: stochastic gradient descent (SGD)

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

- 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  $n = 1$ .

# Outline

- 1 Motivating examples**
- 2 Robbins-Monro algorithm**
  - Algorithm description
  - Illustrative examples
  - Convergence analysis
  - Application to mean estimation
- 3 Stochastic gradient descent**
  - Algorithm description
  - Examples and application**
  - Convergence analysis
  - Convergence pattern
  - A deterministic formulation
- 4 BGD, MBGD, and SGD**
- 5 Summary**

# Stochastic gradient descent – Example and application

We next consider an example:

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

where

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

**Excises:**

- Excise 1: Show that the optimal solution is  $w^* = \mathbb{E}[X]$ .
- Excise 2: Write out the GD algorithm for solving this problem.
- Excise 3: Write out the SGD algorithm for solving this problem.

# Stochastic gradient descent – Example and application

## Answer:

- The GD algorithm for solving the above problem is

$$\begin{aligned}w_{k+1} &= w_k - \alpha_k \nabla_w J(w_k) \\&= w_k - \alpha_k \mathbb{E}[\nabla_w f(w_k, X)] \\&= w_k - \alpha_k \mathbb{E}[w_k - X].\end{aligned}$$

- The SGD algorithm for solving the above problem is

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

- Note:

- It is the same as the mean estimation algorithm we presented before.
- That mean estimation algorithm is a special SGD algorithm.

# Outline

- 1 Motivating examples**
- 2 Robbins-Monro algorithm**
  - Algorithm description
  - Illustrative examples
  - Convergence analysis
  - Application to mean estimation
- 3 Stochastic gradient descent**
  - Algorithm description
  - Examples and application
  - Convergence analysis**
  - Convergence pattern
  - A deterministic formulation
- 4 BGD, MBGD, and SGD**
- 5 Summary**

# Stochastic gradient descent – Convergence

From GD to SGD:

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



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

$\nabla_w f(w_k, x_k)$  can be viewed as a noisy measurement of  $\mathbb{E}[\nabla_w f(w, X)]$ :

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

Since

$$\nabla_w f(w_k, x_k) \neq \mathbb{E}[\nabla_w f(w, X)]$$

whether  $w_k \rightarrow w^*$  as  $k \rightarrow \infty$  by SGD?

# Stochastic gradient descent – Convergence

We next show that SGD is a special RM algorithm. Then, the convergence naturally follows.

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  $g(w) = 0$ .

# Stochastic gradient descent – Convergence

What we can measure is

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

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.
- Therefore, SGD is a special RM algorithm.

# Stochastic gradient descent – Convergence

Since SGD is a special RM algorithm, its convergence naturally follows.

## 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.*

For the proof see the book.

# Outline

- 1 Motivating examples**
- 2 Robbins-Monro algorithm**
  - Algorithm description
  - Illustrative examples
  - Convergence analysis
  - Application to mean estimation
- 3 Stochastic gradient descent**
  - Algorithm description
  - Examples and application
  - Convergence analysis
  - Convergence pattern**
  - A deterministic formulation
- 4 BGD, MBGD, and SGD**
- 5 Summary**

# Stochastic gradient descent – Convergence pattern

**Question:** Since the stochastic gradient is random and hence the approximation is inaccurate, whether the convergence of SGD is slow or random?

To answer this question, we consider the **relative error** between the stochastic and batch gradients:

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

Since  $\mathbb{E}[\nabla_w f(w^*, X)] = 0$ , we further have

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

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

# Stochastic gradient descent – Convergence pattern

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  $\delta_k$  becomes

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

Substituting the above inequality to  $\delta_k$  gives

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

# Stochastic gradient descent – Convergence pattern

Note that

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

The above equation suggests an interesting convergence pattern of SGD.

- The relative error  $\delta_k$  is inversely proportional to  $|w_k - w^*|$ .
- When  $|w_k - w^*|$  is large,  $\delta_k$  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  $w^*$ .

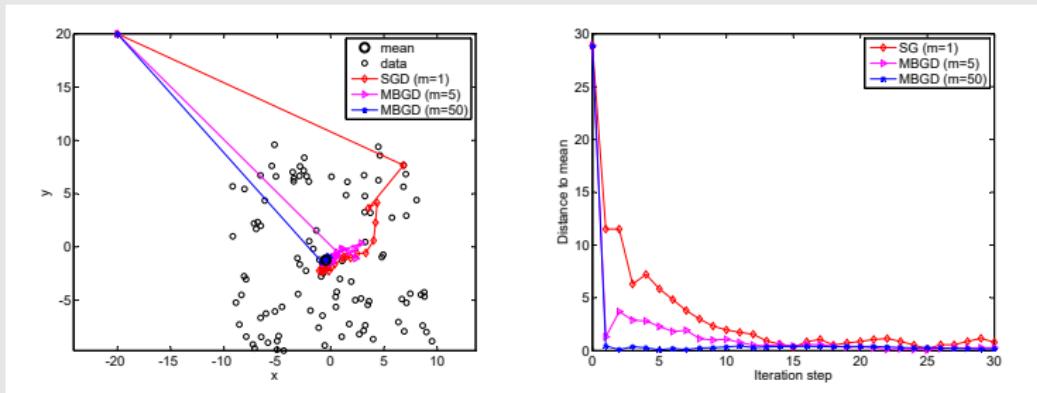
# Stochastic gradient descent – Convergence pattern

Consider an illustrative example:

**Setup:**  $X \in \mathbb{R}^2$  represents a random position in the plane. Its distribution is uniform in the square area centered at the origin with the side length as 20. The true mean is  $\mathbb{E}[X] = 0$ . The mean estimation is based on 100 iid samples  $\{x_i\}_{i=1}^{100}$ .

# Stochastic gradient descent – Convergence pattern

## Result:



- Although the initial guess of the mean is far away from the true value, the SGD estimate can approach the neighborhood of the true value fast.
- When the estimate is close to the true value, it exhibits certain randomness but still approaches the true value gradually.

# Outline

- 1 Motivating examples**
- 2 Robbins-Monro algorithm**
  - Algorithm description
  - Illustrative examples
  - Convergence analysis
  - Application to mean estimation
- 3 Stochastic gradient descent**
  - Algorithm description
  - Examples and application
  - Convergence analysis
  - Convergence pattern
  - A deterministic formulation
- 4 BGD, MBGD, and SGD**
- 5 Summary**

# Stochastic gradient descent – A deterministic formulation

- The formulation of SGD we introduced above involves random variables and expectation.
- One may often encounter a **deterministic** formulation of SGD without involving any random variables.

Consider the optimization problem:

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

- $f(w, x_i)$  is a parameterized function.
- $w$  is the parameter to be optimized.
- a set of real numbers  $\{x_i\}_{i=1}^n$ , where  $x_i$  does not have to be a sample of any random variable.

# Stochastic gradient descent – A deterministic formulation

The gradient descent algorithm for solving this problem is

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

Suppose the set is large and we can only fetch a single number every time. In this case, we can use the following iterative algorithm:

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

## Questions:

- Is this algorithm SGD? It does not involve any random variables or expected values.
- How should we use the finite set of numbers  $\{x_i\}_{i=1}^n$ ? Should we sort these numbers in a certain order and then use them one by one? Or should we randomly sample a number from the set?

# Stochastic gradient descent – A deterministic formulation

A quick answer to the above questions is that we can introduce a random variable manually and convert the *deterministic formulation* to the *stochastic formulation* of 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:

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

- The last equality in the above equation is strict instead of approximate. Therefore, the algorithm is SGD.
- The estimate converges if  $x_k$  is *uniformly* and independently sampled from  $\{x_i\}_{i=1}^n$ .  $x_k$  may repeatedly take the same number in  $\{x_i\}_{i=1}^n$  since it is sampled randomly.

# Outline

- 1 Motivating examples**
- 2 Robbins-Monro algorithm**
  - Algorithm description
  - Illustrative examples
  - Convergence analysis
  - Application to mean estimation
- 3 Stochastic gradient descent**
  - Algorithm description
  - Examples and application
  - Convergence analysis
  - Convergence pattern
  - A deterministic formulation
- 4 BGD, MBGD, and SGD**
- 5 Summary**

# BGD, MBGD, and SGD

Suppose we would like to minimize  $J(w) = \mathbb{E}[f(w, X)]$  given a set of random samples  $\{x_i\}_{i=1}^n$  of  $X$ . The BGD, SGD, MBGD algorithms solving this problem are, respectively,

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

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

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

- **In the BGD algorithm**, all the samples are used in every iteration. When  $n$  is large,  $(1/n) \sum_{i=1}^n \nabla_w f(w_k, x_i)$  is close to the true gradient  $\mathbb{E}[\nabla_w f(w_k, X)]$ .
- **In the MBGD algorithm**,  $\mathcal{I}_k$  is a subset of  $\{1, \dots, n\}$  with the size as  $|\mathcal{I}_k| = m$ . The set  $\mathcal{I}_k$  is obtained by  $m$  times iid samplings.
- **In the SGD algorithm**,  $x_k$  is randomly sampled from  $\{x_i\}_{i=1}^n$  at time  $k$ .

# BGD, MBGD, and SGD

Compare MBGD with BGD and 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.

# BGD, MBGD, and SGD – Illustrative examples

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:

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

The three algorithms for solving this problem are, respectively,

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

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

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

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

## BGD, MBGD, and SGD

Furthermore, if  $\alpha_k = 1/k$ , the above equation can be solved as

$$w_{k+1} = \frac{1}{k} \sum_{j=1}^k \bar{x} = \bar{x}, \quad (\text{BGD})$$

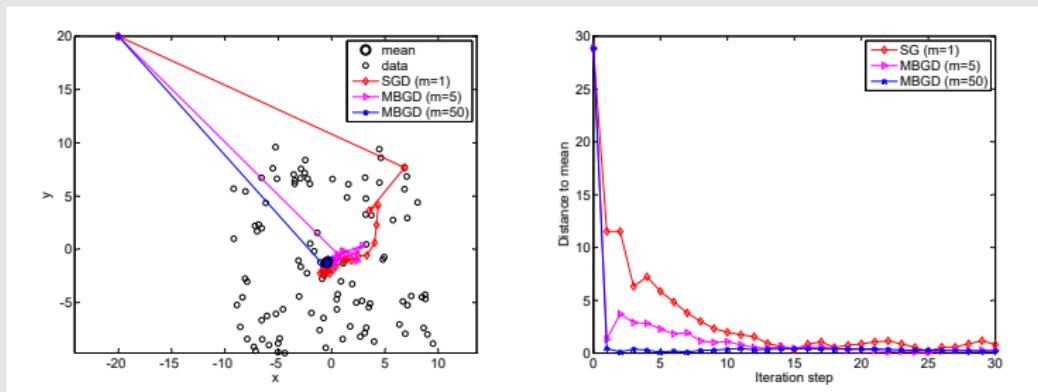
$$w_{k+1} = \frac{1}{k} \sum_{j=1}^k \bar{x}_j^{(m)}, \quad (\text{MBGD})$$

$$w_{k+1} = \frac{1}{k} \sum_{j=1}^k x_j. \quad (\text{SGD})$$

- The estimate of BGD at each step is exactly the optimal solution  
 $w^* = \bar{x}$ .
- The estimate of MBGD approaches the mean faster than SGD because  $\bar{x}_k^{(m)}$  is already an average.

# BGD, MBGD, and SGD

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



**Figure:** An illustrative example for mean estimation by different GD algorithms.

# Outline

- 1 Motivating examples**
- 2 Robbins-Monro algorithm**
  - Algorithm description
  - Illustrative examples
  - Convergence analysis
  - Application to mean estimation
- 3 Stochastic gradient descent**
  - Algorithm description
  - Examples and application
  - Convergence analysis
  - Convergence pattern
  - A deterministic formulation
- 4 BGD, MBGD, and SGD**
- 5 Summary**

# 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  $\{\nabla_w f(w_k, x_k)\}$

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

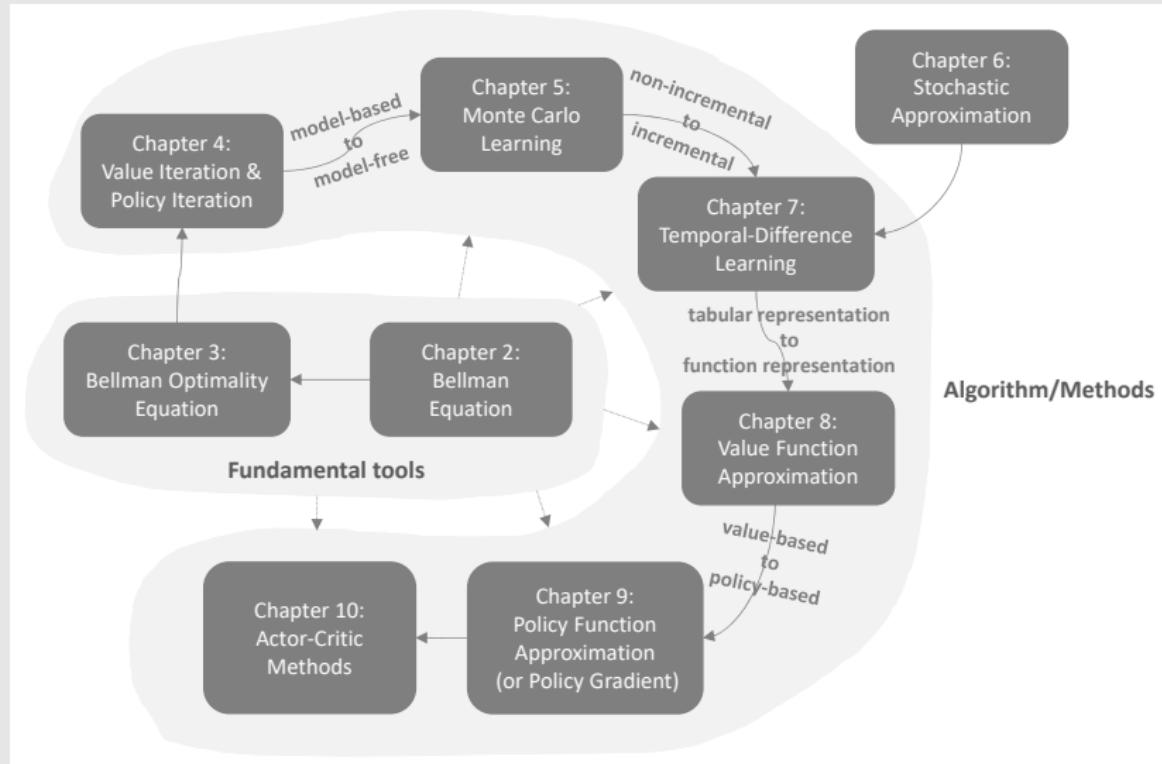
These results are useful:

- We will see in the next chapter that the temporal-difference learning algorithms can be viewed as stochastic approximation algorithms and hence have similar expressions.
- They are important optimization techniques that can be applied to many other fields.

## Lecture 8: Value Function Approximation

Shiyu Zhao

# Introduction



# Outline

- 1 Motivating examples: curve fitting**
- 2 Algorithm for state value estimation**
  - Objective function
  - Optimization algorithms
  - Selection of function approximators
  - Illustrative examples
  - Summary of the story
  - Theoretical analysis
- 3 Sarsa with function approximation**
- 4 Q-learning with function approximation**
- 5 Deep Q-learning**
- 6 Summary**

# Outline

- 1 Motivating examples: curve fitting
- 2 Algorithm for state value estimation
  - Objective function
  - Optimization algorithms
  - Selection of function approximators
  - Illustrative examples
  - Summary of the story
  - Theoretical analysis
- 3 Sarsa with function approximation
- 4 Q-learning with function approximation
- 5 Deep Q-learning
- 6 Summary

# Motivating examples: curve fitting

So far in this book, state and action values are represented by [tables](#).

- For example, action value:

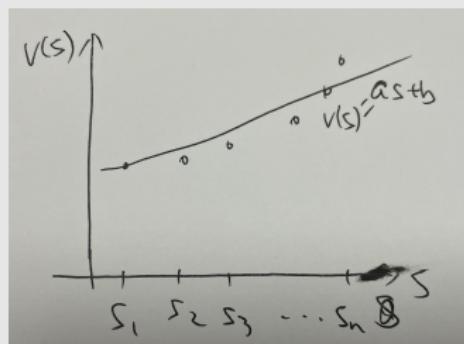
	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$
$s_1$	$q_\pi(s_1, a_1)$	$q_\pi(s_1, a_2)$	$q_\pi(s_1, a_3)$	$q_\pi(s_1, a_4)$	$q_\pi(s_1, a_5)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$s_9$	$q_\pi(s_9, a_1)$	$q_\pi(s_9, a_2)$	$q_\pi(s_9, a_3)$	$q_\pi(s_9, a_4)$	$q_\pi(s_9, a_5)$

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

# Motivating examples: curve fitting

Consider an example:

- Suppose there are one-dimensional states  $s_1, \dots, s_{|\mathcal{S}|}$ .
- Their state values are  $v_\pi(s_1), \dots, v_\pi(s_{|\mathcal{S}|})$ , where  $\pi$  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.



**Figure:** An illustration of function approximation of samples.

# Motivating examples: curve fitting

First, we use the simplest straight line to fit the dots.

Suppose the equation of the straight line is

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

where

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

## Motivating examples: curve fitting

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

The benefits:

- The tabular representation needs to store  $|S|$  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  $\phi^T(s)w$ .
- Such a benefit is **not free**. It comes with a cost: the state values can not be represented accurately. This is why this method is **called value approximation**.

# Motivating examples: curve fitting

Second, we can also fit the points using a **second-order curve**:

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

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  $\phi(s)$ .

## Motivating examples: curve fitting

Third, we can use even **higher-order polynomial curves or other complex curves to fit the dots.**

- **Advantage:** It can better approximate.
- **Disadvantage:** It needs more parameters.

# Motivating examples: curve fitting

Quick summary:

- **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.
- **Key difference:** How to access and assign the value of  $v(s)$
- **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.

# Outline

- 1 Motivating examples: curve fitting
- 2 Algorithm for state value estimation
  - Objective function
  - Optimization algorithms
  - Selection of function approximators
  - Illustrative examples
  - Summary of the story
  - Theoretical analysis
- 3 Sarsa with function approximation
- 4 Q-learning with function approximation
- 5 Deep Q-learning
- 6 Summary

# Outline

- 1 Motivating examples: curve fitting
- 2 Algorithm for state value estimation
  - Objective function
  - Optimization algorithms
  - Selection of function approximators
  - Illustrative examples
  - Summary of the story
  - Theoretical analysis
- 3 Sarsa with function approximation
- 4 Q-learning with function approximation
- 5 Deep Q-learning
- 6 Summary

# Objective function

Introduce in a more formal way:

- Let  $v_\pi(s)$  and  $\hat{v}(s, w)$  be the true state value and a function for approximation.
- Our goal is to find an **optimal**  $w$  so that  $\hat{v}(s, w)$  can best approximate  $v_\pi(s)$  for every  $s$ .
- This is a policy evaluation problem. Later we will extend to policy improvement.
- 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.

# Objective function

The **objective function** is

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

- Our goal is to find the best  $w$  that can minimize  $J(w)$ .
- The expectation is with respect to the random variable  $S \in \mathcal{S}$ . What is the probability distribution of  $S$ ?
  - This is often confusing because we have not discussed the probability distribution of states so far in this book.
  - There are several ways to define the probability distribution of  $S$ .

# Objective function

**The first way is to use a uniform distribution.**

- That is to treat all the states to be **equally important** by setting the probability of each state as  $1/|\mathcal{S}|$ .
- In this case, the objective function becomes

$$J(w) = \mathbb{E}[(v_\pi(S) - \hat{v}(S, w))^2] = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} (v_\pi(s) - \hat{v}(s, w))^2.$$

- **Drawback:**

- The states may not be equally important. For example, some states may be rarely visited by a policy. Hence, this way does not consider the real dynamics of the Markov process under the given policy.

# Objective function

The second way is to use the **stationary distribution**.

- Stationary distribution is an important concept that will be frequently used in this course. In short, it describes the **long-run behavior** of a Markov process.
- Let  $\{d_\pi(s)\}_{s \in \mathcal{S}}$  denote the stationary distribution of the Markov process under policy  $\pi$ . 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.

# Objective function – Stationary distribution

## More explanation about stationary distribution:

- *Distribution*: Distribution of the state
- *Stationary*: Long-run behavior
- *Summary*: after the agent runs a long time following a policy, the probability that the agent is at any state can be described by this distribution.

Remarks:

- Stationary distribution is also called **steady-state distribution**, or **limiting distribution**.
- It is critical to understand the value function approximation method.
- It is also important for the policy gradient method in the next lecture.

# Objective function - Stationary distribution

Illustrative example:

- Given a policy shown in the figure.
- Let  $n_\pi(s)$  denote the number of times that  $s$  has been visited in a very long episode generated by  $\pi$ .
- Then,  $d_\pi(s)$  can be approximated by

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

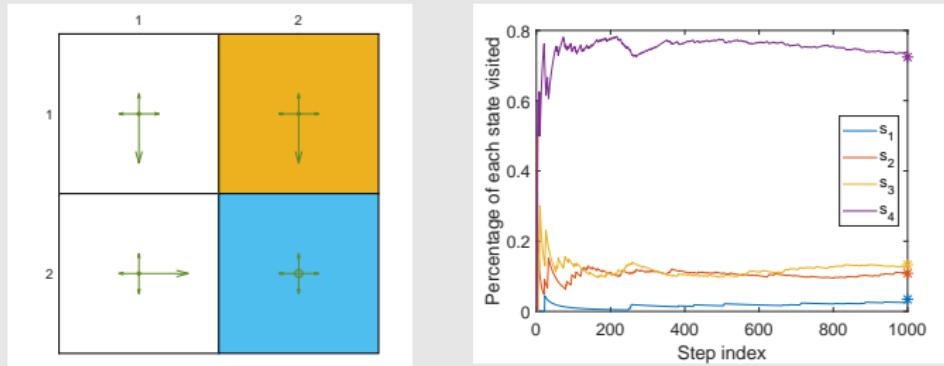


Figure: Long-run behavior of an  $\epsilon$ -greedy policy with  $\epsilon = 0.5$ .

## Objective function - Stationary distribution

The converged values can be predicted because they are the entries of  $d_\pi$ :

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

For this example, we have  $P_\pi$  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

$$d_\pi = [0.0345, 0.1084, 0.1330, 0.7241]^T$$

A comprehensive introduction can be found in the book.

# Outline

- 1 Motivating examples: curve fitting
- 2 Algorithm for state value estimation
  - Objective function
  - Optimization algorithms
  - Selection of function approximators
  - Illustrative examples
  - Summary of the story
  - Theoretical analysis
- 3 Sarsa with function approximation
- 4 Q-learning with function approximation
- 5 Deep Q-learning
- 6 Summary

# 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

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

The true gradient above involves the calculation of an expectation.

# Optimization algorithms

We can use the stochastic gradient to replace the true gradient:

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

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

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

# Optimization algorithms

In particular,

- 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

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

# Optimization algorithms

## Pseudocode: TD learning with function approximation

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

**Aim:** Approximate the true state values of a given policy  $\pi$ .

For each episode generated following the policy  $\pi$ , do

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

        In the general case,

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

        In the linear case,

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

It can only estimate the state values of a given policy, but it is important to understand other algorithms introduced later.

# Outline

- 1 Motivating examples: curve fitting
- 2 Algorithm for state value estimation
  - Objective function
  - Optimization algorithms
  - Selection of function approximators
  - Illustrative examples
  - Summary of the story
  - Theoretical analysis
- 3 Sarsa with function approximation
- 4 Q-learning with function approximation
- 5 Deep Q-learning
- 6 Summary

# Selection of function approximators

An important question that has not been answered: How to select the function  $\hat{v}(s, w)$ ?

- The first approach, which **was widely used before**, is to use a [linear](#) function

$$\hat{v}(s, w) = \phi^T(s)w$$

Here,  $\phi(s)$  is the feature vector, which can be a [polynomial basis](#), [Fourier basis](#), ... (see my book for details). We have seen in the motivating example and will see again in the illustrative examples later.

- The second approach, which is **widely used nowadays**, is to use a neural 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  $w$ .

# Linear function approximation

In the linear case where  $\hat{v}(s, w) = \phi^T(s)w$ , we have

$$\nabla_w \hat{v}(s, w) = \phi(s).$$

Substituting the gradient into the TD algorithm

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

yields

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

which is the algorithm of TD learning with linear function approximation.

It is called **TD-Linear** in our course in short.

# Linear function approximation

- **Disadvantages of linear function approximation:**
  - Difficult to select appropriate feature vectors.
- **Advantages of linear function approximation:**
  - The theoretical properties of the TD algorithm in the linear case can be much better understood than in the nonlinear case.
  - Linear function approximation is still powerful in the sense that the tabular representation is merely a special case of linear function approximation.

# Linear function approximation

We next show that **the tabular representation is a special case of linear function approximation.**

- First, consider the special feature vector for state  $s$ :

$$\phi(s) = e_s \in \mathbb{R}^{|S|},$$

where  $e_s$  is a vector with the  $s$ th entry as 1 and the others as 0.

- In this case,

$$\hat{v}(s, w) = e_s^T w = w(s),$$

where  $w(s)$  is the  $s$ th entry of  $w$ .

# Linear function approximation

Recall that the TD-Linear algorithm is

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

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

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

This is a vector equation that merely updates the  $s_t$ th entry of  $w_t$ .

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

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

which is exactly the tabular TD algorithm.

# Outline

- 1 Motivating examples: curve fitting
- 2 Algorithm for state value estimation
  - Objective function
  - Optimization algorithms
  - Selection of function approximators
  - Illustrative examples
  - Summary of the story
  - Theoretical analysis
- 3 Sarsa with function approximation
- 4 Q-learning with function approximation
- 5 Deep Q-learning
- 6 Summary

# Illustrative examples

Consider a 5x5 grid-world example:

- Given a policy:  $\pi(a|s) = 0.2$  for any  $s, a$
- Our aim is to estimate the state values of this policy (policy evaluation problem).
- There are 25 state values in total. We next show that we can use less than 25 parameters to approximate these state values.
- Set  $r_{\text{forbidden}} = r_{\text{boundary}} = -1$ ,  $r_{\text{target}} = 1$ , and  $\gamma = 0.9$ .

	1	2	3	4	5
1	+	+	+	+	+
2	+	+	+	+	+
3	+	+	+	+	+
4	+	+	+	+	+
5	+	+	+	+	+

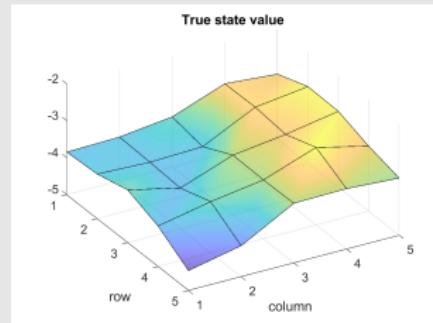
# Illustrative examples

Ground truth:

- The true state values and the 3D visualization

	1	2	3	4	5
1	+	+	+	+	+
2	+	+	+	+	+
3	+	+	+	+	+
4	+	+	+	+	+
5	+	+	+	+	+

	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

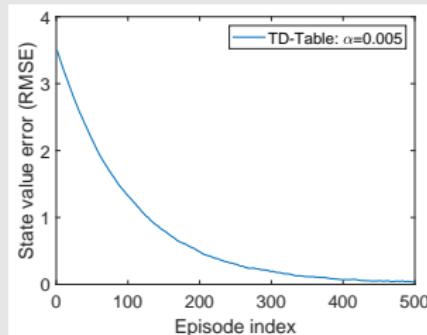
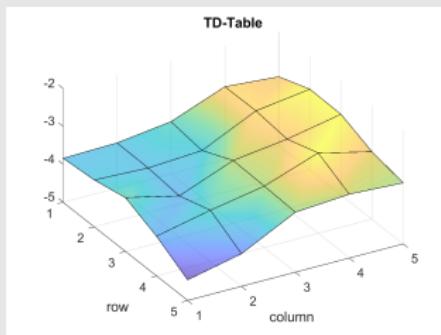


Experience samples:

- 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.

# Illustrative examples

For comparison, the results given by the [tabular TD algorithm](#) (called TD-Table in short):



## Illustrative examples

We next show the results by the TD-Linear algorithm.

Feature vector selection:

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

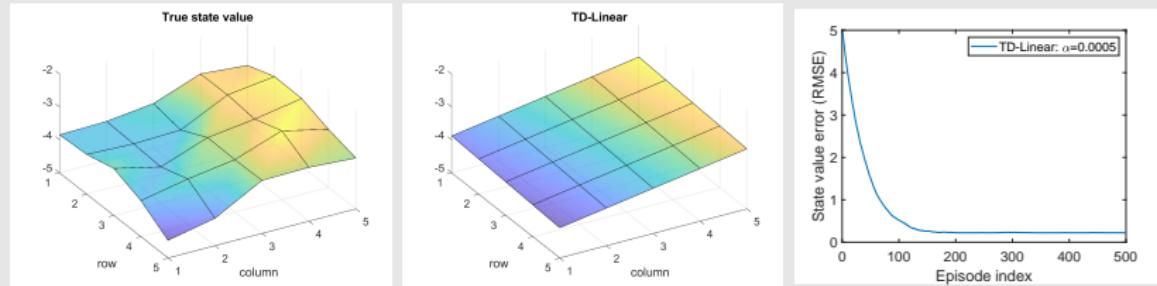
In this case, the approximated state value is

$$\hat{v}(s, w) = \phi^T(s)w = [1, x, y] \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = w_1 + w_2x + w_3y.$$

Notably,  $\phi(s)$  can also be defined as  $\phi(s) = [x, y, 1]^T$ , where the order of the elements does not matter.

# Illustrative examples

Results by the TD-Linear algorithm:



- The trend is right, but there are errors due to **limited approximation ability**!
- We are trying to use a plane to approximate a non-plane surface!

## Illustrative examples

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

- For example, we can consider

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

In this case,

$$\hat{v}(s, w) = \phi^T(s)w = w_1 + w_2x + w_3y + w_4x^2 + w_5y^2 + w_6xy$$

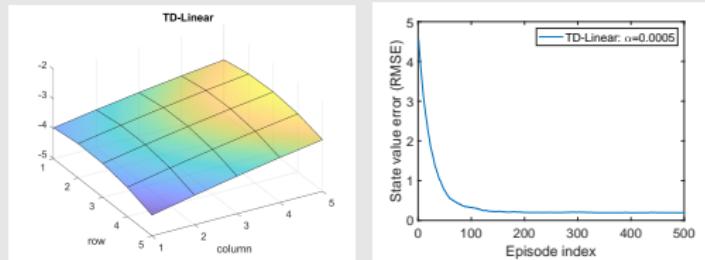
which corresponds to a quadratic surface.

- We can further increase the dimension of the feature vector:

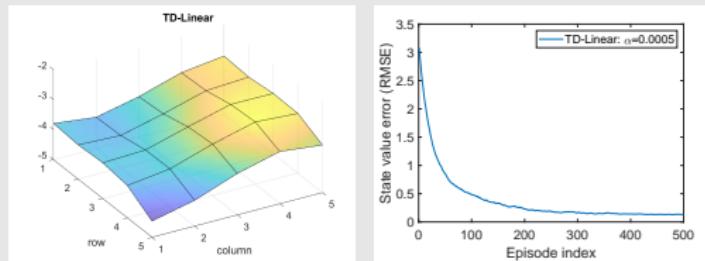
$$\phi(s) = [1, x, y, x^2, y^2, xy, x^3, y^3, x^2y, xy^2]^T \in \mathbb{R}^{10}.$$

# Illustrative examples

Results by the TD-Linear algorithm with higher-order feature vectors:



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



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

More examples and types of features are given in the book.

# Outline

- 1 Motivating examples: curve fitting
- 2 Algorithm for state value estimation
  - Objective function
  - Optimization algorithms
  - Selection of function approximators
  - Illustrative examples
  - Summary of the story
  - Theoretical analysis
- 3 Sarsa with function approximation
- 4 Q-learning with function approximation
- 5 Deep Q-learning
- 6 Summary

# Summary of the story

Up to now, we finished the story of TD learning with value function approximation.

- 1) This story started from the objective function:

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

The objective function suggests that it is a policy evaluation problem.

- 2) The gradient-descent algorithm is

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

- 3) The true value function, which is unknown, in the algorithm is replaced by an approximation, leading to the algorithm:

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

Although this story is helpful to understand the basic idea, it is **not mathematically rigorous**.

# Outline

- 1 Motivating examples: curve fitting
- 2 Algorithm for state value estimation
  - Objective function
  - Optimization algorithms
  - Selection of function approximators
  - Illustrative examples
  - Summary of the story
  - Theoretical analysis
- 3 Sarsa with function approximation
- 4 Q-learning with function approximation
- 5 Deep Q-learning
- 6 Summary

# Theoretical analysis

- The algorithm

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

does not minimize the following objective function:

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

# Theoretical analysis

Different objective functions:

- **Objective function 1: True value error**

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

- **Objective function 2: Bellman error**

$$J_{BE}(w) = \|\hat{v}(w) - (r_\pi + \gamma P_\pi \hat{v}(w))\|_D^2 \doteq \|\hat{v}(w) - T_\pi(\hat{v}(w))\|_D^2,$$

where  $T_\pi(x) \doteq r_\pi + \gamma P_\pi x$

# Theoretical analysis

- **Objective function 3: Projected Bellman error**

$$J_{PBE}(w) = \|\hat{v}(w) - MT_\pi(\hat{v}(w))\|_D^2,$$

where  $M$  is a projection matrix.

The TD-Linear algorithm minimizes the projected Bellman error.

Details can be found in the book.

# Outline

- 1 Motivating examples: curve fitting**
- 2 Algorithm for state value estimation**
  - Objective function
  - Optimization algorithms
  - Selection of function approximators
  - Illustrative examples
  - Summary of the story
  - Theoretical analysis
- 3 Sarsa with function approximation**
- 4 Q-learning with function approximation**
- 5 Deep Q-learning**
- 6 Summary**

# Sarsa with function approximation

So far, we merely considered the problem of state value estimation. That is we hope

$$\hat{v} \approx v_\pi$$

To search for optimal policies, we need to estimate action values.

The Sarsa algorithm with value function approximation is

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

This is the same as the algorithm we introduced previously in this lecture except that  $\hat{v}$  is replaced by  $\hat{q}$ .

# Sarsa with function approximation

To search for optimal policies, we can combine **policy evaluation** and **policy improvement**.

## 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):**

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

**Policy update:**

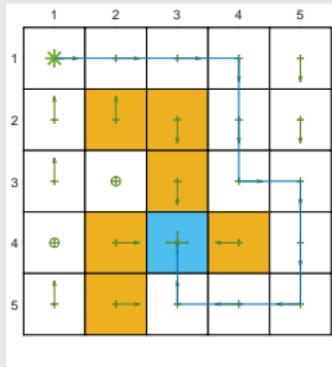
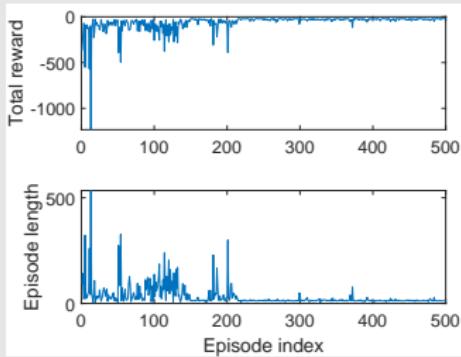
$$\pi_{t+1}(a|s_t) = 1 - \frac{\varepsilon}{|\mathcal{A}(s)|} (|\mathcal{A}(s)| - 1) \quad \text{if } a = \arg \max_{a \in \mathcal{A}(s_t)} \hat{q}(s_t, a, w_{t+1})$$

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

# Sarsa with function approximation

Illustrative example:

- Sarsa with *linear function approximation*.
- $\gamma = 0.9$ ,  $\epsilon = 0.1$ ,  $r_{\text{boundary}} = r_{\text{forbidden}} = -10$ ,  $r_{\text{target}} = 1$ ,  $\alpha = 0.001$ .



For details, please see the book.

# Outline

- 1 Motivating examples: curve fitting**
- 2 Algorithm for state value estimation**
  - Objective function
  - Optimization algorithms
  - Selection of function approximators
  - Illustrative examples
  - Summary of the story
  - Theoretical analysis
- 3 Sarsa with function approximation**
- 4 Q-learning with function approximation**
- 5 Deep Q-learning**
- 6 Summary**

# Q-learning with function approximation

Similar to Sarsa, tabular Q-learning can also be extended to the case of value function approximation.

The q-value update rule is

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

which is the same as Sarsa except that  $\hat{q}(s_{t+1}, a_{t+1}, w_t)$  is replaced by  $\max_{a \in \mathcal{A}(s_{t+1})} \hat{q}(s_{t+1}, a, w_t)$ .

# Q-learning with function approximation

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

**Initialization:** Initial parameter vector  $w_0$ . Initial policy  $\pi_0$ . 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):**

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

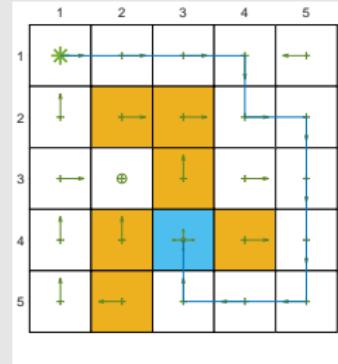
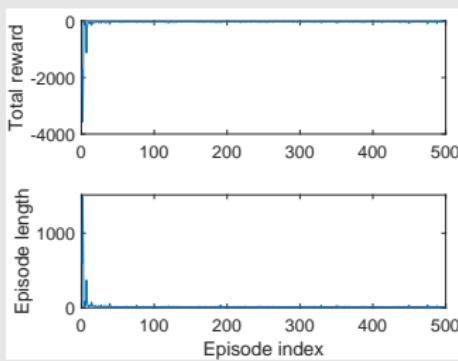
**Policy update:**

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

# Q-learning with function approximation

Illustrative example:

- Q-learning with *linear function approximation*.
- $\gamma = 0.9$ ,  $\epsilon = 0.1$ ,  $r_{\text{boundary}} = r_{\text{forbidden}} = -10$ ,  $r_{\text{target}} = 1$ ,  $\alpha = 0.001$ .



# Outline

- 1 Motivating examples: curve fitting
- 2 Algorithm for state value estimation
  - Objective function
  - Optimization algorithms
  - Selection of function approximators
  - Illustrative examples
  - Summary of the story
  - Theoretical analysis
- 3 Sarsa with function approximation
- 4 Q-learning with function approximation
- 5 Deep Q-learning
- 6 Summary

# Deep Q-learning

Deep Q-learning or deep Q-network (DQN):

- One of the earliest and most successful algorithms that introduce deep neural networks into RL.
- The role of neural networks is to be a nonlinear function approximator.
- Different from the following algorithm:

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

because of the way of training a network.

# Deep Q-learning

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

# Deep Q-learning

How to minimize the objective function? Gradient-descent!

- How to calculate the gradient of the objective function? Tricky!
- That is because, in this objective function

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

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.

# Deep Q-learning

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.

# Deep Q-learning

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

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

- The **basic idea** of deep Q-learning is to use the gradient-descent algorithm to minimize the objective function.
- However, such an optimization process evolves some **important techniques** that deserve special attention.

# Deep Q-learning - Two networks

## First technique:

- Two networks, a main network and a target network.

## Why is it used?

- The mathematical reason has been explained when we calculate the gradient.

## Implementation details:

- 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  $\{(s, a, y_T)\}$ .

# Deep Q-learning - Experience replay

## Another technique:

- Experience replay

**Question:** What is experience replay?

**Answer:**

- 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?)**.

# Deep Q-learning - Experience replay

**Question:** Why is experience replay necessary in deep Q-learning? Why does the replay must follow a uniform distribution?

**Answer:** The answers lie in 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.

# Deep Q-learning - Experience replay

## Answer (continued):

- 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.
- This is the **mathematical reason** *why experience replay is necessary* and *why the experience replay must be uniform*.

# Deep Q-learning - Experience replay

## Revisit the tabular case:

- Question: Why does not tabular Q-learning require experience replay?
  - Answer: No uniform distribution requirement.
- Question: Why Deep Q-learning involves distribution?
  - Answer: The objective function in the deep case is a *scalar* average over all  $(S, A)$ . The tabular case does not involve any distribution of  $S$  or  $A$ . The algorithm in the tabular case aims to solve a set of equations for all  $(s, a)$  (Bellman optimality equation).
- Question: Can we use experience replay in tabular Q-learning?
  - Answer: Yes, we can. And more sample efficient (why?)

# Deep Q-learning

## 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  $\pi_b$ .

Store the experience samples generated by  $\pi_b$  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

Remarks:

- Why no policy update?
- Why not using the policy update equation that we derived?
- The network input and output are different from the DQN paper.

# Deep Q-learning

Illustrative 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.

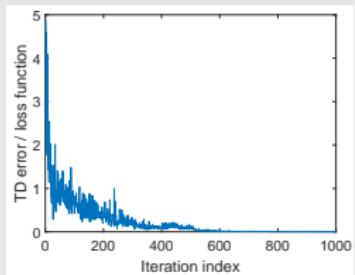
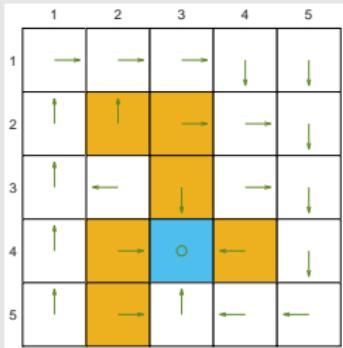
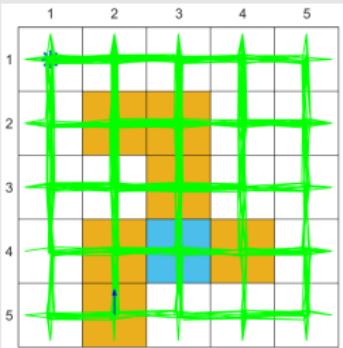
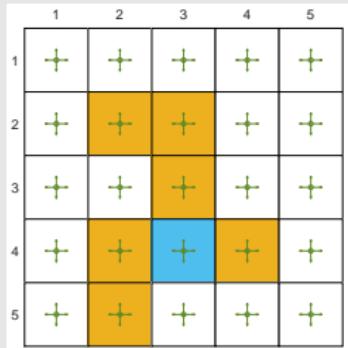
# Deep Q-learning

## Setup:

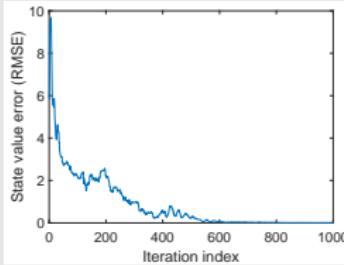
- One single episode is used to train the network.
- This episode is generated by an exploratory behavior policy shown in Figure (a).
- The episode only has 1,000 steps! The tabular Q-learning requires 100,000 steps.
- A shallow neural network with one single hidden layer is used as a nonlinear approximator of  $\hat{q}(s, a, w)$ . The hidden layer has 100 neurons.

See details in the book.

# Deep Q-learning



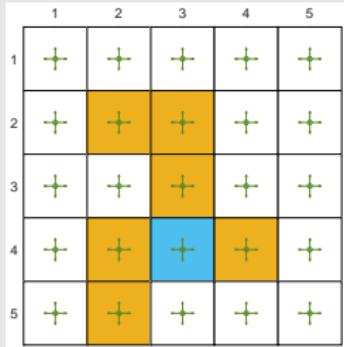
The TD error converges to zero.



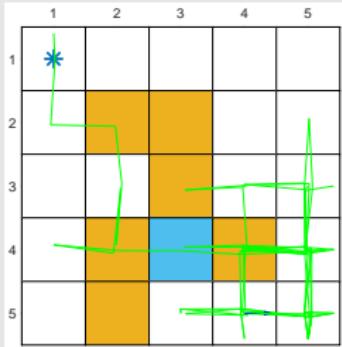
The state estimation error converges to zero.

## Deep Q-learning

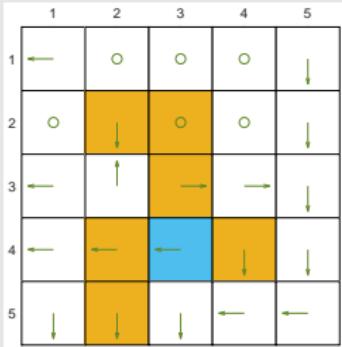
What if we only use a single episode of 100 steps? Insufficient data



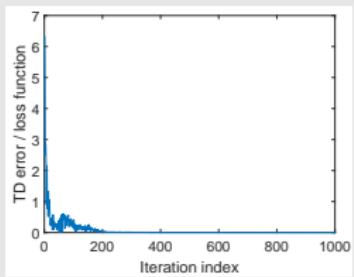
## The behavior policy.



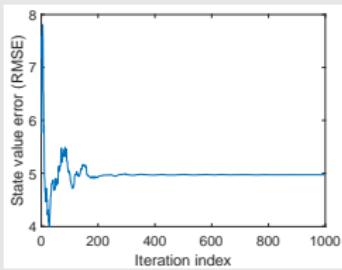
An episode of 100 steps.



## The final policy.



The TD error converges to zero.



The state error does not converge to zero.

# Outline

- 1 Motivating examples: curve fitting**
- 2 Algorithm for state value estimation**
  - Objective function
  - Optimization algorithms
  - Selection of function approximators
  - Illustrative examples
  - Summary of the story
  - Theoretical analysis
- 3 Sarsa with function approximation**
- 4 Q-learning with function approximation**
- 5 Deep Q-learning**
- 6 Summary**

# Summary

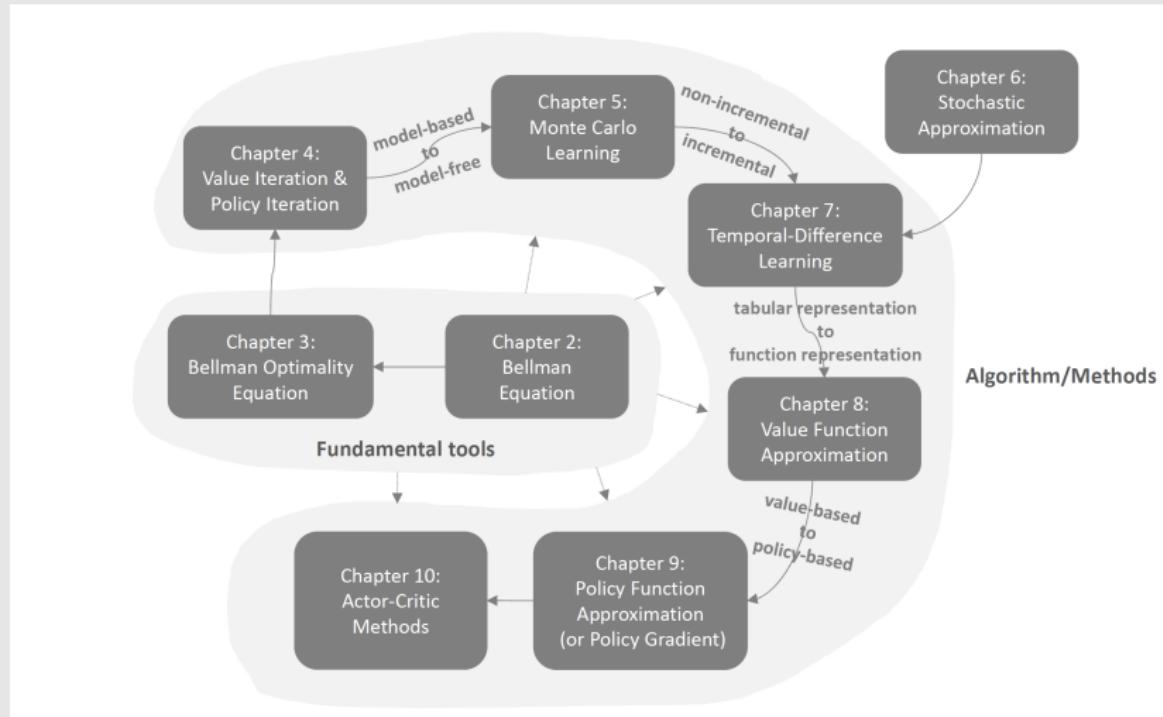
This lecture introduces the method of value function approximation.

- First, understand the basic idea.
- Second, understand the basic algorithms.

# Lecture 7: Temporal-Difference Learning

Shiyu Zhao

# Outline



# Introduction

- This lecture introduces temporal-difference (TD) learning, which is one of the most well-known methods in reinforcement learning (RL).
- Monte Carlo (MC) learning is the first model-free method. TD learning is the second model-free method. TD has some advantages compared to MC.
- We will see how the stochastic approximation methods studied in the last lecture are useful.

# Outline

- 1 Motivating examples
- 2 TD learning of state values
- 3 TD learning of action values: Sarsa
- 4 TD learning of action values: Expected Sarsa
- 5 TD learning of action values:  $n$ -step Sarsa
- 6 TD learning of optimal action values: Q-learning
- 7 A unified point of view
- 8 Summary

# Outline

- 1 Motivating examples
- 2 TD learning of state values
- 3 TD learning of action values: Sarsa
- 4 TD learning of action values: Expected Sarsa
- 5 TD learning of action values:  $n$ -step Sarsa
- 6 TD learning of optimal action values: Q-learning
- 7 A unified point of view
- 8 Summary

# Motivating example: stochastic algorithms

We next consider some stochastic problems and show how to use the RM algorithm to solve them.

**First, consider the simple mean estimation problem:** calculate

$$w = \mathbb{E}[X],$$

based on some iid samples  $\{x\}$  of  $X$ .

- By writing  $g(w) = w - \mathbb{E}[X]$ , we can reformulate the problem to a root-finding problem

$$g(w) = 0.$$

- Since we can only obtain samples  $\{x\}$  of  $X$ , the noisy observation is

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

- Then, according to the last lecture, we know the RM algorithm for solving  $g(w) = 0$  is

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

## Motivating example: stochastic algorithms

**Second, consider a little more complex problem.** That is to estimate the mean of a function  $v(X)$ ,

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

based on some iid random samples  $\{x\}$  of  $X$ .

- To solve this problem, we define

$$g(w) = w - \mathbb{E}[v(X)]$$

$$\tilde{g}(w, \eta) = w - v(x) = (w - \mathbb{E}[v(X)]) + (\mathbb{E}[v(X)] - v(x)) \doteq g(w) + \eta.$$

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

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

# Motivating example: stochastic algorithms

**Third, consider an even more complex problem:** calculate

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

where  $R, X$  are random variables,  $\gamma$  is a constant, and  $v(\cdot)$  is a function.

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

$$g(w) = w - \mathbb{E}[R + \gamma v(X)],$$

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

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

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

# Motivating example: stochastic algorithms

Quick summary:

- The above three examples are more and more complex.
- They can all be solved by the RM algorithm.
- We will see that the TD algorithms have similar expressions.

# Outline

- 1 Motivating examples
- 2 TD learning of state values
- 3 TD learning of action values: Sarsa
- 4 TD learning of action values: Expected Sarsa
- 5 TD learning of action values:  $n$ -step Sarsa
- 6 TD learning of optimal action values: Q-learning
- 7 A unified point of view
- 8 Summary

# TD learning of state values

Note that

- TD learning often refers to a broad class of RL algorithms.
- TD learning also refers to a specific algorithm for estimating state values as introduced below.

# TD learning of state values – Algorithm description

**The data/experience required by the algorithm:**

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

**The TD learning algorithm is**

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

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

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

- At time  $t$ , only the value of the visited state  $s_t$  is updated whereas the values of the unvisited states  $s \neq s_t$  remain unchanged.
- The update in (2) will be omitted when the context is clear.

# TD learning of state values – Algorithm properties

The TD algorithm can be annotated as

$$\underbrace{v_{t+1}(s_t)}_{\text{new estimate}} = \underbrace{v_t(s_t)}_{\text{current estimate}} - \alpha_t(s_t) \left[ \overbrace{v_t(s_t) - [r_{t+1} + \gamma v_t(s_{t+1})]}^{\text{TD error } \delta_t} \right], \quad (3)$$

TD target  $\bar{v}_t$

Here,

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

is called the **TD target**.

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

is called the **TD error**.

It is clear that the new estimate  $v_{t+1}(s_t)$  is a combination of the current estimate  $v_t(s_t)$  and the TD error.

# TD learning of state values – Algorithm properties

First, why is  $\bar{v}_t$  called the TD target?

That is because the algorithm drives  $v(s_t)$  towards  $\bar{v}_t$ .

To see that,

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

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 learning of state values – Algorithm properties

**Second, what is the interpretation of the TD error?**

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

- It is a **difference** between two consequent **time** steps.
- It reflects the **deficiency** between  $v_t$  and  $v_\pi$ . To see that, denote

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

Note that

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

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

# TD learning of state values – Algorithm properties

Other properties:

- The TD algorithm in (3) **only estimates the state value of a given policy.**
  - *It does not estimate the action values.*
  - *It does not search for optimal policies.*
- Later, we will see how to estimate action values and then search for optimal policies.
- Nonetheless, the TD algorithm in (3) is fundamental for understanding the core idea.

# TD learning of state values – The idea of the algorithm

**Q: What does this TD algorithm do mathematically?**

A: It solves the Bellman equation of a given policy  $\pi$ .

# TD learning of state values – The idea of the algorithm

**First, a new expression of the Bellman equation.**

The definition of state value of  $\pi$  is

$$v_\pi(s) = \mathbb{E}[R + \gamma G | S = s], \quad s \in \mathcal{S} \quad (4)$$

where  $G$  is discounted return. Since

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

where  $S'$  is the next state, we can rewrite (4) as

$$v_\pi(s) = \mathbb{E}[R + \gamma v_\pi(S') | S = s], \quad s \in \mathcal{S}. \quad (5)$$

Equation (5) is another expression of the Bellman equation. It is sometimes called the **Bellman expectation equation**, an important tool to design and analyze TD algorithms.

# TD learning of state values – The idea of the algorithm

**Second, solve the Bellman equation in (5) using the RM algorithm.**

In particular, by defining

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

we can rewrite (5) as

$$g(v(s)) = 0.$$

Since we can only obtain the samples  $r$  and  $s'$  of  $R$  and  $S'$ , the noisy observation we have is

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

## TD learning of state values – The idea of the algorithm

Therefore, the RM algorithm for solving  $g(v(s)) = 0$  is

$$\begin{aligned}v_{k+1}(s) &= v_k(s) - \alpha_k \tilde{g}(v_k(s)) \\&= v_k(s) - \alpha_k \left( v_k(s) - [\color{red}r_k + \gamma v_\pi(\color{blue}s'_k)] \right), \quad k = 1, 2, 3, \dots \quad (6)\end{aligned}$$

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

---

The RM algorithm in (6) has two assumptions that deserve special attention.

- We must have the experience set  $\{(s, r, s')\}$  for  $k = 1, 2, 3, \dots$ .
- We assume that  $v_\pi(s')$  is already known for any  $s'$ .

## TD learning of state values – The idea of the algorithm

Therefore, the RM algorithm for solving  $g(v(s)) = 0$  is

$$\begin{aligned}v_{k+1}(s) &= v_k(s) - \alpha_k \tilde{g}(v_k(s)) \\&= v_k(s) - \alpha_k \left( v_k(s) - [\textcolor{red}{r_k} + \gamma \textcolor{blue}{v}_\pi(\textcolor{red}{s'_k})] \right), \quad k = 1, 2, 3, \dots\end{aligned}$$

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

---

To remove the two assumptions in the RM algorithm, we can modify it.

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

# TD learning of state values – Algorithm 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  $t \rightarrow \infty$  if  $\sum_t \alpha_t(s) = \infty$  and  $\sum_t \alpha_t^2(s) < \infty$  for all  $s \in \mathcal{S}$ .

Remarks:

- This theorem says the state value can be found by the TD algorithm for a given policy  $\pi$ .
- $\sum_t \alpha_t(s) = \infty$  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  $\alpha$  is often selected as a small constant. In this case, the condition that  $\sum_t \alpha_t^2(s) < \infty$  is invalid anymore. When  $\alpha$  is constant, it can still be shown that the algorithm converges in the sense of expectation sense.

For the proof of the theorem, see my book.

# TD learning of state values – Algorithm properties

While TD learning and MC learning are both model-free, what are the **advantages and disadvantages** of TD learning compared to MC learning?

TD/Sarsa learning	MC learning
<b>Online:</b> TD learning is online. It can update the state/action values immediately after receiving a reward.	<b>Offline:</b> MC learning is offline. It has to wait until an episode has been completely collected.
<b>Continuing tasks:</b> Since TD learning is online, it can handle both episodic and continuing tasks.	<b>Episodic tasks:</b> Since MC learning is offline, it can only handle episodic tasks that have terminate states.

**Table:** Comparison between TD learning and MC learning.

## TD learning of state values – Algorithm properties

While TD learning and MC learning are both model-free, what are the **advantages and disadvantages** of TD learning compared to MC learning?

TD/Sarsa learning	MC learning
<b>Bootstrapping:</b> TD bootstraps because the update of a value relies on the previous estimate of this value. Hence, it requires initial guesses.	<b>Non-bootstrapping:</b> MC is not bootstrapping, because it can directly estimate state/action values without any initial guess.
<b>Low estimation variance:</b> TD has lower than MC because there are fewer random variables. For instance, Sarsa requires $R_{t+1}, S_{t+1}, A_{t+1}$ .	<b>High estimation variance:</b> To estimate $q_\pi(s_t, a_t)$ , we need samples of $R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$ . Suppose the length of each episode is $L$ . There are $ \mathcal{A} ^L$ possible episodes.

**Table:** Comparison between TD learning and MC learning (continued).

# Outline

- 1 Motivating examples
- 2 TD learning of state values
- 3 TD learning of action values: Sarsa**
- 4 TD learning of action values: Expected Sarsa
- 5 TD learning of action values:  $n$ -step Sarsa
- 6 TD learning of optimal action values: Q-learning
- 7 A unified point of view
- 8 Summary

## TD learning of action values – Sarsa

- The TD algorithm introduced in the last section **can only estimate state values.**
- Next, we introduce, Sarsa, an algorithm that **can directly estimate action values.**
- We will also see **how to use Sarsa to find optimal policies.**

# Sarsa – Algorithm

First, our aim is to estimate the action values of a given policy  $\pi$ .

Suppose we have some experience  $\{(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})\}_t$ .

We can use the following *Sarsa* algorithm to estimate the action values:

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

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

where  $t = 0, 1, 2, \dots$

- $q_t(s_t, a_t)$  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 – Algorithm

- **Why is this algorithm called Sarsa?** That is because each step of the algorithm involves  $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ . **Sarsa is the abbreviation of state-action-reward-state-action.**
- **What is the relationship between Sarsa and the previous TD learning algorithm?** We can obtain Sarsa by replacing the state value estimate  $v(s)$  in the TD algorithm with the action value estimate  $q(s, a)$ . As a result, **Sarsa is an action-value version of the TD algorithm.**
- **What does the Sarsa algorithm do mathematically?** The expression of Sarsa suggests that it is a stochastic approximation algorithm solving the following equation:

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

This is **another expression of the Bellman equation expressed in terms of action values**. The proof is given in my book.

# Sarsa – Algorithm

## 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  $t \rightarrow \infty$  for all  $(s, a)$  if  $\sum_t \alpha_t(s, a) = \infty$  and  $\sum_t \alpha_t^2(s, a) < \infty$  for all  $(s, a)$ .*

Remarks:

- This theorem says the action value can be found by Sarsa for a given a policy  $\pi$ .

# Sarsa – Implementation

The ultimate goal of RL is to find optimal policies.

To do that, we can **combine Sarsa with a policy improvement step**.

The combined algorithm is also called Sarsa.

## 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:*

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

*Update policy:*

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

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

## Remarks about this algorithm:

- 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**.
- The policy is  $\epsilon$ -greedy instead of greedy to well balance exploitation and exploration.

## Be clear about the core idea and complication:

- **The core idea is simple:** that is to use an algorithm to solve the Bellman equation of a given policy.
- **The complication emerges** when we try to find optimal policies and work efficiently.

# Sarsa – Examples

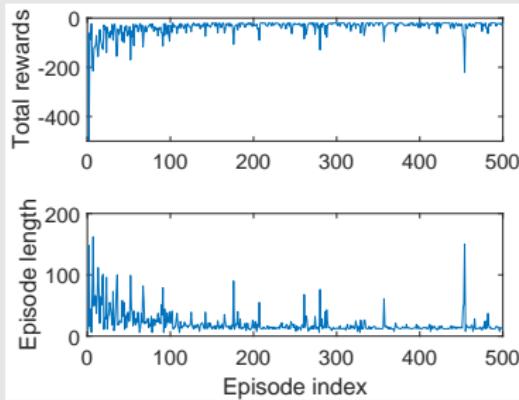
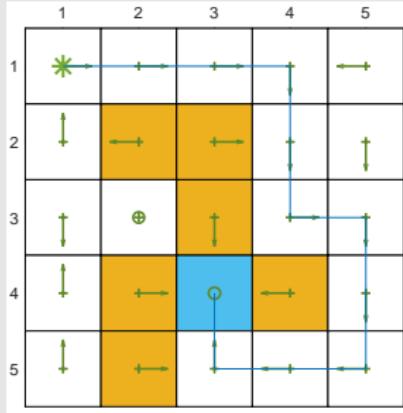
## Task description:

- The task is to find a good path **from a specific starting state to the target state.**
- This task is different from all the previous tasks where we need to find out the optimal policy for every state!
- Each episode starts from the top-left state and end in the target state.
- In the future, pay attention to what the task is.
- $r_{\text{target}} = 0$ ,  $r_{\text{forbidden}} = r_{\text{boundary}} = -10$ , and  $r_{\text{other}} = -1$ . The learning rate is  $\alpha = 0.1$  and the value of  $\epsilon$  is 0.1.

# Sarsa – Examples

## Results:

- 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.



# Outline

- 1 Motivating examples
- 2 TD learning of state values
- 3 TD learning of action values: Sarsa
- 4 TD learning of action values: Expected Sarsa
- 5 TD learning of action values:  $n$ -step Sarsa
- 6 TD learning of optimal action values: Q-learning
- 7 A unified point of view
- 8 Summary

# TD learning of action values: Expected Sarsa

A variant of Sarsa is the **Expected Sarsa** algorithm:

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

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

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  $\pi_t$ .

**Compared to Sarsa:**

- 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  $\{s_t, a_t, r_{t+1}, s_{t+1}\}$ .

# TD learning of action values: Expected Sarsa

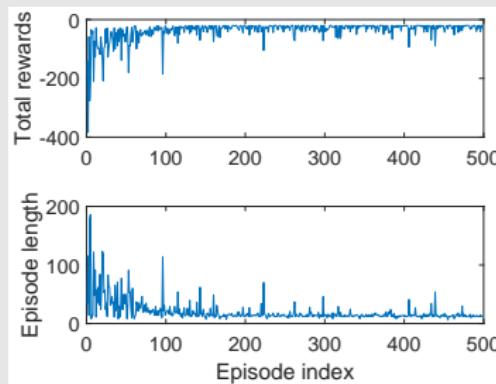
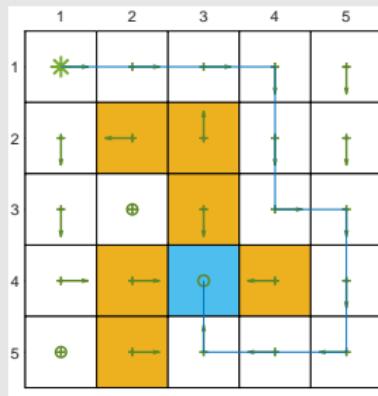
**What does the algorithm do mathematically?** Expected Sarsa is a stochastic approximation algorithm for solving the following equation:

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

The above equation is another expression of the Bellman equation:

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

Illustrative example:



# Outline

- 1 Motivating examples
- 2 TD learning of state values
- 3 TD learning of action values: Sarsa
- 4 TD learning of action values: Expected Sarsa
- 5 TD learning of action values:  $n$ -step Sarsa**
- 6 TD learning of optimal action values: Q-learning
- 7 A unified point of view
- 8 Summary

# TD learning of action values: $n$ -step Sarsa

$n$ -step Sarsa: can unify Sarsa and Monte Carlo 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

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

It should be noted that  $G_t = G_t^{(1)} = G_t^{(2)} = G_t^{(n)} = G_t^{(\infty)}$ , where the superscripts merely indicate the different decomposition structures of  $G_t$ .

# TD learning of action values: $n$ -step Sarsa

- 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

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

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

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

- The algorithm of  $n$ -step Sarsa is

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t)$$

$$- \alpha_t(s_t, a_t) \left[ q_t(s_t, a_t) - [r_{t+1} + \gamma r_{t+2} + \dots + \gamma^n q_t(s_{t+n}, a_{t+n})] \right].$$

$n$ -step Sarsa is more general because it becomes the (one-step) Sarsa algorithm when  $n = 1$  and the MC learning algorithm when  $n = \infty$ .

# TD learning of action values: $n$ -step Sarsa

- $n$ -step Sarsa needs  $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, \dots, r_{t+n}, s_{t+n}, a_{t+n})$ .
- Since  $(r_{t+n}, s_{t+n}, a_{t+n})$  has not been collected at time  $t$ , we are not able to implement  $n$ -step Sarsa at step  $t$ . However, we can wait until time  $t + n$  to update the q-value of  $(s_t, a_t)$ :

$$q_{t+n}(s_t, a_t) = q_{t+n-1}(s_t, a_t)$$

$$- \alpha_{t+n-1}(s_t, a_t) \left[ q_{t+n-1}(s_t, a_t) - [r_{t+1} + \gamma r_{t+2} + \dots + \gamma^n q_{t+n-1}(s_{t+n}, a_{t+n})] \right]$$

- 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.

# Outline

- 1 Motivating examples
- 2 TD learning of state values
- 3 TD learning of action values: Sarsa
- 4 TD learning of action values: Expected Sarsa
- 5 TD learning of action values:  $n$ -step Sarsa
- 6 TD learning of optimal action values: Q-learning
- 7 A unified point of view
- 8 Summary

## TD learning of optimal action values: Q-learning

- Next, we introduce Q-learning, one of the most widely used RL algorithms.
- Sarsa can estimate the action values of a given policy. It must be combined with a policy improvement step to find optimal policies.
- Q-learning can directly estimate optimal action values and hence optimal policies.

# Q-learning – Algorithm

**The Q-learning algorithm is**

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

Q-learning is very similar to Sarsa. They are different only in terms of the 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  $r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})$ .

# Q-learning – Algorithm

**What does Q-learning do mathematically?**

It aims to solve

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

This is the [Bellman optimality equation](#) expressed in terms of action values. See the proof in my book.

# Off-policy vs on-policy

Before further studying Q-learning, we first introduce two important concepts: **on-policy learning** and **off-policy learning**.

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.

# Off-policy vs on-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.

# Off-policy vs on-policy

## **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.

It deserves special attention because it is one of the most confusing problems to beginners.

# Off-policy vs on-policy

Sarsa is on-policy.

- First, Sarsa aims to solve the Bellman equation of a given policy  $\pi$ :

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

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

- Second, the algorithm is

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

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  $\pi_t(s_{t+1})$ !
- $\pi_t$  is both the target and behavior policy.

# Off-policy vs on-policy

Monte Carlo learning is on-policy.

- First, the MC method aims to solve

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

where the sample is generated following a given policy  $\pi$ .

- Second, the implementation of the MC method is

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

- 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.

# Off-policy vs on-policy

Q-learning is off-policy.

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

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

- 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 will converge to the optimal policy.

# Q-learning – Implementation

Since Q-learning is off-policy, it can be implemented in an either off-policy or on-policy fashion.

## 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:

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

Update policy:

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

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

See the book for more detailed pseudocode.

# Q-learning – Algorithm

## 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  $\pi_b$ , do

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

*Update q-value:*

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

*Update target policy:*

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

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

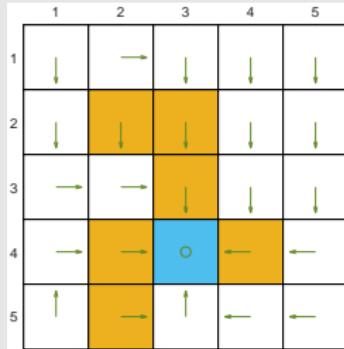
See the book for more detailed pseudocode.

# Q-learning – Examples

## Task description:

- 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  $\alpha = 0.1$ .

**Ground truth:** an optimal policy and the corresponding optimal state values.



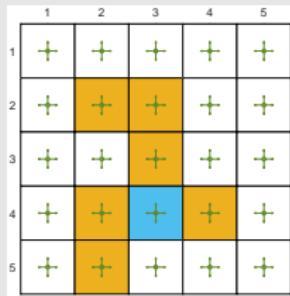
(a) Optimal policy

	1	2	3	4	5
1	5.8	5.6	6.2	6.5	5.8
2	6.5	7.2	8.0	7.2	6.5
3	7.2	8.0	10.0	8.0	7.2
4	8.0	10.0	10.0	10.0	8.0
5	7.2	9.0	10.0	9.0	8.1

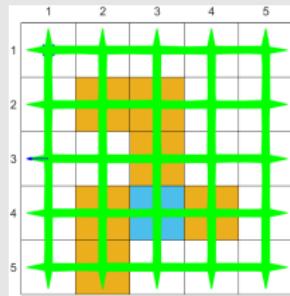
(b) Optimal state value

# Q-learning – Examples

The behavior policy and the generated experience ( $10^5$  steps):

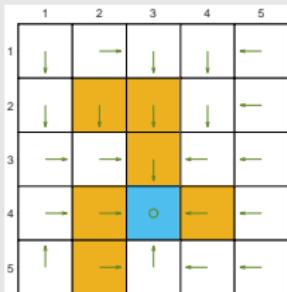


(a) Behavior policy

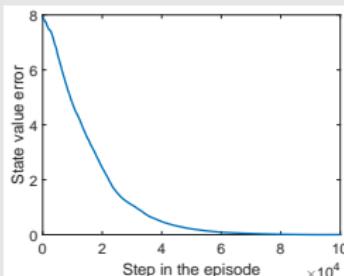


(b) Generated episode

The policy found by off-policy Q-learning:



(a) Estimated policy

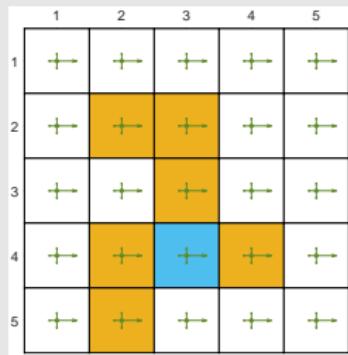


(b) State value error

# Q-learning – Examples

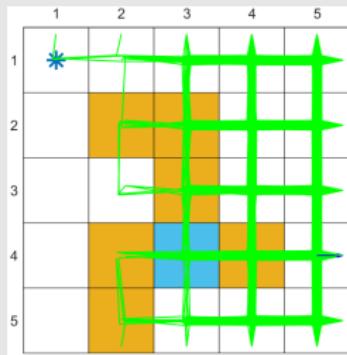
The importance of exploration: episodes of  $10^5$  steps

If the policy is not sufficiently exploratory, the samples are not good.

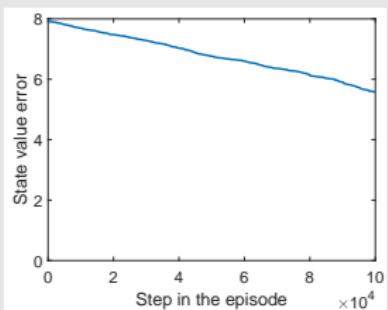


(a) Behavior policy

$$\epsilon = 0.5$$

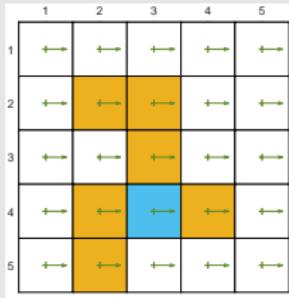


(b) Generated episode



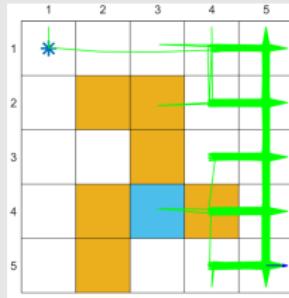
(c) Q-learning result

## Q-learning – Examples

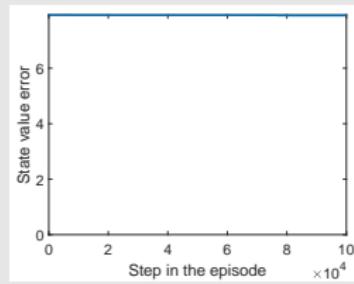


(a) Behavior policy

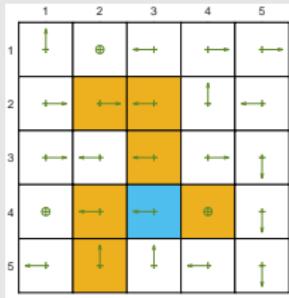
$$\epsilon = 0.1$$



(b) Generated episode

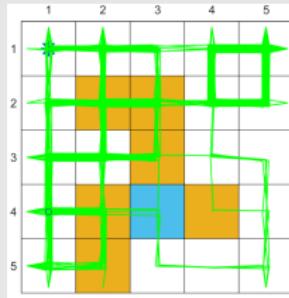


(c) Q-learning result

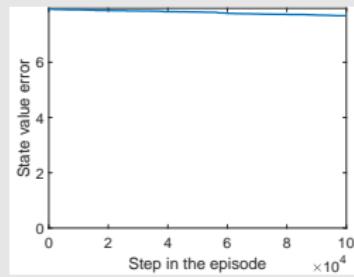


(a) Behavior policy

$$\epsilon = 0.1$$



(b) Generated episode



(c) Q-learning result

# Outline

- 1 Motivating examples
- 2 TD learning of state values
- 3 TD learning of action values: Sarsa
- 4 TD learning of action values: Expected Sarsa
- 5 TD learning of action values:  $n$ -step Sarsa
- 6 TD learning of optimal action values: Q-learning
- 7 A unified point of view
- 8 Summary

# A unified point of view

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	$\bar{q}_t = r_{t+1} + \gamma r_{t+2} + \dots + \gamma^n q_t(s_{t+n}, a_{t+n})$
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	$\bar{q}_t = r_{t+1} + \gamma r_{t+2} + \dots$

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

# A unified point of view

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: $q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})   S_t = s, A_t = a]$
$n$ -step Sarsa	BE: $q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \dots + \gamma^n q_\pi(s_{t+n}, a_{t+n})   S_t = s, A_t = a]$
Expected Sarsa	BE: $q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma \mathbb{E}_{A_{t+1}}[q_\pi(S_{t+1}, A_{t+1})]   S_t = s, A_t = a]$
Q-learning	BOE: $q(s, a) = \mathbb{E}[R_{t+1} + \max_a q(S_{t+1}, a)   S_t = s, A_t = a]$
Monte Carlo	BE: $q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \dots   S_t = s, A_t = a]$

# Outline

- 1 Motivating examples
- 2 TD learning of state values
- 3 TD learning of action values: Sarsa
- 4 TD learning of action values: Expected Sarsa
- 5 TD learning of action values:  $n$ -step Sarsa
- 6 TD learning of optimal action values: Q-learning
- 7 A unified point of view
- 8 Summary

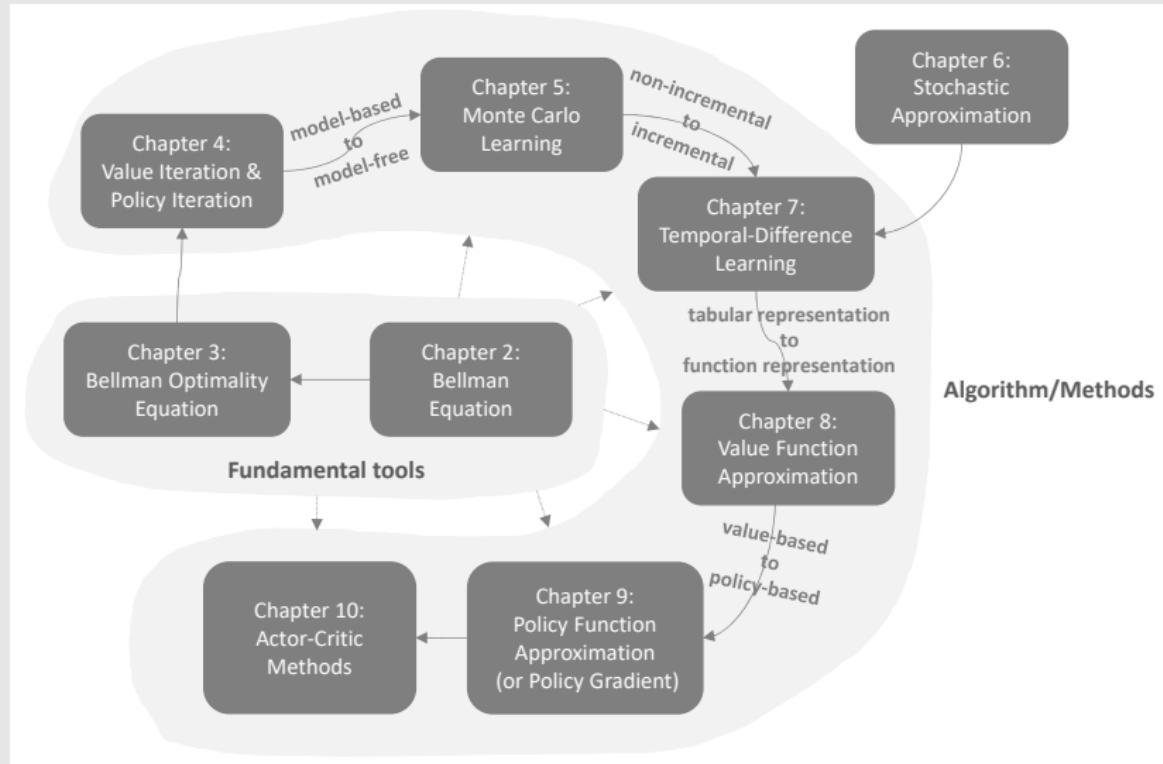
# Summary

- Introduced various TD learning algorithms
- Their expressions, math interpretations, implementation, relationship, examples
- Unified point of view

# Lecture 9: Policy Gradient Methods

Shiyu Zhao

# Introduction



# Introduction

In this lecture, we will move

- from value-based methods to policy-based methods
- from value function approximation to policy function approximation

# Outline

- 1** Basic idea of policy gradient
- 2** Metrics to define optimal policies
- 3** Gradients of the metrics
- 4** Gradient-ascent algorithm (REINFORCE)
- 5** Summary

# Outline

- 1 Basic idea of policy gradient
- 2 Metrics to define optimal policies
- 3 Gradients of the metrics
- 4 Gradient-ascent algorithm (REINFORCE)
- 5 Summary

# Basic idea of policy gradient

Previously, policies have been 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.

# Basic idea of policy gradient

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  $\theta$ .
- **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  $\pi_\theta(a, s)$ .

# Basic idea of policy gradient

## Differences between tabular and function representations:

- First, how to define optimal policies?
  - When represented as a table, a policy  $\pi$  is optimal if it can maximize *every state value*.
  - When represented by a function, a policy  $\pi$  is optimal if it can maximize certain *scalar metrics*.

# Basic idea of policy gradient

## Differences between tabular and function representations:

- 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.

# Basic idea of policy gradient

## Differences between tabular and function representations:

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

# Basic idea of policy gradient

**The basic idea of the policy gradient is simple:**

- 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?

These questions will be answered in detail in this lecture.

# Outline

- 1 Basic idea of policy gradient
- 2 Metrics to define optimal policies
- 3 Gradients of the metrics
- 4 Gradient-ascent algorithm (REINFORCE)
- 5 Summary

# Metrics to define optimal policies - 1) The average value

There are two metrics.

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  $S \sim d$ .

# Metrics to define optimal policies - 1) The average value

**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.

# Metrics to define optimal policies - 1) The average value

**How to select the distribution  $d$ ? There are two cases.**

The first case is that  $d$  is **independent** of the policy  $\pi$ .

- 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,

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

# Metrics to define optimal policies - 1) The average value

**How to select the distribution  $d$ ? There are two cases.**

The second case is that  $d$  depends on the policy  $\pi$ .

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

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

where  $P_\pi$  is the state transition probability matrix.

- The interpretation of selecting  $d_\pi$  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.

## Metrics to define optimal policies - 2) 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  $d_\pi$  is the stationary distribution.
- As its name suggests,  $\bar{r}_\pi$  is simply a weighted average of the one-step immediate rewards.

## Metrics to define optimal policies - 2) The average reward

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

$$\begin{aligned} & \lim_{n \rightarrow \infty} \frac{1}{n} \mathbb{E} \left[ R_{t+1} + R_{t+2} + \dots + R_{t+n} | S_t = s_0 \right] \\ &= \lim_{n \rightarrow \infty} \frac{1}{n} \mathbb{E} \left[ \sum_{k=1}^n R_{t+k} | S_t = s_0 \right] \end{aligned}$$

where  $s_0$  is the starting state of the trajectory.

# Metrics to define optimal policies - Remarks

An important property is that

$$\begin{aligned}\lim_{n \rightarrow \infty} \frac{1}{n} \mathbb{E} \left[ \sum_{k=1}^n R_{t+k} | S_t = s_0 \right] &= \lim_{n \rightarrow \infty} \frac{1}{n} \mathbb{E} \left[ \sum_{k=1}^n R_{t+k} \right] \\ &= \sum_s d_\pi(s) r_\pi(s) \\ &= \bar{r}_\pi\end{aligned}$$

Note that

- The starting state  $s_0$  does not matter.
- The two definitions of  $\bar{r}_\pi$  are equivalent.

See the proof in the book.

# Metrics to define optimal policies - Remarks

## Remark 1 about the metrics:

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

This is the basic idea of policy gradient methods.

# Metrics to define optimal policies - Remarks

## Remark 2 about the metrics:

- One complication is that the metrics can be defined in either the **discounted case** where  $\gamma \in (0, 1)$  or the **undiscounted case** where  $\gamma = 1$ .
- We only consider the discounted case so far in this book. For details about the undiscounted case, see the book.

# Metrics to define optimal policies - Remarks

## Remark 3 about the metrics:

- 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

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

See the proof in the book.

# Metrics to define optimal policies - Excise

## Excise:

You will see the following metric often in the literature:

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

What is its relationship to the metrics we introduced just now?

# Metrics to define optimal policies - Excise

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

**Answer:** First, clarify and understand this metric.

- It starts from  $S_0 \sim d$  and then  $A_0, R_1, S_1, A_1, R_2, S_2, \dots$
- $A_t \sim \pi(S_t)$  and  $R_{t+1}, S_{t+1} \sim p(R_{t+1}|S_t, A_t), p(S_{t+1}|S_t, A_t)$

Then, we know this metric is the same as the average value because

$$\begin{aligned} J(\theta) &= \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t R_{t+1} \right] = \sum_{s \in \mathcal{S}} d(s) \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t R_{t+1} | S_0 = s \right] \\ &= \sum_{s \in \mathcal{S}} d(s) v_{\pi}(s) \\ &= \bar{v}_{\pi} \end{aligned}$$

# Outline

- 1 Basic idea of policy gradient
- 2 Metrics to define optimal policies
- 3 Gradients of the metrics
- 4 Gradient-ascent algorithm (REINFORCE)
- 5 Summary

# Gradients of the metrics

Given a metric, we next

- derive its gradient
- and then, apply gradient-based methods to optimize the metric.

The gradient calculation is one of **the most complicated parts** of policy gradient methods! That is because

- first, we need to **distinguish different metrics**  $\bar{v}_\pi$ ,  $\bar{r}_\pi$ ,  $\bar{v}_\pi^0$
- second, we need to **distinguish the discounted and undiscounted cases**.

The calculation of the gradients:

- We will not discuss the details in this lecture.
- Interested readers may see my book for details.

# 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  $\bar{v}_{\pi}^0$ .
- “=” may denote strict equality, approximation, or proportional to.
- $\eta$  is a distribution or weight of the states.

# Gradients of the metrics

Some specific results:

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

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

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

Details are not given here. Interested readers can read my book.

# Gradients of the metrics

**A compact and useful form of the gradient:**

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

where  $S \sim \eta$  and  $A \sim \pi(A|S, \theta)$ .

**Why is this expression useful?**

- Because we can use samples to approximate the gradient!

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

# Gradients of the metrics

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

**How to prove the above equation?**

Consider the function  $\ln \pi$  where  $\ln$  is the natural logarithm. It is easy to see that

$$\nabla_{\theta} \ln \pi(a|s, \theta) = \frac{\nabla_{\theta} \pi(a|s, \theta)}{\pi(a|s, \theta)}$$

and hence

$$\nabla_{\theta} \pi(a|s, \theta) = \pi(a|s, \theta) \nabla_{\theta} \ln \pi(a|s, \theta).$$

# Gradients of the metrics

Then, we have

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

# Gradients of the metrics

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

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

- This can be achieved by using softmax functions that can normalize the entries in a vector from  $(-\infty, +\infty)$  to  $(0, 1)$ .
- For example, for any vector  $x = [x_1, \dots, x_n]^T$ ,

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

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.

# Gradients of the metrics

## Some remarks:

- Such a form based on the softmax function can be realized by a neural network whose input is  $s$  and parameter is  $\theta$ . 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.

# Outline

- 1 Basic idea of policy gradient
- 2 Metrics to define optimal policies
- 3 Gradients of the metrics
- 4 Gradient-ascent algorithm (REINFORCE)
- 5 Summary

# Gradient-ascent algorithm

Now, we are ready to present **the first policy gradient algorithm** to find optimal policies!

- The gradient-ascent algorithm maximizing  $J(\theta)$  is

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

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

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

# Gradient-ascent algorithm

- Furthermore, since  $q_\pi$  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

# Gradient-ascent algorithm

## Remark 1: How to do sampling?

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

- How to sample  $S$ ?
  - $S \sim d$ , where the distribution  $d$  is a long-run behavior under  $\pi$ .
- 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**.

# Gradient-ascent algorithm

**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:

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

# Gradient-ascent algorithm

It is a gradient-ascent algorithm for maximizing  $\pi(a_t|s_t, \theta)$ :

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

**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  $\beta_t$  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

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

# Gradient-ascent algorithm

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

The coefficient  $\beta_t$  can well balance exploration and exploitation.

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

# REINFORCE algorithm

Recall that

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

is replaced by

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

where  $q_t(s_t, a_t)$  is an approximation of  $q_\pi(s_t, a_t)$ .

- If  $q_\pi(s_t, a_t)$  is approximated by Monte Carlo estimation, the algorithm has a specific name, REINFORCE.
- REINFORCE is one of earliest and simplest policy gradient algorithms.
- Many other policy gradient algorithms such as the actor-critic methods can be obtained by extending REINFORCE (next lecture).

# REINFORCE algorithm

## 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  $k$ th iteration, do

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

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

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

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

$\theta_k = \theta_T$

# Outline

- 1 Basic idea of policy gradient
- 2 Metrics to define optimal policies
- 3 Gradients of the metrics
- 4 Gradient-ascent algorithm (REINFORCE)
- 5 Summary

# Summary

Contents of this lecture:

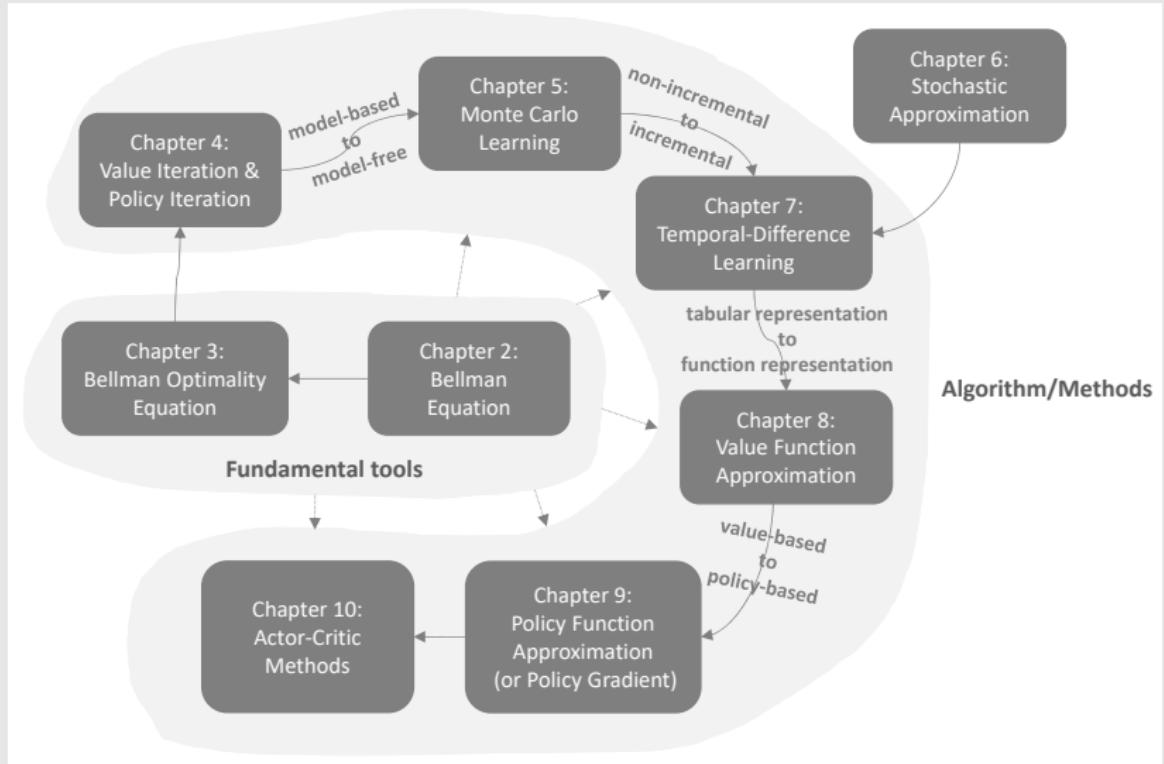
- Metrics for optimality
- Gradients of the metrics
- Gradient-ascent algorithm
- A special case: REINFORCE

Next lecture: Actor-critic

# Lecture 10: Actor-Critic Methods

Shiyu Zhao

# Introduction



# Introduction

Actor-critic methods are still policy gradient methods.

- They emphasize the structure that incorporates the policy gradient and value-based 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.

# Outline

- 1 The simplest actor-critic (QAC)**
- 2 Advantage actor-critic (A2C)**
  - Baseline invariance
  - The algorithm of advantage actor-critic
- 3 Off-policy actor-critic**
  - Illustrative examples
  - Importance sampling
  - The theorem of off-policy policy gradient
  - The algorithm of off-policy actor-critic
- 4 Deterministic actor-critic (DPG)**
  - The theorem of deterministic policy gradient
  - The algorithm of deterministic actor-critic

# Outline

- 1 The simplest actor-critic (QAC)
- 2 Advantage actor-critic (A2C)
  - Baseline invariance
  - The algorithm of advantage actor-critic
- 3 Off-policy actor-critic
  - Illustrative examples
  - Importance sampling
  - The theorem of off-policy policy gradient
  - The algorithm of off-policy actor-critic
- 4 Deterministic actor-critic (DPG)
  - The theorem of deterministic policy gradient
  - The algorithm of deterministic actor-critic

# The simplest actor-critic

Revisit the idea of policy gradient introduced in the last lecture.

- 1) A scalar metric  $J(\theta)$ , which can be  $\bar{v}_\pi$  or  $\bar{r}_\pi$ .
- 2) The gradient-ascent algorithm maximizing  $J(\theta)$  is

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

- 3) The stochastic gradient-ascent algorithm is

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

We can see “actor” and “critic” from this algorithm:

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

# The simplest actor-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](#).
  - We will introduce in this lecture.

# The simplest actor-critic

## 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):**

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

**Actor (policy update):**

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

# The simplest actor-critic

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  $\varepsilon$ -greedy.
- This particular actor-critic algorithm is sometimes referred to as **Q Actor-Critic (QAC)**.
- Though simple, this algorithm reveals the core idea of actor-critic methods. It can be extended to generate many other algorithms as shown later.

# Outline

- 1 The simplest actor-critic (QAC)
- 2 Advantage actor-critic (A2C)
  - Baseline invariance
  - The algorithm of advantage actor-critic
- 3 Off-policy actor-critic
  - Illustrative examples
  - Importance sampling
  - The theorem of off-policy policy gradient
  - The algorithm of off-policy actor-critic
- 4 Deterministic actor-critic (DPG)
  - The theorem of deterministic policy gradient
  - The algorithm of deterministic actor-critic

# Introduction

Next, we extend QAC to advantage actor-critic (A2C)

- The core idea is to introduce a baseline to reduce variance.

# Outline

- 1 The simplest actor-critic (QAC)
- 2 Advantage actor-critic (A2C)
  - Baseline invariance
  - The algorithm of advantage actor-critic
- 3 Off-policy actor-critic
  - Illustrative examples
  - Importance sampling
  - The theorem of off-policy policy gradient
  - The algorithm of off-policy actor-critic
- 4 Deterministic actor-critic (DPG)
  - The theorem of deterministic policy gradient
  - The algorithm of deterministic actor-critic

# Baseline invariance

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

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

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

Next, we answer two questions:

- Why is it valid?
- Why is it useful?

# Baseline invariance

**First, why is it valid?**

That is because

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

The details:

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

# Baseline invariance

## Second, why is the baseline useful?

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

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

We have

- $\mathbb{E}[X]$  is invariant to  $b(S)$ .
- $\text{var}(X)$  is NOT invariant to  $b(S)$ .
  - Why? Because  $\text{tr}[\text{var}(X)] = \mathbb{E}[X^T X] - \bar{x}^T \bar{x}$  and

$$\begin{aligned}\mathbb{E}[X^T X] &= \mathbb{E} [(\nabla_{\theta} \ln \pi)^T (\nabla_{\theta} \ln \pi)(q_{\pi}(S, A) - b(S))^2] \\ &= \mathbb{E} [\|\nabla_{\theta} \ln \pi\|^2 (q_{\pi}(S, A) - b(S))^2]\end{aligned}$$

Imagine  $b$  is huge (e.g., 1 million)

# Baseline invariance

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

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

In the algorithms of REINFORCE and QAC,

- There is no baseline.
- Or, we can say  $b = 0$ , **which is not guaranteed to be a good baseline.**

# Baseline invariance

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

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

See the proof in my book.

- Although this baseline is optimal, it is complex.
- We can remove the weight  $\|\nabla_{\theta} \ln \pi(A|s, \theta_t)\|^2$  and select the suboptimal baseline:

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

which is the state value of  $s$ !

# Outline

- 1 The simplest actor-critic (QAC)
- 2 Advantage actor-critic (A2C)
  - Baseline invariance
  - The algorithm of advantage actor-critic
- 3 Off-policy actor-critic
  - Illustrative examples
  - Importance sampling
  - The theorem of off-policy policy gradient
  - The algorithm of off-policy actor-critic
- 4 Deterministic actor-critic (DPG)
  - The theorem of deterministic policy gradient
  - The algorithm of deterministic actor-critic

# The algorithm of advantage actor-critic

When  $b(s) = v_\pi(s)$ ,

- the gradient-ascent algorithm is

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

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

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

# The algorithm of advantage actor-critic

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  $\delta_t$**  rather than the **absolute value  $q_t$** , which is more reasonable.
- It can still **well balance exploration and exploitation**.

# The algorithm of advantage actor-critic

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

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

- This approximation is **reasonable** because

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

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

# The algorithm of advantage actor-critic

## 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):**

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

It is on-policy. Since the policy  $\pi(\theta_t)$  is stochastic, no need to use techniques like  $\varepsilon$ -greedy.

# Outline

- 1 The simplest actor-critic (QAC)
- 2 Advantage actor-critic (A2C)
  - Baseline invariance
  - The algorithm of advantage actor-critic
- 3 Off-policy actor-critic
  - Illustrative examples
  - Importance sampling
  - The theorem of off-policy policy gradient
  - The algorithm of off-policy actor-critic
- 4 Deterministic actor-critic (DPG)
  - The theorem of deterministic policy gradient
  - The algorithm of deterministic actor-critic

# Introduction

- 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.

# Outline

- 1 The simplest actor-critic (QAC)
- 2 Advantage actor-critic (A2C)
  - Baseline invariance
  - The algorithm of advantage actor-critic
- 3 Off-policy actor-critic
  - Illustrative examples
  - Importance sampling
  - The theorem of off-policy policy gradient
  - The algorithm of off-policy actor-critic
- 4 Deterministic actor-critic (DPG)
  - The theorem of deterministic policy gradient
  - The algorithm of deterministic actor-critic

## Illustrative examples

Consider a random variable  $X \in \mathcal{X} = \{+1, -1\}$ .

If the probability distribution of  $X$  is  $p_0$ :

$$p_0(X = +1) = 0.5, \quad p_0(X = -1) = 0.5$$

then the expectation of  $X$  is

$$\mathbb{E}_{X \sim p_0}[X] = (+1) \cdot 0.5 + (-1) \cdot 0.5 = 0.$$

**Question: how to estimate  $\mathbb{E}[X]$  by using some samples  $\{x_i\}$ ?**

# Illustrative examples

## Case 1 (we are already familiar):

- The samples  $\{x_i\}$  are generated according to  $p_0$ :

$$\mathbb{E}[x_i] = \mathbb{E}[X], \quad \text{var}[x_i] = \text{var}[X]$$

Then, the average value can converge to the expectation:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i \rightarrow \mathbb{E}[X], \quad \text{as } n \rightarrow \infty$$

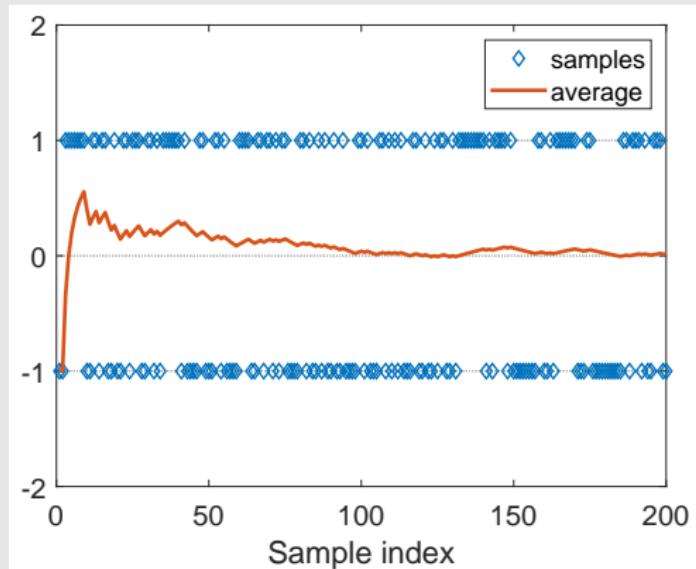
because

$$\mathbb{E}[\bar{x}] = \mathbb{E}[X], \quad \text{var}[\bar{x}] = \frac{1}{n} \text{var}[X]$$

See my book for details (Law of large numbers).

# Illustrative examples

Figure: Samples and  $\bar{x} \rightarrow \mathbb{E}[X]$



# Illustrative examples

## Case 2 (a new case that we want to study):

- The samples  $\{x_i\}$  are generated according to another distribution  $p_1$ :

$$p_1(X = +1) = 0.8, \quad p_1(X = -1) = 0.2$$

The expectation is

$$\mathbb{E}_{X \sim p_1}[X] = (+1) \cdot 0.8 + (-1) \cdot 0.2 = 0.6$$

If we use the average of the samples, then without suprising

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i \rightarrow \mathbb{E}_{X \sim p_1}[X] = 0.6 \neq \mathbb{E}_{X \sim p_0}[X]$$

# Illustrative examples

**Question:** Can we use  $\{x_i\} \sim p_1$  to estimate  $\mathbb{E}_{X \sim p_0}[X]$ ?

- **Why to do that?**

We may want to estimate  $\mathbb{E}_{A \sim \pi}[*]$  where  $\pi$  is the *target policy* based on the samples of a *behavior policy*  $\beta$ .

- **How to do that?**

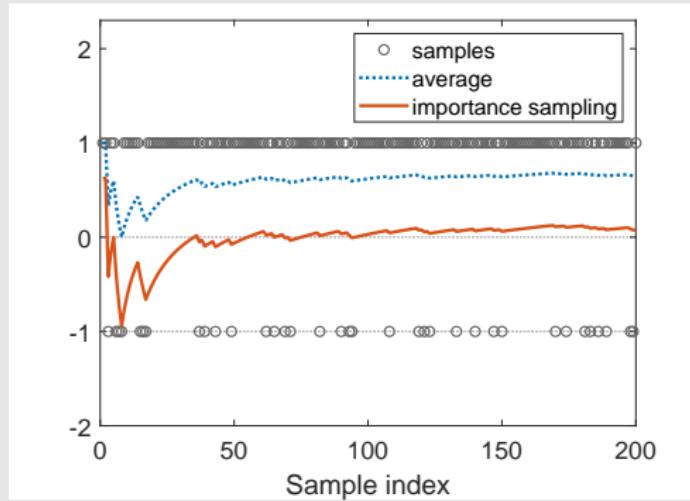
- We can't achieve that if directly using  $\bar{x}$ :

$$\bar{x} \rightarrow \mathbb{E}_{X \sim p_1}[X] = 0.6 \neq \mathbb{E}_{X \sim p_0}[X]$$

- We can achieve that by using the importance sampling technique.

# Illustrative examples

Figure: Samples and  $\bar{x} \rightarrow \mathbb{E}_{X \sim p_1}[X]$  (the dotted line)



# Outline

- 1 The simplest actor-critic (QAC)
- 2 Advantage actor-critic (A2C)
  - Baseline invariance
  - The algorithm of advantage actor-critic
- 3 Off-policy actor-critic
  - Illustrative examples
  - **Importance sampling**
  - The theorem of off-policy policy gradient
  - The algorithm of off-policy actor-critic
- 4 Deterministic actor-critic (DPG)
  - The theorem of deterministic policy gradient
  - The algorithm of deterministic actor-critic

# Importance sampling

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

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

Then,

$$\mathbb{E}_{X \sim p_1}[\bar{f}] = \mathbb{E}_{X \sim p_1}[f(X)]$$

$$\text{var}_{X \sim p_1}[\bar{f}] = \frac{1}{n} \text{var}_{X \sim p_1}[f(X)]$$

# Importance sampling

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.

# Importance sampling

**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.

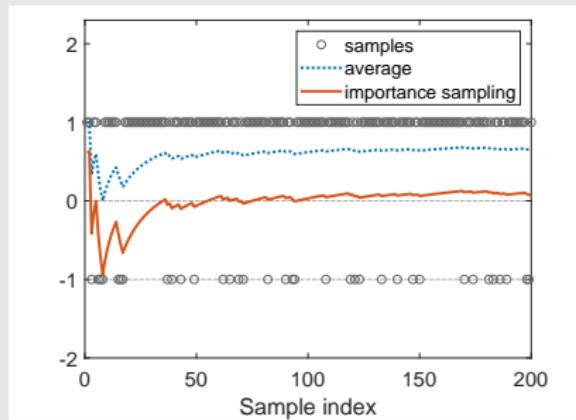
- For example, continuous case, complex expression of  $p_0$ , or no expression of  $p_0$  (e.g.,  $p_0$  represented by a neural network).

# Importance sampling

**Summary:** if  $\{x_i\} \sim p_1$ ,

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i \rightarrow \mathbb{E}_{X \sim p_1}[X]$$

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



**Figure:** Blue dotted line:  $\bar{x}$ ; Red solid line:  $\bar{f}$

# Outline

- 1 The simplest actor-critic (QAC)
- 2 Advantage actor-critic (A2C)
  - Baseline invariance
  - The algorithm of advantage actor-critic
- 3 Off-policy actor-critic
  - Illustrative examples
  - Importance sampling
  - The theorem of off-policy policy gradient
  - The algorithm of off-policy actor-critic
- 4 Deterministic actor-critic (DPG)
  - The theorem of deterministic policy gradient
  - The algorithm of deterministic actor-critic

# The theorem of off-policy policy gradient

Like the previous on-policy case, we need to derive the policy gradient in the off-policy case.

- Suppose  $\beta$  is the behavior policy that generates experience samples.
- Our aim is to use these samples to update a target policy  $\pi$  that can minimize the metric

$$J(\theta) = \sum_{s \in \mathcal{S}} d_\beta(s) v_\pi(s) = \mathbb{E}_{S \sim d_\beta} [v_\pi(S)]$$

where  $d_\beta$  is the stationary distribution under policy  $\beta$ .

# The theorem of 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  $\beta$  is the behavior policy and  $\rho$  is a state distribution.*

*See the details and the proof in my book.*

# Outline

- 1 The simplest actor-critic (QAC)
- 2 Advantage actor-critic (A2C)
  - Baseline invariance
  - The algorithm of advantage actor-critic
- 3 Off-policy actor-critic
  - Illustrative examples
  - Importance sampling
  - The theorem of off-policy policy gradient
  - The algorithm of off-policy actor-critic
- 4 Deterministic actor-critic (DPG)
  - The theorem of deterministic policy gradient
  - The algorithm of deterministic actor-critic

# The algorithm of off-policy actor-critic

The off-policy policy gradient is also **invariant to a baseline  $b(s)$** .

- In particular, we have

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{S \sim \rho, A \sim \beta} \left[ \frac{\pi(A|S, \theta)}{\beta(A|S)} \nabla_{\theta} \ln \pi(A|S, \theta) (q_{\pi}(S, A) - b(S)) \right]$$

- To reduce the estimation variance, we can select the baseline as  $b(S) = v_{\pi}(S)$  and obtain

$$\nabla_{\theta} J(\theta) = \mathbb{E} \left[ \frac{\pi(A|S, \theta)}{\beta(A|S)} \nabla_{\theta} \ln \pi(A|S, \theta) (q_{\pi}(S, A) - v_{\pi}(S)) \right]$$

# The algorithm of off-policy actor-critic

The corresponding stochastic gradient-ascent algorithm is

$$\theta_{t+1} = \theta_t + \alpha_\theta \frac{\pi(a_t|s_t, \theta_t)}{\beta(a_t|s_t)} \nabla_\theta \ln \pi(a_t|s_t, \theta_t) (q_t(s_t, a_t) - v_t(s_t))$$

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

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

# The algorithm of off-policy actor-critic

## 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  $\theta_0$  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):**

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

# Outline

- 1** The simplest actor-critic (QAC)
- 2** Advantage actor-critic (A2C)
  - Baseline invariance
  - The algorithm of advantage actor-critic
- 3** Off-policy actor-critic
  - Illustrative examples
  - Importance sampling
  - The theorem of off-policy policy gradient
  - The algorithm of off-policy actor-critic
- 4** Deterministic actor-critic (DPG)
  - The theorem of deterministic policy gradient
  - The algorithm of deterministic actor-critic

# Introduction

Up to now, the policies used in the policy gradient methods are all **stochastic** since  $\pi(a|s, \theta) > 0$  for every  $(s, a)$ .

Can we use **deterministic policies** in the policy gradient methods?

- Benefit: it can handle continuous action.

# Introduction

The ways to represent a policy:

- 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

$$a = \mu(s, \theta) \doteq \mu(s)$$

- $\mu$  is a mapping from  $\mathcal{S}$  to  $\mathcal{A}$ .
- $\mu$  can be represented by, for example, a neural network with the input as  $s$ , the output as  $a$ , and the parameter as  $\theta$ .
- We may write  $\mu(s, \theta)$  in short as  $\mu(s)$ .

# Outline

- 1** The simplest actor-critic (QAC)
- 2** Advantage actor-critic (A2C)
  - Baseline invariance
  - The algorithm of advantage actor-critic
- 3** Off-policy actor-critic
  - Illustrative examples
  - Importance sampling
  - The theorem of off-policy policy gradient
  - The algorithm of off-policy actor-critic
- 4** Deterministic actor-critic (DPG)
  - The theorem of deterministic policy gradient
  - The algorithm of deterministic actor-critic

# The theorem of deterministic policy gradient

- The policy gradient theorems introduced before are merely valid for stochastic policies.
- If the policy must be deterministic, we must derive a new policy gradient theorem.
- The ideas and procedures are similar.

# The theorem of deterministic policy gradient

Consider the metric of average state value in the discounted case:

$$J(\theta) = \mathbb{E}[v_\mu(s)] = \sum_{s \in \mathcal{S}} d_0(s)v_\mu(s)$$

where  $d_0(s)$  is a probability distribution satisfying  $\sum_{s \in \mathcal{S}} d_0(s) = 1$ .

- $d_0$  is selected to be **independent** of  $\mu$ . The gradient in this case is easier to calculate.
- There are two special yet important cases of selecting  $d_0$ .
  - The first special case is that  $d_0(s_0) = 1$  and  $d_0(s \neq s_0) = 0$ , where  $s_0$  is a specific starting state of interest.
  - The second special case is that  $d_0$  is the stationary distribution of a behavior policy that is different from the  $\mu$ .

# The theorem of 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*

$$\begin{aligned}\nabla_{\theta} J(\theta) &= \sum_{s \in \mathcal{S}} \rho_{\mu}(s) \nabla_{\theta} \mu(s) (\nabla_a q_{\mu}(s, a))|_{a=\mu(s)} \\ &= \mathbb{E}_{S \sim \rho_{\mu}} [\nabla_{\theta} \mu(S) (\nabla_a q_{\mu}(S, a))|_{a=\mu(S)}]\end{aligned}$$

Here,  $\rho_{\mu}$  is a state distribution.

See more details and the proof in my book.

**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.

# Outline

- 1** The simplest actor-critic (QAC)
- 2** Advantage actor-critic (A2C)
  - Baseline invariance
  - The algorithm of advantage actor-critic
- 3** Off-policy actor-critic
  - Illustrative examples
  - Importance sampling
  - The theorem of off-policy policy gradient
  - The algorithm of off-policy actor-critic
- 4** Deterministic actor-critic (DPG)
  - The theorem of deterministic policy gradient
  - The algorithm of deterministic actor-critic

# The algorithm of deterministic actor-critic

Based on the policy gradient, the gradient-ascent algorithm for maximizing  $J(\theta)$  is:

$$\theta_{t+1} = \theta_t + \alpha_\theta \mathbb{E}_{S \sim \rho_\mu} [\nabla_\theta \mu(S) (\nabla_a q_\mu(S, a))|_{a=\mu(S)}]$$

The corresponding stochastic gradient-ascent algorithm is

$$\theta_{t+1} = \theta_t + \alpha_\theta \nabla_\theta \mu(s_t) (\nabla_a q_\mu(s_t, a))|_{a=\mu(s_t)}$$

# The algorithm of deterministic actor-critic

## Deterministic actor-critic algorithm

**Initialization:** A given behavior policy  $\beta(a|s)$ . A deterministic target policy  $\mu(s, \theta_0)$  where  $\theta_0$  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  $\beta$  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):**

$$\theta_{t+1} = \theta_t + \alpha_\theta \nabla_\theta \mu(s_t, \theta_t) (\nabla_a q(s_t, a, w_{t+1}))|_{a=\mu(s_t)}$$

# The algorithm of deterministic actor-critic

Remarks:

- This is an off-policy implementation where the behavior policy  $\beta$  may be different from  $\mu$ .
- $\beta$  can also be replaced by  $\mu + \text{noise}$ .
- 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.

# Summary

- The simplest actor-critic
- Advantage actor-critic
- Off-policy actor-critic
- Deterministic actor-critic

# The end

This is the end of the course, but a start for your journey  
in the field of reinforcement learning!