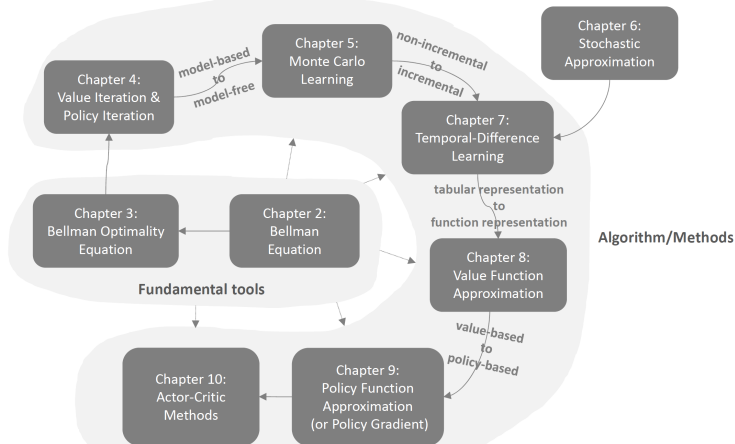


Lecture 7: Temporal-Difference Learning

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, γ 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 π .

The TD learning algorithm is

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

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

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

$$\underbrace{v_{t+1}(s_t)}_{\text{new estimate}} = \underbrace{v_t(s_t)}_{\text{current estimate}} - \alpha_t(s_t) \left[\overbrace{v_t(s_t) - [r_{t+1} + \gamma v_t(s_{t+1})]}^{\text{TD error } \delta_t} \right], \quad (3)$$

$\underbrace{[r_{t+1} + \gamma v_t(s_{t+1})]}_{\text{TD target } \bar{v}_t}$

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 \bar{v}_t is a *target value* that the algorithm attempts to drive $v(s_t)$ to approach.

To see that, we consider the simple case where α and \bar{v} are both constant.

Then, the TD algorithm becomes

$$v_{t+1} = v_t - \alpha(v_t - \bar{v}) = (1 - \alpha)v_t + \alpha\bar{v}.$$

It can be shown that $v_t \rightarrow \bar{v}$ as $t \rightarrow \infty$ if $0 < \alpha < 1$.

Proof.

To prove that, let $\Delta_t = v_t - \bar{v}$. Then the above equation becomes

$\Delta_{t+1} = (1 - \alpha)\Delta_t$. Then,

$\Delta_{t+1} = (1 - \alpha)\Delta_t = (1 - \alpha)^2\Delta_{t-1} = \dots = (1 - \alpha)^t\Delta_1$. Since $(1 - \alpha)^t \rightarrow 0$ as $t \rightarrow \infty$, we know $\Delta_t \rightarrow 0$ and hence $v_t \rightarrow \bar{v}$ as $t \rightarrow \infty$. Therefore, \bar{v} is the target value that v_t will converge to. □

TD learning of state values – Algorithm properties

Second, what is the interpretation of the TD error?

The TD error should be zero in the expectation sense. Note that

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

If $v_t = v_\pi$, then $\delta_t = v(s_t) - [r_{t+1} + \gamma v(s_{t+1})]$ should be zero in the sense of expectation.

Therefore,

- The TD error reflects the **deficiency** between v_t and v_π .
- In a more abstract level, the TD error can be interpreted as **innovation**, which means new information obtained from the experience (s_t, r_{t+1}, s_{t+1}) .
- Innovation is a fundamental concept in many estimation problems such as Kalman filtering.

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 π .

TD learning of state values – The idea of the algorithm

First, a new expression of the Bellman equation.

The definition of state value of π 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

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

TD learning of state values – The idea of the algorithm

Therefore, the RM algorithm for solving $g(v(s)) = 0$ is

$$\begin{aligned} v_{k+1}(s) &= v_k(s) - \alpha_k \tilde{g}(v_k(s)) \\ &= v_k(s) - \alpha_k \left(v_k(s) - [r_k + \gamma v_\pi(s'_k)] \right), \quad k = 1, 2, 3, \dots \end{aligned} \quad (6)$$

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

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

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

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
Online: TD learning is online. It can update the state/action values immediately after receiving a reward.	Offline: MC learning is offline. It has to wait until an episode has been completely collected.
Continuing tasks: Since TD learning is online, it can handle both episodic and continuing tasks.	Episodic tasks: Since MC learning is offline, it can only handle episodic tasks that has terminate states.

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
Bootstrapping: TD bootstraps because the update of a value relies on the previous estimate of this value. Hence, it requires initial guesses.	Non-bootstrapping: MC is not bootstrapping, because it can directly estimate state/action values without any initial guess.
Low estimation variance: TD has lower than MC because there are fewer random variables. For instance, Sarsa requires $R_{t+1}, S_{t+1}, A_{t+1}$.	High estimation variance: To estimate $q_\pi(s_t, a_t)$, we need samples of $R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$. 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 π .

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 π .

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:

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

 Update policy:

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

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 ϵ -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.

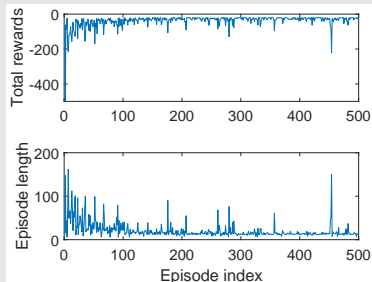
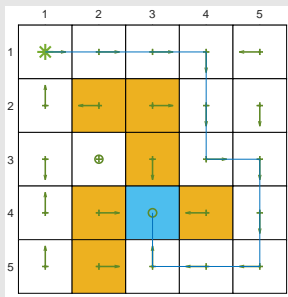
Task description:

- 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 ϵ 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 π_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

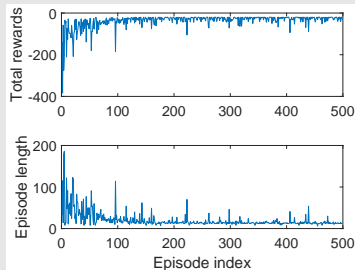
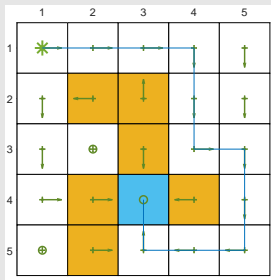
What does the algorithm do mathematically? Expected Sarsa is a stochastic approximation algorithm for solving the following equation:

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

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

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

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

$$\begin{aligned} q_{t+1}(s_t, a_t) &= q_t(s_t, a_t) \\ &\quad - \alpha_t(s_t, a_t) \left[q_t(s_t, a_t) - [r_{t+1} + \gamma r_{t+2} + \dots + \gamma^n q_t(s_{t+n}, a_{t+n})] \right]. \end{aligned}$$

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.

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 $r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})$.

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.

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.

Off-policy vs on-policy

Sarsa is on-policy.

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

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

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 $\pi_t(s_{t+1})$!
- π_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 π .

- 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 is 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*:

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

 Update *policy*:

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

See the book for more detailed pseudocode.

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

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

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

Update q-value:

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

Update target policy:

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

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

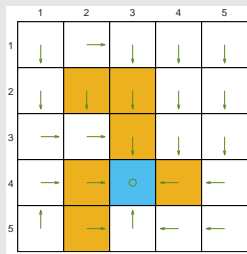
See the book for more detailed pseudocode.

Q-learning – Examples

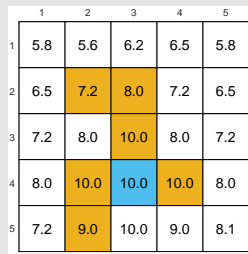
Task description:

- 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.



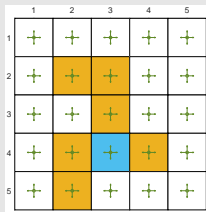
(a) Optimal policy



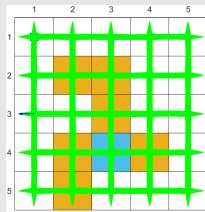
(b) Optimal state value

Q-learning – Examples

The behavior policy and the generated experience (1 million steps):

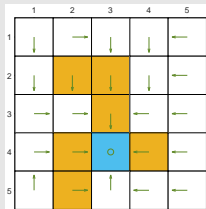


(a) Behavior policy

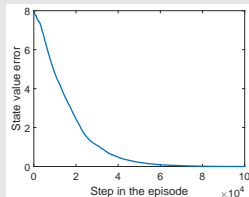


(b) Generated episode

The policy found by off-policy Q-learning:



(a) Estimated policy

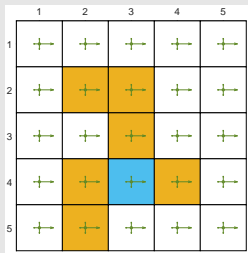


(b) State value error

Q-learning – Examples

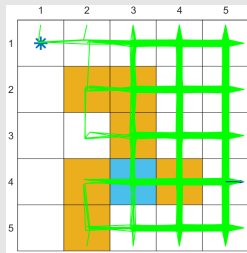
The importance of exploration: episodes of 1 million steps

If the policy is not sufficiently exploratory, the samples are not good.

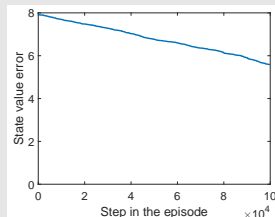


(a) Behavior policy

$\epsilon = 0.5$

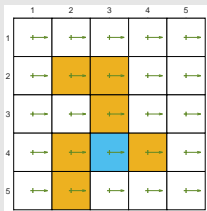


(b) Generated episode



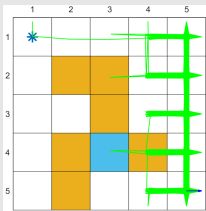
(c) Q-learning result

Q-learning – Examples

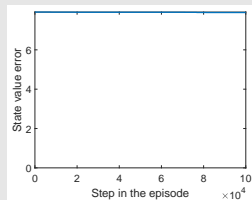


(a) Behavior policy

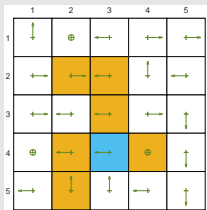
$\epsilon = 0.1$



(b) Generated episode

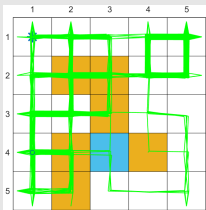


(c) Q-learning result

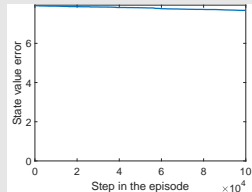


(a) Behavior policy

$\epsilon = 0.1$



(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