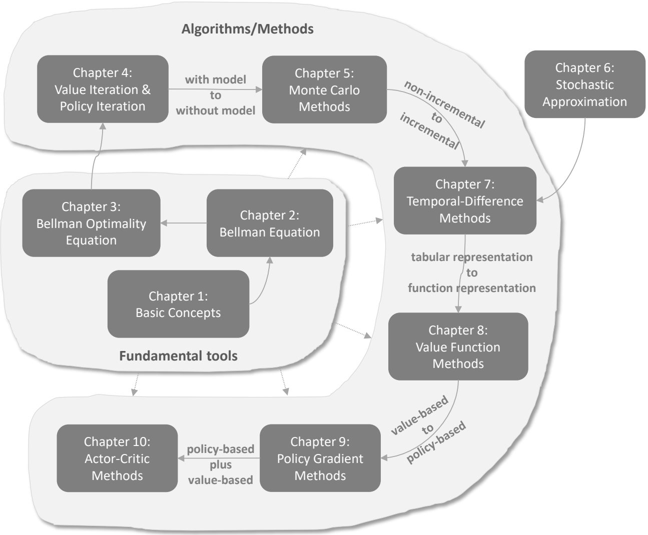
Lecture 8: Value Function Methods

Shiyu Zhao

Department of Artificial Intelligence

Westlake University

Outline



Algorithms/Methods
Chapter 4: Value Iteration & Policy Iteration
with model to without model
Chapter 5: Monte Carlo Methods
non-incremental to incremental
Chapter 6: Stochastic Approximation
Chapter 7: Temporal-Difference Methods
tabular representation to function representation
Chapter 8: Value Function Methods
value-based to policy-based
Chapter 9: Policy Gradient Methods
policy-based plus value-based
Chapter 10: Actor-Critic Methods
Chapter 3: Bellman Optimality Equation
Chapter 2: Bellman Equation
Chapter 1: Basic Concepts
Fundamental tools

Outline

Outline

Motivating examples: from table to function

So far in this book, state and action values are represented by tables.

- For example, state value:

State	s_1	s_2	$\cdots$	s_n
Value	$v_\pi(s_1)$	$v_\pi(s_2)$	$\cdots$	$v_\pi(s_n)$

- 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: from table to function

So far in this book, state and action values are represented by tables.

- For example, state value:

State	s_1	s_2	$\cdots$	s_n
Value	$v_\pi(s_1)$	$v_\pi(s_2)$	$\cdots$	$v_\pi(s_n)$

- 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: from table to function

So far in this book, state and action values are represented by tables.

- For example, state value:

State	s_1	s_2	$\cdots$	s_n
Value	$v_\pi(s_1)$	$v_\pi(s_2)$	$\cdots$	$v_\pi(s_n)$

- 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: from table to function

So far in this book, state and action values are represented by tables.

- For example, state value:

State	s_1	s_2	$\cdots$	s_n
Value	$v_\pi(s_1)$	$v_\pi(s_2)$	$\cdots$	$v_\pi(s_n)$

- 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: from table to function

So far in this book, state and action values are represented by tables.

- For example, state value:

State	s_1	s_2	$\cdots$	s_n
Value	$v_\pi(s_1)$	$v_\pi(s_2)$	$\cdots$	$v_\pi(s_n)$

- For example, action value:

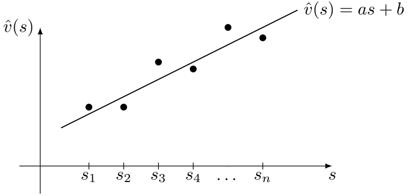
	a_1	a_2	a_3	a_4	a_5
s_1	$q_\pi(s_1, a_1)$	$q_\pi(s_1, a_2)$	$q_\pi(s_1, a_3)$	$q_\pi(s_1, a_4)$	$q_\pi(s_1, a_5)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
s_9	$q_\pi(s_9, a_1)$	$q_\pi(s_9, a_2)$	$q_\pi(s_9, a_3)$	$q_\pi(s_9, a_4)$	$q_\pi(s_9, a_5)$

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

Consider an example:

- There are n states: $s_1, \ldots, s_n$.
- The state values are $v_\pi(s_1), \ldots, v_\pi(s_n)$, where π is a given policy.
- n is very large!
- We hope to use a simple curve to approximate these values.

For example, we can use a simple straight line to fit the dots.



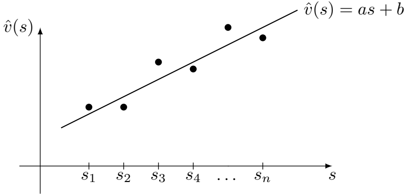
Suppose the equation of the straight line is

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

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

Motivating examples: from table to function

For example, we can use a simple straight line to fit the dots.



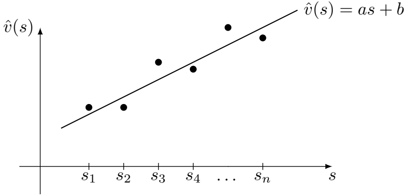
Suppose the equation of the straight line is

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

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

Motivating examples: from table to function

For example, we can use a simple straight line to fit the dots.



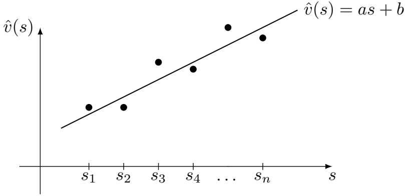
Suppose the equation of the straight line is

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

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

Motivating examples: from table to function

For example, we can use a simple straight line to fit the dots.



Suppose the equation of the straight line is

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

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

Difference between the tabular and function methods:

Difference 1: How to retrieve the value of a state

- When the values are represented by a table, we can directly read the value in the table.
- When the values are represented by a function, we need to input the state index s into the function and calculate the function value.

For example, $s \to \phi(s) \to \phi^T(s)w = \hat{v}(s, w)$

- Benefit: storage. We do not need to store $|\mathcal{S}|$ state values. We only need to store a lower-dimensional w.

Motivating examples: from table to function

Difference between the tabular and function methods:

Difference 1: How to retrieve the value of a state

- When the values are represented by a table, we can directly read the value in the table.
- When the values are represented by a function, we need to input the state index s into the function and calculate the function value.

For example, $s \to \phi(s) \to \phi^T(s)w = \hat{v}(s, w)$

- Benefit: storage. We do not need to store $|\mathcal{S}|$ state values. We only need to store a lower-dimensional w.

Motivating examples: from table to function

Difference between the tabular and function methods:

Difference 1: How to retrieve the value of a state

- When the values are represented by a table, we can directly read the value in the table.
- When the values are represented by a function, we need to input the state index s into the function and calculate the function value.

For example, $s \to \phi(s) \to \phi^T(s)w = \hat{v}(s, w)$

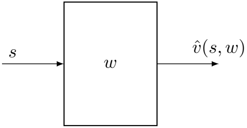
- Benefit: storage. We do not need to store $|\mathcal{S}|$ state values. We only need to store a lower-dimensional w.

Motivating examples: from table to function

Difference between the tabular and function methods:

Difference 1: How to retrieve the value of a state

- When the values are represented by a table, we can directly read the value in the table.
- When the values are represented by a function, we need to input the state index s into the function and calculate the function value.



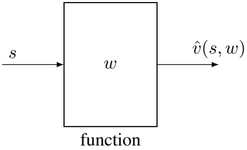
For example, $s \to \phi(s) \to \phi^T(s)w = \hat{v}(s,w)$

- Benefit: storage. We do not need to store $|\mathcal{S}|$ state values. We only need to store a lower-dimensional w.

Difference between the tabular and function methods:

Difference 1: How to retrieve the value of a state

- When the values are represented by a table, we can directly read the value in the table.
- When the values are represented by a function, we need to input the state index s into the function and calculate the function value.



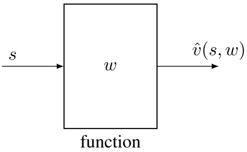
For example, $s \to \phi(s) \to \phi^T(s)w = \hat{v}(s,w)$

- Benefit: storage. We do not need to store $|\mathcal{S}|$ state values. We only need to store a lower-dimensional w.

Difference between the tabular and function methods:

Difference 1: How to retrieve the value of a state

- When the values are represented by a table, we can directly read the value in the table.
- When the values are represented by a function, we need to input the state index s into the function and calculate the function value.



For example, $s \rightarrow \phi(s) \rightarrow \phi^T(s)w = \hat{v}(s,w)$

- Benefit: storage. We do not need to store $|\mathcal{S}|$ state values. We only need to store a lower-dimensional w.

Difference between the tabular and function methods:

Difference 2: How to update the value of a state

- When the values are represented by a table, we can directly rewrite the value in the table.
- When the values are represented by a function, we must update w to change the values indirectly.
  - How to update w will be addressed in detail later.

Difference between the tabular and function methods:

Difference 2: How to update the value of a state

- When the values are represented by a table, we can directly rewrite the value in the table.
- When the values are represented by a function, we must update w to change the values indirectly.
  - How to update w will be addressed in detail later.

Difference between the tabular and function methods:

Difference 2: How to update the value of a state

- When the values are represented by a table, we can directly rewrite the value in the table.
- When the values are represented by a function, we must update w to change the values indirectly.
  - How to update w will be addressed in detail later.

Motivating examples: from table to function

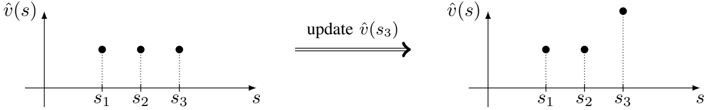
Difference between the tabular and function methods:

Difference 2: How to update the value of a state

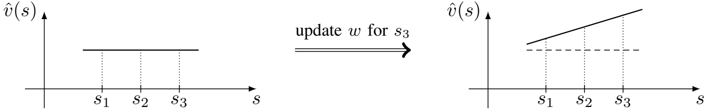
- When the values are represented by a table, we can directly rewrite the value in the table.
- When the values are represented by a function, we must update w to change the values indirectly.
  - How to update w will be addressed in detail later.

Difference between the tabular and function methods:

Difference 2: How to update the value of a state



(a) Tabular method

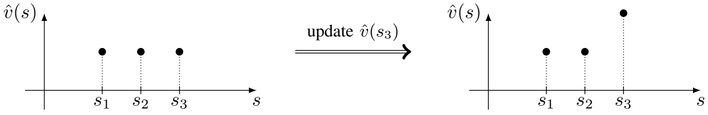


(b) Function method

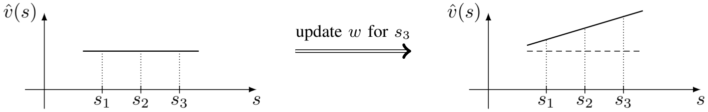
Benefit: generalization ability. When we update $\hat{v}(s, w)$ by changing w, the values of the neighboring states are also changed.

Difference between the tabular and function methods:

Difference 2: How to update the value of a state



(a) Tabular method



(b) Function method

Benefit: generalization ability. When we update $\hat{v}(s, w)$ by changing w, the values of the neighboring states are also changed.

Motivating examples: from table to function

The benefits are not free. It comes with a cost: the state values can not be represented accurately. This is why this method is called approximation.

We can fit the points more precisely using high-order curves:

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

In this case,

- The dimensions of w and $\phi(s)$ increase; 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: from table to function

The benefits are not free. It comes with a cost: the state values can not be represented accurately. This is why this method is called approximation.

We can fit the points more precisely using high-order curves:

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

In this case,

- The dimensions of w and $\phi(s)$ increase; 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: from table to function

The benefits are not free. It comes with a cost: the state values can not be represented accurately. This is why this method is called approximation.

We can fit the points more precisely using high-order curves:

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

In this case,

- The dimensions of w and $\phi(s)$ increase; 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: from table to function

The benefits are not free. It comes with a cost: the state values can not be represented accurately. This is why this method is called approximation.

We can fit the points more precisely using high-order curves:

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

In this case,

- The dimensions of w and $\phi(s)$ increase; 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: from table to function

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 retrieve and change the value of $v(s)$
- **Advantages:**
  1) **Storage:** The dimension of w may be much smaller 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.

Motivating examples: from table to function

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 retrieve and change the value of $v(s)$
- **Advantages:**
  1) **Storage:** The dimension of w may be much smaller 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.

Motivating examples: from table to function

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 retrieve and change the value of $v(s)$
- **Advantages:**
  1) **Storage:** The dimension of w may be much smaller 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.

Motivating examples: from table to function

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 retrieve and change the value of $v(s)$
- **Advantages:**
  1) **Storage:** The dimension of w may be much smaller 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

Outline

Objective function

Introduce in a more formal way:

- Let $v_\pi(s)$ and $\hat{v}(s, w)$ be the true state value and the estimated state value, respectively.
- 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 for optimizing the objective function.

Objective function

Introduce in a more formal way:

- Let $v_\pi(s)$ and $\hat{v}(s, w)$ be the true state value and the estimated state value, respectively.
- 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 for optimizing the objective function.

Objective function

Introduce in a more formal way:

- Let $v_\pi(s)$ and $\hat{v}(s, w)$ be the true state value and the estimated state value, respectively.
- 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 for optimizing the objective function.

Objective function

Introduce in a more formal way:

- Let $v_\pi(s)$ and $\hat{v}(s, w)$ be the true state value and the estimated state value, respectively.
- 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 for optimizing the objective function.

Objective function

Introduce in a more formal way:

- Let $v_\pi(s)$ and $\hat{v}(s,w)$ be the true state value and the estimated state value, respectively.
- 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 for optimizing the objective function.

Objective function

Introduce in a more formal way:

- Let $v_\pi(s)$ and $\hat{v}(s, w)$ be the true state value and the estimated state value, respectively.
- 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 for optimizing the objective function.

Objective function

Introduce in a more formal way:

- Let $v_\pi(s)$ and $\hat{v}(s,w)$ be the true state value and the estimated state value, respectively.
- 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 for 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 new. We have not discussed the probability distribution of states so far.
  - There are several ways to define the probability distribution of S.

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 new. We have not discussed the probability distribution of states so far.
  - There are several ways to define the probability distribution of S.

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 new. We have not discussed the probability distribution of states so far.
  - There are several ways to define the probability distribution of S.

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 new. We have not discussed the probability distribution of states so far.
  - There are several ways to define the probability distribution of S.

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 new. We have not discussed the probability distribution of states so far.
  - 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 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.

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 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. 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 π. By definition, $d_\pi(s) \geq 0$ and $\sum_{s\in\mathcal{S}} d_\pi(s) = 1$.
- The objective function can be rewritten as

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

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

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. 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 π. By definition, $d_\pi(s) \geq 0$ and $\sum_{s\in\mathcal{S}} d_\pi(s) = 1$.
- The objective function can be rewritten as

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

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

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. 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 π. By definition, $d_\pi(s) \geq 0$ and $\sum_{s\in\mathcal{S}} d_\pi(s) = 1$.
- The objective function can be rewritten as

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

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

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. 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 π. By definition, $d_\pi(s) \geq 0$ and $\sum_{s\in\mathcal{S}} d_\pi(s) = 1$.
- The objective function can be rewritten as

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

This function is a weighted squared error.

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

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. 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 π. By definition, $d_\pi(s) \geq 0$ and $\sum_{s\in\mathcal{S}} d_\pi(s) = 1$.
- The objective function can be rewritten as

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

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

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 method.
- It is also important for the policy gradient method in the next lecture.

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 method.
- It is also important for the policy gradient method in the next lecture.

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 method.
- It is also important for the policy gradient method in the next lecture.

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.

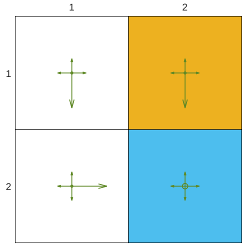
Remarks:

- Stationary distribution is also called steady-state distribution, or limiting distribution.
- It is critical to understand the value function method.
- It is also important for the policy gradient method in the next lecture.

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 π.
- Then, $d_\pi(s)$ can be approximated by

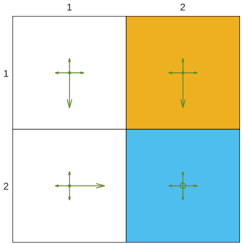
$$d_\pi(s) \approx \frac{n_\pi(s)}{\sum_{s' \in \mathcal{S}} n_\pi(s')}$$



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 π.
- Then, $d_\pi(s)$ can be approximated by

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



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 π.
- Then, $d_\pi(s)$ can be approximated by

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

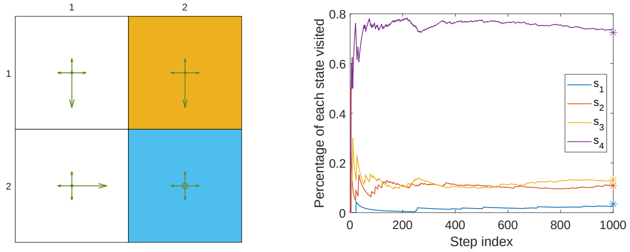


Figure: Long-run behavior of an ϵ-greedy policy with $\epsilon = 0.5$.

Objective function - Stationary distribution

The converged values can be predicted because they are the entries of d_π:

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

For this example, we have P_π as

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

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

$$d_\pi = \left[0.0345, 0.1084, 0.1330, 0.7241\right]^T$$

A comprehensive introduction can be found in my book.

Objective function - Stationary distribution

The converged values can be predicted because they are the entries of d_π:

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

For this example, we have P_π as

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

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

$$d_\pi = \left[0.0345, 0.1084, 0.1330, 0.7241\right]^T$$

A comprehensive introduction can be found in my book.

Objective function - Stationary distribution

The converged values can be predicted because they are the entries of d_π:

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

For this example, we have P_π as

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

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

$$d_\pi = \left[0.0345, 0.1084, 0.1330, 0.7241\right]^T$$

A comprehensive introduction can be found in my book.

Objective function - Stationary distribution

The converged values can be predicted because they are the entries of d_π:

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

For this example, we have P_π as

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

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

$$d_\pi = \left[0.0345, 0.1084, 0.1330, 0.7241\right]^T$$

A comprehensive introduction can be found in my book.

Outline

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

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

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

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

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

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

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

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

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

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

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

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

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:

$$w_{k+1} = w_k + \alpha_k \mathbb{E}[(v_\pi(S) - \hat{v}(S, w))\nabla_w \hat{v}(S, w)]$$

$$\Downarrow$$

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

where s_t is a sample of S. Here, $2\alpha_t$ is merged to α_t.

- The samples are expected to satisfy the stationary distribution. In practice, they may not satisfy.
- This algorithm is not implementable because it requires the true state value v_π, which is the unknown to be estimated.
- We can replace $v_\pi(s_t)$ with an approximation so that the algorithm is implementable.

Optimization algorithms

We can use the stochastic gradient to replace the true gradient:

$$w_{k+1} = w_k + \alpha_k \mathbb{E}[(v_\pi(S) - \hat{v}(S, w))\nabla_w \hat{v}(S, w)]$$

$$\Downarrow$$

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

where s_t is a sample of S. Here, $2\alpha_t$ is merged to α_t.

- The samples are expected to satisfy the stationary distribution. In practice, they may not satisfy.
- This algorithm is not implementable because it requires the true state value v_π, which is the unknown to be estimated.
- We can replace $v_\pi(s_t)$ with an approximation so that the algorithm is implementable.

Optimization algorithms

We can use the stochastic gradient to replace the true gradient:

$$w_{k+1} = w_k + \alpha_k \mathbb{E}[(v_\pi(S) - \hat{v}(S,w))\nabla_w \hat{v}(S,w)]$$

$$\Downarrow$$

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

where s_t is a sample of S. Here, $2\alpha_t$ is merged to α_t.

- The samples are expected to satisfy the stationary distribution. In practice, they may not satisfy.
- This algorithm is not implementable because it requires the true state value v_π, which is the unknown to be estimated.
- We can replace $v_\pi(s_t)$ with an approximation so that the algorithm is implementable.

Optimization algorithms

We can use the stochastic gradient to replace the true gradient:

$$w_{k+1} = w_k + \alpha_k \mathbb{E}[(v_\pi(S) - \hat{v}(S,w))\nabla_w \hat{v}(S,w)]$$

$$\Downarrow$$

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

where s_t is a sample of S. Here, $2\alpha_t$ is merged to α_t.

- The samples are expected to satisfy the stationary distribution. In practice, they may not satisfy.
- This algorithm is not implementable because it requires the true state value v_π, which is the unknown to be estimated.
- We can replace $v_\pi(s_t)$ with an approximation so that the algorithm is implementable.

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

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

In 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

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

In 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

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

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

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

In 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

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

Pseudocode: TD learning of state values with function approximation

Initialization: A function $\hat{v}(s,w)$ that is differentiable in w. Initial parameter w_0.
Goal: Learn the true state values of a given policy π.

For each episode $\{(s_t, r_{t+1}, s_{t+1})\}_t$ generated by π, do
    For each sample (s_t, r_{t+1}, s_{t+1}), do
        In the general case,
        $w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma\hat{v}(s_{t+1}, w_t) - \hat{v}(s_t, w_t)\right] \nabla_w \hat{v}(s_t, w_t)$

It can only estimate the state values of a given policy, but it is important to understand other algorithms introduced later.

Optimization algorithms

Pseudocode: TD learning of state values with function approximation

Initialization: A function $\hat{v}(s, w)$ that is differentiable in w. Initial parameter w_0.
Goal: Learn the true state values of a given policy π.

For each episode $\{(s_t, r_{t+1}, s_{t+1})\}_t$ generated by π, do
  For each sample (s_t, r_{t+1}, s_{t+1}), do
    In the general case,
    $w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma \hat{v}(s_{t+1}, w_t) - \hat{v}(s_t, w_t)\right] \nabla_w \hat{v}(s_t, w_t)$
    In the linear case,
    $w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma \phi^T(s_{t+1}) w_t - \phi^T(s_t) w_t\right] \phi(s_t)$

It can only estimate the state values of a given policy, but it is important to understand other algorithms introduced later.

Optimization algorithms

Pseudocode: TD learning of state values with function approximation

Initialization: A function $\hat{v}(s, w)$ that is differentiable in w. Initial parameter w_0.
Goal: Learn the true state values of a given policy π.

For each episode $\{(s_t, r_{t+1}, s_{t+1})\}_t$ generated by π, do

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

In the general case,

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

In the linear case,

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

It can only estimate the state values of a given policy, but it is important to understand other algorithms introduced later.

Outline

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.
  - For example, the input is s, the output is $\hat{v}(s, w)$, and the parameter is w.

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.
  - For example, the input is s, the output is $\hat{v}(s, w)$, and the parameter is w.

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.
  - For example, the input is s, the output is $\hat{v}(s, w)$, and the parameter is w.

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.
  - For example, the input is s, the output is $\hat{v}(s, w)$, and the parameter is w.

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.
  - For example, the input is s, the output is $\hat{v}(s, w)$, and the parameter is w.

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.
  - For example, the input is s, the output is $\hat{v}(s, w)$, and the 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

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

yields

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

which is the algorithm of TD learning with linear function approximation.
It is called TD-Linear in our course.

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

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

yields

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

which is the algorithm of TD learning with linear function approximation.
It is called TD-Linear in our course.

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

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

yields

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

which is the algorithm of TD learning with linear function approximation. It is called TD-Linear in our course.

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

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

yields

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

which is the algorithm of TD learning with linear function approximation.
It is called TD-Linear in our course.

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

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

yields

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

which is the algorithm of TD learning with linear function approximation.
It is called TD-Linear in our course.

Linear function approximation

- **Disadvantages** of linear function methods:
  - Difficult to select appropriate feature vectors.

- **Advantages** of linear function methods:
  - 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 a special case of linear function representation.

- **Disadvantages** of linear function methods:
  - Difficult to select appropriate feature vectors.

- **Advantages** of linear function methods:
  - 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 a special case of linear function representation.

Linear function approximation

- **Disadvantages** of linear function methods:
  - Difficult to select appropriate feature vectors.
- **Advantages** of linear function methods:
  - 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 a special case of linear function representation.

Linear function approximation

We next show that tabular representation is a special case of linear function representation. Hence, the tabular and function representations are unified!

- Consider a special feature vector for state s:

$$\phi(s) = e_s \in \mathbb{R}^{|\mathcal{S}|},$$

where e_s is a vector with the sth entry as 1 and the others as 0.

- In this case,

$$\hat{v}(s, w) = \phi^T(s)w = e_s^T w = w(s),$$

where $w(s)$ is the sth entry of w.

Linear function approximation

We next show that tabular representation is a special case of linear function representation. Hence, the tabular and function representations are unified!

- Consider a special feature vector for state s:

$$\phi(s) = e_s \in \mathbb{R}^{|\mathcal{S}|},$$

  where e_s is a vector with the sth entry as 1 and the others as 0.

- In this case,

$$\hat{v}(s, w) = \phi^T(s)w = e_s^T w = w(s),$$

  where $w(s)$ is the sth entry of w.

We next show that tabular representation is a special case of linear function representation. Hence, the tabular and function representations are unified!

- Consider a special feature vector for state s:

$$\phi(s) = e_s \in \mathbb{R}^{|\mathcal{S}|},$$

  where e_s is a vector with the sth entry as 1 and the others as 0.

- In this case,

$$\hat{v}(s, w) = \phi^T(s)w = e_s^T w = w(s),$$

  where $w(s)$ is the sth entry of w.

Recall that the TD-Linear algorithm is

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

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

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

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

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

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

  which is exactly the tabular TD algorithm (which is called TD-Table here).

Summary: TD-Linear becomes TD-Table if we select a special feature vector.

Recall that the TD-Linear algorithm is

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

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

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

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

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

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

  which is exactly the tabular TD algorithm (which is called TD-Table here).

Summary: TD-Linear becomes TD-Table if we select a special feature vector.

Linear function approximation

Recall that the TD-Linear algorithm is

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

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

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

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

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

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

  which is exactly the tabular TD algorithm (which is called TD-Table here).

Summary: TD-Linear becomes TD-Table if we select a special feature vector.

Linear function approximation

Recall that the TD-Linear algorithm is

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

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

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

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

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

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

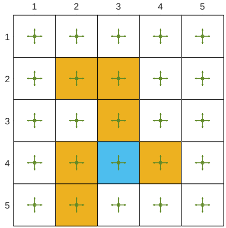
  which is exactly the tabular TD algorithm (which is called TD-Table here).

Summary: TD-Linear becomes TD-Table if we select a special feature vector.

Outline

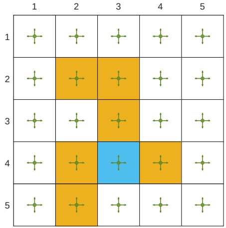
Illustrative examples

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 25 state values.
- Set $r_{\text{forbidden}} = r_{\text{boundary}} = -1$, $r_{\text{target}} = 1$, and $\gamma = 0.9$.

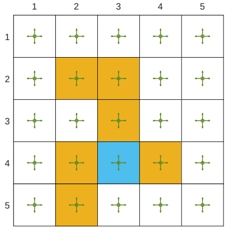
Consider a 5x5 grid-world example:



- 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 25 state values.
- Set $r_{\text{forbidden}} = r_{\text{boundary}} = -1$, $r_{\text{target}} = 1$, and $\gamma = 0.9$.

Illustrative examples

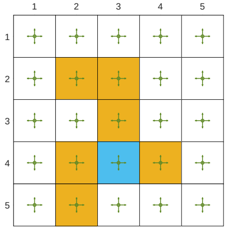
Consider a 5x5 grid-world example:



- 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 25 state values.
- Set $r_{\text{forbidden}} = r_{\text{boundary}} = -1$, $r_{\text{target}} = 1$, and $\gamma = 0.9$.

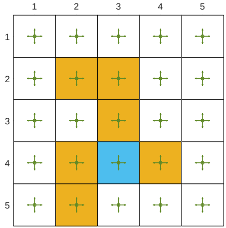
Illustrative examples

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 25 state values.
- Set $r_{\text{forbidden}} = r_{\text{boundary}} = -1$, $r_{\text{target}} = 1$, and $\gamma = 0.9$.

Consider a 5x5 grid-world example:

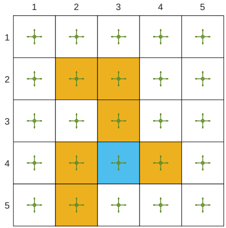


- Given a policy: $\pi(a|s) = 0.2$ for any s, a
- Our aim is to estimate the state values of this policy (policy evaluation problem).
- There are 25 state values in total. We next show that we can use less than 25 parameters to approximate 25 state values.
- Set $r_{\text{forbidden}} = r_{\text{boundary}} = -1$, $r_{\text{target}} = 1$, and $\gamma = 0.9$.

Illustrative examples

Ground truth:

- The true state values and the 3D visualization



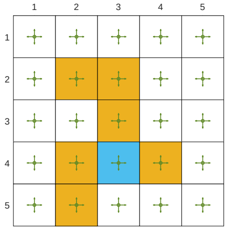
	1	2	3	4	5
1	-3.8	-3.8	-3.6	-3.1	-3.2
2	-3.8	-3.8	-3.8	-3.1	-2.9
3	-3.6	-3.9	-3.4	-3.2	-2.9
4	-3.9	-3.6	-3.4	-2.9	-3.2
5	-4.5	-4.2	-3.4	-3.4	-3.5

Experience samples:

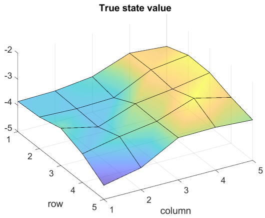
- 500 episodes were generated following the given policy.
- Each episode has 500 steps and starts from a randomly selected state-action pair following a uniform distribution.

Ground truth:

- The true state values and the 3D visualization



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

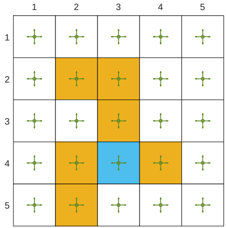


Experience samples:

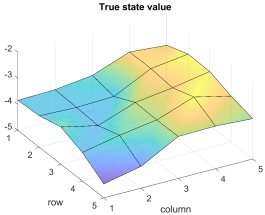
- 500 episodes were generated following the given policy.
- Each episode has 500 steps and starts from a randomly selected state-action pair following a uniform distribution.

Ground truth:

- The true state values and the 3D visualization



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

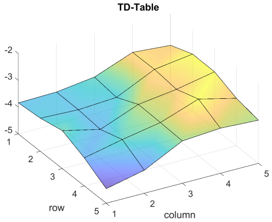


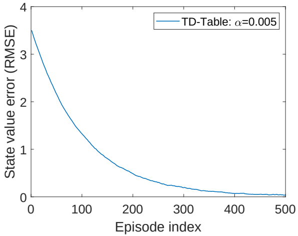
Experience samples:

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

TD-Table:

- For comparison, the results by the tabular TD algorithm (called TD-Table here):





TD-Linear:

- How to apply 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

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

Remark: $\phi(s)$ can also be defined as $\phi(s) = [x, y, 1]^T$, where the order of the elements does not matter.

TD-Linear:

- How to apply 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

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

Remark: $\phi(s)$ can also be defined as $\phi(s) = [x, y, 1]^T$, where the order of the elements does not matter.

TD-Linear:

- How to apply 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

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

Remark: $\phi(s)$ can also be defined as $\phi(s) = [x, y, 1]^T$, where the order of the elements does not matter.

TD-Linear:

- How to apply the TD-Linear algorithm?
  - Feature vector selection:

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

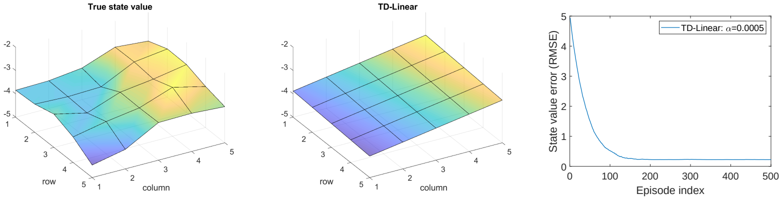
  - In this case, the approximated state value is

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

Remark: $\phi(s)$ can also be defined as $\phi(s) = [x, y, 1]^T$, where the order of the elements does not matter.

TD-Linear:

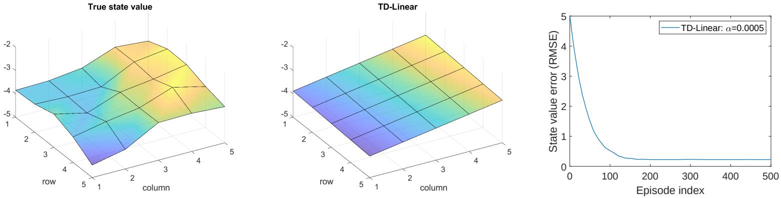
- Results by the TD-Linear algorithm:



- 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!

TD-Linear:

- 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:

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

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:

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

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,

$$\hat{v}(s, w) = \phi^T(s)w = w_1 + w_2 x + w_3 y + w_4 x^2 + w_5 y^2 + w_6 xy$$

  which corresponds to a quadratic surface.

- We can further increase the dimension of the feature vector:

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

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,

$$\hat{v}(s, w) = \phi^T(s)w = w_1 + w_2 x + w_3 y + w_4 x^2 + w_5 y^2 + w_6 xy$$

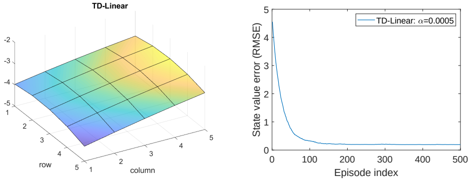
  which corresponds to a quadratic surface.

- We can further increase the dimension of the feature vector:

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

Illustrative examples

Results by the TD-Linear algorithm with higher-order feature vectors:

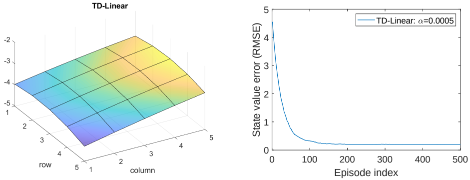


The above figure: $\phi(s) \in \mathbb{R}^6$

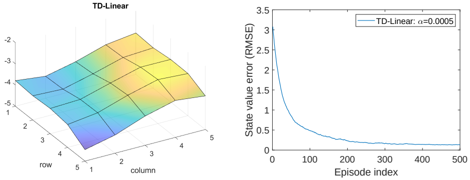
More examples and features are given in the book.

Illustrative examples

Results by the TD-Linear algorithm with higher-order feature vectors:



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

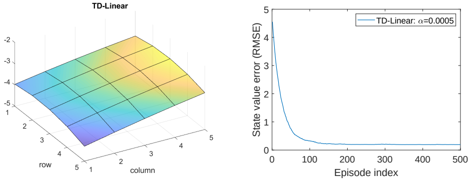


The above figure: $\phi(s) \in \mathbb{R}^{10}$

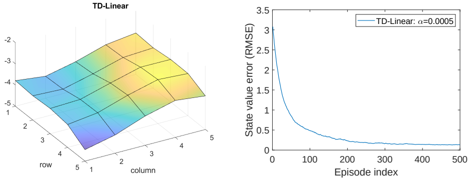
More examples and features are given in the book.

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 features are given in the book.

Outline

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:

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

Although this story is very helpful to understand the basic idea, it is not mathematically rigorous.

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:

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

Although this story is very helpful to understand the basic idea, it is not mathematically rigorous.

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:

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

Although this story is very helpful to understand the basic idea, it is not mathematically rigorous.

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:

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

Although this story is very helpful to understand the basic idea, it is not mathematically rigorous.

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:

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

Although this story is very helpful to understand the basic idea, it is not mathematically rigorous.

Outline

Theoretical analysis (optional)

- The algorithm

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

does not minimize the following objective function:

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

Theoretical analysis (optional)

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 $T_\pi(x) \doteq r_\pi + \gamma P_\pi x$

- **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.

More details are omitted here. Interested readers can check my book.

Theoretical analysis (optional)

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 $T_\pi(x) \doteq r_\pi + \gamma P_\pi x$

- **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.

More details are omitted here. Interested readers can check my book.

Theoretical analysis (optional)

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 $T_\pi(x) \doteq r_\pi + \gamma P_\pi x$

- **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.

More details are omitted here. Interested readers can check my book.

Theoretical analysis (optional)

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 $T_\pi(x) \doteq r_\pi + \gamma P_\pi x$

- **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.

More details are omitted here. Interested readers can check my book.

Theoretical analysis (optional)

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 $T_\pi(x) \doteq r_\pi + \gamma P_\pi x$

- **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.

More details are omitted here. Interested readers can check my book.

Theoretical analysis (optional)

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 $T_\pi(x) \doteq r_\pi + \gamma P_\pi x$

- **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.

More details are omitted here. Interested readers can check my book.

Outline

Sarsa with function approximation

So far, we merely considered state value estimation. That is

$$\hat{v}(s) \approx v_\pi(s), \quad s \in \mathcal{S}$$

To search for optimal policies, we need to estimate action values.

The Sarsa algorithm with value function approximation is

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

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

So far, we merely considered state value estimation. That is

$$\hat{v}(s) \approx v_\pi(s), \quad s \in \mathcal{S}$$

To search for optimal policies, we need to estimate action values.

The Sarsa algorithm with value function approximation is

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

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

So far, we merely considered state value estimation. That is

$$\hat{v}(s) \approx v_\pi(s), \quad s \in \mathcal{S}$$

To search for optimal policies, we need to estimate action values.

The Sarsa algorithm with value function approximation is

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

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

So far, we merely considered state value estimation. That is

$$\hat{v}(s) \approx v_\pi(s), \quad s \in \mathcal{S}$$

To search for optimal policies, we need to estimate action values.

The Sarsa algorithm with value function approximation is

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

This is the same as the algorithm we introduced previously in this lecture except that $\hat{v}$ is replaced by $\hat{q}$.

To search for optimal policies, we can combine policy evaluation and policy improvement.

Sarsa with function approximation

To search for optimal policies, we can combine policy evaluation and policy improvement.

Pseudocode: Sarsa with function approximation

Initialization: Initial parameter w_0. Initial policy π_0. $\alpha_t = \alpha > 0$ for all t. $\epsilon \in (0, 1)$.
Goal: Learn an optimal policy to lead the agent to the target state from an initial state s_0.
For each episode, do

  Generate a_0 at s_0 following $\pi_0(s_0)$

  If s_t $(t = 0, 1, 2, \dots)$ is not the target state, do

    Collect the experience sample $(r_{t+1}, s_{t+1}, a_{t+1})$ given (s_t, a_t): generate r_{t+1}, s_{t+1} by interacting with the environment; generate a_{t+1} following $\pi_t(s_{t+1})$.

    Update q-value (update parameter):

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

    Update policy:

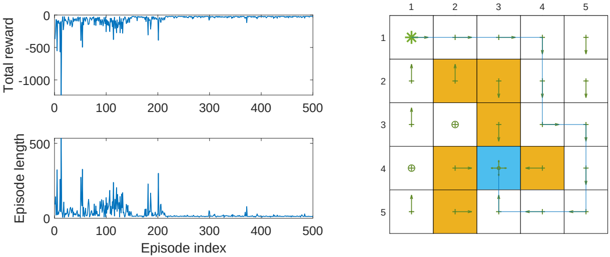
    $\pi_{t+1}(a|s_t) = 1 - \frac{\varepsilon}{|\mathcal{A}(s_t)|}(|\mathcal{A}(s_t)| - 1)$ if $a = \arg\max_{a \in \mathcal{A}(s_t)} \hat{q}(s_t, a, w_{t+1})$

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

    $s_t \leftarrow s_{t+1}$, $a_t \leftarrow a_{t+1}$

Illustrative example:

- Sarsa with linear function approximation: $\hat{q}(s,a,w) = \phi^T(s,a)w$
- $\gamma = 0.9$, $\epsilon = 0.1$, $r_{\text{boundary}} = r_{\text{forbidden}} = -10$, $r_{\text{target}} = 1$, $\alpha = 0.001$.



For details, please see the book.

Outline

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

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

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

Similar to Sarsa, tabular Q-learning can also be extended to the case of value function approximation.

The q-value update rule is

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

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 w_0. Initial policy π_0. $\alpha_t = \alpha > 0$ for all t. $\epsilon \in (0, 1)$.
Goal: Learn an optimal path to lead the agent to the target state from an initial state s_0.

For each episode, do

If s_t $(t = 0, 1, 2, \dots)$ is not the target state, do

Collect the experience sample (a_t, r_{t+1}, s_{t+1}) given s_t: generate a_t following $\pi_t(s_t)$; generate r_{t+1}, s_{t+1} by interacting with the environment.

Update value (update parameter):

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

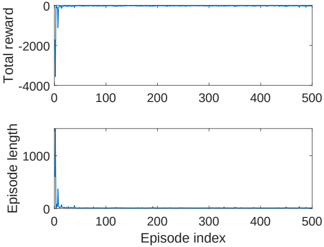
Update policy:

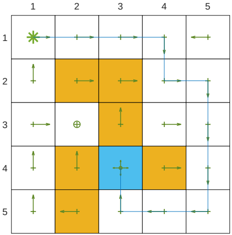
$\pi_{t+1}(a|s_t) = 1 - \frac{\varepsilon}{|\mathcal{A}(s_t)|}(|\mathcal{A}(s_t)| - 1)$ if $a = \arg\max_{a \in \mathcal{A}(s_t)} \hat{q}(s_t, a, w_{t+1})$
$\pi_{t+1}(a|s_t) = \frac{\varepsilon}{|\mathcal{A}(s_t)|}$ otherwise

Q-learning with function approximation

Illustrative example:

- Q-learning with linear function approximation: $\hat{q}(s,a,w) = \phi^T(s,a)w$
- $\gamma = 0.9$, $\epsilon = 0.1$, $r_{\text{boundary}} = r_{\text{forbidden}} = -10$, $r_{\text{target}} = 1$, $\alpha = 0.001$.





Outline

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:

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

because of the way of training a network.

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:

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

because of the way of training a network.

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:

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

because of the way of training a network.

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:

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

because of the way of training a network.

Deep Q-learning

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

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

$$\Downarrow$$

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

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

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

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

- Since the optimal a depends on w,

$$\nabla_w y \neq \gamma \max_{a \in \mathcal{A}(S')} \nabla_w \hat{q}(S', a, w)$$

- To solve this problem, we can assume that w in y is fixed (at least for a while) when we calculate the gradient.

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

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

- Since the optimal a depends on w,

$$\nabla_w y \neq \gamma \max_{a \in \mathcal{A}(S')} \nabla_w \hat{q}(S', a, w)$$

- To solve this problem, we can assume that w in y is fixed (at least for a while) when we calculate the gradient.

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

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

- Since the optimal a depends on w,

$$\nabla_w y \neq \gamma \max_{a \in \mathcal{A}(S')} \nabla_w \hat{q}(S', a, w)$$

- To solve this problem, we can assume that w in y is fixed (at least for a while) when we calculate the gradient.

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

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

- Since the optimal a depends on w,

$$\nabla_w y \neq \gamma \max_{a \in \mathcal{A}(S')} \nabla_w \hat{q}(S', a, w)$$

- To solve this problem, we can assume that w in y is fixed (at least for a while) when we calculate the gradient.

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

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

- Since the optimal a depends on w,

$$\nabla_w y \neq \gamma \max_{a \in \mathcal{A}(S')} \nabla_w \hat{q}(S', a, w)$$

- To solve this problem, 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.

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

To do that, we can introduce two networks.

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

The objective function in this case degenerates to

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

where w_T is the target network parameter.

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

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

- 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.

To do that, we can introduce two networks.

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

The objective function in this case degenerates to

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

where w_T is the target network parameter.

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

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

- 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

To do that, we can introduce two networks.

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

The objective function in this case degenerates to

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

where w_T is the target network parameter.

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

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

- 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

To do that, we can introduce two networks.

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

The objective function in this case degenerates to

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

where w_T is the target network parameter.

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

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

- 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

To do that, we can introduce two networks.

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

The objective function in this case degenerates to

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

where w_T is the target network parameter.

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

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

- 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.

Technique 1: 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).
- For every (s, a, r, s'), we can calculate the desired output as

  $$y_T \doteq r + \gamma \max_{a \in \mathcal{A}(s')} \hat{q}(s', a, w_T)$$

  Therefore, we obtain a mini-batch of data:

  $$\{(s, a, y_T)\}$$

- Use $\{(s, a, y_T)\}$ to train the network so as to minimize $(y_T - \hat{q}(s, a, w))^2$.

Technique 1: 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).
- For every (s,a,r,s'), we can calculate the desired output as

$$y_T \doteq r + \gamma \max_{a\in\mathcal{A}(s')} \hat{q}(s',a,w_T)$$

  Therefore, we obtain a mini-batch of data:

$$\{(s,a,y_T)\}$$

- Use $\{(s,a,y_T)\}$ to train the network so as to minimize $(y_T - \hat{q}(s,a,w))^2$.

Deep Q-learning - Two networks

Technique 1: 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).
- For every (s, a, r, s'), we can calculate the desired output as

$$y_T \doteq r + \gamma \max_{a \in \mathcal{A}(s')} \hat{q}(s', a, w_T)$$

  Therefore, we obtain a mini-batch of data:

$$\{(s, a, y_T)\}$$

- Use $\{(s, a, y_T)\}$ to train the network so as to minimize $(y_T - \hat{q}(s, a, w))^2$.

Technique 1: 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).
- For every (s, a, r, s'), we can calculate the desired output as

$$y_T \doteq r + \gamma \max_{a \in \mathcal{A}(s')} \hat{q}(s', a, w_T)$$

Therefore, we obtain a mini-batch of data:

$$\{(s, a, y_T)\}$$

- Use $\{(s, a, y_T)\}$ to train the network so as to minimize $(y_T - \hat{q}(s, a, w))^2$.

Technique 1: 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).
- For every (s,a,r,s'), we can calculate the desired output as

$$y_T \doteq r + \gamma \max_{a\in\mathcal{A}(s')} \hat{q}(s',a,w_T)$$

  Therefore, we obtain a mini-batch of data:

$$\{(s,a,y_T)\}$$

- Use $\{(s,a,y_T)\}$ to train the network so as to minimize $(y_T - \hat{q}(s,a,w))^2$.

Technique 1: 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).
- For every (s, a, r, s'), we can calculate the desired output as

$$y_T \doteq r + \gamma \max_{a \in \mathcal{A}(s')} \hat{q}(s', a, w_T)$$

Therefore, we obtain a mini-batch of data:

$$\{(s, a, y_T)\}$$

- Use $\{(s, a, y_T)\}$ to train the network so as to minimize $(y_T - \hat{q}(s, a, w))^2$.

Deep Q-learning - Experience replay

Technique 2: 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.

Technique 2: 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.

Deep Q-learning - Experience replay

Technique 2: 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.

Deep Q-learning - Experience replay

Technique 2: 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.

Technique 2: 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.

Technique 2: 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.

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.

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

- $R \sim p(R|S, A), S' \sim p(S'|S, A)$: R and S are determined by the system model.
- $(S, A) \sim d$: (S, A) is an index
- The distribution of the state-action pair (S, A) is assumed to be uniform.
  - Why uniform distribution? Because no prior knowledge.
  - Can we use stationary distribution like before? No, since no policy is given.

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.

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

- $R \sim p(R|S, A), S' \sim p(S'|S, A)$: R and S are determined by the system model.
- $(S, A) \sim d$: (S, A) is an index
- The distribution of the state-action pair (S, A) is assumed to be uniform.
  - Why uniform distribution? Because no prior knowledge.
  - Can we use stationary distribution like before? No, since no policy is given.

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.

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

- $R \sim p(R|S, A), S' \sim p(S'|S, A)$: R and S are determined by the system model.
- $(S, A) \sim d$: (S, A) is an index
- The distribution of the state-action pair (S, A) is assumed to be uniform.
  - Why uniform distribution? Because no prior knowledge.
  - Can we use stationary distribution like before? No, since no policy is given.

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.

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

- $R \sim p(R|S, A), S' \sim p(S'|S, A)$: R and S are determined by the system model.
- $(S, A) \sim d$: (S, A) is an index
- The distribution of the state-action pair (S, A) is assumed to be uniform.
  - Why uniform distribution? Because no prior knowledge.
  - Can we use stationary distribution like before? No, since no policy is given.

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.

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

- $R \sim p(R|S, A), S' \sim p(S'|S, A)$: R and S are determined by the system model.
- $(S, A) \sim d$: (S, A) is an index
- The distribution of the state-action pair (S, A) is assumed to be uniform.
  - Why uniform distribution? Because no prior knowledge.
  - Can we use stationary distribution like before? No, since no policy is given.

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.

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

- $R \sim p(R|S, A), S' \sim p(S'|S, A)$: R and S are determined by the system model.
- $(S, A) \sim d$: (S, A) is an index
- The distribution of the state-action pair (S, A) is assumed to be uniform.
  - Why uniform distribution? Because no prior knowledge.
  - Can we use stationary distribution like before? No, since no policy is given.

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.

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.

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.

Revisit the tabular case:

- Question: Why does not tabular Q-learning require experience replay?
  - Answer: Because it does not require any distribution of S or A.
- Question: Why does Deep Q-learning involve distributions?
  - Answer: Because we need to define a *scalar* objective function $J(w) = \mathbb{E}[*]$, where $\mathbb{E}$ is for all (S, A).
  - The tabular case aims to solve a set of equations for all (s, a) (Bellman optimality equation), whereas the deep case aims to optimize a scalar objective function.
- Question: Can we use experience replay in tabular Q-learning?
  - Answer: Yes, we can. And more sample efficient (why?)

Revisit the tabular case:

- Question: Why does not tabular Q-learning require experience replay?
  - Answer: Because it does not require any distribution of S or A.
- Question: Why does Deep Q-learning involve distributions?
  - Answer: Because we need to define a *scalar* objective function $J(w) = \mathbb{E}[*]$, where $\mathbb{E}$ is for all (S, A).
  - The tabular case aims to solve a set of equations for all (s, a) (Bellman optimality equation), whereas the deep case aims to optimize a scalar objective function.
- Question: Can we use experience replay in tabular Q-learning?
  - Answer: Yes, we can. And more sample efficient (why?)

Revisit the tabular case:

- Question: Why does not tabular Q-learning require experience replay?
  - Answer: Because it does not require any distribution of S or A.
- Question: Why does Deep Q-learning involve distributions?
  - Answer: Because we need to define a *scalar* objective function $J(w) = \mathbb{E}[*]$, where $\mathbb{E}$ is for all (S, A).
  - The tabular case aims to solve a set of equations for all (s, a) (Bellman optimality equation), whereas the deep case aims to optimize a scalar objective function.
- Question: Can we use experience replay in tabular Q-learning?
  - Answer: Yes, we can. And more sample efficient (why?)

Revisit the tabular case:

- Question: Why does not tabular Q-learning require experience replay?
  - Answer: Because it does not require any distribution of S or A.
- Question: Why does Deep Q-learning involve distributions?
  - Answer: Because we need to define a *scalar* objective function $J(w) = \mathbb{E}[*]$, where $\mathbb{E}$ is for all (S, A).
  - The tabular case aims to solve a set of equations for all (s, a) (Bellman optimality equation), whereas the deep case aims to optimize a scalar objective function.
- Question: Can we use experience replay in tabular Q-learning?
  - Answer: Yes, we can. And more sample efficient (why?)

Revisit the tabular case:

- Question: Why does not tabular Q-learning require experience replay?
  - Answer: Because it does not require any distribution of S or A.
- Question: Why does Deep Q-learning involve distributions?
  - Answer: Because we need to define a *scalar* objective function $J(w) = \mathbb{E}[*]$, where $\mathbb{E}$ is for all (S, A).
  - The tabular case aims to solve a set of equations for all (s, a) (Bellman optimality equation), whereas the deep case aims to optimize a scalar objective function.
- Question: Can we use experience replay in tabular Q-learning?
  - Answer: Yes, we can. And more sample efficient (why?)

Deep Q-learning - Experience replay

Revisit the tabular case:

- Question: Why does not tabular Q-learning require experience replay?
  - Answer: Because it does not require any distribution of S or A.
- Question: Why does Deep Q-learning involve distributions?
  - Answer: Because we need to define a *scalar* objective function $J(w) = \mathbb{E}[*]$, where $\mathbb{E}$ is for all (S, A).
  - The tabular case aims to solve a set of equations for all (s, a) (Bellman optimality equation), whereas the deep case aims to optimize a scalar objective function.
- Question: Can we use experience replay in tabular Q-learning?
  - Answer: Yes, we can. And more sample efficient (why?)

Revisit the tabular case:

- Question: Why does not tabular Q-learning require experience replay?
  - Answer: Because it does not require any distribution of S or A.
- Question: Why does Deep Q-learning involve distributions?
  - Answer: Because we need to define a *scalar* objective function $J(w) = \mathbb{E}[*]$, where $\mathbb{E}$ is for all (S, A).
  - The tabular case aims to solve a set of equations for all (s, a) (Bellman optimality equation), whereas the deep case aims to optimize a scalar objective function.
- 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)

Initialization: A main network and a target network with the same initial parameter.
Goal: Learn an optimal target network to approximate the *optimal* action values from the experience samples generated by a given behavior policy π_b.

Store the experience samples generated by π_b in a replay buffer $\mathcal{B} = \{(s, a, r, s')\}$
For each iteration, do
Uniformly draw a mini-batch of samples from $\mathcal{B}$
For each sample (s, a, r, s'), calculate the target value as $y_T = r + \gamma \max_{a \in \mathcal{A}(s')} \hat{q}(s', a, w_T)$, where w_T is the parameter of the target network
Update the main network to minimize $(y_T - \hat{q}(s, a, w))^2$ using the mini-batch of samples
Set $w_T = w$ every C iterations

Remarks:

- Why no policy update?
- The network input and output are different from the DQN paper.

Deep Q-learning

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

Initialization: A main network and a target network with the same initial parameter.
Goal: Learn an optimal target network to approximate the *optimal* action values from the experience samples generated by a given behavior policy π_b.

Store the experience samples generated by π_b in a replay buffer $\mathcal{B} = \{(s, a, r, s')\}$
    For each iteration, do
        Uniformly draw a mini-batch of samples from $\mathcal{B}$
        For each sample (s, a, r, s'), calculate the target value as $y_T = r + \gamma \max_{a \in \mathcal{A}(s')} \hat{q}(s', a, w_T)$, where w_T is the parameter of the target network
        Update the main network to minimize $(y_T - \hat{q}(s, a, w))^2$ using the mini-batch of samples
    Set $w_T = w$ every C iterations

Remarks:

- Why no policy update?
- The network input and output are different from the DQN paper.

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

Initialization: A main network and a target network with the same initial parameter.
Goal: Learn an optimal target network to approximate the *optimal* action values from the experience samples generated by a given behavior policy π_b.

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

For each iteration, do

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

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

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

Set $w_T = w$ every C iterations

Remarks:

- Why no policy update?
- The network input and output are different from the DQN paper.

Illustrative example:

- We need 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 Fig. (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.

Setup:

- One single episode is used to train the network.
- This episode is generated by an exploratory behavior policy shown in Fig. (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.

Setup:

- One single episode is used to train the network.
- This episode is generated by an exploratory behavior policy shown in Fig. (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.

Setup:

- One single episode is used to train the network.
- This episode is generated by an exploratory behavior policy shown in Fig. (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.

Setup:

- One single episode is used to train the network.
- This episode is generated by an exploratory behavior policy shown in Fig. (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.

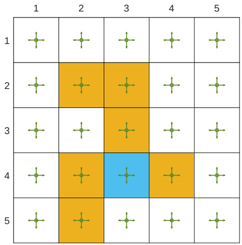
See details in the book.

Deep Q-learning

Setup:

- One single episode is used to train the network.
- This episode is generated by an exploratory behavior policy shown in Fig. (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.



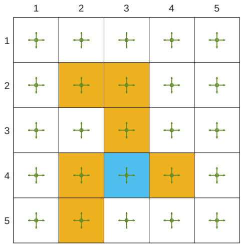
The behavior policy.

An episode of 1,000 steps.

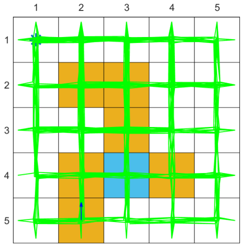
The obtained policy.

The TD error converges to zero.

The state estimation error converges to zero.



The behavior policy.



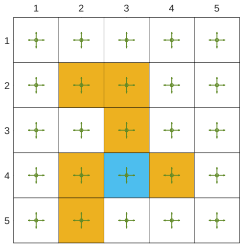
An episode of 1,000 steps.

The obtained policy.

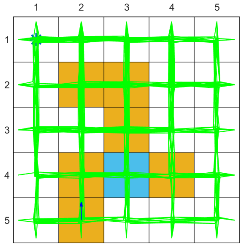
The TD error converges to zero.

The state estimation error converges to zero.

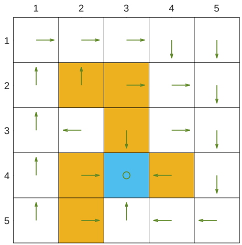
Deep Q-learning



The behavior policy.



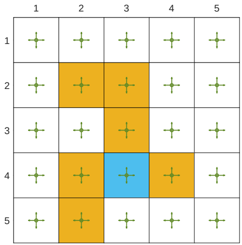
An episode of 1,000 steps.



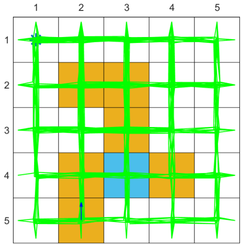
The obtained policy.

The TD error converges to zero.

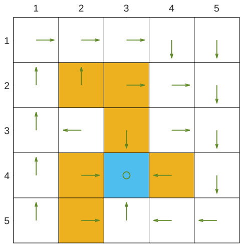
The state estimation error converges to zero.



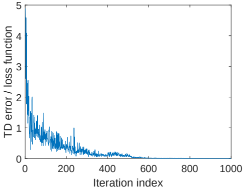
The behavior policy.



An episode of 1,000 steps.



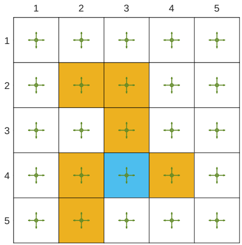
The obtained policy.



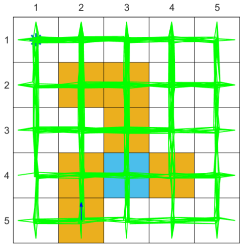
The TD error converges to zero.

The state estimation error converges to zero.

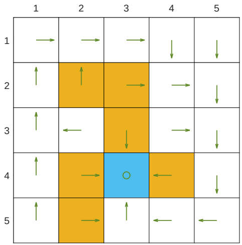
Deep Q-learning



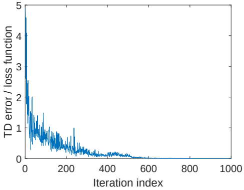
The behavior policy.



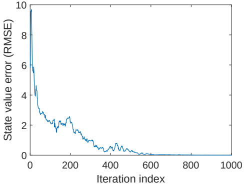
An episode of 1,000 steps.



The obtained policy.

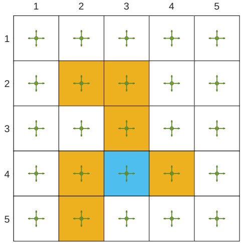


The TD error converges to zero.



The state estimation error converges to zero.

What if we only use a single episode of 100 steps? Insufficient data



The behavior policy.

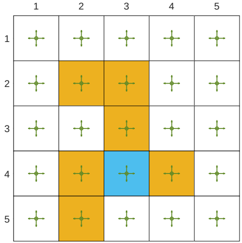
An episode of 100 steps.

The final policy.

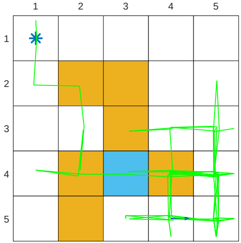
The TD error converges to zero.

The state error does not converge to zero.

What if we only use a single episode of 100 steps? Insufficient data



The behavior policy.



An episode of 100 steps.

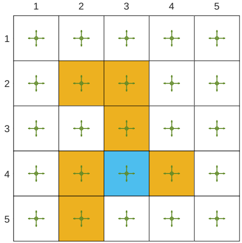
The final policy.

The TD error converges to zero.

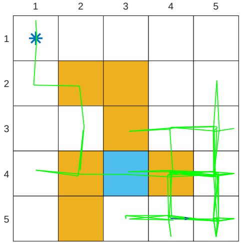
The state error does not converge to zero.

Deep Q-learning

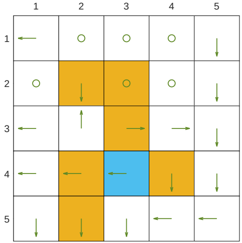
What if we only use a single episode of 100 steps? Insufficient data



The behavior policy.



An episode of 100 steps.

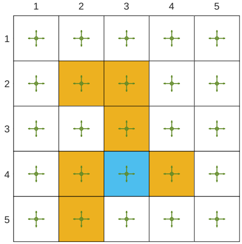


The final policy.

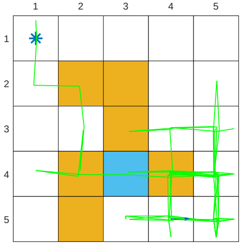
The TD error converges to zero.

The state error does not converge to zero.

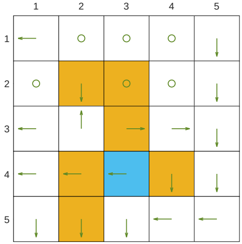
What if we only use a single episode of 100 steps? Insufficient data



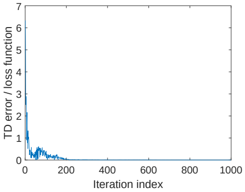
The behavior policy.



An episode of 100 steps.



The final policy.

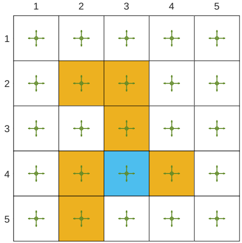


The TD error converges to zero.

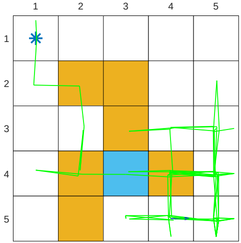
The state error does not converge to zero.

Deep Q-learning

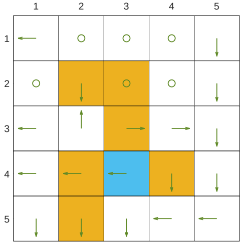
What if we only use a single episode of 100 steps? Insufficient data



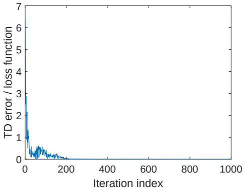
The behavior policy.



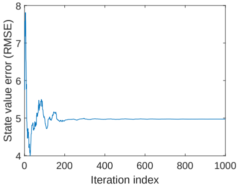
An episode of 100 steps.



The final policy.



The TD error converges to zero.



The state error does not converge to zero.

Outline

Summary

This lecture introduces the method of value function approximation.

- First, understand the basic idea.
- Second, understand the basic algorithms.