Lecture 8: Value Function Methods

Indirect RL with Func Approximation

Dr. Wen Fuxi

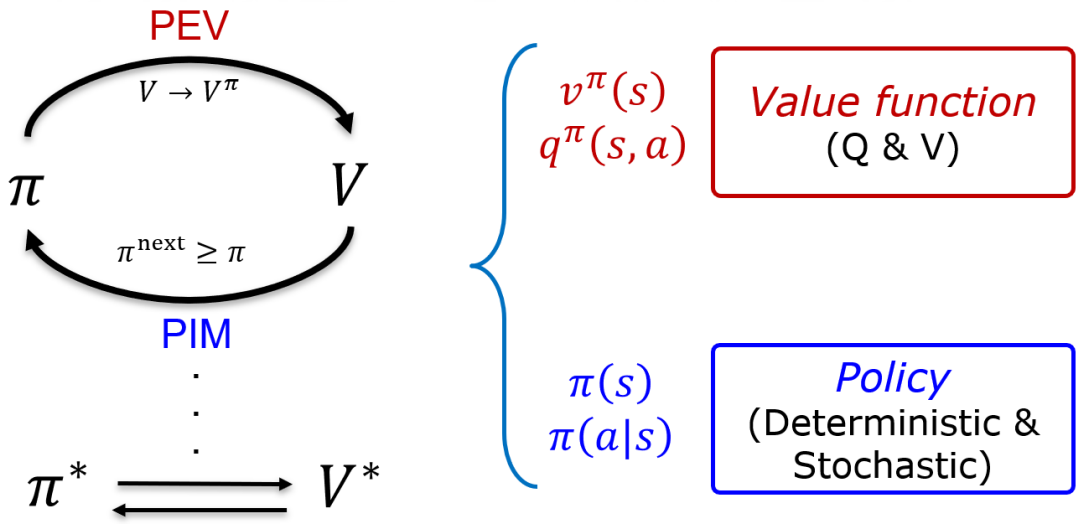
Basis of Indirect Reinforcement Learning

- **Indirect RL** seeks to find the solution to the Bellman Optimal equation

$$v^*(s) = \max_{\pi} \left\{ \mathbb{E}_{\pi} \left\{ r + \gamma v^* \left(s' \right) \right\} \right\}$$

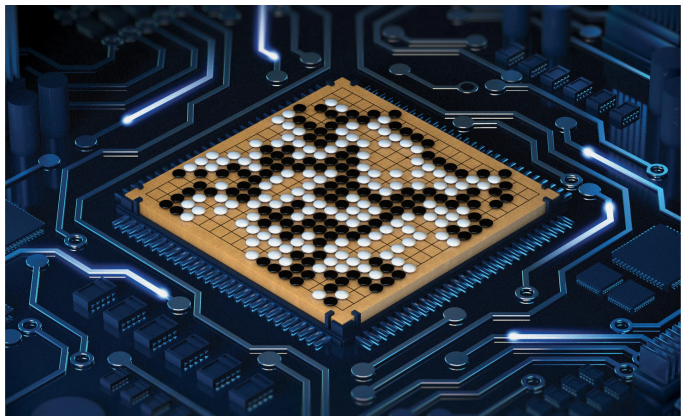
- **Generalized Policy Iteration (GPI)** framework

  Policy EValuation (PEV) + Policy IMprovement (PIM)

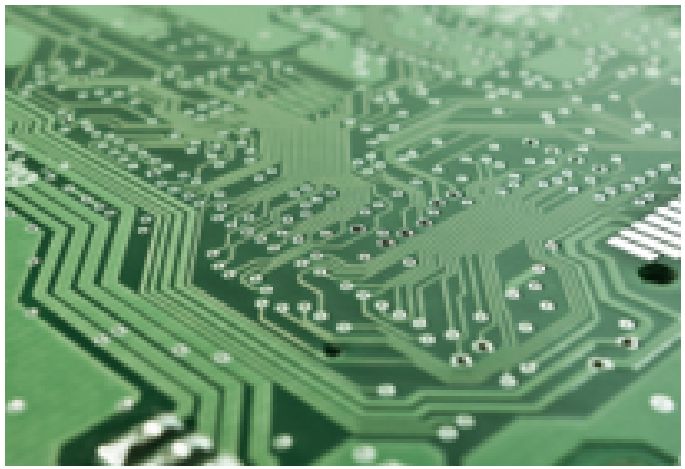


Motivating examples: Large-scale problems

- AlphaGo learned to master the ancient Chinese game of Go, with an astonishing 10 to the power of 170 (10^{170}) possible board configurations.

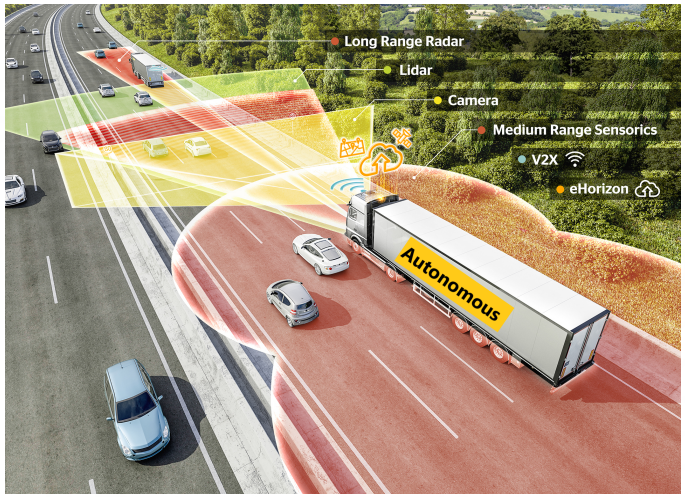


- Flexible 3D Floorplanning, Integrated Circuit (IC) design.



Motivating examples: Large-scale problems

- Robot / Autonomous car: Continuous state space.



Motivating examples: from table to function

State and action values are represented by tables so far $\rightarrow$ *Tabular method*.

- For a given policy, state value:

State	s_1	s_2	$\cdots$	s_n
True value	$v_\pi(s_1)$	$v_\pi(s_2)$	$\cdots$	$v_\pi(s_n)$
Estimated value	$\hat{v}(s_1)$	$\hat{v}(s_2)$	$\cdots$	$\hat{v}(s_n)$

- For example, action value:

	a_1	a_2	$\cdots$	a_m
s_1	$q_\pi(s_1, a_1)$	$q_\pi(s_1, a_2)$	$\cdots$	$q_\pi(s_1, a_m)$
s_2	$q_\pi(s_2, a_1)$	$q_\pi(s_2, a_2)$	$\cdots$	$q_\pi(s_2, a_m)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
s_n	$q_\pi(s_n, a_1)$	$q_\pi(s_n, a_2)$	$\cdots$	$q_\pi(s_n, a_m)$

  - 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

Tabular RL suffers from the "Curse of Dimensionality"

- Too many states and actions to store in memory
- Too slow to learn the value of each state individually

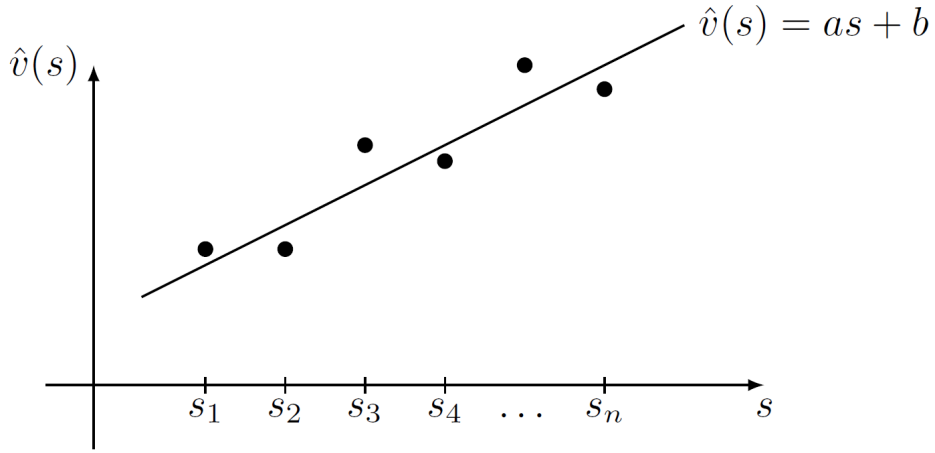
Consider an example:

- There are n states: $s_1, s_2, \ldots, s_n$.
- 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.

Motivating examples: from table to function

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



Suppose the equation of the straight line is

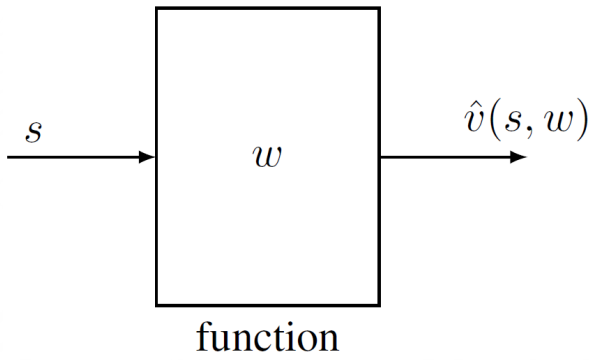
$$\hat{v}(s, w) = as + b = \underbrace{[a, b]}_{w^T} \underbrace{\begin{bmatrix} s \\ 1 \end{bmatrix}}_{f(s)} = w^T f(s)$$

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

Motivating examples: from table to function

Difference between the tabular and function methods:

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



- For example, $s \xrightarrow{(1)} f(s) \xrightarrow{(2)} w^T f(s) = \hat{v}(s, w)$

Benefit: Store $|\mathcal{S}|$ state values vs store a lower-dimensional w.

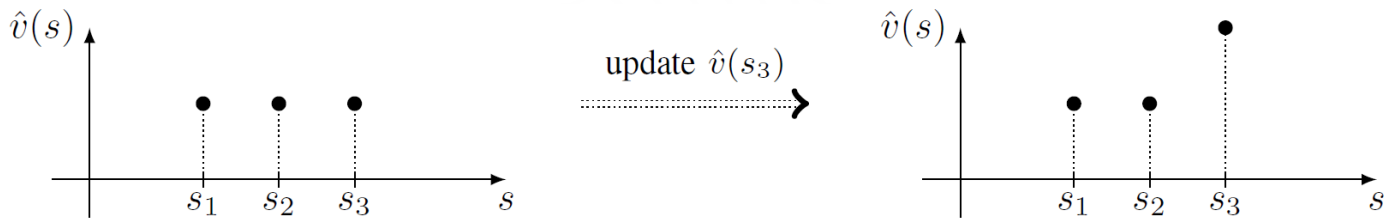
Motivating examples: from table to function

Difference 2: How to update the value of a state?

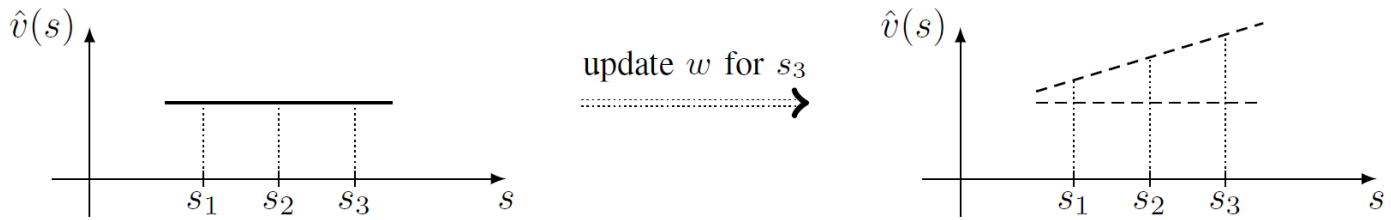
- If the values are represented by a table $\rightarrow$ we can directly rewrite the value in the table.
- If the values are represented by a function $\rightarrow$ we must update w to change the values indirectly.

Motivating examples: from table to function

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

Cost - There is no free lunch!

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:

$$\hat{v}(s, \boldsymbol{w}) = w_1 s^2 + w_2 s + w_3 = \underbrace{[w_1, w_2, w_3]}_{w^T} \underbrace{\begin{bmatrix} s^2 \\ s \\ 1 \end{bmatrix}}_{f(s)} = \boldsymbol{w}^T f(s).$$

In this case,

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

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.

RL with Function Approximation

(1) Value approximation Only

- Substitute tabular values by a parameterized function.
- e.g., Deep Q-learning

(2) Policy approximation Only

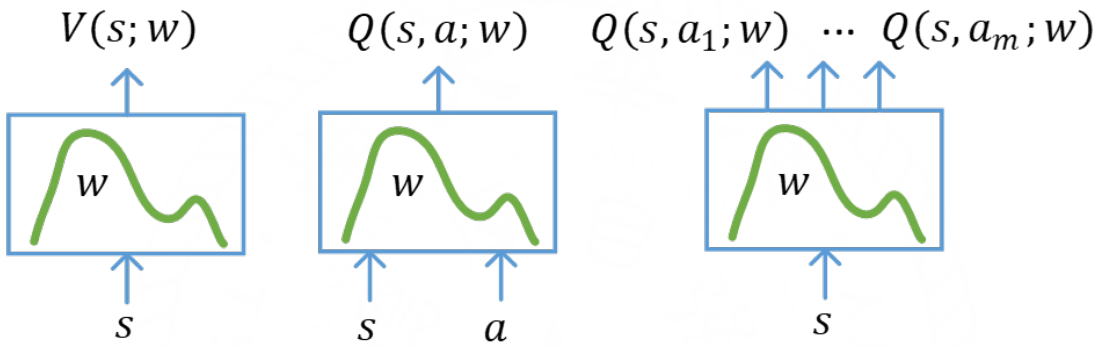
- Represent policy by a parameterized function.
- e.g., REINFORCE, finite-horizon policy gradient

(3) Actor-Critic (AC) architecture

- Approximate both value and policy.
- e.g., A3C, DDPG, PPO, TRPO, SAC, DSAC, etc

Three Types of Value Approximation

Parameterized value function



- (A) State-value approximation

$$V(s;w) \approx v^{\pi}(s), \forall s \notin \mathcal{S}$$

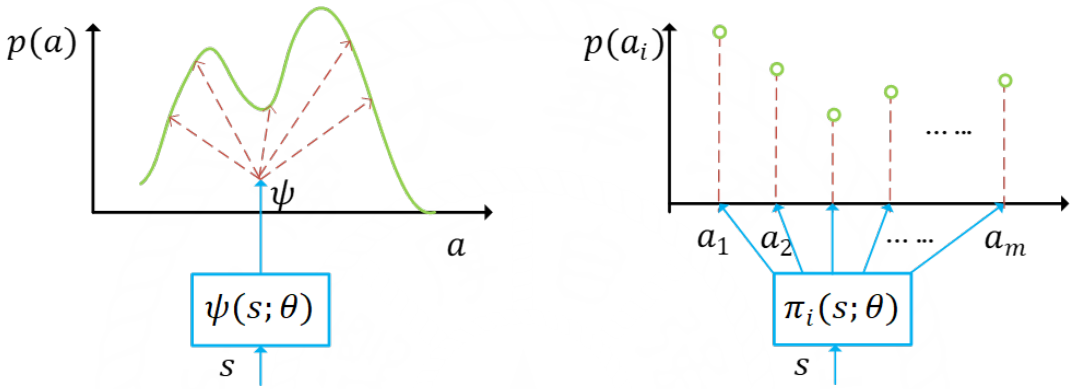
- (B.1) Action-value approximation (continuous/discrete action space)

$$Q(s,a;w) \approx q^{\pi}(s,a), \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$$

- (B.2) Action-value approximation (discrete action space)

Three Types of Policy Approximation

Parameterized policy



- (A) Deterministic policy

$$a = \pi(s;\theta) \approx \pi(s), \forall s \in \mathcal{S}$$

- (B.1) Stochastic policy (continuous action space)

$$p(a;\psi(s,\theta)) \approx \pi(a \mid s), \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$$

- (B.2) Stochastic policy (discrete action space)

$$p\left(a_i \mid s;\theta\right) \approx \pi\left(a_i \mid s\right), \forall s \in \mathcal{S}, i = 1, \cdots, m$$

Selection of function approximators

Function Approximation

Select a parameterized function that closely matches a target function.

(1) The target function is directly known, (2) Only have a set of target points.

An important question has not been answered:

$\rightarrow$ How to select the function $\hat{v}(s, w)$?

- The first approach, which was widely used before, is to use a linear function

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

  Here, $f(s)$ is the feature vector, which can be a polynomial basis, a Fourier basis, Radial basis function, ... We have seen this in the motivating example and will see it again in the illustrative examples later.

- The second approach, which is widely used nowadays, uses a neural network (or deep) as a nonlinear function approximator.

Selection of function approximators

Approximate a target function as a linear combination of features

$$g(\cdot; w) = \boldsymbol{w}^{\mathrm{T}} \cdot \boldsymbol{F}(s)$$

- $\boldsymbol{w} = [w_1, w_2, \cdots, w_l]^{\mathrm{T}} \in \mathbb{R}^l$: weights to be learned.
- $\boldsymbol{F}(s) = [f_1(s), f_2(s), \cdots, f_l(s)]^{\mathrm{T}} \in \mathbb{R}^l$: features of state s (basis function).

Choices of basis functions:

- (1) Polynomial basis function
- (2) Fourier basis function
- (3) Radial basis function (RBF)
- (4) Radial basis function network (RBFN)

Polynomial basis function

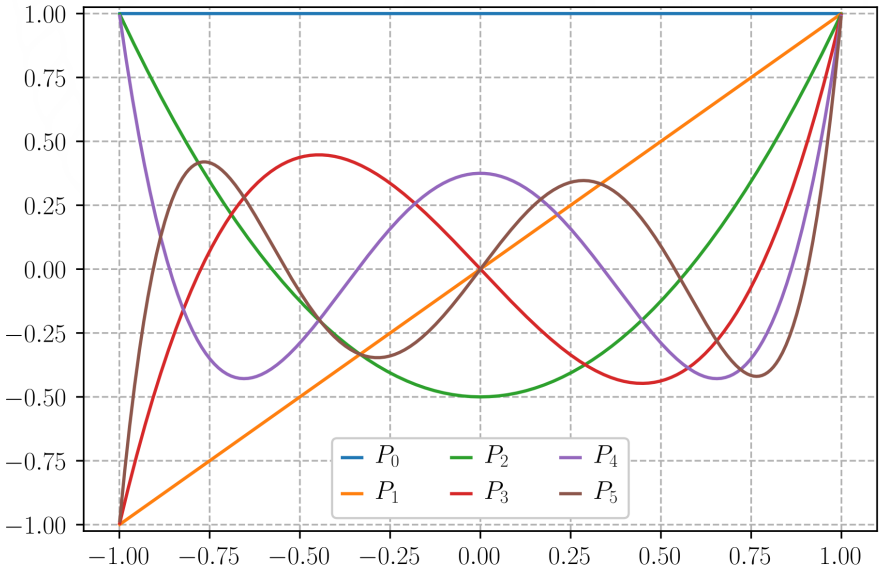
Suitable for continuous space

Polynomials with d-order

$$s = [s_1, s_2, \ldots, s_n]^{\mathrm{T}} \in \mathbb{R}^n$$

$$f_i(s) = \prod_{j=1}^{n} \prod s_j^{c_{i,j}}, \sum_j c_{i,j} \leq d, c_{i,j} \in \{0, 1, \ldots, n\}$$

Legendre polynomial basis



Fourier basis function

Fourier transform

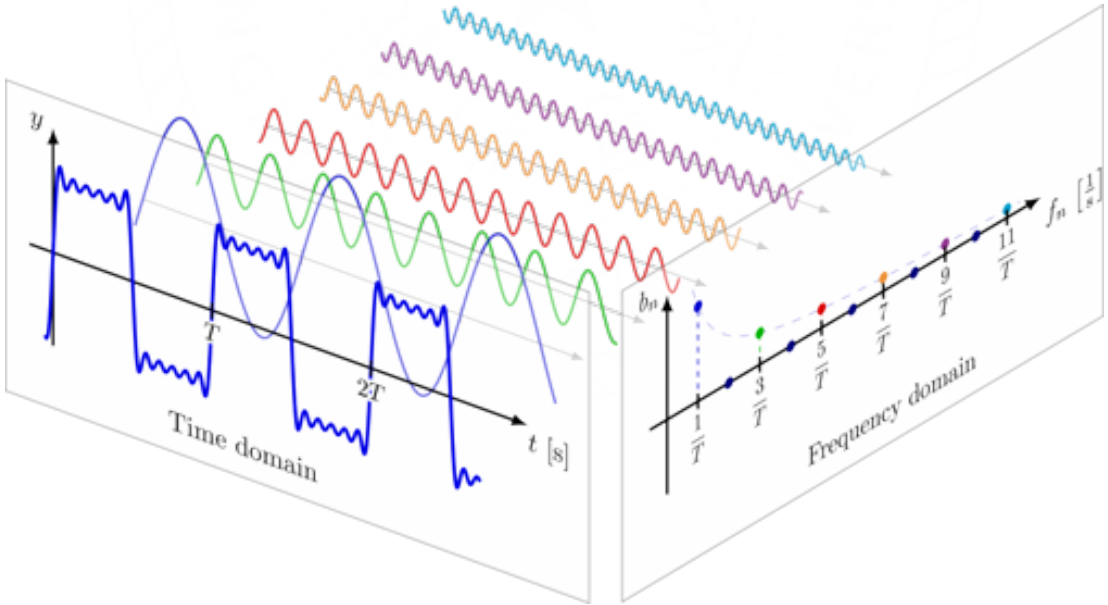
d-order Fourier cosine approximation

$$f_i(s) = \cos\left(\pi c_i^{\mathrm{T}} s\right)$$

$$c_i = [c_{i,1}, c_{i,2}, \ldots, c_{i,n}]^{\mathrm{T}}, c_{i,j} \in \{0, 1, \ldots, d\}$$

$$s = [s_1, s_2, \ldots, s_n]^{\mathrm{T}}, 0 \leq s_j \leq 1$$

Fourier transform



Radial basis functions (RBF)

Michael Powell in 1977

A real-valued function whose output depends only on the distance between the input and some fixed points.

Gaussian RBF

$$f_i(s) = \exp\left(-\frac{\|s-\mu_i\|^2}{2\sigma_i^2}\right)$$

$$s = [s_1, s_2, \cdots, s_n]^{\mathrm{T}} \in \mathbb{R}^n$$

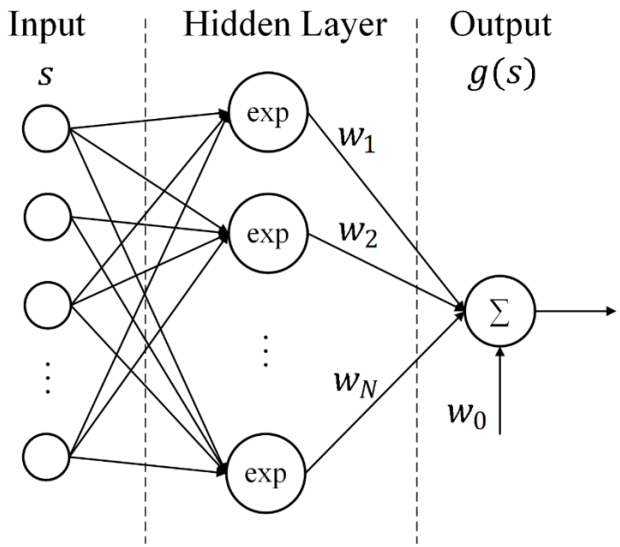
Radial basis function network (RBFN)

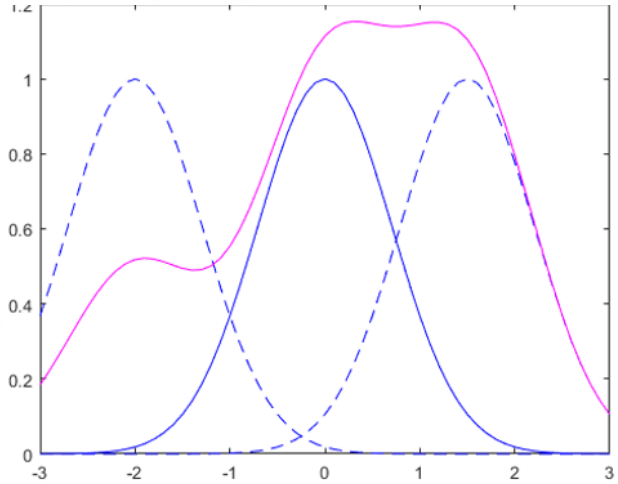
Broomhead & Lowe in 1988

Three-layer RBF network

$$g(s) = w_0 + \sum_{i=1}^{N} w_i \exp\left(-\frac{\|s-\mu_i\|^2}{2\sigma_i^2}\right)$$

A linear input layer, a hidden layer with a nonlinear RBF activation function, and a linear output layer.





Objective function

Introduce more formally:

- 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 the policy improvement. (Policy evaluation $\rightarrow$ Policy improvement)

To find the optimal w, we need two steps.

1. The first step is to define an objective function.
2. The second step is to derive algorithms for optimizing the objective function.

Objective function

The objective function is

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

- 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 ?

- There are several ways to define the probability distribution of S.

Value approximation problem

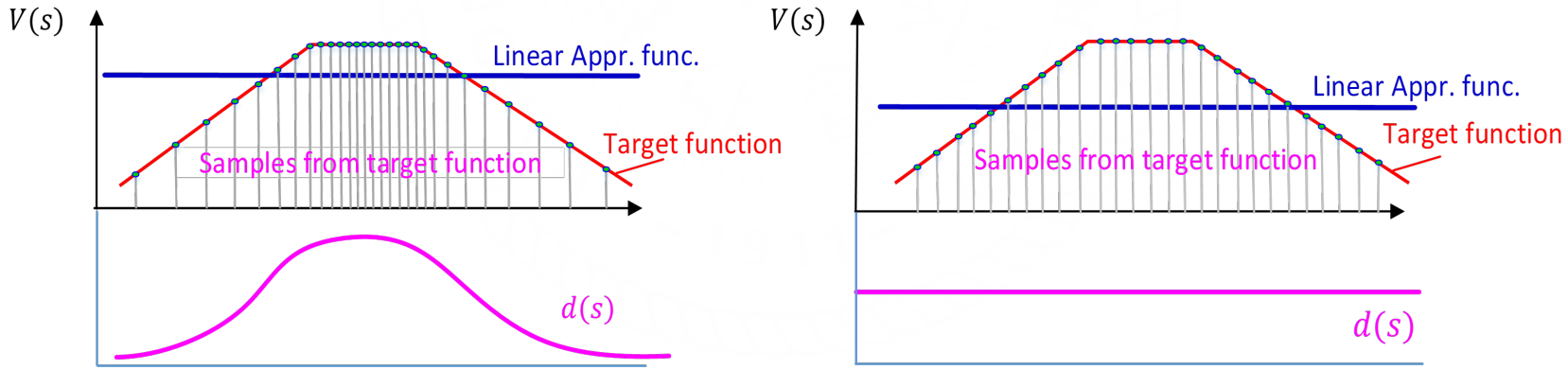
Minimize the difference between the target and approximate functions under a specific measure.

$$V(s;w) \approx v^{\pi}(s), \forall s \in \mathcal{S}$$

$$\downarrow$$

$$\min_{w} J(w) = \mathbb{E}_{s\sim d(s)}\left\{\phi\left(v^{\pi}(s),\, V(s;w)\right)\right\} = \sum_{s} d(s)\phi\left(v^{\pi}(s),\, V(s;w)\right)$$

State distribution $d(s)$ in the expectation really matters



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

$$J(w) = \mathbb{E}\left[(v_\pi(S) - \hat{v}(S, w))^2\right] = \frac{1}{|\mathcal{S}|}\sum_{s\in\mathcal{S}}(v_\pi(s) - \hat{v}(s, w))^2 .$$

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

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

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 of rarely visited states.

Objective function - Stationary distribution

More explanation about the stationary distribution:

Distribution: Distribution of the state

Stationary: Long-run behavior

After the agent runs for 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 following lecture.

Objective function - Stationary distribution

Illustrative example: Given a policy shown in the figure.

- Let $n_\pi(s)$ denote the number of times that s has been visited in a very long episode generated by π.
- Then, $d_\pi(s)$ can be approximated by

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

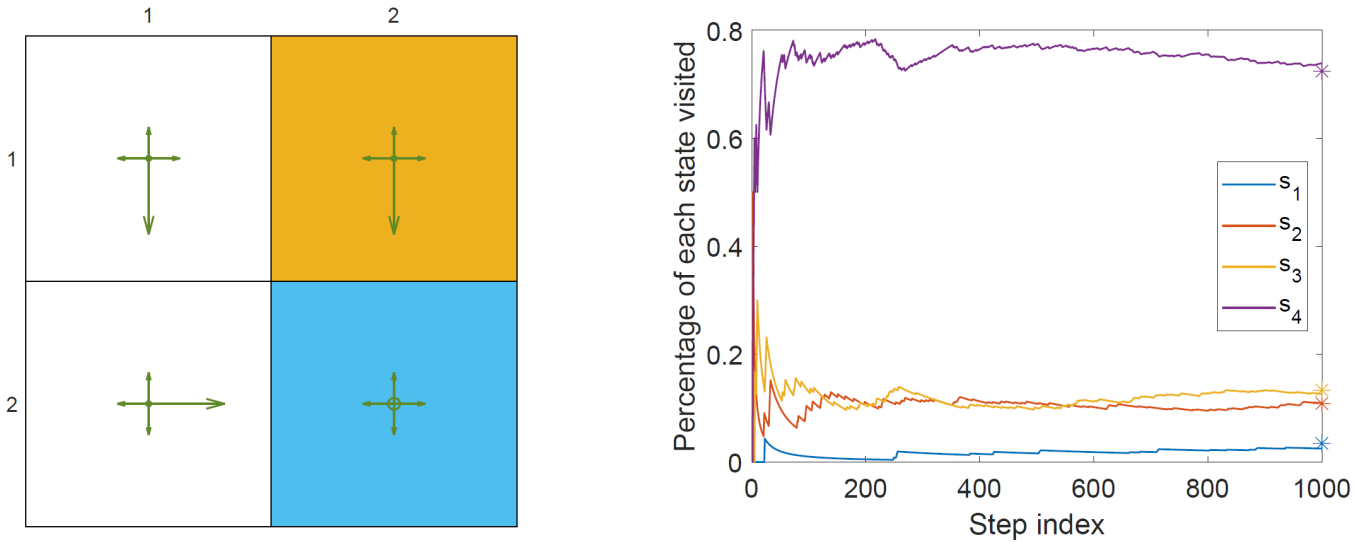


Figure: Long-run behavior of an ϵ-greedy policy with $\epsilon = 0.5$.

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

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

On-policy objective function

- One common solution is to minimize mean squared error (MSE) under the stationary state distribution (SSD) of the target policy.

$$\min_{w} J(w) = \mathbb{E}_{s\sim d_\pi}\left\{(v^\pi(s) - V(s;w))^2\right\}$$

- Its gradient (called value gradient) is

$$\nabla_w J(w) \propto -\mathbb{E}_{s\sim d_\pi}\left\{(v^\pi(s) - V(s;w))\frac{\partial V(s;w)}{\partial w}\right\}$$

- Questions to calculate value gradients
  - True value of $v^\pi(s)$ is usually unknown
  - How to handle a variable's randomness

Q1: Replace true value with its estimate

Substitute the actual value with the value estimate.

$$v^{\pi}(s) \cong \mathbb{E}_{\pi}\left\{R_t \mid s\right\}$$

$$R_t = \begin{cases} G_{t:T} & \text{for} \quad \text{MC} \\ r + \gamma V(s'; w) & \text{for} \quad \text{TD}(0) \end{cases}$$

- For MC

$$\nabla J_{\text{MC}}(w) = -\mathbb{E}_{s\sim d_{\pi}}\left\{\left(\mathbb{E}_{\pi}\left\{G_{t:T}\right\} - V(s; w)\right)\frac{\partial V(s; w)}{\partial w}\right\}$$

- For TD

$$\nabla J_{\text{TD}}(w) = -\mathbb{E}_{s\sim d_{\pi}}\left\{\left(\mathbb{E}_{a\sim\pi, s'\sim\mathcal{P}}\left\{r + \gamma V\left(s'; w\right)\right\} - V(s; w)\right)\frac{\partial V(s; w)}{\partial w}\right\}$$

- It is also called "semi-gradient"

Q2 (A): Stochastic Gradient Descent (SGD)

The update rule for gradient descent is:

$$\theta = \theta - \eta \nabla J(\theta)$$

where:

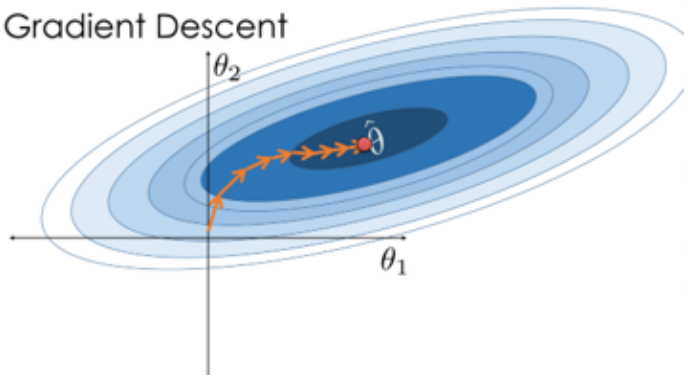
- θ represents the parameters of the model.
- η is the learning rate (step size).
- $\nabla J(\theta)$ is the gradient of the loss function $J(\theta)$ with respect to θ.

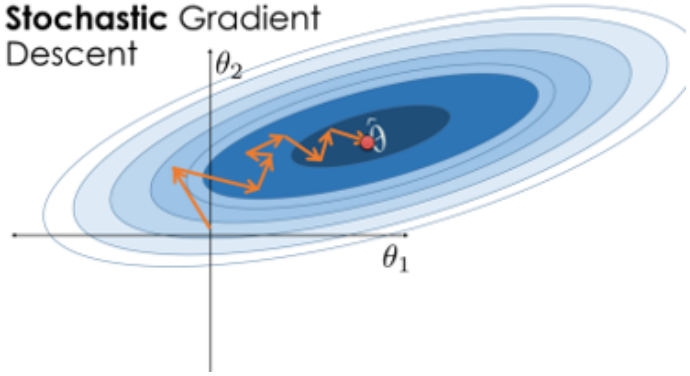
The update rule for SGD is:

$$\theta = \theta - \eta \nabla J(\theta; x_i, y_i)$$

where:

- (x_i, y_i) is a single training example (or a mini-batch).
- $\nabla J(\theta; x_i, y_i)$ is the gradient of the loss function for that example.





Q2 (A): Stochastic Gradient Descent (SGD)

- Minimize when randomness is present

$$\min_x J(x) = \mathbb{E}_\zeta\{l(x,\zeta)\}, \text{where } \zeta \text{ is a randomness.}$$

- Step 1: Gradient descent algorithm

$$x \leftarrow x - \alpha \nabla_x J(x)$$

$\nabla_x J(x) = \mathbb{E}_\zeta \{\nabla_x l(x,\zeta)\}$ is called "expected gradient"

- Step 2: Estimation of expected gradient $\rightarrow$ Average-based estimation

$$\mathbb{E}_\zeta \{\nabla_x l(x,\zeta)\} \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_x l(x_i,\zeta_i)$$

Q2 (A): Stochastic Gradient Descent (SGD)

- (1) Use all data (Batch GD)

$$x \leftarrow x - \alpha \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \nabla_x J(x_i, \zeta_i)$$

  - Computationally expensive
  - Delayed update after collecting all data

- (2) Use only one sample (SGD)

$$x \leftarrow x - \alpha \nabla_x J(x_i, \zeta_i)$$

  - Simple and fast convergence
  - Possibly escape from a local minimum
  - Randomly choose a small subsets with size $\ll |\mathcal{D}|$ (Mini-batch GD)

Q2 (B): Least Squares Estimation

Batch Least Squares

- Formulate function approximation as a regression problem

$$\min_{w} J(w) = \sum_{\mathcal{D}} \left(R_t - w^{\mathrm{T}} \cdot F(s_t) \right)^2,$$

$$\text{subject to } \mathcal{D} = \{(s_t, R_t)\}, t \in \{1 : T\}$$

$$R_t = \begin{cases} G_{t:T} & \text{for} \quad \text{MC} \\ r + \gamma w^{\mathrm{T}} F(s_{t+1}) & \text{for} \quad \text{TD(0)} \end{cases}$$

- The solution is a fixed point in value function space

$$\nabla_w J(w) = 0$$

$$w = \left(\sum_{t=1}^{T} F(s_t) F(s_t)^{\mathrm{T}} \right)^{-1} \sum_{t=1}^{T} F(s_t) R_t$$

Q2 (B): Least Squares Estimation

Incremental Least Squares

- For n features, inverse computation is $O\left(n^3\right)$ in batch LS

$$D_t^{-1} = \left(\sum_{t=1}^{T} F(s_t)\, F(s_t)^{\mathrm{T}} \right)^{-1}$$

- Apply Shermann-Morrison formula to inverse computation

$$\left(M + UV^{\mathrm{T}}\right)^{-1} = M^{-1} - \frac{M^{-1} U V^{\mathrm{T}} M^{-1}}{1 + V^{\mathrm{T}} M^{-1} U}$$

$$D_t^{-1} = [\underbrace{D_{t-1}}_{M} + \underbrace{F(s_t)}_{U} \underbrace{F(s_t)^{\mathrm{T}}}_{V}]^{-1} = D_{t-1}^{-1} - \frac{D_{t-1}^{-1} F(s_t)\, F(s_t)^{\mathrm{T}}\, D_{t-1}^{-1}}{1 + F(s_t)^{\mathrm{T}}\, D_{t-1}^{-1} F(s_t)^{\mathrm{T}}}$$

- Incremental LS is $O\left(n^2\right)$ by Shermann-Morrison formula

SGD vs LS

SGD	LS
Linear/nonlinear	Only linear
Iteratively solve with randomly shuffled samples	Directly solve with batch data
Computationally expensive	Computationally cheap
High variance	Low variance
Oscillation around minimum	Stable

Off-policy value approximation

- Use data from behavior policy b to evaluate target policy π - One solution is to use the same PEV criterion of on-policy approximation

$$\nabla_w J(w) \propto -\mathbb{E}_{s\sim d_b}\left\{\frac{d_\pi(s)}{d_b(s)}\left(v^\pi(s) - V(s;w)\right)\nabla V(s;w)\right\}$$

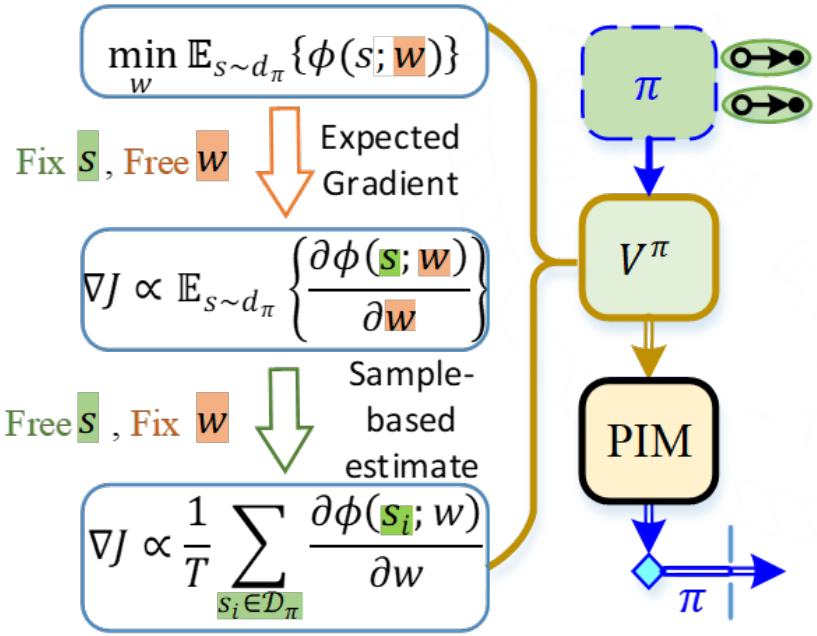


Figure: On-policy approximation

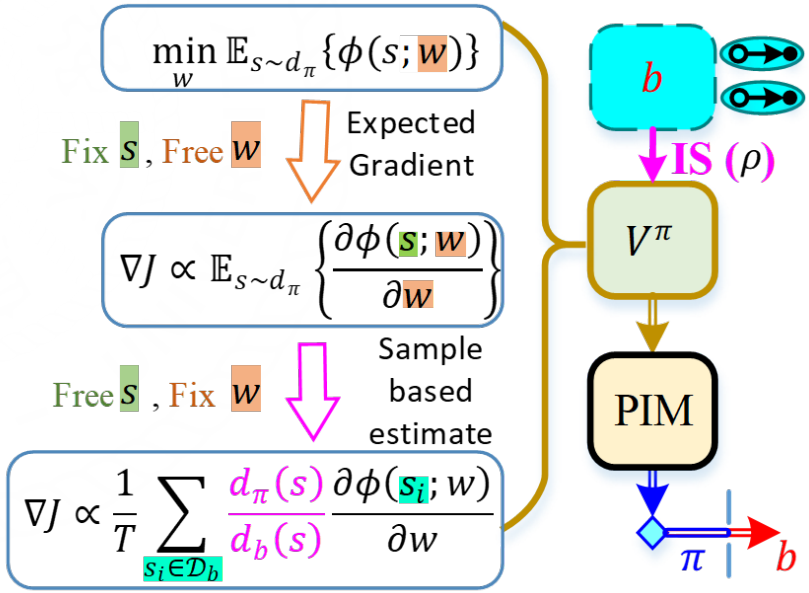


Figure: Off-policy approximation

Off-policy value approximation

- $d_\pi(s)/d_b(s)$ is computationally intractable because we have no access to the SSDs of both the target policy and the behavior policy
- An alternative is to change the PEV criterion to be weighted by the behavior state distribution

$$\min_w J(w) = \mathbb{E}_{s\sim d_b}\left\{\left(v^\pi(s) - V(s;w)\right)^2\right\}$$

$$\nabla_w J(w) \propto -\mathbb{E}_{s\sim d_b}\left\{\left(v^\pi(s) - V(s;w)\right)\frac{\partial V(s;w)}{\partial w}\right\}$$

$$v^\pi(s) = \mathbb{E}_{a\sim b, s'\sim\mathcal{P}}\left\{\rho_{t:t}\left(r+\gamma V(s';w)\right) \mid s\right\}$$

Take one-step TD as the target

$$\nabla_w J(w) = -\mathbb{E}_b\left\{\left(\rho_{t:t}\left(\boldsymbol{r}+\gamma \boldsymbol{V}\left(\boldsymbol{s}';\boldsymbol{w}\right)\right) - V(s;w)\right)\frac{\partial V(s;w)}{\partial w}\right\}$$

Off-policy value approximation

Variance reduction in off-policy TD

$$\nabla_W J(w) \propto -\mathbb{E}_b \left\{ \rho_{t:t} \left(\boldsymbol{r} + \boldsymbol{\gamma} \boldsymbol{V}\left(\boldsymbol{s}'; \boldsymbol{w}\right) - \boldsymbol{V}(\boldsymbol{s}; \boldsymbol{w}) \right) \frac{\partial V(s; w)}{\partial w} \right\}$$

For $\rho_{t:t}$, $\boldsymbol{S}$ is a fixed variable

- Proof:

$$\begin{aligned}
&\mathbb{E}_b \left\{ (1 - \rho_{t:t})\ V(s; w) \frac{\partial V(s; w)}{\partial w} \right\} \\
&= \mathbb{E}_{s\sim d_b} \left\{ \mathbb{E}_{a\sim b} \left\{ (1 - \rho_{t:t})\ V(s; w) \frac{\partial V(s; w)}{\partial w} \right\} \right\} \\
&= \mathbb{E}_{s\sim d_b} \left\{ V(s; w) \frac{\partial V(s; w)}{\partial w} \mathbb{E}_{a\sim b} \left\{ (1 - \rho_{t:t}) \right\} \right\} \\
&= \mathbb{E}_{s\sim d_b} \left\{ V(s; w) \frac{\partial V(s; w)}{\partial w} \sum_a \left[b(a \mid s) \left(1 - \frac{\pi(a \mid s)}{b(a \mid s)} \right) \right] \right\} \\
&= \mathbb{E}_{s\sim d_b} \left\{ V(s; w) \frac{\partial V(s; w)}{\partial w} \cdot 0 \right\} \\
&= 0
\end{aligned}$$

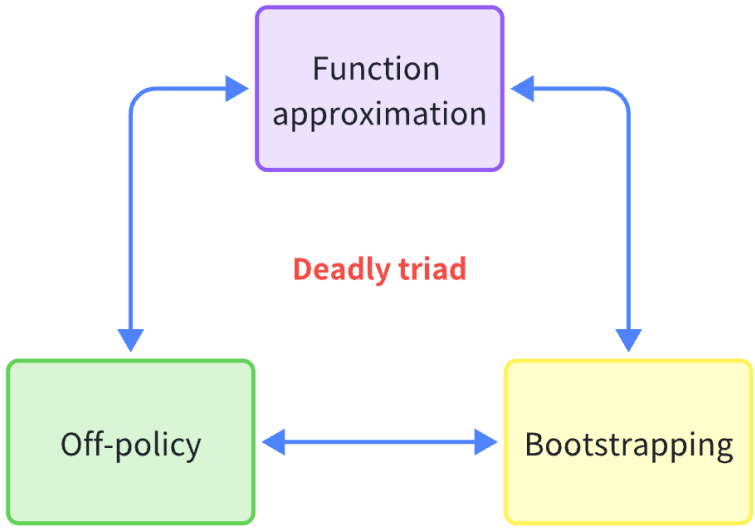
Error accumulation for target value during RL iteration

The risk of instability arises.

(1) Function approximation

(2) Bootstrapping

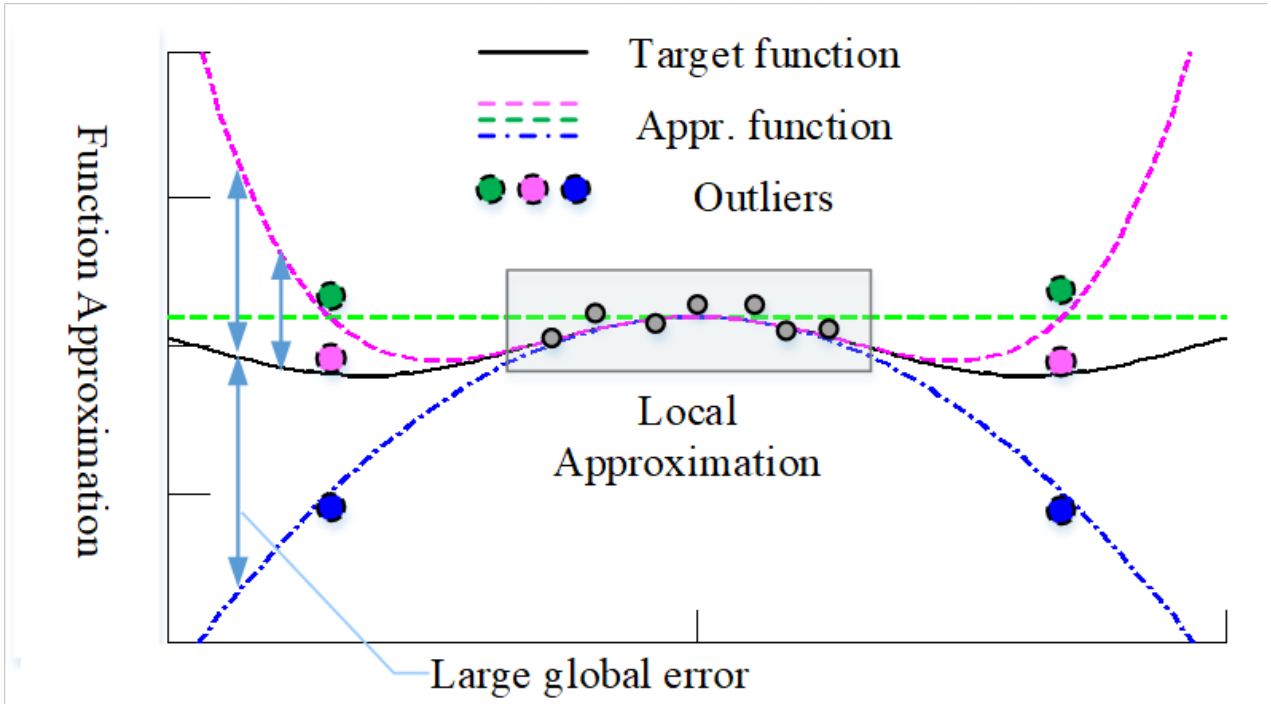
(3) Off-policy



Error accumulation for target value during RL iteration

$$v^{\text{target}} = \frac{\pi(a \mid s)}{b(a \mid s)} \left(r + \gamma V\left(s'; w\right)\right)$$

(1) The function approximation error is extremely large because of a few outliers in a less explored area



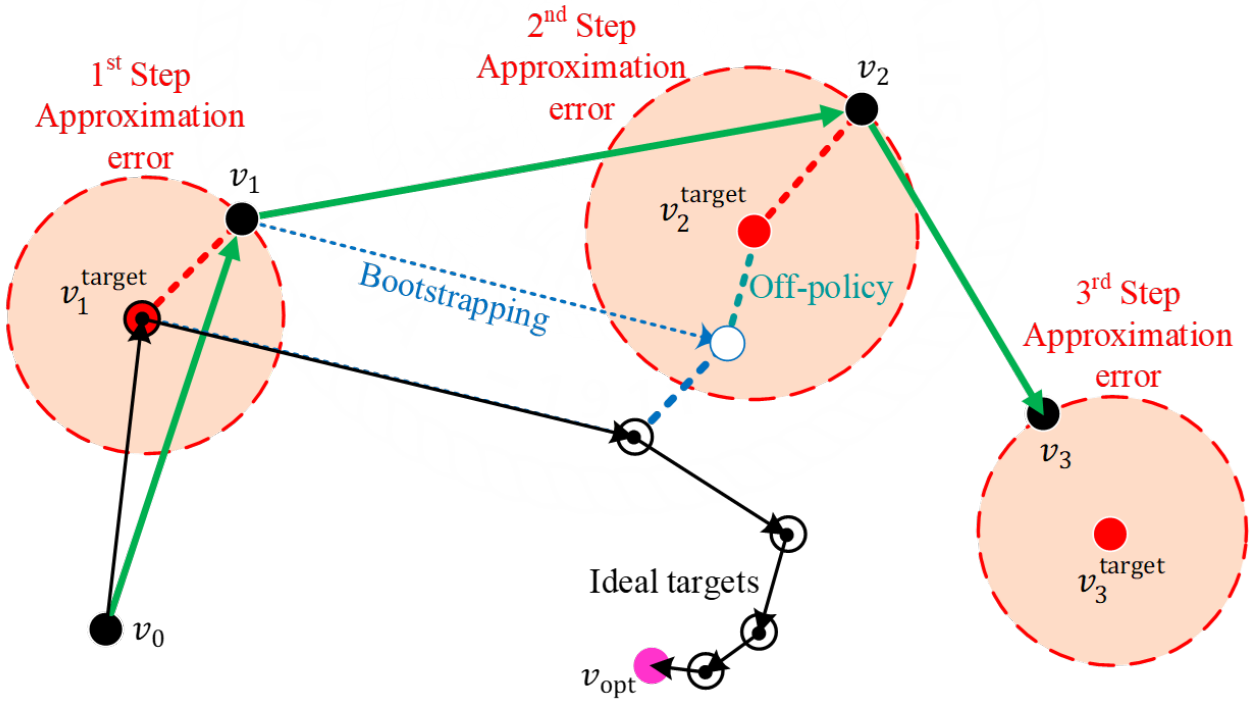
Error accumulation for target value during RL iteration

(2) Off-policy enlarges the error with the importance sampling ratio

Error accumulation for the target value during RL iteration

$$v^{\text{target}} = \frac{\pi(a \mid s)}{b(a \mid s)} \left(r + \gamma V\left(s'; w\right)\right)$$

(3) Bootstrapping induces current-step approximation error into the next target value



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

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

The true gradient above involves the calculation of an expectation.

Optimization algorithms

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

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

$$\Downarrow$$

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

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 unknown and cannot be estimated.

- We can replace $v_\pi\left(s_t\right)$ 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

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

- 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

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,

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

- In the linear case,

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

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

Linear function approximation

In the linear case where $\hat{v}(s, w) = w^T f(s)$, we have

$$\nabla_w \hat{v}(s, w) = f(s).$$

Substituting the gradient into the TD algorithm

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

yields

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

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.

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 :

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

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

- In this case,

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

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

Linear function approximation

Recall that the TD-Linear algorithm is

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

- When $f(s_t) = e_s$, 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 vector equation merely updates the s_t th entry of w_t.

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

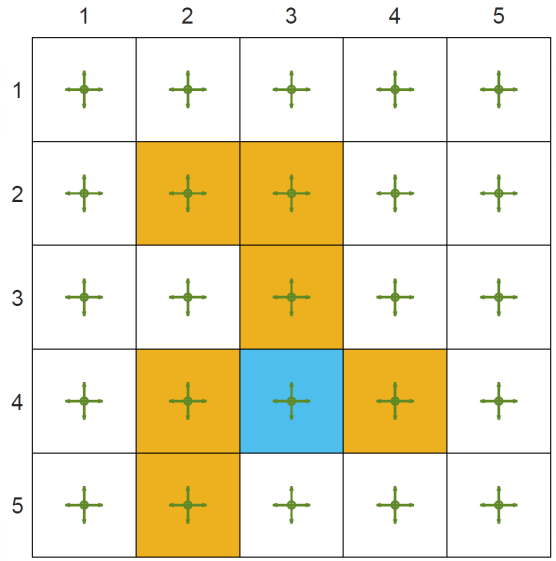
$$w_{t+1} (s_t) = w_t (s_t) + \alpha_t \left(r_{t+1} + \gamma w_t (s_{t+1}) - w_t (s_t) \right),$$

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

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

Illustrative examples

Consider a 5×5 grid-world example:

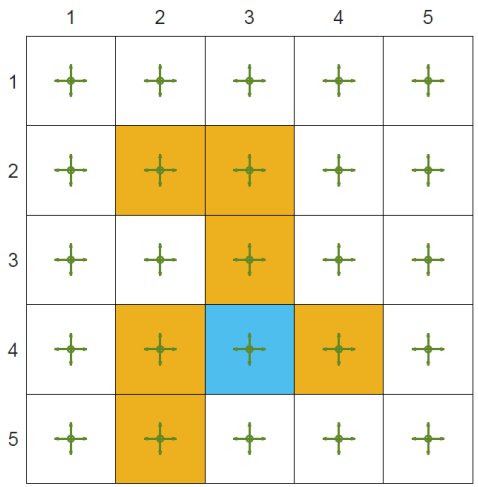


- Given a policy: $\pi(a \mid s) = 0.2$ for any s, a
- Our aim is to estimate the state values of this policy (policy evaluation problem).
- There are 25 state values in total. We next show that we can use less than 25 parameters to approximate 25 state values.
- Set $r_{\text{forbidden}} = r_{\text{boundary}}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

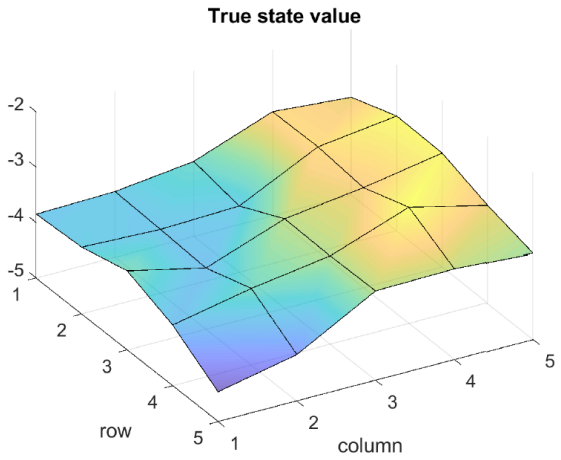
Illustrative examples

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



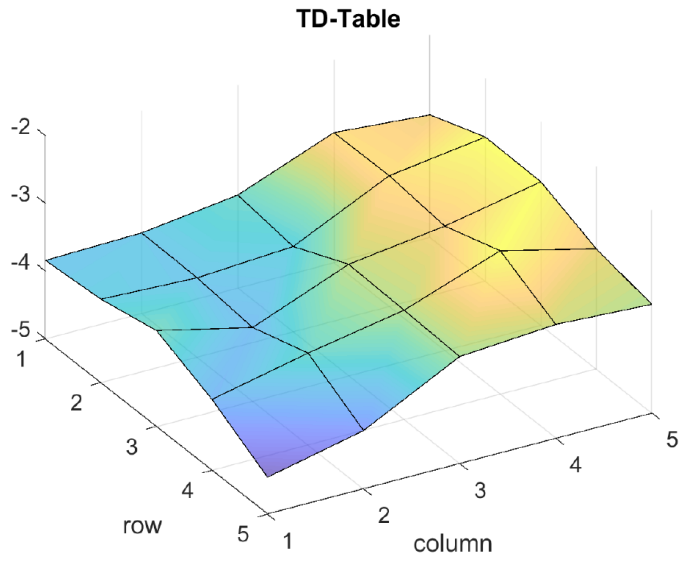
Experience samples:

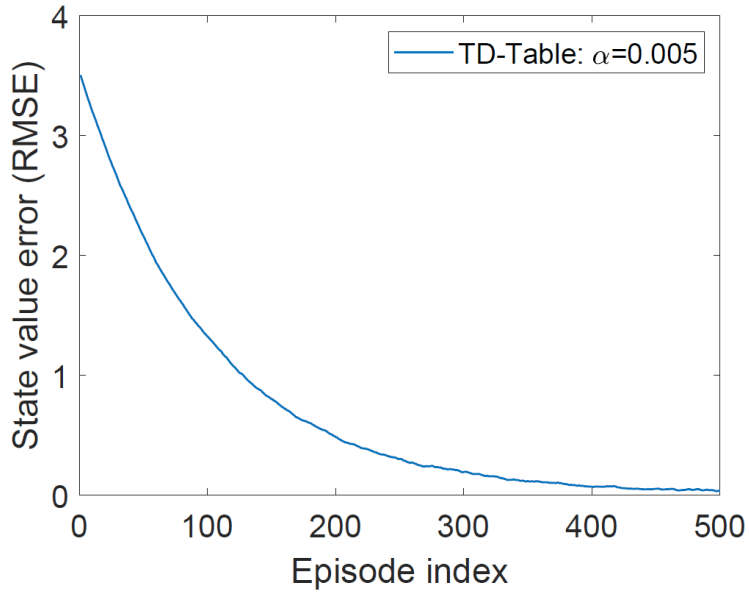
- 500 episodes were generated following the given policy.
- Each episode has 500 steps and starts from a randomly selected state-action pair following a uniform distribution.

Illustrative examples

TD-Table:

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





Illustrative examples

TD-Linear:

How to apply the TD-Linear algorithm?

- Feature vector selection:

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

- In this case, the approximated state value is

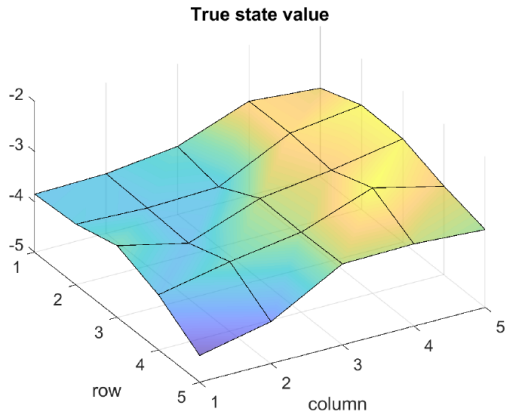
$$\hat{v}(s, w) = f^T(s)w = [1, x, y] \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = w_1 + w_2 x + w_3 y.$$

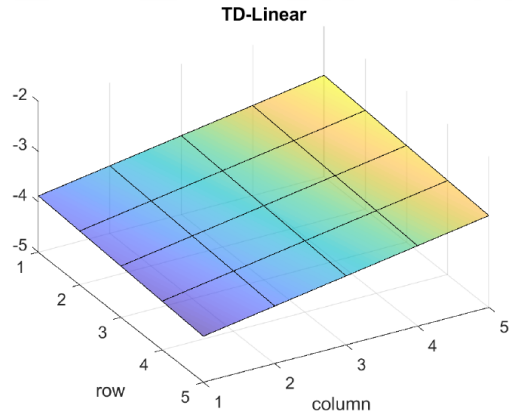
Remark: $f(s)$ can also be defined as $f(s) = [x, y, 1]^T$, where the order of the elements does not matter.

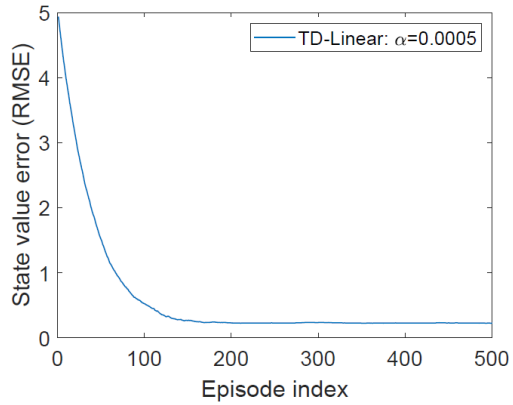
Illustrative examples

TD-Linear:

Results by the TD-Linear algorithm:







- The trend is correct, but there are errors due to limited approximation ability!
- We are trying to use a plane to approximate a non-plane surface!

Illustrative examples

We can use high-order feature vectors and hence more parameters to enhance the approximation ability.

- For example, we can consider

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

In this case,

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

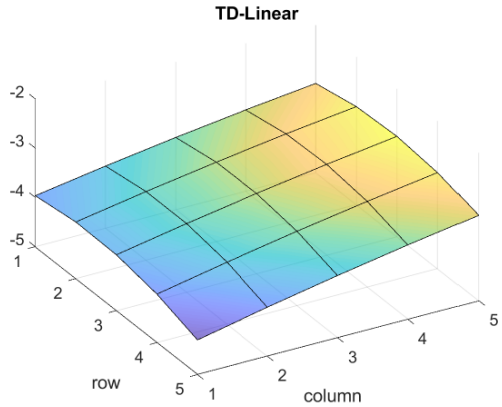
which corresponds to a quadratic surface.

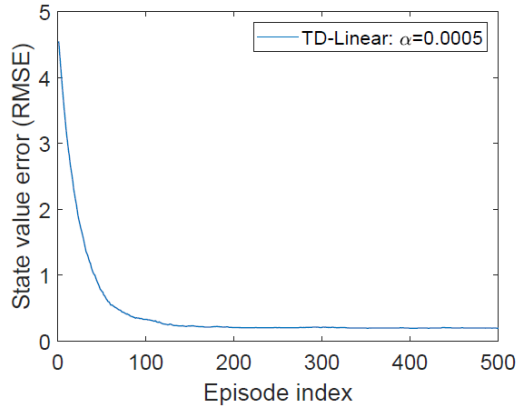
- We can further increase the dimension of the feature vector:

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

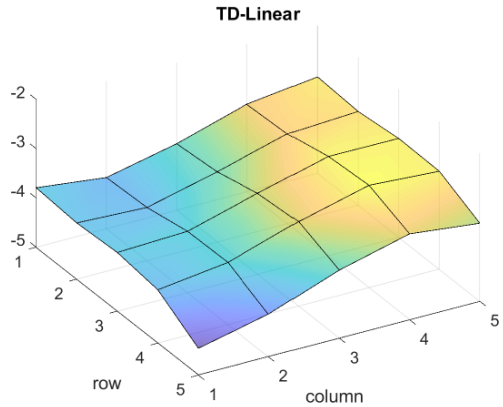
Illustrative examples

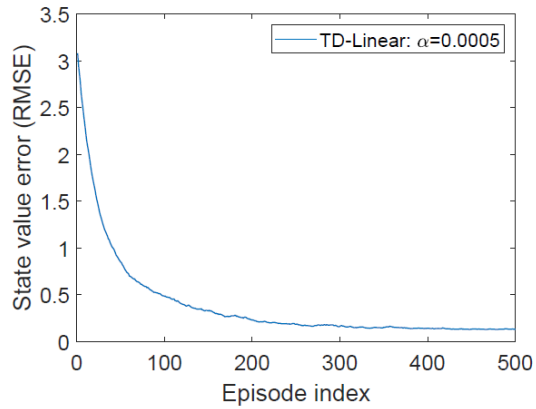
Results by the TD-Linear algorithm with higher-order feature vectors:





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





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

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:

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

   The objective function suggests that it is a policy evaluation problem.

2. The gradient-descent algorithm is

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

3. The true value function, which is unknown, in the algorithm is replaced by an approximation, leading to the algorithm:

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

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

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

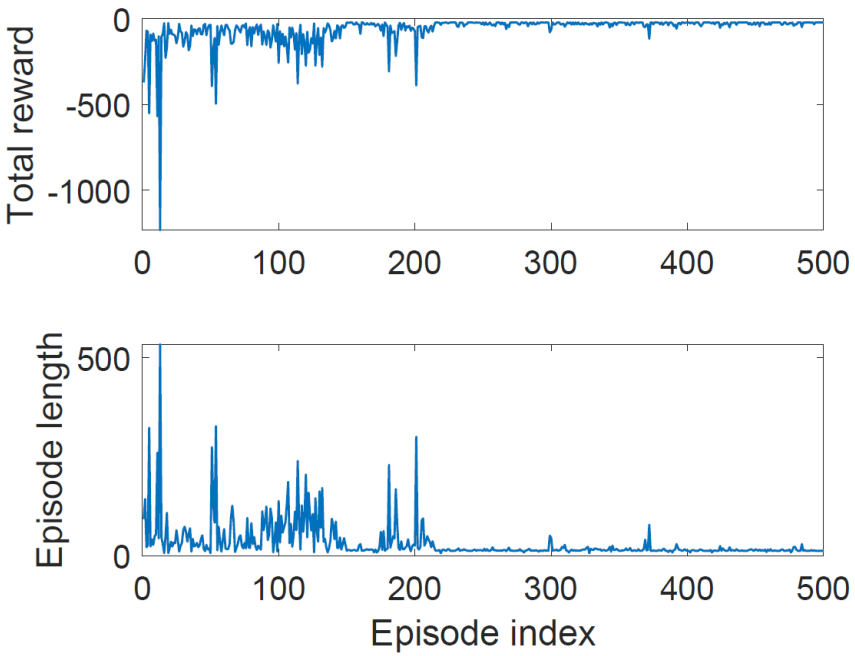
$$w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma \hat{q}\left(s_{t+1}, a_{t+1}, w_t\right) - \hat{q}\left(s_t, a_t, w_t\right) \right] \nabla_w \hat{q}\left(s_t, a_t, w_t\right).$$

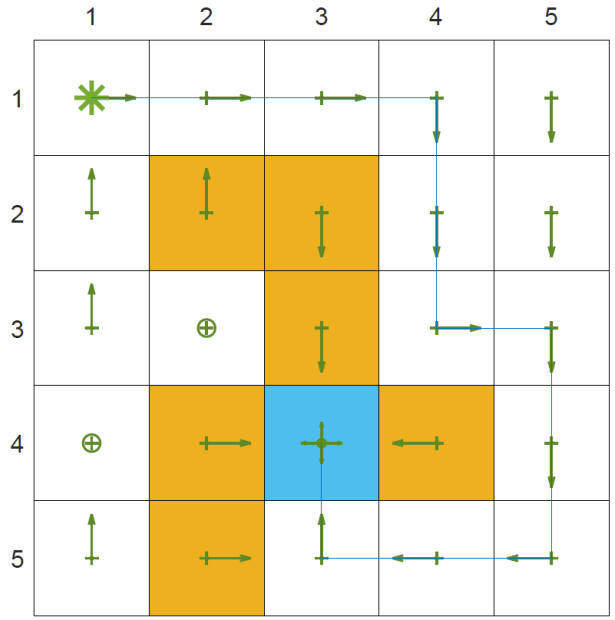
This is the same as the algorithm we introduced previously in this lecture except that $\hat{v}$ is replaced by $\hat{q}$.

Sarsa with function approximation

Illustrative example:

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





Q-learning with function approximation

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

The q-value update rule is

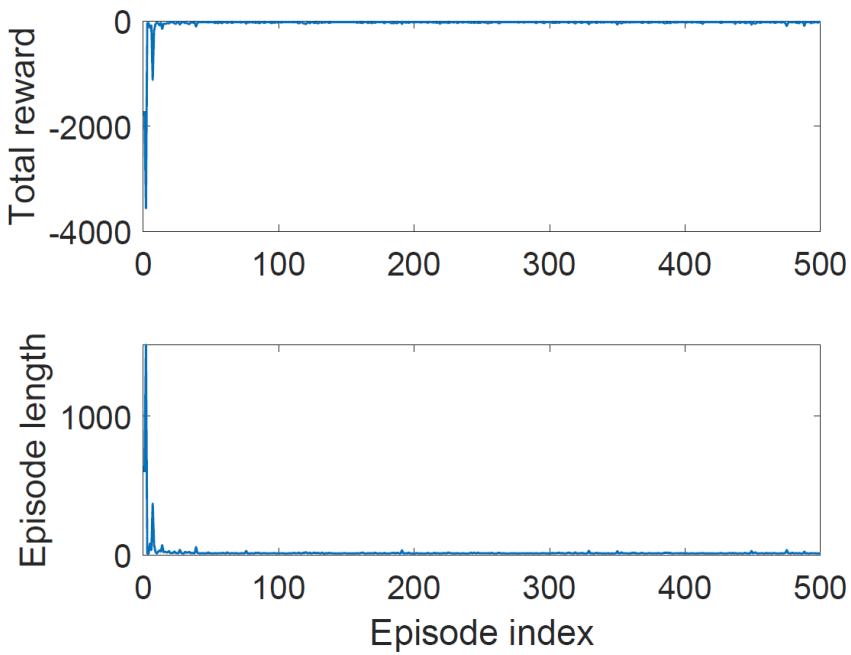
$$w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma \max_{a \in \mathcal{A}(s_{t+1})} \hat{q}\left(s_{t+1}, a, w_t\right) - \hat{q}\left(s_t, a_t, w_t\right) \right] \nabla_w \hat{q}\left(s_t, a_t, w_t\right),$$

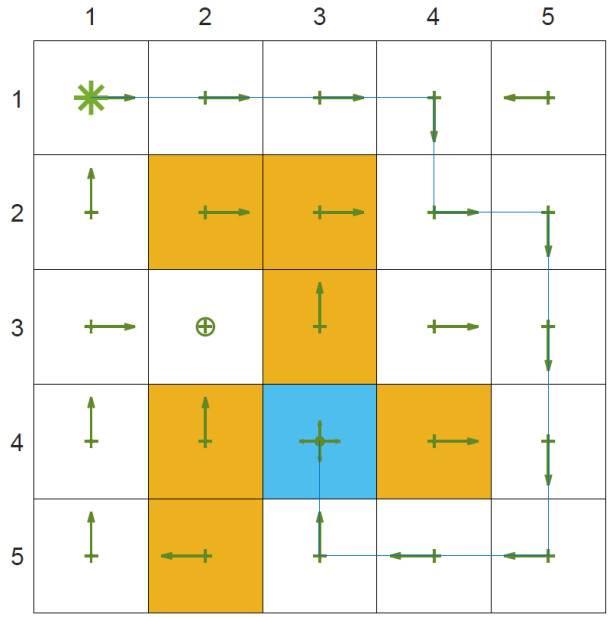
which is the same as Sarsa except that $\hat{q}\left(s_{t+1}, a_{t+1}, w_t\right)$ is replaced by $\max_{a \in \mathcal{A}(s_{t+1})} \hat{q}\left(s_{t+1}, a, w_t\right)$.

Q-learning with function approximation

Illustrative example:

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





Deep Q-learning

Deep Q-learning or deep Q-network (DQN):

- One of the earliest and most successful algorithms that introduce deep neural networks into RL.
- The role of neural networks is to be a nonlinear function approximator.
- Different from the following algorithm:

  $w_{t+1} =$
  $w_t + \alpha_t \left[r_{t+1} + \gamma \max_{a \in \mathcal{A}(s_{t+1})} \hat{q}(s_{t+1}, a, w_t) - \hat{q}(s_t, a_t, w_t) \right] \nabla_w \hat{q}(s_t, a_t, w_t)$
  because of the way of training a network.

Deep Q-learning

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

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

$$\Downarrow$$

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

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

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

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

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

- Since the optimal a depends on w,

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

- 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

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

  where w_T is the target network parameter.

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

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

- The basic idea of deep Q-learning is to use the gradient-descent algorithm to minimize the objective function.
- However, such an optimization process involves some important techniques that deserve special attention.

Deep Q-learning - Two networks

Technique 1: Two networks, a main network and a target network.

Why is it used?

- The mathematical reason is 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

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

  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?

- 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.

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

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

Deep Q-learning - Experience replay

Answer (continued):

- However, the samples are not uniformly collected because they are generated consequently by certain policies.
- To break the correlation between consequent samples, we can use the experience replay technique by uniformly drawing samples from the replay buffer.
- This is the mathematical reason why experience replay is necessary and why the experience replay must be uniform.

Deep Q-learning - Experience replay

Revisit the tabular case:

- Question: Why does tabular Q-learning not 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

Illustrative example:

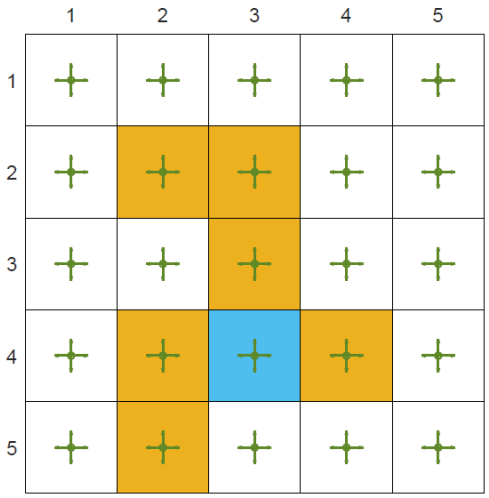
- We need to learn optimal action values for every state-action pair.
- Once the optimal action values are obtained, the optimal greedy policy can be obtained immediately.

Deep Q-learning

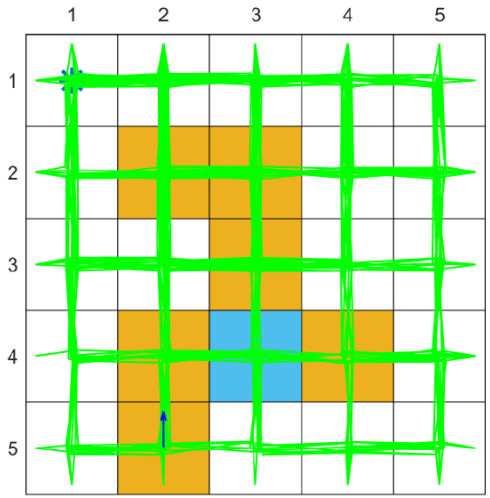
Setup:

- 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.

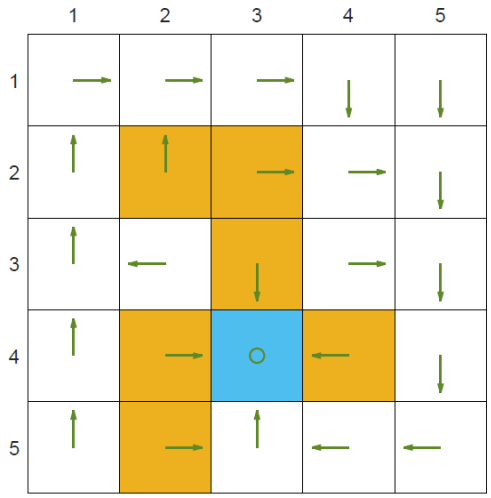
Deep Q-learning



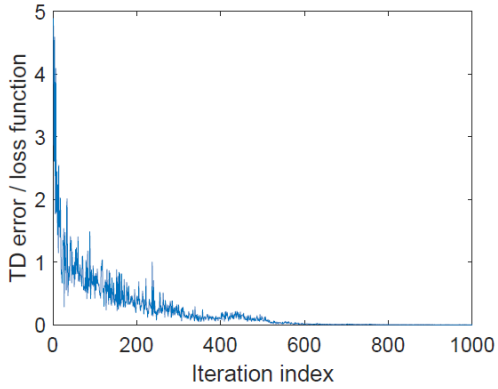
The behavior policy.



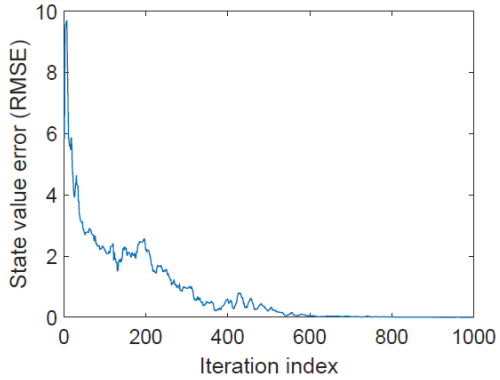
An episode of 1,000 steps.



The obtained policy.



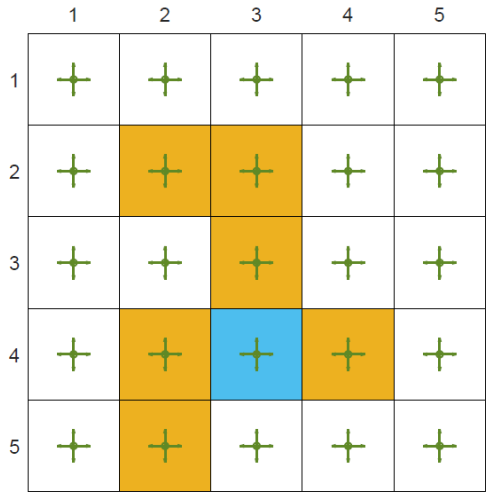
The TD error converges to zero.



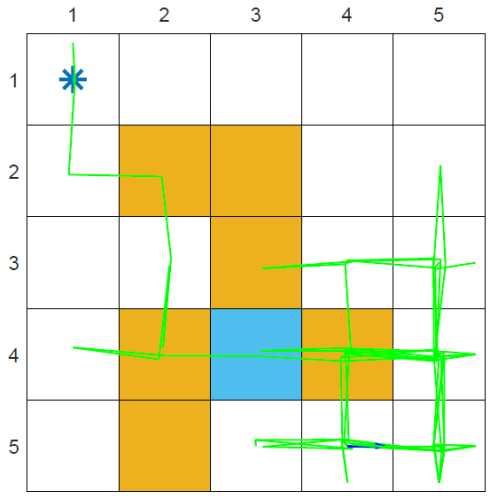
The state estimation error converges to zero.

Deep Q-learning

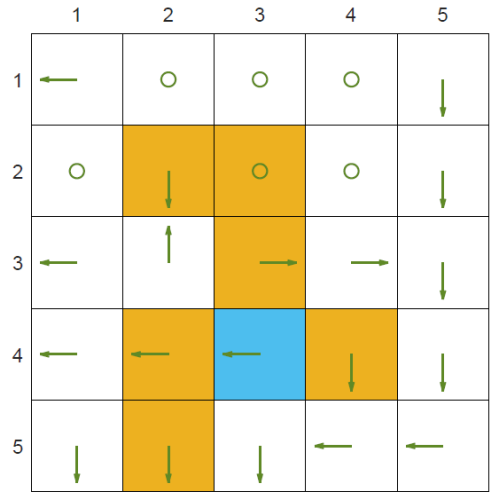
What if we only use a single episode of 100 steps? Insufficient data



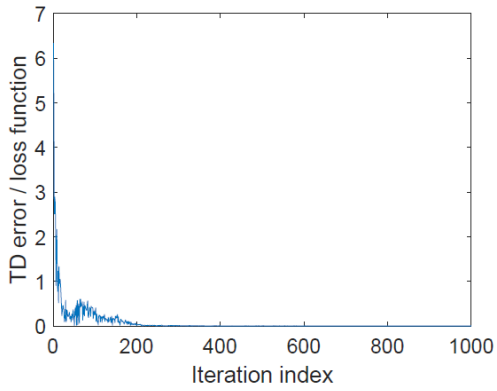
The behavior policy.



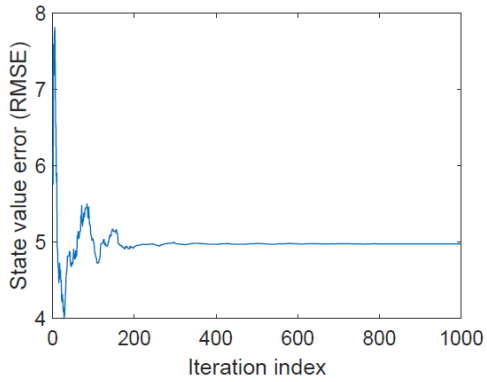
An episode of 100 steps.



The final policy.



The TD error converges to zero.



The state error does not converge to zero.

Summary

This lecture introduces the method of value function approximation.

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