Note for Deep Reinoforcement Learing

Wang Hui

Amateur Machine Learning Researcher in NUDT

wanghuichn1234@gmail.com

Contents

PART

I

Math Foundation

Section 1

Basic Concept

Subsection 1.1

Agent-Environment interaction cycle

The basic process in RL is called Agent-Environment interaction cycle. The interaction can go on for serveral cycles. Each cycle is called a time-step. Agent-Enviroment interaction cycle inludes steps which are listed below:

1. Enviroment recieves agent actions
2. Depending on current state and the agent chosen action, environment transitions to a new state
3. The new state and reward will pass to the agenet, at most time these signals are passed through a filter because agent is forbidden to perceive the true environment.
4. According to the new state and reward, agent do the next action. [1]
5. **Deep Reinforcement Learning**

[1] *Agent can do actions like learning or randomly choose next action*

Subsection 1.2

Episodic task, Continuing task and Return

Tasks that agent is trying to solve have a natural ending, are *episodic tasks*. Tasks that don't, are *continuing tasks*. A *time-step* limit is often added to *continuing tasks*.

An **episode** is a sequence of time-steps from the beginnig to the end of an **episodic task**.[2]. Agent may take serveral episodes to solve the *episodic task*. [3]

Return is the sum of all rewards that agent gets in an episode.

[2] *like playing a game and in the end you can win or lose*

[3] *you should play again and again to win the game.*

Subsection 1.3

States

A **state** is a specific configuration of the problem (or environment). The set of all possible states is defined as the **state space**.

State space is a set of sets. We can treated it as two parts: the inner set and the outer set. The inner set is must be finite, the elements of inner set are variables compose the states (variables do not have to be all inclusive). Elements in outer set represent possible states. Since the task can be a *continuing tasks*, the outer set may be infinite.

Independence

States must satisfy the **independence**. It means that the representation of states does not have to include all variables which compose the states, however, it must **contain all the variables necessary to make it independent of all other states**.

MDPs vs POMDPs

MDPs is Markov Decision Process and **POMDPs** is Partially observable Markov Decision Process.

1. MDPs: the observation is the same as the states.
2. POMDPs (*more general*): the observation is part of the states

Markov Property

Current RL or DRL systems are designed to take advantage of the Markov property. Variables (or the inner sets) you feed to the agent must **hold Markov assumption as tightly as possible**.

Definition 1 (Markov assumpiton) The probability of the next state, given the current state and action, is dependent of the history of interactions.

$$p(S_{t+0}|S_t, A_t) = p(S_{t+1}|S_t, A_t, S_{t-1}, A_{t-1}, \cdots) \tag{1.1}$$

States in MDPs

The set of all states in the MDP is denoted as S^+. Two unique states in MDPs are starting state and **Terminal State**.[4]

A RL system can have only one terminal state, and also can have multiple terminal states.

[4] *Terminal state is one unique state in the set of states in a MDP. It must have all available actions transitioning, with probability 0, to itself, and these transitions must not provide any reward (reward equals 0).*

Subsection 1.4

Actions

Actions are mechanisms to influence states. MDPs make available a set of actions $A(s)$ that depends on states. $A(s)$ is a function taking a state as its argument, and return some available actions base on the state.

The Action space with States space can be finite or infinite, but one action in action space must have finite variables to describe itself.

Unlike the number of variables compose the state, number of variables compose actions may not be constant because actions are depend on the state.

The environment makes all available actions known by agents in advance. Agent can select actions deterministically (from a look-up table) or stochastically (from a per-state probability distribution).

Subsection 1.5

Transition Function

When agent takes one action, environment must respond to the action. The respond is transitioning current state to next state.

The way of changing is referred to as the **state-transition probabilities**, or for simplicity, **transition function**. According to its definition, transition function should have three parameters, current state s, action a taking at s and the next state s'. Transition function maps the tuple: (s, a, s') to a probability.

Transition function describes a probability distribution determining how RL system will evolve from current state to next state (with what probability change to a next specific state). As a probability distribution, if we sum or integrate over all next state

s', the result must be one.

$$P(s'|s,a) = P(S_t = s'|S_{t-2} = s, A_{t-1} = a)$$

$$\sum_{s \in S} P(s'|s,a) = 0, \forall s \in S, a \in A(s)$$

Subsection 1.6
Rewards

The **reward function** maps a transition tuple: s, s', a to a scalar.

While the reward function can be represented by $R(s, s', a)$, which is explicit, it also can be represented by $R(s, s', a)$ or even $R(s)$, depending on needs.

We can compute the marginalization over next state s', to obtain $R(s, a)$, and the marginalization over action a, to obtain $R(s)$. For a stochastic process, reward function is defined by expectation.

$$r(s,a) = E[R_t|S_{t-2} = s, A_{t-1} = a]$$

$$r(s,a,s') = E[R_t|S_{t-2} = s, A_{t-1} = a, S_t = s']$$

Subsection 1.7
Horizon

A time-step, also referred to as **epoch, cycle, iteration or even interaction,** is a global clock syncing all parties and discrete time. Having a clock gives rise to a couple of types of task, *episodic task* and *continuing task*.

Planning horizon is agent's perspective that defines the *episodic task* and *continuing task*.

Finite Horizon is a planning horizon in which agent knows task will terminate in a finite number of steps. If the finite number of steps equals one, we call this planning horizon *greedy horizon*.

Infinite Horizon is the other planning horizon in which agent does not have predetermined time step limit.

Although agent plans for infinite, interaction may be stopped by environment (common practice is adding an artificial terminal state). We refer the sequence of consecutive time steps from the beginning to the end of an episodic task as an **episode, trial, period, or stage**.[5]

In infinite horizon, episode also has its meaning that is a collection contains all interactions between an initial and a terminal state [6].

[5] *the same as the definition in section 1.2*

[6] *We can add an artificial terminal state*

Subsection 1.8
Discount Factor

The **discount factor** adjust the importance of rewards, it helps reduce the degree to which future rewards affect our value function estimates and stabilize learning. Because that the future is uncertain, and the further we look into the future, the more stochastic we accumulate and the more variance our value function estimates will have.

Discount factor is part of MDP, not the agent. It is also a hyper parameter.

The **return** G_t is sum of all rewards obtained during an episode with discount factor and can be defined as below:

$$G_t = R_{t+0} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \tag{1.2}$$

The **recursive definition of return** is:

$$G_t = R_{t+0} + \gamma G_{t+1} \tag{1.3}$$

PART

II

Balancing immediate and long-term goals

This is about the basic algorithm for solving MDPs. It is a technique called **dynamic programming**, includes **Value Iteration (VI)** and **Policy Iteration (PI)**.

Although VI and PI require full access to the MDP, like knowing the dynamics of the environment, which is in some cheating way, they are the foundations from which virtually every other RL algorithm originates.

Section 2

Policy

Solid plan can not account for stochastic environment. What agent needs to come up with is called **policy**. Comparing with a plan, **policy** has a action for all possible states (only states in the plan have a action).

Policies can be stochastic (return action-probability distribution) or deterministic (return single actions for a given state). A policy is a function that prescribes actions to take for a given non-terminal state. Now, one immediate question is:*how good is a policy?*

Section 3

State-value function: What to expect from here?

If we are given a policy and the MDP, we should be able to calculate the expected return staring from every single state. That is, a number should be put into every state in the MDP for a given policy. The number must tell agent whether a state is better than another and precisely how much better it is.

Remember that the value of states depends on the policy which the agent follows. We define the value of states as the expectation of returns that the agent would get if it follows the policy π.

Definition 2 (State-value function)

$$V_\pi(s) = E[G_t|S_t = s] \tag{3.1}$$

Considering the recursive definition of returns, the value of states can be rewrite as:

$$V_\pi(s) = E[R_{t+1} + \gamma G_{t+1}|S_t = s] \tag{3.2}$$

and this will derive famous Bellman Equation.

$$V_\pi(s) = \sum_a \pi(a|s)[\sum_r rP(r|s,a) + \gamma \sum_{s'} V_\pi(s')P(s'|s,a)] \quad \forall s, s' \in S \tag{3.3}$$

We can use margin probability to rewrite equation above, because

$$P(r|s,a) = \sum_{s'} P(r,s'|s,a) \quad P(s'|s,a) = \sum_r P(r,s'|s,a) \tag{3.4}$$

Then

$$V_\pi(s) = \sum_a \pi(a|s) \sum_r \sum_{s'} P(r, s'|s, a)[r + \gamma V_\pi(s')] \quad \forall s, s' \in S \tag{3.5}$$

Definition 3 (Bellman Equation) We start from here:

$$V_\pi(s) = E[R_{t+1} + \gamma G_{t+1}|S_t = s]$$

According to expectation definition and properties:

$$V_\pi(s) = E[R_{t+1}|S_t = s] + \gamma E[G_{t+1}|S_t = s]$$

The first item on equation right side can be expand as (according to Law of Total Probability[7]):

$$\begin{aligned} E[R_{t+1}|S_t = s] &= \sum_r rP(r|s) \\ &= \sum_r r(\sum_a \pi(a|s)P(r|s, a)) \end{aligned} \tag{3.6}$$

For simplicity, we write $S_t = s, A_t = a$ as s, a. The second item on equation right side can be expand as (according to Properties of Expectation[8])

$$\begin{aligned} E[G_{t+1}|S_t = s] &= \sum_{s'} E[G_{t+1}|S_t = s, S_{t+1} = s']P(S_{t+1} = s'|S_t = s) \\ &= \sum_{s'} E[G_{t+1}|S_{t+1} = s']P(s'|s) \ (Markov\ Property) \\ &= \sum_{s'} V_\pi(s') \sum_a \pi(a|s)P(s'|s, a) \ (Law\ of\ Total\ Probability) \end{aligned} \tag{3.7}$$

We combine equation (3.6) and (3.7), then get Bellman Equation.

$$V_\pi(s) = \sum_a \pi(a|s) \left[\sum_r rP(r|s, a) + \gamma \sum_{s'} V_\pi(s')P(s'|s, a) \right] \quad \forall s, s' \in S$$

[7] *If $B_1, B_2, \cdots$ form a partition of the sample space S, then we can calculate the probability of event A as:*

$$P(A) = \sum_{B_i} P(A|B_i)P(B_i)$$

[8]

$$E[X] = \sum_y E[X|Y = y]P(y)$$

$$E[X] = E[E[X|Y]]$$

Proof (Properties of Expectation)

$$\begin{aligned} E[X] &= \sum_x xP(x) \\ &= \sum_x \sum_y xP(x|Y = y)P(y) \\ &= \sum_y P(y) \sum_x xP(x|Y = y) \\ &= \sum_y E[X|Y = y]P(y) \end{aligned}$$

For $E[X] = E[E[X|Y]]$, we can use above euqation:

$$\begin{aligned} E[X] &= \sum_y E[X|Y = y]P(y) \\ &= E[E[X|Y]] \quad \text{(Definition of Expectation)} \end{aligned}$$

□

Section 4

Action-value function: What should I expect from here if I do this action?

Another important question is the value of taking action a in a state s. By comparing **two actions under the same policy**, we can use **action-value function** to select a better one and then improve our policy.
Action-value function can be defined as:

Definition 4 (Action-value function)

$$q_\pi(s,a) = E[G_t|S_t = s, A_t = a] \tag{4.1}$$

We expand right side (use recursive definition of Retrun and Properties of Expectation):

$$\begin{aligned} E[G_t|S_t = s, A_t = a] &= \sum_r rP(r|s,a) + \gamma \sum_{s'} E[G_{t+1}|s,a,s']P(s'|s,a) \\ &= \sum_r rP(r|s,a) + \gamma \sum_{s'} E[G_{t+1}|s']P(s'|s,a) \\ &= \sum_r rP(r|s,a) + \gamma \sum_{s'} V_\pi(s')P(s'|s,a) \end{aligned}$$

By comparing with eq(3.5), we have

$$q_\pi(s,a) = \sum_r rP(r|s,a) + \gamma \sum_{s'} V_\pi(s')P(s'|s,a) \tag{4.2}$$

and

$$V_\pi(s) = \sum_a \pi(a|s) q_\pi(s,a) \tag{4.3}$$

Section 5

Optimal Policy

An optimal policy is an policy that for every state s can obtain expectation returns **greater than or equal to any other policy.** Now, an **Optimal State-value function** of state s and **Optimal action-value function** in state s of action a can be defined as:

$$\begin{aligned} V_*(s) &= \max_\pi V_\pi(s) \\ q_*(s,a) &= \max_\pi q_\pi(s,a) \end{aligned}$$

According to eq(3.5), the Optimal State-value funciton can be defined as:

Definition 5 (Optimal State-value function)

$$\begin{aligned} V_*(s) &= \max_\pi V_\pi(s) \\ &= \max_\pi \sum_a \pi(a|s) \sum_{s'} \sum_r P(s', r|s, a)[r + \gamma V_\pi(s')] \end{aligned} \tag{5.1}$$

According to equation above, Optimal State-value function of state is determined by selecting an optimal policy that makes $\sum_a \pi(a|s) \sum_{s'} \sum_r P(s', r|s, a)[r + \gamma V_\pi(s')]$ maximum (this is the meaning of $\max_\pi$). Because π is an optimal policy, $V_\pi(s') = V_*(s')$. Then we should **clarify what action should be in an optimal policy**. In any states s, there must exist an action that maximizes $\sum_{s'} \sum_r P(s', r|s, a)[r + \gamma V_*(s')]$. Since $\sum_a \pi(a|s) = 1$, we have:

$$\begin{aligned} &\sum_a \pi(a|s) \sum_{s'} \sum_r P(s', r|s, a)[r + \gamma V_*(s')] \\ \leq &\sum_a \pi(a|s) \max_a \sum_{s'} \sum_r P(s', r|s, a)[r + \gamma V_*(s')] \\ = &\max_a \sum_{s'} \sum_r P(s', r|s, a)[r + \gamma V_*(s')] \end{aligned}$$

Now, we have the optimal policy $\pi^*(a|s)$,

$$\pi^*(a|s) = \begin{cases} 1 \text{ if } a = a^*, \\ 0 \text{ if } a \neq a^*. \end{cases} \tag{5.2}$$

here $a^* = \arg\max_a \sum_{s'} \sum_r P(s', r|s, a)[r + \gamma V_*(s')]$. So we rewrite eq (5.1)as below:

$$V_*(s) = \max_a \sum_{s'} \sum_r P(s', r|s, a)[r + \gamma V_*(s')] \tag{5.3}$$

And according to eq.(4.2)and (4.3)

$$\begin{aligned} q_\pi(s, a) &= \sum_r rP(r|s, a) + \gamma \sum_{s'} V_\pi(s')P(s'|s, a) \\ &= \sum_r rP(r|s, a) + \gamma \sum_{s'} \sum_{a'} \pi(a'|s')q_\pi(s', a')P(s'|s, a) \\ &= \sum_r \sum_{s'} rP(r, s'|s, a) + \gamma \sum_r \sum_{s'} \left[\sum_{a'} \pi(a'|s')q_\pi(s', a')P(r, s'|s, a) \right] \\ &= \sum_r \sum_{s'} P(r, s'|s, a) \left[r + \gamma \sum_{a'} \pi(a'|s')q_\pi(s', a') \right] \end{aligned} \tag{5.4}$$

So Optimal Action-value function can be defined as:

Definition 6 (Optimal Action-value function)

$$\begin{aligned} q_*(s,a) &= \max_\pi q_\pi(s,a) \\ &= \max_\pi \sum_r \sum_{s'} P(r,s'|s,a) \left[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s',a') \right] \\ &= \sum_r \sum_{s'} P(r,s'|s,a) \left[r + \gamma \max_{a'} q_*(s',a') \right] \end{aligned} \tag{5.5}$$

So far, we have defined the equation for Optimal State-value function and Optimal Action-value function. They are also the objectives we are after. We can start exploring the methods for finding these objectives.

PART

III

Two algorithm to find objects

The key point of both algorithms is the iterative method for solving Bellman optimality equation. In particular, the algorithm is:

$$\overleftarrow{\texttt{iteratively compute}}$$

$$\begin{aligned} V_{k+1}(s) &= \max_{\pi} \sum_{a} \pi(a|s) \sum_{s'} \sum_{r} P(s', r|s, a)[r + \gamma V_k(s')] \\ &= \max_{\pi} \sum_{a} \pi(a|s) q_k(s, a) \end{aligned} \tag{5.6}$$

In the end of this part, we will demonstrate why the iterative method can solve optimal Bellman Equation and find the object.

Section 6

Value Iteration Algorithm

Accroding to the eq.(5.6), we immeidately get the Value Iteration Algorithm which is listed below:

Algorithm 1: Value Iteration Algorithm

Input: The probability models $P(r|s,a)$ and $P(s'|s,a)$ for all (s,a). Initial guess v_0.

Output: The optimal state value and optimal policy.

while V_k **has not converged in the sense that** $\|V_k - V_{k-1}\|$ **is greater than a predefined threshold do**
  foreach state $s \in S$ **do**
    foreach $a \in A(s)$ **do**
      $q_k(s,a) \leftarrow \sum_r P(r|s,a)r + \gamma \sum_{s'} P(s'|s,a)V_k(s')$
    end
    $a_k^*(s) = \arg\max_a q_k(s,a)$ `// Maximum action value`
    $\pi_{k+1}(a|s) = 1$ if $a = a_k^*(s)$, else $\pi_{k+1}(a|s) = 0$ `// Policy updates`
    $V_{k+1}(s) \leftarrow \max_a q_k(s,a)$ `// Value updates`
  end
  $k \leftarrow k + 1$
end

In summary, the above steps can be illustrated as :

$V_k(s) \rightarrow q_k(s,a) \rightarrow$ greedly update policy $\pi_{k+1}(a|s) \rightarrow$ new value $V_{k+1}(s) = \max_a q_k(s,a)$

What should be noticed is that in Value Iteration algorithm $V_{k+1}(s)$ is not the value function because it can not guarantee[9]

$$V_{k+1}(s) = \max_{\pi} \sum_{a} \pi(a|s) q_{k+1}(s,a)$$

[9] *In this equation, the index of* $q(s,a)$ *is* k *not* $k+1$.

If we want $V_{k+1}(s)$ to be a value function, in the value updates step of Value Iteration algorithm, we should use an iterative procedure in place of the one step computation to get the value function under the updated policy π_{k+1}.

Why? Although $V_{k+1}(s)$ is merely an intermediate value generated by the algorithm, the iterative algorithm can guarantee that V_{k+1} converge to the optimal state value function.

Section 7

Policy Iteration Algorithm

Not like Value Iteration algorithm that directly solve the Bellman optimality equation, **P**olicy Iteration algorithm has an intimate relationship with **Value Iteration algorithm**.

Algorithm 2: Policy Iteration Algorithm

Input: The system model, $P(r|s,a)$ and $P(s'|s,a)$ for all (s,a), is known.
Initial guess π_0.

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

while V_{π_k} **has not converged, for the** kth **iteration do**
  Initialization: randomly guess $V^0_{\pi_k}$
  `/* Iteratively find the approximation of the true state value function */`
  while $V^j_{\pi_k}$ **has not converged, for the jth iteration do**
    foreach state $s \in S$ **do**
      $V^{j+1}_{\pi_k} \leftarrow \sum_a \pi(a|s) \sum_r \sum_{s'} \left[P(r|s,a) + \gamma P(s'|s,a) V^j_{\pi_k}\right]$
    end
  end
  `/* Policy improvement */`
  foreach state $s \in S$ **do**
    foreach action $a \in A(s)$ **do**
      $q_{\pi_k}(s,a) \leftarrow \sum_r P(r|s,a)r + \gamma \sum_{s'} P(s'|s,a) V_{\pi_k}(s')$
    end
    $a^*_k(s) = \arg\max_a q_{\pi_k}(s,a)$
    $\pi_{k+1}(a|s) = 1$ if $a = a^*_k(s)$, else $\pi_{k+1}(a|s) = 0$ `// Policy updates`
  end
  $k \leftarrow k+1$
end

The embedded iterative algorithm of policy evaluation requires an infinite steps that is $j \to \infty$ to converge to the true state value function. But it is impossible. We actually get an approximation of the true state value function but this does not cause any problems.[10]

[10] *$V_{k+1}(s)$ in Value Iteration algorithm is more far from the true state value than that in policy evaluation, but it also can find the solution for optimal Bellman Equation.*

Section 8

Comparison of two algorithms

Table below clearly illustrates the steps of two algorithms. We can see they are similar (if the initial guess of V_0 in Value Iteration algorithm is replaced with V_{π_0} which is iteratively computed in Policy Iteration algorithm). What different between Value

Table 1. Comparison of two algorithm

	Policy iteration algorithm	Value iteration algorithm	Comments
1) Policy	π_0	N/A	
2) Value	V_{π_0} **(iteratively compute under π_0)**	$V_0 = V_{\pi_0}$	replace random initial guess V_0 in Value iteration with V_{π_0}
3) Policy	$\pi_1 = \arg\max_\pi (r_\pi + \gamma P_\pi V_{\pi_0})$	$\pi_1 = \arg\max_\pi (r_\pi + \gamma P_\pi V_{\pi_0})$	two operations are the same
4) Value	$V_{\pi_1} = r_{\pi_1} + \gamma P_{\pi_1} V_{\pi_1}$	$V_1 = r_{\pi_1} + \gamma P_{\pi_1} V_{\pi_0}$	V_{π_1} is iteratively computed whereas V_1 is just an one-step assignment
⋮	⋮	⋮	

Iteration and Policy Iteration is in value compute, the V_{π_j} in Policy iteration is an approximation of Value State function but the V_j is just an intermediate result.

If we explicitly write the whole iterative procedure, we have

$$V_{\pi_1}^{(0)} = V_0$$

$$\text{Value iteration} \leftarrow V_1 \leftarrow V_{\pi_1}^{(1)} = r_{\pi_1} + \gamma P_{\pi_1} V_{\pi_1}^{(0)}$$

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

$$\vdots$$

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

$$\vdots$$

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

and can have an observation from above process: if we truncated policy iteration, for example, limit the maximum number of iteration, we can get a new algorithm which is a middle ground between two extremes. It is called **Truncated Policy Iteration Algorithm**.

Algorithm 3: Truncated Policy Iteration algorithm

Input: T

he probability models $P(r|s,a)$ and $P(s'|s,a)$ for all (s,a) are known. Initial guess π_0. **Output:** T

he optimal State Value funtion and Policy.

while V_k **(not** V_{π_k}**) has not converged, for the kth iteration do**

  Select the initial value as $V_k^{(0)} = V_{k-1}$

  `/* truncate infinite j to a fixed number` $j_{truncate}$ `*/`

  while $j < j_{truncate}$ **do**

    $V_k^{(j)} = \sum_a \pi(a|s) \sum_r \sum_{s'} \left[P(r|s,a)r + \gamma P(s'|s,a) V_k^{(j-1)}(s') \right]$

  end

  Set $V_k = V_k^{j_{truncate}}$

  `/* Policy improvement */`

  foreach state $s \in S$ **do**

    foreach action $\in A(s)$ **do**

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

    end

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

    $\pi_{k+1}(a|s) = 1$ if $a = a_k^*$, else $\pi_{k+1}(a|s) = 0$ `// Policy updates`

  end

end

Section 9

Correctness of Algorithm

Nomatter Value Iteration or Policy Iteration, iterative method(5.6) for solving optimal Bellman Equation is the key point.The correctness of it is assured by **Contract Mapping Theorem**.

Definition 7 (Contract Mapping) The function $f(x)$ is a **contract mapping** if there exists $\gamma \in (0,1)$ such that

$$\|f(x_1) - f(x_2)\| \leq \gamma \|x_1 - x_2\| \tag{9.1}$$

for any $x_1, x_2 \in R^d$.

Theorem 1 (Contract mapping theorem) For any equation that has the form $x = f(x)$, where x and $f(x)$ are real vectors, if$f$is a contract mapping, then the following properties hold:

1. Existence: there exists a fixed point x^* satisfying $f(x^*) = x^*$.
2. Uniqueness: the fixed point is unique.
3. Algorithm: the following iterative algorithm

$$x_{k+1} = f(x_k) \tag{9.2}$$

where $(k = 0, 1, \cdots)$ will converge to (x^*) when $(k \to \infty)$

Before proving this theorem, we give the definition of **fixed point** and **Cauchy sequence**.

Definition 8 (Fixed point) Consider a function $f(x)$, where $x \in R^d$ and $f : R^d \to R^d$. A point is x^* is a fixed point if $f(x^*) = x^*$

Definition 9 (Cauchy sequence) A sequence $x_1, x_2, \cdots \in R$ is called **Cauchy Sequence** if for $\forall \epsilon > 0$, there exists an integer N such that $\|x_m - x_n\| < \epsilon$, for all $m, n > N$.
Cauchy Sequence is guaranteed to converge to a limit.

PROOF

1. Convergent
For any $m > n$,

$$\begin{aligned}\|x_m - x_n\| &= \|x_m - x_{m-1} + x_{m-1} - x_{m-2} + x_{m-2} - \cdots + x_{n+1} - x_n\| \\ &\leq \|x_m - x_{m-1}\| + \|x_{m-1} - x_{m-2}\| + \cdots + \|x_{n+1} - x_n\|\end{aligned}$$

and f is a contract mapping, we have

$$\begin{aligned}\|x_{k+1} - x_k\| &= \|f(x_k) - f(x_{k-1})\| \\ &\leq \gamma\|x_k - x_{k-1}\| \\ &= \gamma\|f(x_{k-1}) - f(x_{k-2})\| \\ &\leq \gamma^2\|x_{k-1} - x_{k-2}\| \\ &\cdots \\ &\leq \gamma^k\|x_1 - x_0\|\end{aligned}$$

So

$$\begin{aligned}\|x_m - x_n\| &\leq \gamma^{m-1}\|x_1 - x_0\| + \gamma^{m-2}\|x_1 - x_0\| + \cdots + \gamma^n\|x_1 - x_0\| \\ &= \frac{\gamma^n(1 - \gamma^{m-n})}{1 - \gamma} \\ &\leq \frac{\gamma^n}{1 - \gamma}\|x_1 - x_0\| \qquad (9.3)\end{aligned}$$

Now, for any $\epsilon > 0$, we can always find an integer N such that $\|x_m - x_n\| < \epsilon$ for all $m, n > N$. So $\{x_{k+1} = f(x_k)\}_{k=0}^{\infty}$ is a **Cauchy sequence** and hence converge to a limit point which is denoted as $x^* = \lim_{k\to\infty} x_k$

2. The limit point is a fixed point
Because

$$\|f(x_k) - x_k\| = \|x_{k+1} - x_k\| \leq \gamma^k\|x_1 - x_0\|$$

$\|f(x_k) - x_k\|$ converges to zero exponentially. We get $f(x^*) = x^*$

3. Uniqueness
Suppose there is another fixed point x' which satisfies $f(x') = x'$, then

$$\|x' - x^*\| = \|f(x') - f(x^*)\| \leq \gamma\|x' - x^*\| \quad \text{(definition of contract mapping)} \qquad (9.4)$$

Since $0 < \gamma < 1$, this inequality holds if and only if $\|x' - x^*\| = 0$ □

Contract mapping theorem tells us, if a mapping is a contract mapping, a Cauchy

sequence $x_k{}_{k=0}^{\infty}$ can be generated by an iterative method($x_{k+1} = f(x_k)$) and its limit is the fixed point of that mapping.

Here we demonstrate the right-hand side of equation $V_*(s)$ is a contract mapping. Firstly, we define $r_\pi(s)$ and $p_\pi(s'|s)$ based on eq.(3.5).

$$r_\pi(s) = \sum_a \pi(a|s) \sum_r rP(r|s,a)$$
$$P_\pi(s'|s) = \sum_a \pi(a|s) P(s'|s,a)$$

Suppose that the states are indexed as s_i with $i = 1, 2, \cdots, n$, for state s_i in eq.(3.5)

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

Let $V_\pi = [V_\pi(s_1), V_\pi(s_1), \cdots, V_\pi(s_n)]^T \in \mathbb{R}^n$, $r_\pi = [r_\pi(s_1), r_\pi(s_1), \cdots, r_\pi(s_n)]^T \in \mathbb{R}^n$, and $P_\pi \in \mathbb{R}^{n\times n}$ with $[P_\pi]_{ij} = P_\pi(s_i|s_j)$ Then we get the matrix form of eq.(3.5)

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

Here is an example with four states:

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

Since under different state we should have different optimal policy, the value of policy π is determined by the states. The matrix form of Bellman Optimality Equation is:

$$V^* = \max_\pi (r_\pi + \gamma P_\pi V^*) \tag{9.5}$$

Right now, we can see if taking right-hand side of eq.(9.5) as a funtion f, optimal value funciton (V^*) will be the fixed point of $f(V)$. That is why we will demostrate f is a contract mapping.

Considering given any two vectors $V_1, V_2 \in \mathbb{R}^n$, and $\pi_1 = \arg\max_\pi(r_\pi + \gamma P_\pi V_1)$ and $\pi_2 = \arg\max_\pi(r_\pi + \gamma P_\pi V_2)$ (notice that police π is determined by states and that is why the mapping taking only V as its variable rather than V and π). Then

$$f(V_1) = r_{\pi_1} + \gamma P_{\pi_1} V_1 \geq r_{\pi_2} + \gamma P_{\pi_2} V_1$$
$$f(V_2) = r_{\pi_2} + \gamma P_{\pi_2} V_2 \geq r_{\pi_1} + \gamma P_{\pi_1} V_2$$

where $\geq$ is an elementwise comparison.

As a result,

$$\begin{aligned} f(V_1) - f(V_2) &\leq r_{\pi_1} + \gamma P_{\pi_1} V_1 - r_{\pi_1} + \gamma P_{\pi_1} V_2 \\ &= \gamma P_{\pi_1}(V_1 - V_2) \end{aligned}$$

$$\begin{aligned} f(V_2) - f(V_1) &\leq r_{\pi_2} + \gamma P_{\pi_2} V_2 - r_{\pi_2} + \gamma P_{\pi_2} V_1 \\ &= \gamma P_{\pi_2}(V_2 - V_1) \end{aligned}$$

Therefore,

$$\gamma P_{\pi_2}(V_1 - V_2) \leq f(V_1) - f(V_2) \leq \gamma P_{\pi_1}(V_1 - V_2)$$

Define $z \doteq \max(|\gamma P_{\pi_1}(V_1 - V_2)|, |\gamma P_{\pi_2}(V_1 - V_2)|)$ and use infinite norm ($\|\cdot\|_\infty \doteq \max_i |x_i|$), we get

$$\|f(V_1) - f(V_2)\|_\infty \leq \|z\|_\infty$$

and

$$z_i = \max(\gamma |P_{\pi_1}[i](V_1 - V_2)|, \gamma |P_{\pi_2}[i](V_1 - V_2)|)$$

where z_i is the ith element in z and $P_{\pi_{1,2}}[i]$ is the ith row in matrix $P_{\pi_{1,2}}$. Because $P_{\pi_{1,2}}[i]$ is a vector with all non-negative elements and the sum of the elements are equal to 1, it follows that

$$|P_{\pi_1}[i](V_1 - V_2)| = P_{\pi_1}[i]|(V_1 - V_2)| \leq \|V_1 - V_2\|_\infty,$$

Similarly,

$$|P_{\pi_2}[i](V_1 - V_2)| = P_{\pi_2}[i]|(V_1 - V_2)| \leq \|V_1 - V_2\|_\infty.$$

So

$$\|z\|_\infty = \max_i |z_i| \leq \gamma \|V_1 - V_2\|_\infty$$

Finally we conclude that $f(V)$ is a contract mapping because

$$\|f(V_1) - f(V_2)\|_\infty \leq \gamma \|V_1 - V_2\|_\infty \leq \|V_1 - V_2\|_\infty$$

According to Contract mapping theorem, we can easily get the theorem:

Theorem 2 For the Bellman Optimality Equation $V = f(V) = \max_\pi(r_\pi + \gamma P_\pi V)$, there always exists a unique solution V^*, which can be iteratively solved by

$$V_{k+1} = \max_\pi(r_\pi + \gamma P_\pi V_k) \tag{9.6}$$

The value of V_k is converged exponentially to V^* as $k \to \infty$ given any initial state V_0

For the two algorithm Value Iteration and Policy Iteration, Value Iteration directly uses iterative method to solve BOE (Bellman Optimality Equation) whereas Policy Iteration just improve policy. So we can confirm that Value Iteration can get the optimal state value function and optimal policy[11], but for Policy Iteration, we need to prove that it eventually find an optimal policy and optimal state value funciton.

[11] *Solving π^* can be easily obtained by $\pi^* = argmax_\pi(r_\pi + \gamma P_\pi V^*)$ once optimal State value function is obtained.*

Lemma 1 (Policy improvement) In the policy improvement step, π_{k+1} is better than π_k. That is, if $\pi_{k+1} = \arg\max_\pi(r_\pi + \gamma P_\pi V_{\pi_k})$, then $V_{\pi_{k+1}} \geq V_{\pi_k}$.

Proof Since $V_{\pi_{k+1}}$ and V_{π_k} are state value function, they satisfy the Bellman Equation:

$$\begin{aligned} V_{\pi_{k+1}} &= r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} V_{\pi_{k+1}} \\ V_{\pi_k} &= r_{\pi_k} + \gamma P_{\pi_k} V_{\pi_k} \end{aligned}$$

Because $\pi_{k+1} = \arg\max_\pi (r_\pi + \gamma P_\pi V_{\pi_k})$, we know that,

$$r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} V_{\pi_k} \geq r_{\pi_k} + \gamma P_{\pi_k} V_{\pi_k}$$

It then follows that,

$$\begin{aligned} V_{\pi_k} - V_{\pi_{k+1}} &= (r_{\pi_k} + \gamma P_{\pi_k} V_{\pi_k}) - \left(r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} V_{\pi_{k+1}}\right) \\ &\leq \left(r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} V_{\pi_{k+1}}\right) - \left(r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} V_{\pi_k}\right) \\ &= \gamma P_{\pi_{k+1}} (V_{\pi_k} - V_{\pi_{k+1}}) \end{aligned}$$

Therefore,

$$\begin{aligned} V_{\pi_k} - V_{\pi_{k+1}} &\leq \gamma^2 P_{\pi_{k+1}}^2 (V_{\pi_k} - V_{\pi_{k+1}}) \leq \cdots \\ &\leq \gamma^n P_{\pi_{k+1}}^n (V_{\pi_k} - V_{\pi_{k+1}}) \\ &\leq \lim_{n\to\infty} \gamma^n P_{\pi_{k+1}}^n (V_{\pi_k} - V_{\pi_{k+1}}) = 0 \end{aligned}$$

The limit is due to $\gamma < 1$ and $P_{\pi_{k+1}}^n$ is a non-negative stochastic matrix for any n □

Before demonstrating that policy iteration alogorithm can get optimal value function, we recall converge of sequence. For monotonic sequence, its convergence can be guaranteed by theorem below:

Theorem 3 (Convergence of monotonic sequences) If the sequence $\{x_k\}$ is non-increasing and bounded from below:

1. Non-increasing: $x_{k+1} \leq x_k$
2. Similarly, if $\{x_k\}$ is non-decreasing and have a upper bound, then the sequence is convergent

then x_k converges to a value, which is the limit of $\{x_k\}$, as $k \to \infty$.

Similarly, if $\{x_k\}$ is non-decreasing and have a upper bound, then the sequence is convergent.

Considering the non-monotonic sequence, we also have a similiar converegence.

Theorem 4 (Convergence of non-monotonic sequence) if the sequences is non-negative $x_k \geq 0$ and satisfies:

$$\sum_{k=1}^{\infty} (x_{k+1} - x_k)^+ < \infty \tag{9.7}$$

then $\{x_k\}$ converges as $k \to \infty$.
The definition of symbol $(\cdot)^+$ is: For any $z \in \mathbb{R}$, define

$$z^+ \doteq \begin{cases} z & \text{if } z \geq 0, \\ 0 & \text{if } z < 0. \end{cases}$$

Similarly, the definition of symbol $(\cdot)^-$ is:

$$z^- \doteq \begin{cases} z & \text{if } z \leq 0, \\ 0 & \text{if } z > 0. \end{cases}$$

Proof To analyze the convergence of non-monotonic sequences $\{x_k\}$, we can use symbols above to rewrite it as:

$$\begin{aligned} x_k &= x_k - x_{k-1} + x_{k-1} - x_{k-2} + \cdots + x_2 - x_1 + x_1 \\ &= \sum_{i=1}^{k-1}(x_{i+1} - x_i) + x_1 \\ &\doteq S_k + x_1 \end{aligned}$$

Note $S_k = \sum_{i=1}^{k-1}(x_{i+1} - x_i)$ can be decomposed as:

$$S_k = S_k^+ + S_k^-$$

where

$$\begin{aligned} S_k^+ &= \sum_{i=1}^{k-1}(x_{i+1} - x_i), \text{ for all } i \text{ which } x_{i+1} - x_i \geq 0 \\ S_k^- &= \sum_{i=1}^{k-1}(x_{i+1} - x_i), \text{ for all } i \text{ which } x_{i+1} - x_i < 0 \end{aligned}$$

Obviously, sequence $\{S_k^+\}$ is non-decreasing whereas $\{S_k^-\}$ is non-increasing.

Because $x_k \geq 0$, then $S_k^- \geq -S_k^+ - x_1$. So if $\{S_k^+\}$ has a upper bound, $\{S_k^-\}$ also has a lower bound, and vice versa.

Now the convergence of non-negative and non-monotonic sequences which satisfy the condition 9.7 can be easily proved. First, the condition 9.7 indicates that $\{S_k^+\}$ has a upper bound. Since $\{S_k^+\}$ is non-decreasing, the convergence of $\{S_k^+\}$ immediately follows from theorem 1. Suppose that $\{S_k^+\}$ converges to S_*^+.

Second, because $\{S_k^+\}$ has a upper bound, the non-increasing sequence $\{S_k^-\}$ also has a lower bound. We also immediately get the convergence of $\{S_k^-\}$ based on theorem 1. Suppose that $\{S_k^-\}$ converges to S_*^-.

Finally, because $x_k = S_k^+ + S_k^- + x_1$, the convergence of $\{S_k^+\}$ and $\{S_k^-\}$ implies that $\{x_k\}$ converges to $x_1 + S_*^+ + S_*^-$. □

In Policy iteration, we obtain two sequence, one is the sequence of policies:

$$\{\pi_1, \pi_2, \cdots, \pi_k, \cdots\}$$

and the other is the sequence of state value function:

$$\{V_{\pi_1}, V_{\pi_2}, \cdots, V_{\pi_k}, \cdots\}.$$

Based on the Policy Improvement Lemma, we know that the sequence of state value function is a non-decreasing sequence and bounded by V_* and it follows from Convergence of monotonic sequences that V_{π_k} converges to a constant value V_∞, when $k \to \infty$. The final step is to demonstrate $V_\infty = V_*$[12].

[12] *Convergent value may not equal to bound!*

Remark The idea of the proof is to show that the policy algorithm has a faster convergent speed than the value iteration. What the idea means is that as value iteration can converges to the optimal value sate V_*, with **squeeze theorem**(V_* is the lower bound of the sequence of state value function which is generated by policy iteration algorithm since policy iteration has a faster convergent speed than value iteration and according to the definition, V_* is the upper bound of the sequence), policy iteration is able to converge to V_*.

Here is the proof,

PROOF In particular, to prove the convergence of $\{V_{\pi_k}\}_{k=0}^{\infty}$, we introduce another sequence $\{V_k\}_{k=0}^{\infty}$ generated by

$$V_{k+1} = \max_{\pi_k}(r_{\pi_k} + \gamma P_{\pi_k} V_k) \tag{9.8}$$

In fact, $\{V_k\}_{k=0}^{\infty}$ is exactly generated by value iteration. We use mathematical induction to demonstrate that the policy iteration can faster converge.

1) For $k = 0$, we can always find a value V_0 that $V_{\pi_0} \geq V_0$ under any initial guess π_0
2) Suppose that for $k \geq 0$, we have $V_{\pi_k} \geq V_k$
3) For $k + 1$,

$$\begin{aligned} V_{\pi_{k+1}} - V_{k+1} &= (r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} V_{\pi_{k+1}}) - \max_{\pi_k}(r_{\pi_k} + \gamma P_{\pi_k} V_k) \\ &\geq (r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} V_{\pi_k}) - \max_{\pi_k}(r_{\pi_k} + \gamma P_{\pi_k} V_k) \quad (\textit{Lemma 1 } V_{\pi_{k+1}} \geq V_{\pi_k}) \end{aligned}$$

Let $\pi'_k = \max_{\pi_k}(r_{\pi_k} + \gamma P_{\pi_k} V_k)$. We should notice that in policy improvement step of both value and policy iteration, they travel throughout every state and every action. That means the candidate policy set for the optimal policy in the *kth* iteration of policy iteration algorithm is as same as that in value iteration.

So if π_k^* which is equal to π_{k+1} (policy update step in Policy Iteration algorithm) is the optimal policy generated by the *kth* iteration of policy iteration algorithm and $\pi'_k = \max_{\pi_k}(r_{\pi_k} + \gamma P_{\pi_k} V_k)$, we have $(r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} V_{\pi_k}) \geq (r_{\pi'_k} + \gamma P_{\pi'_k} V_{\pi_k})$ and,

$$\begin{aligned} V_{\pi_{k+1}} - V_{k+1} &\geq (r_{\pi'_k} + \gamma P_{\pi'_k} V_{\pi_k}) - (r_{\pi'_k} + \gamma P_{\pi'_k} V_k) \\ &= \gamma P_{\pi'_k}(V_{\pi_k} - V_k) \end{aligned}$$

Because $V_{\pi_k} \geq V_k$ and $P_{\pi' k}$ is non-negative, we have $V_{\pi_{k+1}} \geq V_{k+1}$. Therefore we can show by induction that $V_k \leq V_{\pi_k} \leq V^*$. Since $\{V_k\}_{k=0}^{\infty}$ converges to V^*, V_{π_k} also converges to V^*. □

Now we have

Theorem 5 (Convergence of Policy Iteration) The state value sequence $\{V_{\pi_k}\}_{k=0}^{\infty}$generated by Policy Iteration converges to the optimal state value V_*. As a result, the policy sequence $\{\pi_k\}_{k=0}^{\infty}$ converges to the optimal policy π^*

PART

IV

Important formulas in Probability Theory

Section 10

Probability

1. Joint probability

$$\sum_y p(x,y) = p(x)$$

2. Law of total probability

$$p(x) = \sum_y p(x,y) = \sum_y p(x|y)p(y)$$
$$p(x|a) = \sum yp(x,y|a)$$

3. Chain rule of conditional probability

$$\begin{aligned} p(a,b,c) &= p(a|b,c)p(b,c) \\ &= p(a|b,c)p(b|c)p(c) \end{aligned}$$

And because $p(a,b,c)/p(c) = p(a,b|c)$, we obtain

$$p(a|b,c)p(b|c) = p(a,b|c) \tag{10.1}$$

Equation 10.1 is very important in DRL because it tells us how to put a random variable to condition. Now according to **jointprobability** and **law of total probability**, we have

$$\begin{aligned} p(x|a) &= \sum_b p(x,b|a) \\ &= \sum_b p(x|b,a)p(b|a) \end{aligned} \tag{10.2}$$

Section 11

Expectation

1. Law of total expectation

$$\begin{aligned} E[X] &= \sum_x xp(x) \\ &= \sum_x x \sum_a p(x|a)p(a) \\ &= \sum_a \left(\sum_x xp(x|a)p(a) \right) \\ &= \sum_a E[X|A=a]p(a) \end{aligned} \tag{11.1}$$

2. Lemmas of Expectation
   Before introducing lemmas, consider three cases:
   (a) $E[X|Y=a]$(a is a given number): this conditional expectation is a specific *number*.
   (b) $E[X|Y=y]$(y is a normal variable): this conditional expectation is a *funtion of y ie. f(y)*.
   (c) $E[X|Y]$: this conditional expectation is a *random variable* and we can calculate its expectation.

Lemma 2 (Basic properties) Let X, Y, Z be random varialbes, the following properties hold.

(a) $E[a|Y] = a$, where a is a given number.

(b) $E[aX + bY|Z] = aE[X|Z] + bE[Y|Z]$

(c) $E[X|Y] = E[X]$, if X,Y are independent

(d) $E[Xf(Y)|Y] = f(Y)E[X|Y]$

(e) $E[f(Y)|Y] = f(Y))$

(f) $E[X|f(Y), Y] = E[X|Y]$

(g) If $X \leq 0$, then $E[X|Y] \leq 0$

(h) If $X \leq Z$, then $E[X|Y] \leq E[Z|Y]$

Here we only prove some important properties.

PROOF **(d)** $E[Xf(Y)|Y] = f(Y)E[X|Y]$

For all y, we have

$$\begin{aligned} E[Xf(Y=y)|Y=y] &= \sum_x xf(y)p(x|Y=y) \\ &= f(y)\sum_x xp(x|Y=y) \\ &= f(y)E[X|y=y] \end{aligned}$$

| Since equation above is valid for any y, the proof is complete. □

PROOF **(e)** $E[f(Y)|Y] = f(Y)$

According to (d), we know $E[Xf(Y)|Y] = f(Y)E[X|Y]$. If we set X=1 and according to (a), we immediately get $E[f(Y)|Y] = f(Y)$ □

PROOF **(f)** $E[X|Y, f(Y)] = E[X|Y]$

According to definition of conditional expectation, for all y

$$\begin{aligned} E[X|Y = y, f(Y = y)] &= \sum_x xp(x|y, f(y)) \\ &= \sum_x x\frac{p(x, f(y)|y)}{p(f(y)|y)} \qquad (according\ to\ 10.1) \\ &= \sum_x x\frac{p(f(y)|x, y)p(x|y)}{p(f(y)|y)} \\ &= \sum_x xp(x|y) \qquad (p(f(y)|y) = 1, p(f(y)|x, y) = 1) \\ &= E[X|Y = y] \end{aligned}$$

So $E[X|Y, f(Y)] = E[X|Y]$. □

Lemma 3 Let X,Y,Z be random variables. The following properties hold.

(a) $E[E[X|Y]] = E[X]$

(b) $E[E[X|Y, Z]] = E[X]$

(c) $E[E[X|Y]|Y] = E[X|Y]$

You should keep in mind that $E[X|Y]$ is a random variable which is a function of random variable Y.

PROOF **(a)** $E[E[X|Y]] = E[X]$

Because $E[X|Y]$ is a function of random variable Y, we have

$$\begin{aligned} E[E[X|Y]] &= E[f(Y)] \\ &= \sum_y f(y)p(y) \\ &= \sum_y E[X|Y = y]p(y) \\ &= \sum_y \sum_x xp(x|y)p(y) \\ &= \sum_y \sum_x xp(x, y) \\ &= \sum_x xp(x) \\ &= E[X] \end{aligned}$$

□

PROOF **(b)** $E[E[X|Y, Z]] = E[X]$

$E[X|Y,Z]$ is a function of Y,Z, so we have

$$\begin{aligned}
E[E[X|Y,Z]] &= E[f(Y,Z)] \\
&= \sum_y \sum_z f(y,z)p(y,z) \\
&= \sum_y \sum_z E[X|Y=y,Z=z]p(y,z) \\
&= \sum_y \sum_z \sum_x xp(x|y,z)p(y,z) \\
&= \sum_y \sum_z \sum_x xp(x,y,z) \\
&= \sum_x xp(x) \\
&= E[X]
\end{aligned}$$

From the proof, we can see that the number of conditional random variables (Y,Z) can be expanded to any finite number. So we have $E[E[X|Y_1,\cdots,Y_n]] = E[X]$. □

PROOF **(c)** E[E[X|Y]|Y]=E[X|Y] This proof follows immediately from property Lemma 2(e). Because $E[X|Y]$ is a function of Y which is also a random variable, we denote it as $f(Y) = E[X|Y]$. So we have

$$\begin{aligned}
E[E[X|Y]|Y] &= E[f(Y)|Y] \\
&= f(Y) \qquad (Lemma\ 2(e)) \\
&= E[X|Y]
\end{aligned}$$

□

3. Definition of stochastic convergence ($\Omega issample\ space$)

   (a) Sure convergence

Definition 10 $\{X_k\}$ converges surely (or *everywhere* or *pointwise*) to X if

$$\lim_{k\to\infty} X_k(\omega) = X(\omega), \text{ for all } \omega \in \Omega$$

It means that $\lim_{k\to\infty} X_k(\omega) = X(\omega)$ is valid for all points in Ω. It also can be equivalently stated as:

$$A = \Omega \text{ where } A = \left\{\omega \in \Omega : \lim_{k\to\infty} X(\omega) = X(\Omega)\right\}$$

   (b) Almost sure convergence

Definition 11 $\{X_k\}$ converges almost surely (or *almost everywhere* or *with probability 1*) to X if

$$P(A) = 1 \text{ where } A = \omega \in \Omega : \lim_{k\to\infty} X_k(\omega) = X(\omega)$$

This definition means when $k \to \infty$, points in Ω almost satisfy

$$\lim_{k\to\infty} X_k(\omega) = X(\omega)$$

The set of poinsts for which the limit is invalid has a zero measure. Almost sure convergence **first evaluates the convergence of every point** in Ω, then **check the measure of set** which is formed by the points that can converge.

(c) Convergence in probability

Definition 12 $\{X_k\}$ converges in probability to X if for any $\epsilon > 0$,

$$\lim_{k\to\infty} P(A_k) = 0 \text{ where } A_k = \{\omega \in \Omega : \|X_k(\omega) - X(\omega)\| > \epsilon\}$$

The difference between **Almost sure converge** and **Convergence in probability** is that for convergence in probability, we check the **whole sample space** for a given k and form set A_k by the points that is valid for $\|X_k(\omega) - X(\omega)\| > \epsilon$. From this procedure, we have a set sequence $\{A_k\}$ and if the measure of this sequence converges to zero,X_k converges in probability. That is to say, **Almost sure converge** only cares about the result (last set), **Convergence in probability** cares about process (the sequence of A_k).

Section 12

Convergence of stochastic sequences

1. Convergence of martingale sequence

Definition 13 A stochastic sequence $\{X_k\}_{k=1}^{\infty}$ is called a martingle if $E[|X_k|] < \infty$ and

$$E\left[X_{k+1}|X_1, \cdots, X_k\right] = X_k$$

almost surely for all k. In application, $E\left[X_{k+1}|X_1, \cdots, X_k\right]$ is often written as $E\left[X_{k+1}|\mathcal{H}_k\right]$ for short where $\mathcal{H}_k = X_1, \cdots, X_k$

Here "almost surely" is due to that $E[X_{k+1}|\mathcal{H}_k]$ is a random variable (function of $\mathcal{H}_k$).

Definition 14 A stochastic sequence $\{X_k\}_{k=1}^{\infty}$ is called a *submartingle* if $E[|X_k|] < \infty$ and

$$E\left[X_{k+1}|X_1, \cdots, X_k\right] \geq X_k$$

for all k.

Definition 15 A stochastic sequence $\{X_k\}_{k=1}^{\infty}$ is called a *supermartingale* if $E[|X_k|] < \infty$ and

$$E\left[X_{k+1}|X_1, \cdots, X_k\right] \leq X_k$$

for all k.

Theorem 6 (Martingale convergence theorem). If $\{X_k\}$ is a *submartingale (or supermartingale)*, then there is a finite random variable X such that $X_k \to X$ almost surely.

2. Convergence of quasimartingale sequence

The event A_k is defined as $A_k \doteq \{\omega \in \Omega : E[X_{k+1-X_k}|\mathcal{H}_k] \geq 0\}$, where $\mathcal{H}_k =$

$\{X_1 \cdots, X_k\}$. Intuitively, A_k indicates that X_{k+1} is greater than X_k in expectation. Let $\mathbb{1}_{A_k}$ be an indicator function:

$$\mathbb{1}_{A_k} = \begin{cases} 1 \;\; E[X_{k+1} - X_k | \mathcal{H}_k] \geq 0, \\ \\ 0 \;\; E[X_{k+1} - X_k | \mathcal{H}_k] < 0. \end{cases}$$

Theorem 7 (Quasimartingale convergence theorem). For a *nonnegative stochastic sequence* $\{X_k \geq 0\}$, if

$$\sum_{k=1}^{\infty} E\left[X_{k+1} - X_k \mathbb{1}_{A_k}\right] < \infty$$

then there is a *finite random variable* such that $X_k \to X$ almost surely as $k \to \infty$.

PART

V

Monte Carlo Methods

Optimal policies based on a system model can be obtained by solving the optimal Bellman equation. This chapter introduces model-free algorithms that do not presume system models.

The key point is how to learn from data which can be observed from system rather than a model. The simplest example of learning from data is mean estimation problem. Now we demonstrate the differences between model-based and model-free methods. Consider a random variable X. If we know the probability distribution of X, then the mean of X can be directly calculated based on the definition of the expected value (model-based approach):

$$E[X] = \sum_x p(x)x$$

When the probability distribution is unknown, and we have some samples $\{x_1, x_2, \cdots, x_n\}$, the mean can be approximated as:

$$E[X] \approx \overline{x} = \frac{1}{n}\sum_{j=1}^{n} x_j \tag{12.1}$$

When n is small, the estimation may not be accurate. However, as n increase, the estimation will approach the actual expectation. The correctness is guaranteed by Law of large numbers (8).

Theorem 8 (Law of large numbers) For a random variable X, suppose that ${x_n}_{i=1}^n$ are i.i.d (the abbreviation of independent identically distribution) samples. Lets $\overline{x} = \frac{1}{n}\sum_{i=1}^n x_i$ be the average of the samples. Then

$$E[\overline{x}] = E[X]$$
$$var[\overline{x}] = \frac{1}{n}var[X]$$

Law of large numbers show that $\overline{x}$ is an unbiased estimate of $E[X]$, and its variance decreases to zero as n increases to infinity.

Section 13

MC Basics: The simplest MC-based algorithm

This algorithm is obtained by replacing the model-based policy evaluation step with a model-free MC estimation step. There are two steps in every iteration of the policy iteration algorithm. The first step is policy evaluation, which aims to evaluate v_{π_k} by solving $v_{\pi_k} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}$. The second step is policy improvement, which aims to evaluate the greedy policy $\pi_{k+1} = \arg\max_\pi(r_\pi + \gamma P_\pi v_{\pi_k})$. The elementwise form of

policy improvement step is:

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

Action value functions can be calculated in two crucial steps. In the first step, value functions are determined for the purpose of calculating action-value functions. In the second step, action-value functions are employed to select an optimal policy. Here are two methods for calculating action-value functions:

1. Model-based approach. This approach is adopted by the policy iteration algorithm. First, we obtain the value function by solving the Bellman equation. Next, we calculate the action-value function using the equation below:

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

2. Model-free approach. Recall the definition of action-value function:

$$\begin{aligned}q_{\pi_k} &= E[G_t|s,a] \\ &= E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3}, \cdots |s,a]\end{aligned}$$

According to equation (12.1), the action-value function represents an expected reward of actions (a) given a state (s). We can approximate this function using Monte Carlo (MC) methods. The agent interacts with the system from state (s) and action (a), following policy π_k. Assuming n episodes, the returns received in each episode are denoted as $g^{(i)}_{\pi_k}(s,a)$. Consequently, the estimated action-value function can be calculated as:

$$\begin{aligned}q_{\pi_k} &= E[G_t|s,a] \\ &\approx \frac{1}{n}\sum_{i=1}^{n} g^{(i)}_{\pi_k}(s,a)\end{aligned}$$

This is the fundamental idea of MC-based method.

Subsection 13.1

The basic MC algorithm

Starting from an initial policy π_0, the algorithm has two steps in the kth iteration:

1. Step 1: policy evaluation. In this step, the algorithm estimate $q_{\pi_k}(s,a)$ for **all (s,a)**. Specifically, for every (s,a), we collect sufficient many episodes and use the average of returns denoted as $q_k(s,a)$ to approximate $q_{\pi_k}(s,a)$.

2. Step 2: policy update. Policy is updated in this step by solving π_{k+1} for all $s \in S$. The greedy optimal policy is $\pi_{k+1}(a^*|s) = 1$ where $a^* = \arg\max_a = q_k(s,a)$.

This algorithm is similar to the policy iteration algorithm, but unlike the latter, basic Monte Carlo (MC) algorithm **directly estimates action-value functions** without requiring a value function v_{π_k}.

Since policy iteration algorithms converge, MC algorithms also converge. With sufficient episodes, the average return approximates the action-value function accurately.

In practice, however, we don't usually need an excessive number of episodes for every state-action pair, and the estimation may not be very accurate. Nonetheless, the algorithm still functions, similar to truncated policy iteration methods.

Algorithm 4: MC basic Policy Iteration Algorithm

Input: Initial guess π_0

Output: Search for the optimal policy

```
. for kth iteration (k = 1, 2, 3, ···) do
    for every state s do
        for every action a ∈ A(s) do
            Collect sufficient many episodes staring from (s,a) by following
             policy π_k.
            /* Policy evaluation                                        */
            q_{π_k}(s, a) ≈ q_k(s, a) (the average return from all the episodes staring
             from (s,a)).
        end
        /* Policy improvement                                          */
        a*_k = arg max_a q_k(s, a)
         π_{k+1}(a|s) = 1 if a = a*_k, and π_{k+1}(a|s) = 0, otherwise
    end
end
```

Subsection 13.2

MC exploring stars

MC basic algorithm has too low sample efficiency (many same state-action pair are calculated in different episodes). The key point of improving MC basic algorithm is to **use samples efficiently**. Suppose that we have an episode following a policy π

$$s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_4} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \cdots$$

Every time a state-action pair appears in an episode, it is referred to as a **visit** of that state-action pair. If the MC basic algorithm only uses the episode return to estimate the action-value function, and only the starting state-action pair, which is (s_1, a_2) in this episode, can obtain an estimation of the action-value function, then this strategy is referred to as the initial visit strategy which is not sample efficient.

For the purpose of improving sample efficiency, we can decompose this episode into many multiple subepisodes:

$$\begin{aligned}
&s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_4} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \cdots && \text{origin episode}\\
&\quad s_2 \xrightarrow{a_4} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \cdots && \text{subepisode starting from } (s_2, a_4)\\
&\quad\quad s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \cdots && \text{subepisode starting from } (s_1, a_2)\\
&\quad\quad\quad s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \cdots && \text{subepisode starting from } (s_2, a_3)\\
&\quad\quad\quad\quad s_5 \xrightarrow{a_1} \cdots && \text{subepisode starting from } (s_5, a_1)
\end{aligned}$$

If we regard these subepisodes as distinct from the original one and employ the initial visit strategy, more accurate action value functions can be estimated. It's essential to note that for the same state-action pair, it may be visited in various subepisodes (for instance, (s_1, a_2)). If only the first visit of this state-action pair is considered, the strategy is referred to as the first visit strategy. Conversely, if every visit of this state-action pair is taken into account, the strategy is known as the every visit strategy.

In practice, we don't need to decompose original episodes into different subepisodes nor apply the initial visit strategy for each one. Instead, we can compute returns from back to front in the original episodes, achieving action-value functions for all state-action pairs in a single traversal. The detailed algorithm is presented below:

Algorithm 5: MC Exploring Starts

Input: Initial policy $\pi_0(a|s)$ and initial value $q(s,a)$ for all (s,a).
$Returns(s,a) = 0$ and $Num(s,a) = 0$ for all (s,a)

Output: Search for an optimal policy

while **Policy is not convergent** **do**
  Episode generation: Select a staring state-action pair (s_0, a_0) and ensure all pairs can be possibly selected **exploring-starts condition**.
  Following the current policy, generate an episode of length T: $s_0, a_0, r_1, \cdots, s_{T-1}, a_{T-1}, r_T$.
  for **every episode** **do**
    $g \leftarrow 0$
    for **each step of the episode,** $t = T-1, T-2, \cdots, 1, 0,$ **do**
      $g \leftarrow \gamma g + r_{t+1}$
      `/* every visit strategy */`
      $Returns(s_t, a_t) \leftarrow Returns(s_t, a_t) + g$
      $Num(s_t, a_t) \leftarrow Num(s_t, a_t) + 1$
      `/* Policy evaluation */`
      $q(s_t, a_t) \leftarrow Returns(s_t, a_t)/Num(s_t, a_t)$
      `/* Policy improvement */`
      $\pi(a|s_t) = 1$ if $a = \arg\max_a q(s_t, a)$, otherwise, $\pi(a|s_t) = 0$
    end
  end
end

Exploring-starts condition requires every staring state-action pair should have sufficient many episodes. Only if every state-action pair is well explored can the accurate action-value function be estimated. However, meeting this condition is difficult in many applications, particularly those involving physical interactions with environments. $\epsilon-$greedy policy removes the exploring-starts condition requirements.

The reason $\epsilon-$greedy policy can eliminate the need for exploring-starts condition is its soft nature. A soft policy assigns a positive probability to every action at every state. Consequently, generating one episode following a soft policy, the length of the episode will sufficient long so that every state-action can be visited and also well explored, thereby allowing us to remove the exploring-starts condition requirement.

But how to assign the probability to the action? We also need to guarantee that the action with greatest action-value function should have bigger probability. Suppose that $\epsilon \in [0, 1]$, the corresponding $\epsilon-$greedy policy has the following form:

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

where $|\mathcal{A}(s)|$ denotes the number of actions associated with state s.

The $\epsilon-$greedy policy has properties listed below:

1. Become greedy policy when $\epsilon = 0$ and taking all actions with equal probability when $\epsilon = 1$ (At this point, the agent behavior is completely random). Now, we can see $\epsilon-$greedy policy provide a way to balance exploration and exploitation (greedy policy).

2. The probability of taking the greedy action is always than that of taking other actions because:

$$1-\frac{\epsilon}{|\mathcal{A}(s)|}\left(|\mathcal{A}(s)|-1\right)=1-\epsilon+\frac{\epsilon}{|\mathcal{A}(s)|}\geq\frac{\epsilon}{|\mathcal{A}(s)|}$$

Using $\epsilon-$greedy policy to modify MC Exploring Starts algorithm and get the MC $\epsilon-$Greedy algorithm:

Algorithm 6: MC Exploring Starts

Input: Initial policy $\pi_0(a|s)$ and initial value $q(s,a)$ for all (s,a).
$Returns(s,a)=0$ and $Num(s,a)=0$ for all (s,a)

Output: Search for an optimal policy

while **Policy is not convergent** **do**
  Episode generation: Select a staring state-action pair (s_0,a_0) and ensure all pairs can be possibly selected **exploring-starts condition**.
  Following the current policy, generate an episode of length T: $s_0,a_0,r_1,\cdots,s_{T-1},a_{T-1},r_T$.
  for **every episode** **do**
    $g\leftarrow 0$
    for **each step of the episode,** $t=T-1,T-2,\cdots,1,0,$ **do**
      $g\leftarrow\gamma g+r_{t+1}$
      `/* every visit strategy */`
      $Returns(s_t,a_t)\leftarrow Returns(s_t,a_t)+g$
      $Num(s_t,a_t)\leftarrow Num(s_t,a_t)+1$
      `/* Policy evaluation */`
      $q(s_t,a_t)\leftarrow Returns(s_t,a_t)/Num(s_t,a_t)$
      `/* Policy improvement */`
      let $a^*=\arg\max_a q(s_t,a)$ and

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

    end
  end
end

PART

VI

Stochastic Approximation

This part lays the necessary foundation for learning Temporal-Difference Methods. We reconsider the mean estimation problem, suppose there is a sequence of i.i.d samples $\{x_i\}_{i=1}^n$, the expectation of X can be approximated by:

$$E[X] \approx \overline{x} \doteq \frac{1}{n}\sum_{i=1}^{n} x_i \tag{13.1}$$

There are two methods to solve 13.1:

1. Non-incremental method. This is a standard MC method that collects all the samples first and then calculates the average. For an accurate estimation, a sufficient number of samples should be collected, which may require a significant amount of time.

2. Incremental method. To avoid the drawbacks of non-incremental method, this method calculate the average incrementally instead of waiting for all samples. Specially,

$$\omega_{k+1} \doteq \frac{1}{k}\sum_{i=1}^{k} x_i$$

and hence

$$\omega_k = \frac{1}{k-1}\sum_{i=1}^{k-1} x_i$$

The ω_{k+1} can be expressed as:

$$\omega_{k+1} = \frac{1}{k}(\sum_{i=1}^{k-1} x_i + x_k) = \frac{1}{k}\left[(k-1)\omega_k + x_k\right] = \omega_k - \frac{1}{k}(\omega_k - x_k) \tag{13.2}$$

The advantage of 13.2 is that average can be calculated immediately upon receiving a sample. As more samples are obtained, the accuracy of estimation can be gradually improved according to the law of large numbers. Moreover, replacing k in 13.2 is replaced with α results in the main algorithm we will discuss in this part.

$$\omega_{k+1} = \omega_k - \alpha(\omega_k - x_k) \tag{13.3}$$

Subsection 13.3

Robbins-Monro algorithm

Algorithm 13.3 is a typical representation of the stochastic approximation method. In fact, the term "stochastic approximation" refers to a broad class of **stochastic iterative** algorithms for solving root-finding or optimization problems. Compared to many other root-finding algorithms, stochastic approximation is powerful because it does not require knowledge of objective functions or their derivatives. This section will introduce the famous Robbins-Monro (RM) algorithm.

Suppose we want to find the root of the equation:

$$g(\omega) = 0$$

Before introducing the details of RM algorithm, it is essential to note that root-finding problem is significant as many other problems can be formulated into its form. For example, if we want to find the maximum or minimum of $J(\omega)$, we can let $g(\omega) = \nabla J(\omega)$ and solve the equation $g(\omega) = 0$ which is a root-finding problem.

As mentioned above, RM is powerful because it does not require the expression of objective function and its derivatives [13]. Even the function has a noisy observation, RM algorithm still solve the problem. Suppose that noisy observation of $g(\omega)$ is expressed as:

$$\tilde{g}(\omega) = g(\omega) + \eta$$

where η is the observation error, which is Gaussian or not.

[13] *for instance, function represented by neural networks whose structure and parameters are unknown*

The RM algorithm that solve $g(\omega) = 0$ is:

$$\omega_{k+1} = \omega_k - \alpha\tilde{g}(\omega) \tag{13.4}$$

The reason for RM algorithm can find the root is very simple. That is ω_{k+1} is closer to ω^* than ω_k. The explanation is listed below:

1. If $\omega_k > \omega^*$, then $\tilde{g}(\omega) > 0$. Then $\omega_{k+1} = \omega_k - \alpha\tilde{g}(\omega) < \omega_k$. If $\alpha\tilde{g}(\omega)$ is sufficient small, we have $\omega^* < \omega_{k+1} < \omega_k$.
2. If $\omega_k < \omega^*$, then $\tilde{g}(\omega) < 0$. Then $\omega_{k+1} = \omega_k - \alpha\tilde{g}(\omega) > \omega_k$. If $\alpha\tilde{g}(\omega)$ is sufficient small, we have $\omega_k < \omega_{k+1} < \omega^*$.

In either case, ω_{k+1} is closer to ω^* than ω_k. A rigorous convergence result is given below:

Theorem 9 (Robbins-Monro theorem) In the Robbins-Monro algorithm 13.4, if

(a) $0 < c_1 \leq \nabla_\omega g(\omega) \leq c_2$ for all ω;

(b) $\sum_{k=1}^{\infty} a_k = \infty$ and $\sum_{k=1}^{\infty} a_k^2 < \infty$

(c) $E[\eta_k|\mathcal{H}_k] = 0$ and $E[\eta^2|\mathcal{H}_k] < \infty$

where $\mathcal{H}_k = \{\omega_k, \omega_{k-1}, \cdots\}$, then ω_k almost surely converges to ω^* satisfying $g(\omega^*) = 0$.

The three conditions in theorem 9 are explained as follows:

1. $0 < c_1 \leq \nabla_\omega g(\omega) \leq c_2$ for all ω. This condition indicates that $g(\omega)$ is a monotonically increasing function which guarantee that $g(\omega) = 0$ has a unique root. $\nabla g(\omega) \leq c_2$ indicates that the gradient of $g(\omega)$ is bounded from above.
2. The second condition is common in reinforcement learning. $\sum_{k=1}^{\infty} a_k^2 < \infty$ requires a_k converges to zero as $k \to \infty$, whereas $\sum_{k=1}^{\infty} a_k = \infty$ requires that a_k should not converge to zero too fast. Here is insight into why $\{a_k\}$ should be configured this way.

   First, suppose that the observation