동적계획법과 강화학습 Lecture 3장

- 이태헌 –

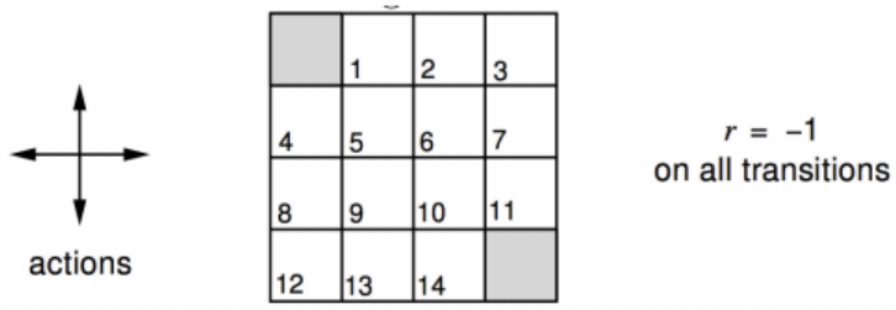
<Value Evaluation and Policy Iteration>

MDP를 알고 있을 때(Reward, Transition Probability를 알고 있음) 사용하는 방법에는 3가지가 있다.

1. Policy evaluation
2. Policy iteration (Policy improve)
3. Value iteration

Policy evaluation

Small Grid World Example을 통해 Policy evaluation을 적용할 때는 S, A의 크기가 작고 R, T에 대해 알고 있으므로 테이블 기반 방법론(벨만 기대 방정식을 반복적으로 사용하여 테이블 값을 업데이트)을 활용하여 문제를 풀 수 있다.



주어진 Grid World는 아래와 같으며, uniform random plicy를 택했을 경우 value function 값을 구하는 것이 목적이다.

S : 1~14 nonterminal state, 2 teriminal state

A : 4방향 1칸 이동, 바깥쪽 state에서 grid-world를 벗어나는 action의 경우, 제자리로 돌아옴

R : -1

T : $P_{ss'}^{a} = 1$

γ : 1

Policy : 4방향 uniform random policy

$$\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25$$

1) 모든 state의 값을 임의의 값 0으로 초기화한다.

v_k for the Random Policy

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

2) 개별 state 값 업데이트

State1 에서 현재의 V_0을 one-step씩 이동 가능한 모든 next state의 value function으로 다음 iteration의 V_1 을 구하여 update 한다.

v_k for the Random Policy

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

State1

Up : $V_1(s) = 0.25 * (-1 + 0)$

Down : $V_1(s) = 0.25 * (-1 + 0)$

Left : $V_1(s) = 0.25 * (-1 + 0)$

Right : $V_1(s) = 0.25 * (-1 + 0)$

> $V_1(s) = 4 * 0.25 * (-1) = -1$

위의 과정을 모두 계산하면 1번째 iteration 결과 update 된 value function은 아래와 같다.

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

다음 iteration 통해 state2, state3의 $V_2(s)$ 값을 각각 구하면,

State2

Up : $V_2(s) = 0.25 * (-1 + (-1))$

Down : $V_2(s) = 0.25 * (-1 + (-1))$

Left : $V_2(s) = 0.25 * (-1 + 0)$

Right : $V_2(s) = 0.25 * (-1 + (-1))$

> $V_2(s) = 3 * 0.25 * (-2) + 0.25 * (-1) = -1.75$

State3

Up : $V_2(s) = 0.25 * (-1 + (-1))$

Down : $V_2(s) = 0.25 * (-1 + (-1))$

Left : $V_2(s) = 0.25 * (-1 + (-1))$

Right : $V_2(s) = 0.25 * (-1 + (-1))$

> $V_2(s) = 4 * 0.25 * (-2) = -2$

'

값이 나오게 된다. 이를 개별 state마다 update 하면 2번째 iteration은 다음과 같아진다.

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

수렴할 때까지 위와 같은 방법의 iteration 통해 evaluation 결과를 얻을 수 있으며, 해당 policy에 대한 value function을 구하는 과정이 policy evaluation

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

Policy Improvement

현재 Policy을 평가하는 이유는 더 나은 Policy를 찾기 위함, Iteration을 통해 value function 찾은 후 이 Policy의 효과를 판단하고 Policy를 update 해야함. 이를 통해 더 나은 Policy를 따르게 되고 Optimal Policy에 가까워지는 과정을 Policy improvement라

고 함.

Policy π에 대한 Policy evaluation value는 아래와 같으며,

$$v_\pi(s) = E[R_{t+1} + \gamma R_{t+2} + \cdots | S_t = s]$$

이를 Greedy policy improvement (value가 높은 state로 이동, max 값 선택)하면,

$$\pi' = greedy(v_\pi)$$
$$\pi'(s) = argmax\ q_\pi(s,a)$$

One step 과정을 거치며 모든 state를 improve하면 아래와 같은 식으로 정의할 수 있다.

$$Q_\pi(s,\pi'(s)) = max\ q_\pi(s,a) \geq q_\pi(s,\pi(s)) = v_\pi(s)$$

Grid-world에서 greedy policy improvement를 아래와 같이 Policy evaluation 결과로 얻은 (state1, state5) value function 값에 적용하면, 아래 식과 같다.

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

State1

Up : $q_\pi(1,0) = -1 + (-14)$

Down : $q_\pi(1,1) = -1 + (-18)$

Left : $q_\pi(1,2) = -1 + (0)$

Right : $q_\pi(1,3) = -1 + (-20)$

> $maxq_\pi(1,a) = q_\pi(1,left)$

State5

Up : $q_\pi(5,0) = -1 + (-14)$

Down : $q_\pi(5,1) = -1 + (-20)$

Left : $q_\pi(5,2) = -1 + (-14)$

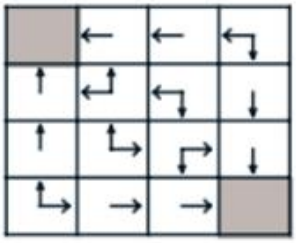
Right : $q_\pi(5,3) = -1 + (-20)$

> $maxq_\pi(5,a) = q_\pi(5,up)\ \ or\ \ q_\pi(5,left)$

Sate 별로 취할 수 있는 action 중 greedy policy improvement를 통해 선택된 optimal policy는 아래와 같다.

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



Grid-world는 단순하고 사이즈가 작은 문제이기 때문에 Policy evaluation과 Policy improvement를 적은 수로 반복하여도 optimal 값을 찾을 수 있다. $\pi' = \pi^*$ 하지만 일반적인 경우에는 π^* 값으로 수렴하기 위해 더 많은 Policy iteration이 필요함.

Value Iteration

Bellman Optimality Equation을 이용하여 적용. Value iteration은 각 state에서 action을 취할 확률을 곱하여 summation하는 방법을 사용하는 대신, max 값을 취함.

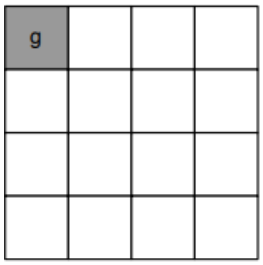
Theorem (Principle of Optimality)

A policy $\pi(a|s)$ achieves the optimal value from state s, $v_\pi(s) = v_*(s)$, if and only if

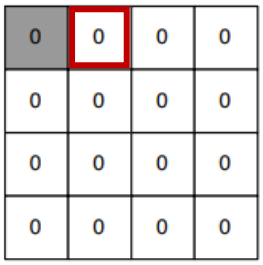
- *For any state s' reachable from s*
- *π achieves the optimal value from state s', $v_\pi(s') = v_*(s')$*

$$v_*(s) \leftarrow max R_s^a + \sum_{s' \in S} P_{ss'}^a v_*(s')$$

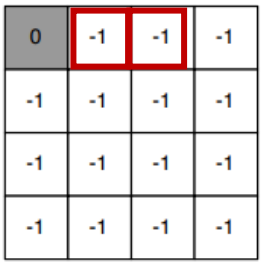
Value iteration을 grid-world에 적용하면 다음과 같다.



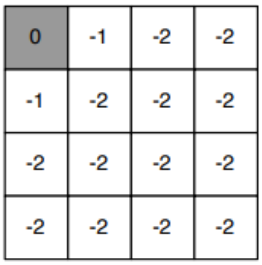
Problem



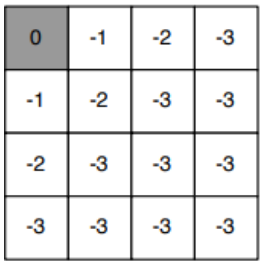
V_1



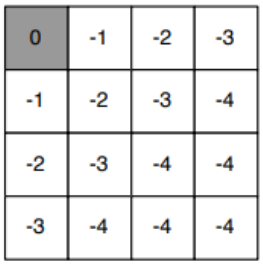
V_2



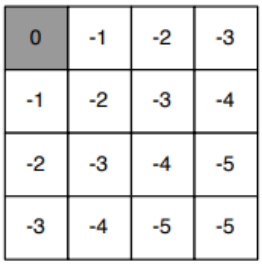
V_3



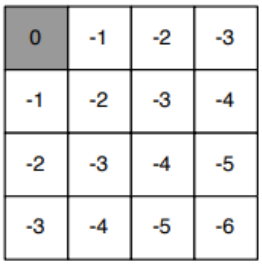
V_4



V_5



V_6



V_7

State1

Up $V_1(s) = -1 + 0$

Down $V_1(s) = -1 + 0$

Left $V_1(s) = -1 + 0$

Right $V_1(s) = -1 + 0$

$\therefore V_1(s) = \max V_1(s) = -1$

State1

Up $V_2(s) = -1 + (-1)$

Down $V_2(s) = -1 + (-1)$

Left $V_2(s) = -1 + (0)$

Right $V_2(s) = -1 + (-1)$

$\therefore V_2(s) = \max V_2(s) = -1$

State2

Up $V_2(s) = -1 + (-1)$

Down $V_2(s) = -1 + (-1)$

Left $V_2(s) = -1 + (-1)$

Right $V_2(s) = -1 + (-1)$

$\therefore V_2(s) = \max V_2(s) = -2$

Policy iteration과 다르게 Value iteration은 명시적인 Policy가 없음. Value iteration의 경우 현재 Policy가 optimal 하다는 전제로 Value function 값을 취하기 때문에 deterministic한 action이 된다.

Reference

[1] 이웅원, 양혁렬, 김건우, 이영무, 이의령, "파이썬과 케라스로 배우는 강화학습", 위키북스, 2020

[2] 노승은, "바닥부터 배우는 강화학습", 영진닷컴, 2020