Appendix

내가 몰라서 만드는 Appendix

Dynamic Programming ⇒ to understand MP

DP는 재귀를 이용하여 최적화 솔루션을 얻어내는 알고리즘이다. DP는 주어진 문제를 작은 문제로 나누어, 중간 과정을 저장하며, 반복하여 문제를 풀어내고, 가장 좋은 결과를 찾아내는 최적화 알고리즘이다.

하나의 예시로, 무게 W까지 물건 i개를 담을 수 있는 가방이 있다. 우리는 개별 item에 대한 중요도(가중치)를 알고 있을 때, 가방에 담은 item의 중요도의 합이 높아지도록 가방을 싸는게 목적이다. 이 때, 가방에 담을 수 있는 n개의 물건이 있다면 조사해야 할 부분집합의 수는 2^n으로 계산량이 어마어마하다.

반면 DP는 “다음 물건을 담았을 때 중요도가 올라가는가?”로 문제를 재정의한다. 즉, 이전까지의 물건 i개와 무게 w를 담았을 때의 가치를 $V[i,w], w \leq W$로 정의하겠다. 그리고 item을 바꾼 $v_i + V[i-1, w-w_i]$가 $V[i,w]$보다 큰지를 확인하는 문제로 바꾼 것이다. 또한 i와 w를 점차 증가시키는 bottom-up 접근법을 이용해 재귀적으로 탐색한다. 대충 아래 그림처럼 탐색한다.

Let $W = 10$ and

i	1	2	3	4
v_i	10	40	30	50
w_i	5	4	6	3

$V[i,w]$	0	1	2	3	4	5	6	7	8	9	10
$i=0$	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	10	10	10	10	10	10
2	0	0	0	0	40	40	40	40	40	50	50
3	0	0	0	0	40	40	40	40	40	50	70
4	0	0	0	50	50	50	50	90	90	90	90

이러한 특징 때문에 저장하는 용량이 커지면 시간이 감소하는 trade-off 관계를 담고 있다. RL에서는 Reward를 최대화 할 수 있는 최적 알고리즘으로 생각할 수 있다.

확률의 안정화 ⇒ to understand MRP

Markov Chain에서 다음의 예시를 생각해보겠다.

오늘/내일	맑음	흐림
맑음	0.7	0.3
흐림	0.25	0.75

위 경우는 다음과 같이 전이확률 행렬로 적을 수 있다. $P = \begin{bmatrix} 0.7 & 0.3 \\ 0.25 & 0.75 \end{bmatrix}$

이 전이 확률은 “오늘의 날씨를 알 때 내일의 날씨는?”에 대한 답을 할 수 있게 도와준다.그런데 만약 “오늘과 내일의 날씨를 알 때 모레의 날씨는?”이라고 묻는다면 어떻게 구할 수 있을까? 위의 전이행렬을 2번 곱하면 될 것이다.

$P_2 = \begin{bmatrix} 0.7 & 0.3 \\ 0.25 & 0.75 \end{bmatrix} \begin{bmatrix} 0.7 & 0.3 \\ 0.25 & 0.75 \end{bmatrix} = \begin{bmatrix} 0.565 & 0.435 \\ 0.362 & 0.637 \end{bmatrix}$이 된다. 그렇다면 “오늘과 내일과 ... k-1 시점의 날씨를 알 때 k 시점의 날씨는?”이라고 묻는다면 P 전이확률을 k-1번 곱할 것이다. 만약 k가 충분히 큰 숫자라면, 이 전이행렬은 변하지 않는 “안정 상태" 혹은 “정적 분포"가 되어있을 것이다.

위의 전이행렬의 경우, Python을 이용해서 반복한 결과$P_{stationary} = \begin{bmatrix} 0.4545 & 0.5454 \\ 0.4545 & 0.5454 \end{bmatrix}$가 되었다.

```
import numpy as np
matrix = np.array([[0.7, 0.3], [0.25,0.75]])
current_matrix = matrix.copy()
epsilon = 1e-5
iteration = 0
while True :
    next_matrix = np.dot(current_matrix, matrix)

    if abs(current_matrix - next_matrix).sum() > epsilon :
        current_matrix = next_matrix.copy()
        iteration += 1
    else :
        print(f'At Iteration {iteration}, Stationary Matrix is Formed')
        display(current_matrix)
        break;
# At Iteration 14, Stationary Matrix is Formed
# array([[0.45454888, 0.54545112],
#        [0.4545426 , 0.5454574 ]])
```

Monotonic Improvement ⇒ to understand PI

$V^{\pi_{i+1}} \geq V^{\pi_i}$: $V^{\pi_{i+1}}(s) \geq V^{\pi_i}(s)$를 이용해서 증명할 수 있다.

$$
\begin{aligned}
&V^{\pi_i}(s) \\
&\leq \underset{a}{max} Q^{\pi_i}(s,a) \\
&= \underset{a}{max} R(s,a) + \gamma \underset{s' \in S}{P}(s'|s,a) V^{\pi_i}(s') \\
&= R(s,\pi_{i+1}(s)) + \gamma \sum_{s' \in S} P(s'|s,\pi_{i+1}(s)) V^{\pi_i}(s') \\
&\leq R(s,\pi_{i+1}(s)) + \gamma \sum_{s' \in S} P(s'|s,\pi_{i+1}(s)) (\underset{a}{max} Q^{\pi_i}(s',a')) \\
&= R(s,\pi_{i+1}(s)) + \gamma \sum_{s' \in S} P(s'|s,\pi_{i+1}(s)) (\underset{a}{max} R(s',a') + \cdots) \\
&\vdots \\
&= V^{\pi_{i+1}}(s)
\end{aligned}
$$

Step을 만들어 살펴보자.

Step 1 :

$V^{\pi_i}(s)$

π_i를 따를 때, s에서의 Value를 의미한다.

Step 2 :

$$
\begin{aligned}
&\underset{a}{max} Q^{\pi_i}(s,a) \\
&= \underset{a}{max} R(s,a) + \gamma \underset{s' \in S}{P}(s'|s,a) V^{\pi_i}(s') \\
&= R(s,\pi_{i+1}(s)) + \gamma \sum_{s' \in S} P(s'|s,\pi_{i+1}(s)) V^{\pi_i}(s')
\end{aligned}
$$

$\underset{a}{max} Q^{\pi_i}(s,a)$

⇒ 모든 action 을 가지고, π_i를 따랐을 때 얻을 수 있는 Q값

first equation

⇒ 그 값은 $Q^{\pi_i}(s,a)$의 정의를 풀어낸 값과 동일하며,

second equation

⇒ 그 값을 최대화 한다는 것은, 다음 state부터는 다음 policy를 따랐을 때의 행동을 따랐을 때의 Value와 같다.

Step 3 :
$R(s,\pi_{i+1}(s)) + \gamma \sum_{s' \in S} P(s'|s,\pi_{i+1}(s))(\underset{a}{max} Q^{\pi_i}(s',a'))$
$= R(s,\pi_{i+1}(s)) + \gamma \sum_{s' \in S} P(s'|s,\pi_{i+1}(s))(\underset{a}{max} R(s',a') + \cdots)$
$\vdots$
$= V^{\pi_{i+1}}(s)$

s'일 때는 Step 2의 값을 이용해서 알아보았다면,
s''일 때는 Step 2의 과정을 중첩해서 사용하는 것과 동일하다.

그리고 이 과정을 무한히 반복한다면,
결국에는 π는 사라지고, π_{i+1}만 남게되는데,
그 값이 $V^{\pi_{i+1}}$이란 뜻이다.

그래서 $V^{\pi_i}(s) \leq V^{\pi_{i+1}}(s)$가 성립한다.

∴ 모든 $s \in S$에 대해서 $V^{\pi_i}(s) \leq V^{\pi_{i+1}}(s)$를 성립하기 때문에, $V^{\pi_i} \leq V^{\pi_{i+1}}(s)$를 성립한다.

Bellman Equation

이름만 봐도 너무 하기 싫게 생겼습니다.
Bellman... 공식 이름에 사람 이름이 들어가면 다 어렵던데...
이것도 얼마나 어려울까요 ㅠ-ㅠ
그래도 시작해봅시다.

Bellman Expectation Equation (벨만 기대방정식)

벨만 기대 방정식은 현재 state의 V와 다음 state의 V사이의 관계를 나타낸다.
다음과 같이 유도할 수 있다.

$$
\begin{aligned}
&V(s) \\
&= \mathbb{E}[G_t|S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \cdots |S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} \cdots)|S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma G_{t+1}|S_t = s]
\end{aligned}
$$

$= \mathbb{E}[R_{t+1} + \gamma V(s_{t+1}) | S_t = s]$
$\therefore V(s) = \mathbb{E}[R_{t+1} + \gamma V(s_{t+1}) | S_t = s]$

π를 따르는 벨만 기대방정식

$V^\pi(s) = \mathbb{E}^\pi[R_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s]$

π를 따르는 Q함수의 벨만 기대방정식

$Q^\pi(s, a) = \mathbb{E}^\pi[R_{t+1} + \gamma Q^\pi(s_{t+1}, A_{t+1}) | s_t = s, A_t = a]$

엇 우리가 본 것과 차이가 없는데요? 왜 Bellman인가요?
⇒ 저 $V(s)$와 $Q(s)$ 모두 Bellman이 도출해낸 공식이기 때문
그냥 똑같다. 하지만 여기서 끝이 아니겠죠? ㅎㅎ

Bellman (Backup) Operator

기본적으로 Bellman Equation과 동일하지만,
수학적으로 좀 더 편하게 사용하기 위해 만든 표시자

S공간에 state가 n개 있다고 하자.
그럼 V는 다음과 같이 표현해야 한다.

$$\begin{bmatrix} V^\pi(s_1) \\ \vdots \\ V^\pi(s_n) \end{bmatrix} = \begin{bmatrix} R^\pi(s_1) \\ \vdots \\ R^\pi(s_n) \end{bmatrix} + \gamma \begin{bmatrix} P^\pi_{11}(s_1) \cdots P^\pi_{1n} \\ \vdots \\ P^\pi_{n1}(s_1) \cdots P^\pi_{nn} \end{bmatrix} \begin{bmatrix} V^\pi(s_1) \\ \vdots \\ V^\pi(s_n) \end{bmatrix}$$

그런데 이 연산을 줄여서 다음과 같이 표기하기도 한다.
$V^\pi = R^\pi + \gamma P^\pi V^\pi$

이렇게 보면, 결국 V^π를 이용해서 V^π를 만들어내는 $\mathbb{R}^n \to \mathbb{R}^n$의 구조를 가진다.
그렇다면, $\mathbb{R}^n$에 속해있는 어떤 V에 대해서 다르게 표기할 수 있다.
이를 bellman operator라고 하고, τ로 표기하며, 다음을 만족한다.

$\tau^\pi : \mathbb{R}^n \to \mathbb{R}^n$

즉, Bellman Operator는 한 point에서 다른 point를 $\mathbb{R}^n$에 있는 state value의 벡터 공간을 가지고 표시해 주는 mapping과 동일하다. (애초에 Operator라는 것이 “교환자"의 의미이다.)

τ를 이용해서 bellman optimality equation을 다시 적으면 다음과 같다.

$$V^\pi(s) = \max_{a\in A}(R^a(s) + \gamma \sum_{s'\in S} P(s'|s,a)V^\pi(s'))$$
$$\Rightarrow \tau^*(v) = \max_{a\in A}(\mathcal{R}^a + \gamma\mathcal{P}^a\mathcal{V})$$
$$\Rightarrow \lim_{k\to\infty}(\tau^*)^k v = V_*$$

여기서 가장 마지막 줄로 요약해서 적을 수 있는데,
그 이유는 Bellman Operator가 Contraction의 특성을 가지고 있기 때문이다.

즉, Bellman Operator는 한 point에서 다른 point를 $\mathbb{R}^n$에 있는 state value의 벡터 공간을 가지고 표시해주는 mapping과 동일하다. (애초에 Operator라는 것이 “교환자"의 의미이다.”

Contraction : 수축

RL
어떤 v에서 어떤 policy를 가지고 시작하더라도
무한히 반복하면 해당 policy의 v^π로 수렴하게 된다는 뜻

⇒ Contraction Mapping Theory를 통해 증명되며, Branch Fixed Point Theorem을 봐라...

- Contraction Mapping Theory

> 거리 함수 d를 가지는 거리 공간(metric space) X에서 정의된 함수 φ에 대하여
> $\forall x, y \in X, d(\varphi(x), \varphi(y)) \le c \cdot d(x,y)$을 만족하는 $c < 1$이 존재하면 φ를 축약사상(contraction mapping)이라고 한다.

- Branch Fixed Point Theorm

X가 완비거리공간(complete metric space)이고, φ가 X에서 정의된 축약사상이면 φ는 유일한 고정점을 가진다.

⇒ Bracnfh Fixed Point Theorm은 뻗어나가지 않는 공간에서, 1보다 작은 값들을 반복적으로 곱해줄 경우, 그 값이 하나의 fixed point로 수렴한다는 내용이다.
⇒ 그리고 Bellman Operation이 이 이론을 만족한다.

⇒ 두 Value의 차이는 반드시 하나의 점으로 수렴한다.

Bellman Operator의 Contraction 증명

Policy Iteration as Bellman Operations

그러면 Policy Iteration과 관련해 Bellman Backup을 이용해 설명해보자.

B : Bellman Backup (Operator)로 정의하자.

Policy Iteration은 크게 Policy Evaluation과 Policy Iteration으로 나뉘었다.

- Policy Evaluation
  Bellman Backup을 사용하면 다음과 같다.
  $B^\pi V(s) = R^\pi(s) + \gamma \sum_{s \in S} P^\pi(s'|s)V(s)$

  ⇒ 즉, B^π의 고정점을 찾는 문제와 동일하다.

  ⇒ V가 그만 바뀔 때 까지 B^π를 곱해주겠다
  ⇒ $V^\pi = B^\pi B^\pi \cdots B^\pi V$
  ⇒ 최종적으로 Contraction 후 V^π 결정

- Policy Iteration

  $\pi_{k+1}(s) = \underset{a}{argmax} R(s,a) + \gamma \sum s' \in S \sum P(s'|s,a)V^{\pi_k}(s')$

Value가 항상 수렴하는가?

Contraction Operator

- O : Operator
- $|x|$: norm of x, value function 같은 vector 등, norm은 어떤거든 괜찮다.

O가 Contraction Operator라면, 다음을 만족한다.
$|OV - OV'| \leq |V - V'|$

⇒ 즉, 원래 V와 V'의 차이의 Normalize보다 O를 적용했을 때 작아진다면, 그 O는 Contraction Operator다.

γ가 1보다 작을 때, Bellman Backup이 Contraction Operator이기 때문에, 수축할 수 밖에 없다.

증명

- γ는 항상 1보다 작거나 같다.
- $||V - V'||$이 $max_s ||V(s) - V'(s)||$ 라고 하자.

Step 1 :

$$||BV_k - BV_j||$$
$$= \left|\left| \underset{a}{max} \left(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V_k(s')) \right) - \underset{a'}{max} \left(R(s,a') - \gamma \underset{s' \in S}{P} (s'|s,a') V_j(s') \right) \right|\right|$$

Bellman Operation을 적용한 $||BV_k - BV_j||$를 풀면 아래와 같다.

Step 1 $\leq$ Step2

$$\left|\left| \underset{a}{max} \left(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V_k(s') - R(s,a) - \gamma \sum_{s' \in S} P(s'|s,a) V_j(s') \right) \right|\right|$$
$$= \left|\left| \underset{a}{max} \gamma \sum_{s' \in S} P(s'|s,a) (V_k(s') - V_j(s')) \right|\right|$$

$\leq$인 이유는 V를 최대화하기 위해 a, a'을 각각 구한 값들의 차보다는, 하나의 a만을 사용해 구한 차이가 더 크거나 같기 때문 ($a \neq a'$이면 두 V의 차는 $a = a'$일 때보다 크거나 같다.)

equation 오른쪽은 서로 겹치는 $R(s,a)$를 제거하여 정리하여 성립하며,
항상 $V_k(s') - V_j(s')$이 가장 커지도록 만들겠다는 뜻이다.

Step2 $\leq$ Step3

$$\left|\left| \underset{a}{max} \gamma \sum_{s' \in S} P(s'|s,a) ||V_k - V_j|| \right|\right| = \left|\left| \gamma ||V_k - V_j|| \underset{a}{max} \sum_{s' \in S} P(s'|s,a) \right|\right| = \gamma ||V_k - V_j||$$

정의에 의해서, $||V - V'||$이 $||V(s) - V'(s)||$의 max이기 때문에, 항상 크거나 같다.
첫번째 equation은 V_k와 V_j가 a와 관계 없기 때문에, 앞으로 빼준 것이며
두번째 equation은 $\sum_{s' \in S} P(s'|s,a)$는 항상 1이기 때문에 표기에서 제외했다.

위의 증명을 통해서 $||BV_k - BV_j|| \leq \gamma ||V_k - V_j||$임을 보였다.
따라서 Bellman Backup Operator는 Contraction Operator이며,
Bellman Operator를 적용한 값은 항상 작아져야함을 보였다.