



[지능 시스템 1차 과제]

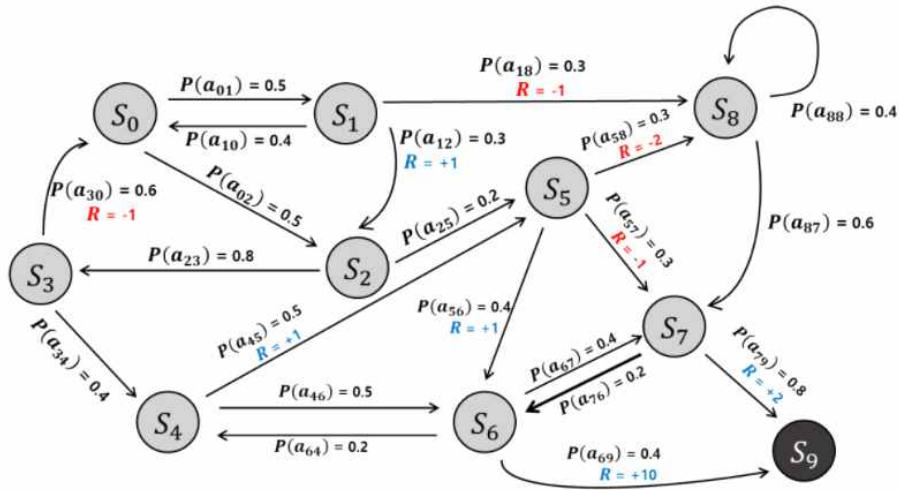
| | |
|--------|---------|
| 과목명 | 지능 시스템 |
| 담당 교수님 | 조영완 교수님 |

| | |
|----|------------|
| 전공 | 컴퓨터공학과 |
| 학번 | 2018305068 |
| 이름 | 정동기 |

| | |
|-----|------------|
| 제출일 | 2021.03.31 |
|-----|------------|

과제1 : 벨만기대방정식 수립 및 풀이

- 다음의 State diagram으로 나타난 MDP에 대해 벨만 기대방정식을 세우고 이를 행렬식을 이용하여 푸시오.
(4월 1일까지 Google classroom으로 제출)



이번 과제의 문제는 위의 다이어그램을 MDP에 대해 벨만기대방정식을 세우고 이를 행렬식을 이용하여 푸는 문제이다.

벨만기대방정식은 현재 상태의 가치함수와 다음 상태의 가치함수와의 관계를 나타내는 방정식이다. 이 방정식에서 기댓값을 얻기 위해선 위에서 설명한 벨만 방정식을 이용하여야 한다.

벨만 방정식은

$$v(s) = \sum_{a \in A} \pi(a | s) (R_{t+1} + \gamma \sum_{s' \in S} P_{ss'}^a v(s'))$$

이런식으로 나타낼 수 있다.

여기서 이 식을 설명하자면 기댓값(s) = 정책(확률)*(보상 + 감가율*상태변환확률*기댓값(s'))이다.

아래는 감가율을 0.9로 두고, 상태변환확률이 1인 위의 다이어그램을 벨만기대방정식에 넣어서 푸는 값이다.

예제 1

$$V(S_0) = P(u_{0.1})(0 + 0.9 \times V(S_1)) + P(u_{0.2})(0 + 0.9 \times V(S_2))$$

$$= 0.5 \times 0.9 \times V(S_1) + 0.5 \times 0.9 \times V(S_2)$$

$$V(S_1) = 0.4 \times 0.9 \times V(S_0) + 0.3(1 + 0.9 \times V(S_2)) + 0.3(-1 + 0.9 \times V(S_8))$$

$$V(S_2) = 0.8 \times 0.9 \times V(S_3) + 0.2 \times 0.9 \times V(S_5)$$

$$V(S_3) = 0.6(-1 + 0.9 \times V(S_0)) + 0.4 \times 0.9 \times V(S_4)$$

$$V(S_4) = 0.5(1 + 0.9 \times V(S_5)) + 0.5 \times 0.9 \times V(S_6)$$

$$V(S_5) = 0.4(1 + 0.9 \times V(S_6)) + 0.3 \times (-1 + 0.9 \times V(S_7)) + 0.3(-2 + 0.9 \times V(S_8))$$

$$V(S_6) = 0.2 \times 0.9 \times V(S_4) + 0.4 \times 0.9 \times V(S_7) + 0.4(10 + 0.9 \times V(S_9))$$

$$V(S_7) = 0.2 \times 0.9 \times V(S_6) + 0.8(2 + 0.9 \times V(S_9))$$

$$V(S_8) = 0.6 \times 0.9 \times V(S_7) + 0.4 \times 0.9 \times V(S_8)$$

$$V(S_9) = 0$$

↓

| | | | | | | | | | | | |
|----------|------|------|------|------|------|------|------|------|------|----------|------|
| $V(S_0)$ | 0 | 0.45 | 0.45 | 0 | 0 | 0 | 0 | 0 | 0 | $V(S_0)$ | 0 |
| $V(S_1)$ | 0.36 | 0 | 0.27 | 0 | 0 | 0 | 0 | 0 | 0.27 | $V(S_1)$ | 0 |
| $V(S_2)$ | 0 | 0 | 0 | 0.72 | 0 | 0.18 | 0 | 0 | 0 | $V(S_2)$ | 0 |
| $V(S_3)$ | 0.54 | 0 | 0 | 0 | 0.36 | 0 | 0 | 0 | 0 | $V(S_3)$ | -0.6 |
| $V(S_4)$ | 0 | 0 | 0 | 0 | 0 | 0.45 | 0.45 | 0 | 0 | $V(S_4)$ | +0.5 |
| $V(S_5)$ | 0 | 0 | 0 | 0 | 0 | 0 | 0.36 | 0.27 | 0.27 | $V(S_5)$ | -0.5 |
| $V(S_6)$ | 0 | 0 | 0 | 0 | 0.18 | 0 | 0 | 0.36 | 0 | $V(S_6)$ | 4 |
| $V(S_7)$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.18 | 0 | $V(S_7)$ | 1.6 |
| $V(S_8)$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.54 | 0.36 | $V(S_8)$ | 0 |

이후 행렬식을 보기 편하게 바꾸어 준 것이 아래 그림이다.

| | | | | | | | | | | | |
|----------|-------|-------|-------|-------|-------|-------|-------|-------|-------|------|---|
| $V(S_0)$ | 1 | -0.45 | -0.45 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $V(S_1)$ | -0.36 | 1 | -0.27 | 0 | 0 | 0 | 0 | 0 | -0.27 | 0 | 0 |
| $V(S_2)$ | 0 | 0 | 1 | -0.72 | 0 | -0.18 | 0 | 0 | 0 | 0 | 0 |
| $V(S_3)$ | -0.54 | 0 | 0 | 1 | -0.36 | 0 | 0 | 0 | 0 | -0.6 | 0 |
| $V(S_4)$ | 0 | 0 | 0 | 0 | 1 | -0.45 | -0.45 | 0 | 0 | 0.5 | 0 |
| $V(S_5)$ | 0 | 0 | 0 | 0 | 0 | 1 | -0.36 | -0.27 | -0.27 | -0.5 | 0 |
| $V(S_6)$ | 0 | 0 | 0 | 0 | -0.18 | 0 | 1 | -0.36 | 0 | 4 | 0 |
| $V(S_7)$ | 0 | 0 | 0 | 0 | 0 | 0 | -0.18 | 1 | 0 | 1.6 | 0 |
| $V(S_8)$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -0.54 | 0.64 | 0 | 0 |

이제 이를 행렬식을 이용하여 풀어야되는데, 여기서는 기댓값 오른쪽에 있는 행렬의 역행렬을 곱해주는 방법을 이용하였다. 여기서 왜 역행렬을 이용하는데 왜 행렬식 이용이 되는 거냐는 이야기가 나올 수 있는데, 역행렬을 수반행렬을 이용하여 풀 때 행렬식 값을 마지막에 나누어 주기 때문이다. 그럼 이제 역행렬을 구하여보자. 역행렬을 구할 때 파이썬을 이용하였는데 파이썬에 있는 numpy함수인 numpy.linalg.inv()함수를 이용하였다.

```
import numpy as np
MDP_array = np.array([[1, -0.45, -0.45, 0, 0, 0, 0, 0, 0],
                      [-0.36, 1, -0.27, 0, 0, 0, 0, 0, -0.27],
                      [0, 0, 1, -0.72, 0, -0.18, 0, 0, 0],
                      [-0.54, 0, 0, 1, -0.36, 0, 0, 0, 0],
                      [0, 0, 0, 0, 1, -0.45, -0.45, 0, 0],
                      [0, 0, 0, 0, 0, 1, -0.36, -0.27, -0.27],
                      [0, 0, 0, 0, -0.18, 0, 1, -0.36, 0],
                      [0, 0, 0, 0, 0, 0, -0.18, 1, 0],
                      [0, 0, 0, 0, 0, 0, 0, -0.54, 0.64]])
# numpy.array를 사용하여 9x9 행렬 생성
MDP_array_inv = np.linalg.inv(MDP_array)
# d_array 행렬의 역행렬 연산
print("행렬의 역행렬\n")
print(MDP_array_inv)
```

아래는 함수를 사용하고 남은 결과값을 깔끔하게 편집해놓은 값이다. 함수를 사용하면 소숫점 8자리까지만 결과가 나오는 것 같다.

```
[[1.62390176 0.73075579 0.92805985 0.66820309 0.2982193 0.30124946 0.32036771 0.43177342 0.43537721]
 [0.75507534 1.3397839 0.70152556 0.5050984 0.23253253 0.23091424 0.2816506 0.52156573 0.66263828]
 [0.631373 0.28411785 1.36082967 0.97979736 0.43017703 0.438529 0.43027764 0.43793077 0.30486664]
 [0.87690695 0.39460813 0.50115232 1.36082967 0.57272765 0.34793486 0.46016096 0.42876093 0.31326032]
 [0. 0. 0. 0. 1.1435812 0.51461154 0.79767334 0.54334246 0.21710174]
 [0. 0. 0. 0. 0.09896175 1.04453279 0.54978748 0.71790497 0.44066227]
 [0. 0. 0. 0. 0.22010759 0.09904841 1.22281993 0.48952271 0.04178605]
 [0. 0. 0. 0. 0.03961937 0.01782871 0.22010759 1.08811409 0.00752149]
 [0. 0. 0. 0. 0.03342884 0.01504298 0.18571578 0.91809626 1.56884626]]
```

이 후 위에서 구한 역행렬을 등호 왼쪽과 오른쪽에 곱하여 주면 왼쪽 등호는 역행렬의 성질의 의해 원래 행렬과 역행렬을 곱하여 1이 되게 되고 기댓값만 있는 행렬이 된다. 오른쪽은 상수 행렬과 역행렬이 곱해지는데 9*1 행렬과 9*9 행렬이 곱해져 9*1 행렬인 값이 나오게 된다, 여기서 numpy함수인 numpy.dot(a, b) 함수를 이용하여 행렬곱을 해주었다.

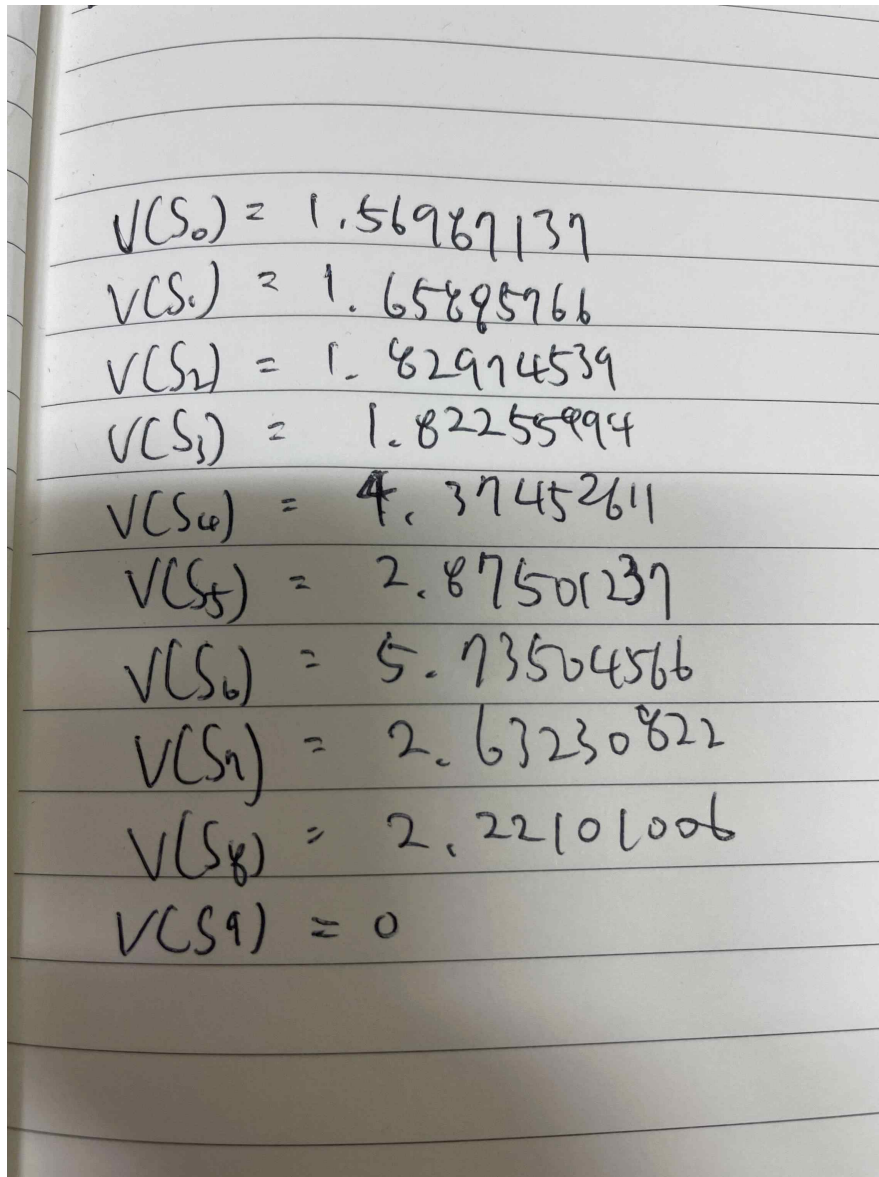
```
#구한 행렬의 왼쪽 오른쪽에 역행렬을 곱함
#왼쪽은V(S)가 남고 오른쪽은 상수행렬과 역행렬의 곱으로 나옴
MDP_nuber_array = np.array([[0], [0], [0], [-0.6], [0.5], [-0.5], [4], [1.6], [0]])
final_array = np.dot(MDP_array_inv, MDP_nuber_array)
print("\nV(S) 값\n")
print(final_array)
```

아래는 함수를 사용하고 남은 결과값이다.

V(S)값

```
[[1.56987137]
 [1.65885766]
 [1.82974539]
 [1.82255994]
 [4.37452611]
 [2.87501237]
 [5.73504566]
 [2.63230822]
 [2.22101006]]
```

그러므로 결과적으로 벨만기대방정식을 이용하여 구한 기댓값은 아래처럼 나오게 된다.



A photograph of a piece of lined paper with handwritten mathematical expressions. The expressions are listed vertically, showing the expected values for states S0 through S9. The handwriting is in black ink on white lined paper. The values are: V(S0) = 1.56987137, V(S1) = 1.65895766, V(S2) = 1.82974539, V(S3) = 1.82255994, V(S4) = 4.37452611, V(S5) = 2.87501237, V(S6) = 5.73504566, V(S7) = 2.63230822, V(S8) = 2.22101006, and V(S9) = 0.

$$\begin{aligned} V(S_0) &= 1.56987137 \\ V(S_1) &= 1.65895766 \\ V(S_2) &= 1.82974539 \\ V(S_3) &= 1.82255994 \\ V(S_4) &= 4.37452611 \\ V(S_5) &= 2.87501237 \\ V(S_6) &= 5.73504566 \\ V(S_7) &= 2.63230822 \\ V(S_8) &= 2.22101006 \\ V(S_9) &= 0 \end{aligned}$$

$V(S_0) \sim V(S_8)$ 의 값은 위에서 구하였고, $V(S_9)$ 는 terminal state여서 아무 일을 하지 않기 때문에 기댓값이 0이 나왔다.