# E3

Reinforcement Learning Study

2021-02-02

## 차 례

## Preparation

[전원 비슷하게 코드를 제출하였습니다.]

```
gamma = 1
states = np.arange(0,80,10).astype('str')
P_normal=pd.DataFrame(np.matrix([[0,1,0,0,0,0,0,0],
                                 [0,0,1,0,0,0,0,0],
                                 [0,0,0,1,0,0,0,0],
                                 [0,0,0,0,1,0,0,0],
                                 [0,0,0,0,0,1,0,0],
                                 [0,0,0,0,0,0,1,0],
                                 [0,0,0,0,0,0,0,1],
                                 [0,0,0,0,0,0,0,1]]), index=states,columns=states)
P_speed=pd.DataFrame(np.matrix([[.1,0,.9,0,0,0,0,0],
                                [.1,0,0,.9,0,0,0,0],
                                [0,.1,0,0,.9,0,0,0],
                                [0,0,.1,0,0,.9,0,0],
                                [0,0,0,.1,0,0,.9,0],
                                [0,0,0,0,.1,0,0,.9],
                                [0,0,0,0,0,.1,0,.9],
                                [0,0,0,0,0,0,0,1]]), index=states, columns=states)
R_s_a=pd.DataFrame(np.array([-1,-1,-1,-1,0.0,-1,-1,0,
                             -1.5,-1.5,-1.5,-1.5,-0.5,-1.5,-1.5,0]).reshape(len(states),2,order='F'),columns
```

## Implementation

**Initialize V**

[v_old 선언방법에서 약간의 차이는 있었습니다(np.repeat, np.zeros)]

---

**교수님 Feedback:**

1번 방식이 가장 좋습니다.

---

1.

**V_old = np.zeros(len(states)).T**

**V_old = pd.DataFrame(V_old,states)**

2.

V_old = np.zeros(states.shape[0]).reshape(states.shape[0],1)

3.

```
V_old=pd.DataFrame(np.repeat(0,len(states)).reshape(len(states),1),index=states)
V_old.T
```

```
##    0  10  20  30  40  50  60  70
## 0  0   0   0   0   0   0   0   0
```

**Evaluate the Q-function**

[열을 합치는 방법이 나뉘었습니다 (np.c, np.hstack)]

————————————————————-

**교수님 Feedback:**

1번 방식이 가장 좋습니다. hstack이 더 구체적임. np.c- alternative way

————————————————————-

**1. (np.hstack)**

**q_s_a=R_s_a+np.hstack((np.dot(gamma*P_normal,V_old),np.dot(gamma*P_speed,V_old)))**

2. (np.c_)

```
q_s_a = R_s_a+np.c_[gamma*np.dot(P_normal,V_old), gamma*np.dot(P_speed,V_old)]
q_s_a
```

```
##     normal  speed
## 0     -1.0   -1.5
## 10    -1.0   -1.5
## 20    -1.0   -1.5
## 30    -1.0   -1.5
## 40     0.0   -0.5
## 50    -1.0   -1.5
## 60    -1.0   -1.5
## 70     0.0    0.0
```

**Find the best action for each state** [전원 동일]

```
V_new=np.matrix(q_s_a.apply(max,axis=1)).reshape(len(states),1)
V_new.T
```

```
## matrix([[-1., -1., -1., -1.,  0., -1., -1.,  0.]])
```

### Value Iteration -Implementation

[반복문 종료하는 if 문에서 np.linalg.norm, np.max(np.abs) 사용차이가 있었습니다]

———————————————————-

**교수님 Feedback:**

np.linalg.norm에 ord라는 argument에 "inf"를 넣으면 np.max(np.abs)가 나옵니다. 그러므로 1번은 이해하기 쉬운 코드이고 2번은 일반화된 코드입니다.equally Good 하다는 생각입니다.

———————————————————-

```
cnt=0
epsilon=10**(-8)
# p.8 like Init V_old, difference between using np.repeat or np.zeros
V_old=pd.DataFrame(np.repeat(0,len(states)).reshape(len(states),1),index=states)
results=V_old.T
while True:
    q_s_a=R_s_a+np.c_[np.dot(gamma*P_normal,V_old),np.dot(gamma*P_speed,V_old)]
    V_new=np.matrix(q_s_a.apply(max,axis=1)).reshape(len(states),1)

    # using linalg.norm function
    # if(np.linalg.norm(V_new-V_old)<epsilon):

    # using np.max(np.abs) function
    if np.max(np.abs(V_new-V_old)).item() < epsilon :
        break

    results=np.r_[results, V_new.T]
    V_old=V_new

    cnt+=1
```

```
# result codes are all the same!
value_iter_process = results
results = pd.DataFrame(results, columns=states)
results.head()
```

```
##      0   10   20   30   40    50   60   70
```

```
## 0  0.0  0.0  0.0  0.0  0.0  0.00  0.0  0.0
## 1 -1.0 -1.0 -1.0 -1.0  0.0 -1.00 -1.0  0.0
## 2 -2.0 -2.0 -1.6 -1.0 -1.0 -1.50 -1.0  0.0
## 3 -3.0 -2.6 -2.0 -2.0 -1.5 -1.60 -1.0  0.0
## 4 -3.6 -3.0 -3.0 -2.5 -1.6 -1.65 -1.0  0.0
```

```
results.tail()
```

```
##            0        10        20        30        40        50   60   70
## 17 -5.107743 -4.410774 -3.441077 -2.666667 -1.666667 -1.666667 -1.0  0.0
## 18 -5.107744 -4.410774 -3.441077 -2.666667 -1.666667 -1.666667 -1.0  0.0
## 19 -5.107744 -4.410774 -3.441077 -2.666667 -1.666667 -1.666667 -1.0  0.0
## 20 -5.107744 -4.410774 -3.441077 -2.666667 -1.666667 -1.666667 -1.0  0.0
## 21 -5.107744 -4.410774 -3.441077 -2.666667 -1.666667 -1.666667 -1.0  0.0
```

## Visualization

[Visualization 코드 전원 동일]

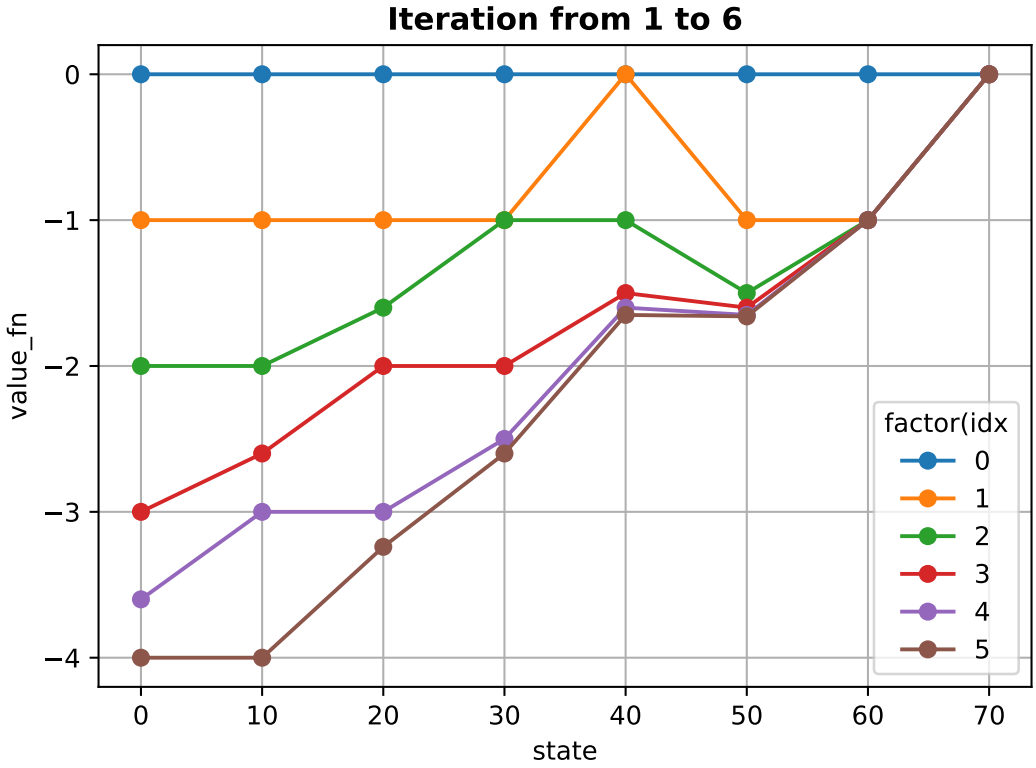
1. Iteration from 6 to 12

```
for i in range(6):
    plt.plot(results.columns,results.iloc[i], label=i,marker='o')

plt.grid(True)
plt.legend(title='factor(idx')
plt.xlabel('state')
plt.ylabel('value_fn')
plt.title('Iteration from 1 to 6', fontweight='bold')
plt.yticks([0,-1,-2,-3,-4])
```

```
## ([<matplotlib.axis.YTick object at 0x000000002DA329B0>, <matplotlib.axis.YTick object at 0x000000002DA32588>,
```

```
plt.show()
```


Iteration from 1 to 6
value_fn
state
0
−1
−2
−3
−4
0
10
20
30
40
50
60
70
factor(idx
0
1
2
3
4
5

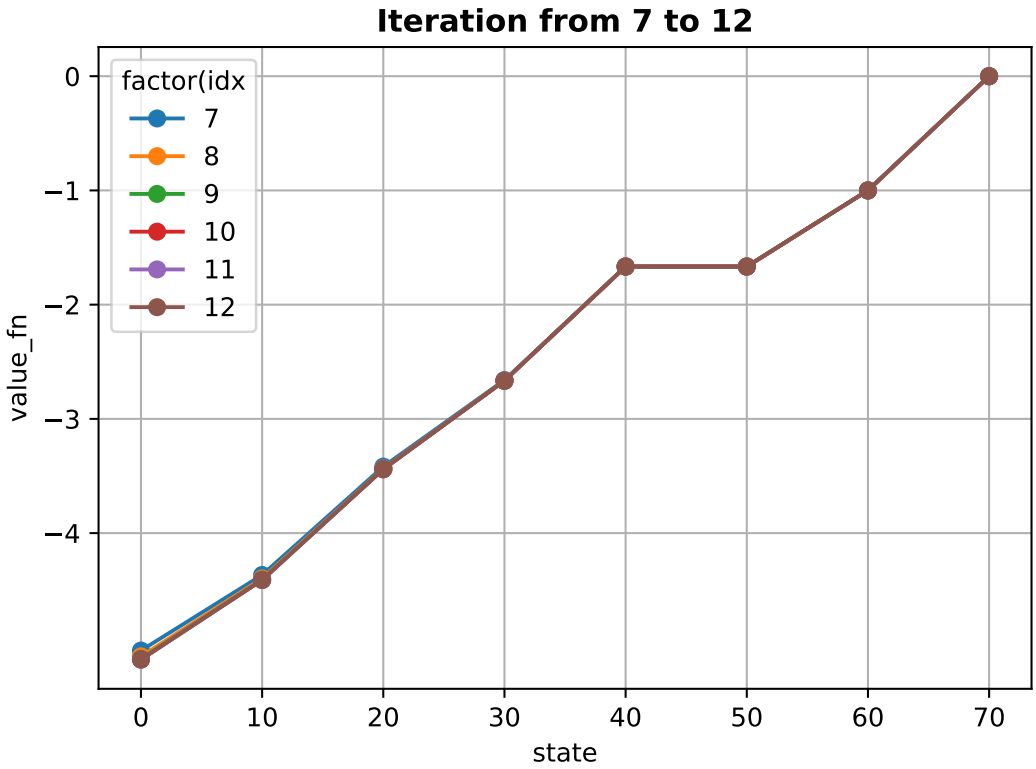
2. Iteration from 7 to 12

```
for i in range(7,13):
    plt.plot(results.columns,results.iloc[i], label=i,marker='o')

plt.grid(True)
plt.legend(title='factor(idx')
plt.xlabel('state')
plt.ylabel('value_fn')
plt.title('Iteration from 7 to 12', fontweight='bold')
plt.yticks([0,-1,-2,-3,-4])
```

```
## ([<matplotlib.axis.YTick object at 0x000000002DB7EFD0>, <matplotlib.axis.YTick object at 0x000000002DB7EBA8>,
```

```
plt.show()
```



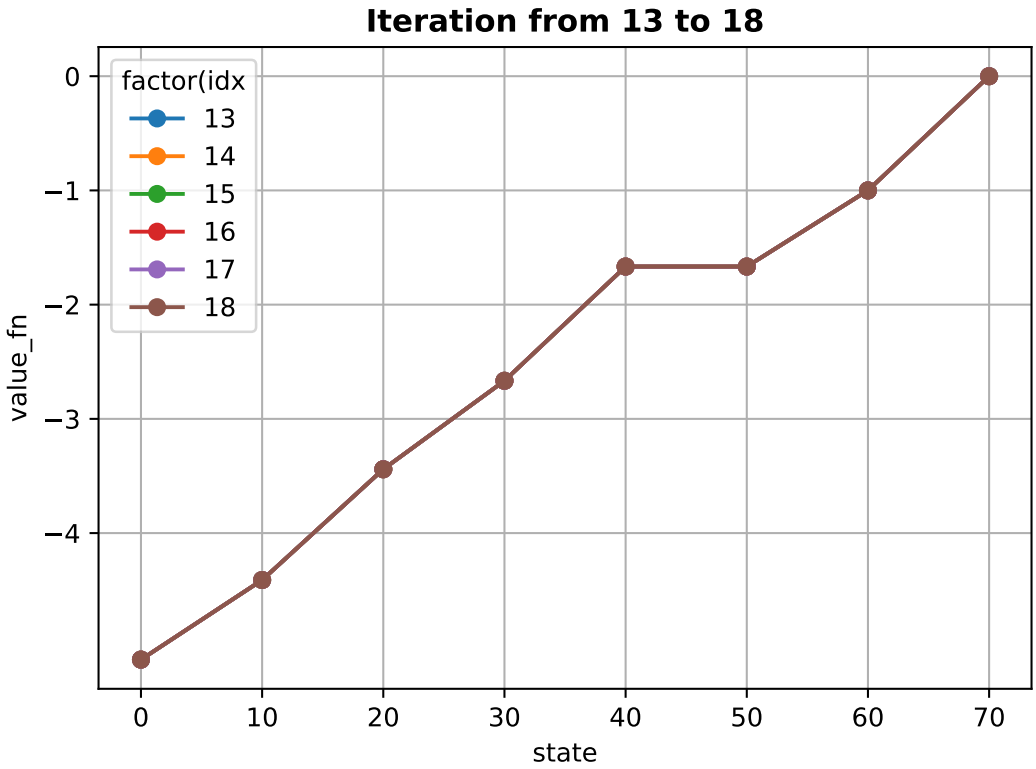
3. Iteration from 13 to 18

```
for i in range(13,19):
    plt.plot(results.columns,results.iloc[i], label=i,marker='o')

plt.grid(True)
plt.legend(title='factor(idx')
plt.xlabel('state')
plt.ylabel('value_fn')
plt.title('Iteration from 13 to 18', fontweight='bold')
plt.yticks([0,-1,-2,-3,-4])
```

```
## ([<matplotlib.axis.YTick object at 0x000000002DA3F3C8>, <matplotlib.axis.YTick object at 0x000000002DA3F898>,
```

```
plt.show()
```



### Optimal Value function → Optimal policy

[코드 전원 비슷합니다]

```
V_opt=results.tail(1).T
V_opt.T
```

```
##            0        10        20        30        40        50   60   70
## 21 -5.107744 -4.410774 -3.441077 -2.666667 -1.666667 -1.666667 -1.0  0.0
```

```
q_s_a=R_s_a+np.c_[np.dot(gamma*P_normal,V_opt), np.dot(gamma*P_speed, V_opt)]
q_s_a
```

```
##       normal     speed
## 0  -5.410774 -5.107744
## 10 -4.441077 -4.410774
## 20 -3.666667 -3.441077
## 30 -2.666667 -3.344108
## 40 -1.666667 -1.666667
## 50 -2.000000 -1.666667
## 60 -1.000000 -1.666667
## 70  0.000000  0.000000
```

```
pi_opt_vec=q_s_a.idxmax(axis=1)
pi_opt_vec
```

```
## 0      speed
## 10     speed
## 20     speed
## 30    normal
## 40    normal
## 50     speed
## 60    normal
## 70    normal
## dtype: object
```

```
pi_opt=pd.DataFrame(np.zeros((len(states),2)), index=states, columns=['normal','speed'])
for i in range(len(pi_opt_vec)):
    pi_opt.iloc[i][pi_opt_vec[i]]=1

pi_opt.T
```

```
##           0   10   20   30   40   50   60   70
## normal  0.0  0.0  0.0  1.0  1.0  0.0  1.0  1.0
## speed   1.0  1.0  1.0  0.0  0.0  1.0  0.0  0.0
```