기초 인공지능(CSE4185) Assignment #6

2분반 20200183 박정원

1. 각 함수마다 구현한 방법에 대한 간략한 설명

```python
def OOB(x, y, M, N): # check if the coordinates out of bound, returns True if it is, False otherwise.
    if x < 0 or x >= M or y < 0 or y >= N:
        return True
    else:
        return False

dx = [0,-1,0,1] # left, up, right, down
dy = [-1,0,1,0] # left, up, right, down
```

각 함수에 대한 구현을 설명하기 이전에, 전역에 그리드에서 벗어난 여부를 판단하는 OOB 함수와 네 가지 action에 대한 x 좌표와 y 좌표의 변화를 담는 dx와 dy를 선언하였고, 이는 과제에서 부여된 순서와 동일하게 left(0), up(1), right(2), down(3)으로 하였다.

1) Transition matrix

```python
def compute_transition_matrix(model):
    """
    Parameters:
    model - the MDP model returned by load_MDP()

    Output:
    P - An M x N x 4 x M x N numpy array. P[r, c, a, r', c'] is the probability that the agent will move from cel
    """
    # raise RuntimeError("You need to write this part!")
    M, N = model.M, model.N
    P = np.zeros([M, N, 4, M, N])

    for i in range(M):
        for j in range(N):
            if model.T[i, j] == True: # if the state is the terminal state, make the transition probability zero
                P[i,j,:,:,:] = 0
                continue
            for a in range(4):
                # intended direction
                nx = i + dx[a]
                ny = j + dy[a]
                if OOB(nx, ny, M, N) or model.W[nx, ny]: # if next state is OOB or wall
                    P[i,j,a,i,j] += model.D[i,j,0] # probability is added to the same state
                else:
                    P[i,j,a,nx,ny] += model.D[i,j,0] # probability is added to the next state
                # unintended counter-clockwise
                nx = i + dx[(a+3)%4]
                ny = j + dy[(a+3)%4]
                if OOB(nx, ny, M, N) or model.W[nx, ny]: # if next state is OOB or wall
                    P[i,j,a,i,j] += model.D[i,j,1] # # probability is added to the same state
                else:
                    P[i,j,a,nx,ny] += model.D[i,j,1] # probability is added to the next state
                # unintended clockwise
                nx = i + dx[(a+1)%4]
                ny = j + dy[(a+1)%4]
                if OOB(nx, ny, M, N) or model.W[nx, ny]: # if next state is OOB or wall
                    P[i,j,a,i,j] += model.D[i,j,2] # probability is added to the same state
                else:
                    P[i,j,a,nx,ny] += model.D[i,j,2] # probability is added to the next state

    return P
```

먼저, 주어진 grid world 환경의 모든 state에 대해서 for 문으로 돌면서, transition probability를 계산하여야 하고, 네 가지의 의도한 action에 대한 다음 state으로의 확률을 만들어 나가면 된다. 따라서, transition matrix는 (m,n,4,m,n)의 shape을 갖는다. 중요한 것은 각각의 의도한 방향에 대해서 의

도한 방향의 반시계 방향과 시계 방향으로 또한 고려하여 해당 entry에 확률을 더해주어야 한다는 것이다. 간단히 예를 들어서, 현재 state에서 의도한 방향이 아래인데, 아래는 grid 범위 밖이고, 오른쪽은 벽으로 막혀있다면, 해당 action을 취했을 때 다음 state는 현재 state와 같은 자리이므로, 현재 state에 두 확률을 모두 더한 것이 확률이 될 것이다. 이러한 사실을 고려하여 1. 의도한 방향 2. 의도한 방향의 반시계 방향 3. 의도한 방향의 시계 방향을 고려하여 확률을 더해주는데, 각각 범위에서 벗어나거나 벽을 만나게 되면 현재 state의 좌표와 동일한 곳으로, 아니면 현재 state의 좌표에서 각각 dx와 dy를 더한 좌표로의 확률에 해당 확률을 더해주면 된다. 현재 의도한 방향이 a 라면, 의도한 방향의 반시계 방향은 (a+3)%4이고, 의도한 방향의 시계 방향은 (a+1)%4이다. 의도한 반대 방향으로는 가지 않기 때문에 해당 방향의 확률은 0이 될 것이고, 후에 update_utility를 통해 s prime 상태에 대해 곱해줄 때 0 으로 곱해지기에 구현에 있어 상당히 편하다. (예외 처리를 굳이 하지 않아도 되기 때문이다.)

예외적으로, 현재 state가 terminal state라면, 해당 state의 transition probability는 모두 0으로 설정하고, continue 하도록 구현하면 된다.

2) Update utility

```python
def update_utility(model, P, U_current):
    """
    Parameters:
    model - The MDP model returned by load_MDP()
    P - The precomputed transition matrix returned by compute_transition_matrix()
    U_current - The current utility function, which is an M x N array

    Output:
    U_next - The updated utility function, which is an M x N array
    """
    U_next = np.zeros_like(U_current) # U_next same shape as U_current
    # raise RuntimeError("You need to write this part!")
    M, N = model.M, model.N
    for i in range(M):
        for j in range(N):
            Max = -10000000 # initialize with very small value
            for a in range(4): # to pull out the max out of all actions
                # intended direction
                Sum = 0
                # stay at the same spot
                Sum += (P[i,j,a,i,j] * U_current[i,j])

                # check all four directions
                for s_p in range(4):
                    nx = i + dx[s_p]
                    ny = j + dy[s_p]
                    if OOB(nx,ny, M, N) or model.W[nx, ny]: # if the next state is OOB
                        continue
                    Sum += (P[i,j,a,nx,ny] * U_current[nx,ny]) # even if the direction is 180 degree opposite of intended action, transition probability is 0, resulting in 0 addition, so it is fine.

                if Sum > Max: # renew the max value
                    Max = Sum
            U_next[i, j] = model.R[i,j] + model.gamma * Max # update the utility of that state

    return U_next
```

Update_utility 함수는 과제 PDF에서 나온 최종 수식을 그대로 참고하여 구현했다.

최종 수식)

$$U_{i+1}(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U_i(s')$$

이 수식을 해석하면, 네 가지의 각각의 action에 대해서 시그마를 통하여 값을 더해줄 것인데, 이때 s'은 다음 상태를 의미하는 것으로, 현재 상태에서 나올 수 있는 가능한 다음 상태는 총 4 가지이다. 현재 상태에 그대로 머무는 경우, 의도한 방향으로 이동하는 경우, 의도한 방향의 반시계 방향으로 이동하는 경우, 의도한 방향의 시계방향으로 이동하는 경우이다. 반대 방향은 이동하지 않으므로, 고려하지 않지만, 해당 방향까지 포함해도 상관 없는 이유는 transition matrix에서 해당

방향에 대한 확률은 0으로 처리되어 있을 것이기 때문에 (zeros로 초기화) U(s')와 곱할 때 0으로 처리되므로 영향을 주지 못한다. 예외 처리의 수고로움을 덜기 위해 해당 방향까지 포함하여 for 문을 사용하여 구현했다. 구현할 때에는 현재 상태에 머무르는 경우에 대한 값을 먼저 Sum에 더해주고 시작했다. 그 이유는 나중에 네 방향에 대한 체크를 할 때, 범위 밖이거나 벽인 경우에는 현재 상태를 더하는데, 두 번 더하는 것을 방지하기 위해서이다. For 문 안에서는 범위 밖이거나 벽인 경우 continue를 통해서 앞서 이미 더해준 현재 상황에 대한 확률을 두 번 더하는 것을 방지한다. 만약 세 가지 방향이 모두 이동 가능한 상태라면 현재 상태에 대한 확률을 더 하면 안되는 것 아니냐는 생각을 할 수 있지만, 그렇다면 transition matrix에서 이미 0으로 초기화되어 있을 것이기에 상관이 없다.

3) Value iteration

update_utility 함수를 반복적으로 호출하면서 모든 state에 대해 $|U_{i+1}(s) - U_i(s)| < \epsilon$이면, 수렴되었다고 판단하여 iteration을 종료하도록 **value_iteration** 함수를 구현한다.
transition probability P는 **value_iteration** 함수 내에서 **compute_transition_matrix** 함수를 호출하여 사용한다. iteration은 수렴되지 않아도 최대 100회로 제한한다. ($\epsilon = 1e-3$로 코드 내 정의 되어 있음)

Value iteration 역시 과제 PDF의 설명 그대로를 참고하여 구현했다.

```python
def value_iteration(model):
    """
    Parameters:
    model - The MDP model returned by load_MDP()

    Output:
    U - The utility function, which is an M x N array
    """
    P = compute_transition_matrix(model) # calculate the transition probabiltiy
    M, N = model.M, model.N
    U_current = np.zeros([M,N])
    U_next = np.zeros_like(U_current) # U_next same shape as U_current
    iter_num = 0 # track the number of iteration
    converged = False # check if it is converged
    while iter_num < 100 and converged == False: # repeat until the iteration num is smaller than 100 and it is not converged yet
        U_next = update_utility(model, P, U_current) # update the utility
        check = True
        for i in range(M):
            for j in range(N):
                if abs(U_next[i,j] - U_current[i,j]) >= epsilon: # if one of the state's difference in utility is greater than or equal to epsilon, break instantly
                    check = False
                    break
            if check == False:
                break

        if check == True: # if any of the state's difference in utility is greater than or equal to epsilon, it means it is converged, so terminate the loop
            converged = True

        U_current = U_next
        iter_num += 1
    # raise RuntimeError("You need to write this part!")
    return U_next # return the result of the value iteration
```

Value iteration의 종료 조건으로 다음 utility와 현재 utility 의 모든 상태의 차이의 절댓값이 epsilon 보다 작거나, 즉 수렴하거나, 수렴되지 않아도 100번 iteration을 한 경우 종료하도록 하면 된다. 이는 while 문을 통해서 구현했고, iter_num이라는 변수를 두어 한번의 iteration 마다 1씩 증가시키고, converged 라는 변수를 두어 만약 모든 update에 대해서 모든 상태에 대해 수렴되었는지 확인하고 만약 그렇다면 converged 변수를 True로 바꾸어주어 while 문의 조건에 걸려 빠져나오도록 하였다.
모든 상태에 대한 수렴 판단 여부는 2중 for 문으로 모든 상태를 돌면서, epsilon과 U_next와

U_current의 차의 절댓값을 비교해보면 되고, epsilon 보다 크거나 같으면 수렴되지 않았다는 뜻이므로, 더이상 다른 상태를 체크할 필요없이 check를 False로 바꾸고 break로 빠져나와 또 한번 iteration을 하도록 한다. 만약 모든 상태가 수렴 상태라면, check 변수가 True로 유지되어 있을 것이고, 해당 경우에 converged를 True로 바꾸어 주어 while 문을 빠져나오도록 할 수 있다. U_next라는 U_current와 shape이 같은 변수를 두어 update_utility의 결과 값을 받아오고, 해당 U_next는 다음 iteration의 U_current가 되므로 while 문의 마지막에 U_current = U_next를 해주는 식으로 구현했다.

2. 실행 결과 후 저장된 사진 첨부

[그림 6] 초기화된 Utility 한 번 업데이트

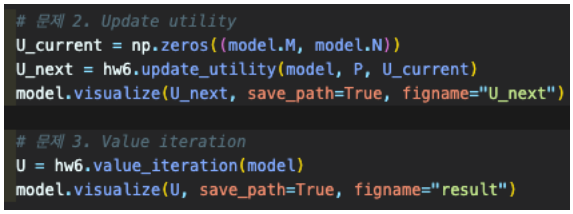
-0.040	-0.040	-0.040	1.000
-0.040	x	-0.040	-1.000
-0.040	-0.040	-0.040	-0.040

[그림 8] 최종 utility 업데이트 결과

0.812	0.868	0.918	1.000
0.762	x	0.660	-1.000
0.705	0.655	0.611	0.387

```
~/Desktop/AI6 python main.py
2023-12-06 11:16:40.283 python[96819:4945795] +[CATransaction synchronize] called within transaction
2023-12-06 11:16:41.030 python[96819:4945795] +[CATransaction synchronize] called within transaction
2023-12-06 11:16:58.717 python[96819:4945795] +[CATransaction synchronize] called within transaction
2023-12-06 11:16:58.837 python[96819:4945795] +[CATransaction synchronize] called within transaction
```

위 두 사진 모두, 제 코드를 실행, visualize 하여 나온 화면의 저장 기능을 활용하여 저장한 png 이미지입니다.



```python
# 문제 2. Update utility
U_current = np.zeros((model.M, model.N))
U_next = hw6.update_utility(model, P, U_current)
model.visualize(U_next, save_path=True, figname="U_next")

# 문제 3. Value iteration
U = hw6.value_iteration(model)
model.visualize(U, save_path=True, figname="result")
```