Finite Markove Decision Processes

February 15, 2023

In general, we seek to maximize the *expected return*, where the return G_t is defined as some specific function of the reward sequence.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{1}$$

A state signal that succeeds in retaining all relevant information is said to be *Markov*, or to have the *Markov property*. For example, the current position and velocity of a cannonball is all that matters for its future flight, It doesn't matter how that position and velocity came about. All that matters is in the current state signal.

$$p(s', r|s, a) = Pr\{R_{t+1} = r, S_{t+1} = s'|S_t, A_t\} \tag{2}$$

The probability of $S_{t+1} = s'$ and $R_{t+1} = r$ given $S_t = s$ and $A_t = a$. These quantities completely specify the dynamics of a finite MDP. One can compute **anything** one might want to know about the environment. The expected rewards for state-action pairs:

$$r(s, a) = E[R_{t+1}|S_t = s, A_t = a] = \sum_{r \in R} r \sum_{s' \in S} p(s', r|s, a) \tag{3}$$

The state-transition probabilities:

$$p(s'|s, a) = Pr\{S_{t+1} = s'|S_t = s, A_t = a\} = \sum_{r \in R} p(s', r|s, a) \tag{4}$$

The expected rewards for state-action-next-state triples:

$$r(s, a, s') = E[R_{t+1}|S_t = s, A_t = a, S_{t+1} = s'] = \frac{\sum_{r \in R} rp(s', r|s, a)}{p(s'|s, a)} \tag{5}$$

A policy π, is a mapping from each state $s \in S$ and action $a \in A(s)$, to the probability $\pi(a|s)$ of taking action a when in state s. $\pi(a|s)$ - the probability of taking action a when in state s $v_\pi(s)$ - the expected return when starting in s and following π thereafter. (*state-value function for policy* π)

$$v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s] \tag{6}$$

Note: the value of the terminal state, if any, is always zero. $q_\pi(s, a)$ - the expected return starting from s, taking the action a, and thereafter following policy π. (*action-value function for policy* π)

$$q_\pi(s, a) = E_\pi[G_t|S_t = s, A_t = a] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a] \tag{7}$$

Review: $E[X] = \sum_{i=0}^{\infty} x_i p_i$, $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$ The following consistency condition holds between the value of s and the value of its possible successor states:

$$v_\pi(s) = \sum_{a,r,s'} \pi(a|s)p(s',r|s,a)[r + \gamma v_\pi(s')] \tag{8}$$

This equation is called the **Bellman Equation**. By solving system of linear equation, we can get the value-function $v_\pi(s)$ of each state s under policy π.

1 Grid World

page 60, example 3.5 There are a total of 25 states, 4 actions and $\pi(a|s) = 0.25$ for all state.

[1]:
```
import numpy as np

n_row = 5
n_col = 5
n_state = n_row * n_col # (i, j) for i in range(5) for j in range(5)
n_action = 4 # 0: up, 1: down, 2: left, 3: right
n_reward = 4 # 0: -1, 1: 0, 2: 5, 3: 10
policy = 0.25
gamma = 0.9
state_A = 0 * n_col + 1
state_A_ = 4 * n_col + 1
state_B = 0 * n_col + 3
state_B_ = 2 * n_col + 3

table = np.zeros([n_state, n_reward, n_state, n_action])
for s in range(n_state):
    if s == state_A:
        table[state_A_, 3, state_A, :] = 1
        continue
    elif s == state_B:
        table[state_B_, 2, state_B, :] = 1
        continue
    row = s // n_col
    col = s % n_row
    for a in range(n_action):
        if a == 0: # go up
            if row == 0:
                table[s, 0, s, 0] = 1
            else:
                s_ = (row - 1) * n_col + col
                table[s_, 1, s, 0] = 1
        elif a == 1: # go down
            if row == n_row - 1:
                table[s, 0, s, 1] = 1
            else:
                s_ = (row + 1) * n_col + col
```

```
                table[s_, 1, s, 1] = 1
        elif a == 2: # go left
            if col == 0:
                table[s, 0, s, 2] = 1
            else:
                s_ = s - 1
                table[s_, 1, s, 2] = 1
        elif a == 3: # go right
            if col == n_col - 1:
                table[s, 0, s, 3] = 1
            else:
                s_ = s + 1
                table[s_, 1, s, 3] = 1
```

$$\begin{bmatrix} v_\pi(s_1) \\ v_\pi(s_2) \\ v_\pi(s_3) \\ \dots \\ v_\pi(s_{25}) \end{bmatrix} = \begin{bmatrix} \sum_{a,r,s'} \pi(a|s_1)p(s',r|s_1,a)r \\ \sum_{a,r,s'} \pi(a|s_2)p(s',r|s_2,a)r \\ \sum_{a,r,s'} \pi(a|s_3)p(s',r|s_3,a)r \\ \dots \\ \sum_{a,r,s'} \pi(a|s_{25})p(s',r|s_{25},a)r \end{bmatrix} + \begin{bmatrix} \sum_{a,r} \pi(a|s_1)p(s_1,r|s_1,a) & \sum_{a,r} \pi(a|s_1)p(s_2,r|s_1,a) & \dots & \sum \\ \sum_{a,r} \pi(a|s_2)p(s_1,r|s_2,a) & \sum_{a,r} \pi(a|s_2)p(s_2,r|s_2,a) & \dots & \sum \\ \sum_{a,r} \pi(a|s_3)p(s_1,r|s_3,a) & \sum_{a,r} \pi(a|s_3)p(s_2,r|s_3,a) & \dots & \sum \\ \dots & & & \\ \sum_{a,r} \pi(a|s_{25})p(s_1,r|s_{25},a) & \sum_{a,r} \pi(a|s_{25})p(s_2,r|s_{25},a) & \dots & \sum_{a,} \end{bmatrix}$$

$$A = I - \gamma \begin{bmatrix} \sum_{a,r} \pi(a|s_1)p(s_1,r|s_1,a) & \sum_{a,r} \pi(a|s_1)p(s_2,r|s_1,a) & \dots & \sum_{a,r} \pi(a|s_1)p(s_{25},r|s_1,a) \\ \sum_{a,r} \pi(a|s_2)p(s_1,r|s_2,a) & \sum_{a,r} \pi(a|s_2)p(s_2,r|s_2,a) & \dots & \sum_{a,r} \pi(a|s_2)p(s_{25},r|s_2,a) \\ \sum_{a,r} \pi(a|s_3)p(s_1,r|s_3,a) & \sum_{a,r} \pi(a|s_3)p(s_2,r|s_3,a) & \dots & \sum_{a,r} \pi(a|s_3)p(s_{25},r|s_3,a) \\ \dots & & & \\ \sum_{a,r} \pi(a|s_{25})p(s_1,r|s_{25},a) & \sum_{a,r} \pi(a|s_{25})p(s_2,r|s_{25},a) & \dots & \sum_{a,r} \pi(a|s_{25})p(s_{25},r|s_{25},a) \end{bmatrix}$$

$$B = \begin{bmatrix} \sum_{a,r,s'} \pi(a|s_1)p(s',r|s_1,a)r \\ \sum_{a,r,s'} \pi(a|s_2)p(s',r|s_2,a)r \\ \sum_{a,r,s'} \pi(a|s_3)p(s',r|s_3,a)r \\ \dots \\ \sum_{a,r,s'} \pi(a|s_{25})p(s',r|s_{25},a)r \end{bmatrix}$$

[15]:
```
matrix_A = np.zeros((n_state, n_state))
for s in range(n_state):
    for s_ in range(n_state):
        val = 0
        for r in range(n_reward):
            for a in range(n_action):
                val += policy * table[s_, r, s, a]
        matrix_A[s, s_] = val
matrix_A = np.eye(n_state) - gamma * matrix_A
matrix_B = np.zeros((n_state, 1))
for s in range(n_state):
```

```
    val = 0
    for s_ in range(n_state):
        for r in range(n_reward):
            for a in range(n_action):
                if r == 0:
                    val += policy * table[s_, r, s, a] * (-1)
                elif r == 1:
                    val += policy * table[s_, r, s, a] * 0
                elif r == 2:
                    val += policy * table[s_, r, s, a] * 5
                elif r == 3:
                    val += policy * table[s_, r, s, a] * 10
    matrix_B[s, 0] = val
x = np.linalg.solve(matrix_A, matrix_B) # solve system of linear equations
grid_world = np.zeros((n_row, n_col))
for i in range(n_state):
    row = i // n_col
    col = i % n_row
    grid_world[row, col] = x[i, 0]
print(grid_world)
```

```
[[ 3.30899634  8.78929186  4.42761918  5.32236759  1.49217876]
 [ 1.52158807  2.99231786  2.25013995  1.9075717   0.54740271]
 [ 0.05082249  0.73817059  0.67311326  0.35818621 -0.40314114]
 [-0.9735923  -0.43549543 -0.35488227 -0.58560509 -1.18307508]
 [-1.85770055 -1.34523126 -1.22926726 -1.42291815 -1.97517905]]
```

A policy π is defined to be better than or equal to a policy π' if its expected return is greater than or equal to that of π' for all states. In other words, $\pi >= \pi'$ if and only if $v_\pi(s) >= v_{\pi'}(s)$ for all $s \in S$. π_* - all the optimal policies. $v_*(s)$ - optimal state-value function $q_*(s,a)$ - optimal action-value function

However, in reality, we can rarely solve the optimal policy due to time and space complexity of the problem. We can only approximate.

[ ]: