ASSIGNMENT 2

Soheil Shirvani 862465192

Problem 1. Consider a Markov decision process with 5 states s_0, s_1, s_2, s_3, s_4 and 2 actions a_0, a_1. The reward function and transition probability are given as follows

$$r(s,a) = \begin{bmatrix} 0 & 0.2 \\ 0 & 0.2 \\ 0 & 0.2 \\ 0 & 0.2 \\ 1 & 0.2 \end{bmatrix} \quad p(s'|s,a_0) = \begin{bmatrix} 0 & 0.8 & 0.2 & 0 & 0 \\ 0 & 0 & 0.8 & 0.2 & 0 \\ 0 & 0 & 0.2 & 0.8 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad p(s'|s,a_1) = \begin{bmatrix} 0.9 & 0.1 & 0 & 0 & 0 \\ 0.9 & 0.1 & 0 & 0 & 0 \\ 0.9 & 0 & 0.1 & 0 & 0 \\ 0.9 & 0 & 0.1 & 0 & 0 \\ 0.9 & 0 & 0.1 & 0 & 0 \end{bmatrix}$$

Each cell in the $r(s,a)$ table is the reward for that state-action pair. For example, $r(s_0, a_1) = 0.2$. Each row of the two p tables is the probability distribution of s' given an state-action pair. For example, $p(\cdot|s_0, a_0) = [0, 0.8, 0.2, 0, 0]$. The initial state distribution is $[1, 0, 0, 0, 0]$, that is, the initial state is always s_0. There is no terminal state for the MDP. Finally, the discount factor is 0.95.

Find the optimal Q values for all state-action pairs using dynamic programming and Q learning with epsilon greedy exploration. Compare your results obtained by the two methods. The programming language is up to you.

Dynamic programming implementation:

```
# Discount factor
gamma = 0.95

# Value iteration algorithm
iterations = 1000  # Maximum number of iterations
threshold = 1e-6  # Convergence threshold

# Initialize Q-values to zero
Q = np.zeros((num_states, num_actions))

for i in range(iterations):
    Q_prev = Q.copy()
    for s in range(num_states):
        for a in range(num_actions):
            if a == 0:
                Q[s, a] = np.sum(transition_a0[s] * (reward[s, a] + gamma * np.max(Q_prev, axis=1)))
            else:
                Q[s, a] = np.sum(transition_a1[s] * (reward[s, a] + gamma * np.max(Q_prev, axis=1)))
    if np.max(abs(Q - Q_prev)) < threshold:
        break

    # if i >= 3:
    #     exit()

print(Q)
```

Q Learning Implementation:

```
iterations = 1000000  # Maximum number of iterations

Q = np.zeros((num_states, num_actions))

# Simulate the process of Q-learning
for i in range(iterations):
    # Start from a random state
    s = np.random.choice(num_states)

    # Choose action using epsilon-greedy policy
    if np.random.rand() < epsilon:
        a = np.random.choice(num_actions)
    else:
        a = np.argmax(Q[s])

    # Simulate taking action 'a' in state 's' and landing in a new state 's_prime' with probability of state transition
    if a == 0:
        s_prime = np.random.choice(num_states, p=transition_a0[s])
    else:
        s_prime = np.random.choice(num_states, p=transition_a1[s])

    # Receive the reward for the transition
    r = reward[s, a]

    # Q-learning update
    Q[s, a] = Q[s, a] + alpha * (r + gamma * np.max(Q[s_prime]) - Q[s, a])

print(Q)
```

Results:

Dynamic programming State Values:

dynamic_programming = [[7.87481849 , 7.88077864],
[9.92330339 , 9.87522034],
[12.53470385, 12.4213358],
[15.83331496, 15.63748163],
[19.99998163, 19.19998163]]

Q Learning State Values (100000 iteration):

q_learning = [[8.47847407, 8.09561857],
[9.73689351, 9.52777817],
[11.65207207, 12.61854581],
[16.28098114, 15.21584888],
[20. , 19.2]]

Conclusion:

The results from the dynamic programming and Q-learning approaches exhibit noteworthy differences in the computation of state values, underlining the distinctions between these methods. Dynamic programming, by leveraging a model of the environment's transitions, is able to compute the optimal state values directly. This is evident in the relatively stable and consistent values obtained across states, such as the value of 19.99998163 in the final state, compared to 20.0 in Q-learning. In contrast, Q-learning, employing an epsilon-greedy policy, iteratively approximates the state values based on sampled experiences. This method does not require a known transition model and instead relies on the exploration (through epsilon-greedy actions) and exploitation of gathered experiences. Consequently, the values derived from Q-learning, such as 20.0 in the final state, are achieved after 100,000 iterations, indicating a convergence towards an optimum soft policy rather than the absolute optimum.

Q-learning and dynamic programming represent two fundamental approaches in reinforcement learning, each with its unique mechanism and implications. Dynamic programming assumes complete knowledge of the environment’s dynamics and computes the exact values needed for an optimal policy through a process known as value iteration. This process iteratively updates the values of each state until they converge to the optimal values, as seen in the provided dynamic programming results. On the other hand, Q-learning is a model-free approach that estimates the values of state-action pairs through experiences. It does not assume prior knowledge of the environment's dynamics and adjusts its estimates based on the outcomes of actions taken, guided by an epsilon-greedy policy. This policy mixes exploration of new actions with the exploitation of known rewarding actions, which is crucial for learning optimal behavior in unknown environments. With fewer iterations or higher epsilon values, Q-learning may not converge to the optimal values, reflecting in less consistent or lower state values compared to those from dynamic programming.

In practice, the choice between Q-learning and dynamic programming depends on the availability of model information and the computational resources. While dynamic programming can swiftly reach optimal solutions with known model transitions, Q-learning offers flexibility and adaptability in environments where such information is absent or incomplete. Over time, and with sufficient iterations, Q-learning can converge to produce robust policy decisions, albeit typically requiring more computational time to explore and learn from the environment effectively.

Codes are attached to my submission in zipfile.