Exercise 1

Breno Aberle - 876438
ELEC-E8125 - Reinforcement Learning

September 28, 2020

1 Question 1

In that Reinforcement Learning exercise the agent is the sailor. The sailor wants to reach the harbour because there is a positive reward. The agent learns with every episode. Given the current state it has to take an action. The agent can take four actions: left, down, right, up. The wind probability, effects of wind, and exact location of the harbour is perfectly known to the agent.

The environment is the whole cell grid with harbour, rocks, water with less wind and water with more wind. The harbour has a reward of +10, rocks -2 and episode termination and the rest a reward of 0. The sea has a low wind probability, while the probability of wind in the narrow passage between the rocks higher is.

2 Task 1

The following code represents the value iteration:

```
gamma = 0.9 # discount factor value
number_iterations = 100
actions = [0,1,2,3]
value_est_history = []
policy_history = []

for i in range(number_iterations):
    env.clear_text() # remove previously drawn values
    value_est_copy = value_est.copy()
    policy_copy = policy.copy()
    value_est_history.append(value_est_copy)
    policy_history.append(policy_copy)

    for x in range(env.w): # loop through all states/cell in
        the environment
        for y in range(env.h):
            state_values_of_actions = [] # keep track of
                calculated state values of each action
            for action in actions:
                transitions = env.transitions[x,y,action]
                state_value_of_action = 0
                for transition in transitions:
                    state_next = transition[0]
                    reward = transition[1]
                    done = transition[2]
                    probability = transition[3]
                    if (state_next == None):
                        state_value_of_action += 0
                    else:
                        state_value_of_action += probability *
                            (reward + gamma * value_est_copy[
                            state_next])
                state_values_of_actions.append(
                    state_value_of_action)
            value_est[x][y] = np.max(state_values_of_actions)
            policy[x][y] = np.argmax(state_values_of_actions)

    max_diff_val = np.max(abs(abs(value_est) - abs(
        value_est_copy)))
    if (max_diff_val < epsilon):
        print("Converged! Value state converged in iteration: "
            , i+1)
        break
    max_diff_policy = np.max(abs(policy_copy - policy))
    if (max_diff_policy < epsilon):
        print("Converged! Policy converged in iteration: ", i
            +1)
```

The figure 1 shows an example render after convergence.

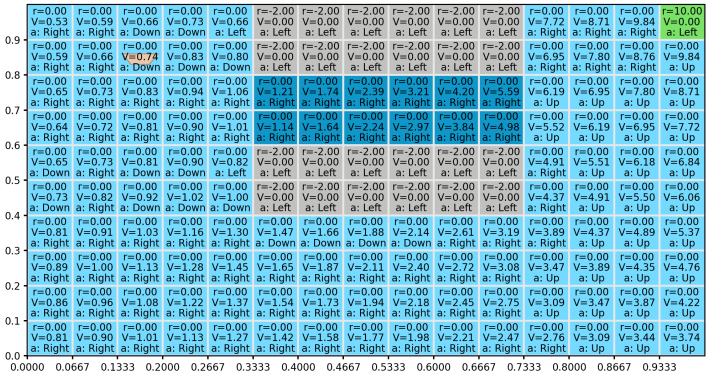

Figure 1: Environment with reward, state value, and policy for each cell.

3 Question 2

The state value of the harbour as well as the rocks is 0. That's because after one of these are reached the episode gets terminated. So there won't follow any update.

4 Task 2

The optimal policy is also displayed in figure 1. To run with the policy the following code snippet needs to be updated:

```
# TODO: Use the policy to take the optimal action (Task 2)
action = policy[state]
```

I rerun several episodes and observed the behavior. The boat is able to reach the harbour. However, it is not able to always reach the harbour. The sailor tends to take the narrow passage between the rocks. Due to the wind probability it sometimes crashes against the rocks.

5 Question 3

The sailor chose most of the times path through the narrow passage between the rocks. However, if we change the rock penalty to -10,

```
# Set up the environment
env = SailingGridworld(rock_penalty=-10)
```

the sailor tends to avoid narrow passage and goes most of the time below it.

6 Question 4

This question is conducted with rock penality = -2. I rerun the algorithm with 10, 20, 30, 40, 50 iterations. For 10, 20, 30 iterations it didn't converge. However, for 40 iterations and higher it converged. The policy converged for 20 but not for 30 iterations. The policy needs less iterations to converge. That makes sense because as soon as it is figured which action take, the state values get fine tuned within the next iterations while the policy stays the same.

7 Task 3

The termination condition after convergence is integrated in the code:

```
1 max_diff_val = np.max(abs(abs(value_est) - abs(value_est_copy))
    )
2 if (max_diff_val < epsilon):
3     print("Converged! Value state converged in iteration: ", i
        +1)
4     break
```

So as soon as the maximal updated difference is smaller than epsilon the value iteration stops.

8 Task 4

The code got updated to run the program for 1000 episodes and calculate the mean and variance of the discounted returns of the initial state. The mean is 0.6701 and the variance 1.3472.

```python
# TODO: Run multiple episodes and compute the discounted returns (Task 4)
number_episodes = 1000
discounted_return_history = []

for i in range(number_episodes):
    done = False
    counter_discount = 0
    discounted_return = 0

    while not done:
        # Select a random action
        # TODO: Use the policy to take the optimal action (Task 2)
        action = policy[state]

        # Step the environment
        state, reward, done, _ = env.step(action)

        # calculate discounted reward
        discounted_return += reward * gamma**counter_discount
        counter_discount += 1

        # Render and sleep
        #env.render()
        #sleep(0.5)

    discounted_return_history.append(discounted_return)
    state = env.reset()

print("discounted return (initial state) - mean: ", np.mean(discounted_return_history))
print("discounted return (initial state) - std: ", np.std(discounted_return_history))
```

9 Question 5

In the book [1] we can see in equation (3.8) that the discounted returns depend on the rewards in each step and gamma. Furthermore, the state value depends on which policy we execute. The equation (3.12) states the value function of a state s under a policy π is the expected return when starting in s and following π thereafter. That's the relationship between the discounted return and the value function.

10 Question 6

No, the value iteration approach can't be applied to an unknown environment. It is a prerequisite to have complete knowledge of the environment which is stated in chapter 4.4 in book [1]. That includes probability of all transactions, actions, reward, and states. Then we can predict the next state value and reward. In this exercise the whole environment is known and therefore we can apply value iteration.

To be able to learn in an unknown environment is addressed with Monte Carlo estimation. The robot can learn with given experience from episodes.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.