Soleiman BashirGanji
810103077

Interactive Learning Homework 2

Q1.

https://chatgpt.com/share/673347c5-25c0-8012-83e8-7b72173f8421

In supervised learning, we need labeled data to train a model. The model learns a mapping between the data and the labels and usually becomes highly specialized for that specific data. A supervised learning model aims to replicate the patterns in the labeled data as accurately as possible.

In contrast, reinforcement learning gathers data through active exploration of an environment, and there are no fixed labels. Instead, the model receives rewards based on its actions, which may vary and can depend on a sequence of actions (delayed rewards). In supervised learning, the "reward" (correctly predicting a label) is immediate, while in reinforcement learning, rewards are not necessarily instant.

For example, consider a self-driving car. It needs to classify traffic signs, estimate distances, and detect lane lines. These tasks are often handled with supervised learning, where the model is trained with labeled images to recognize objects like signs.

However, the act of driving cannot be fully handled as a supervised learning problem because it involves navigating a continuous environment. Each action moves the car into a new state, and the results of actions (rewards) are not always immediate. For driving, the car needs to learn by interacting with the environment and gaining experience over time.

Q2.

A positive linear transformation on the reward function does not change the optimal policy, because it does not affect the relative values of the rewards and only re-scales them.
But a negative linear transformation can change the order of the rewards, meaning that the policy with the least expected reward, will be transformed into the optimal policy.

Being continuous or episodic does not change anything here.

$$\pi^* = \underset{a}{\operatorname{argmax}}\, q_\pi(s,a) = \underset{a}{\operatorname{argmax}} \sum_{s'r} p(s',r|s,a)\left[r + \gamma v_\pi(s')\right]$$

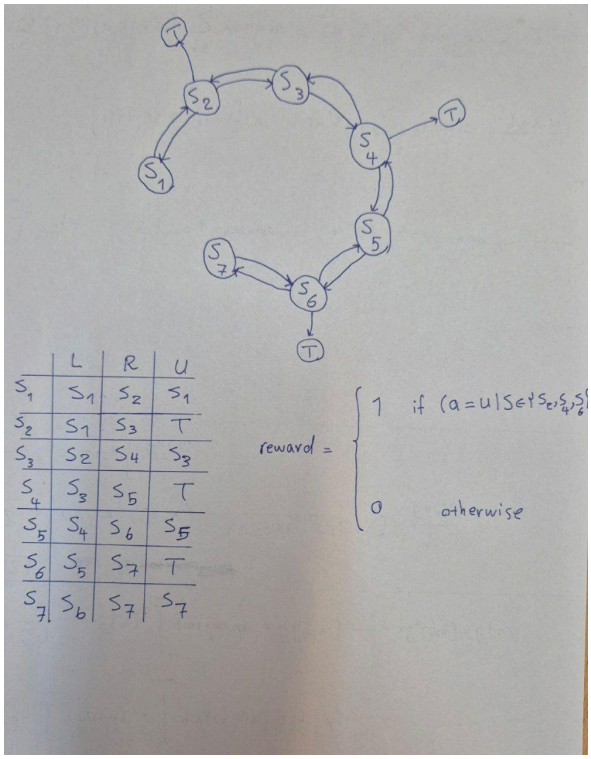
$$\xrightarrow{\alpha \cdot X + b} \underset{a}{\operatorname{argmax}} \sum_{s'r} p(s',\alpha r+b|s,a)\left[\alpha r + b + \gamma v_\pi(s')\right]$$

$$\longrightarrow \underset{a}{\operatorname{argmax}}\, q_\pi(s,a) = \underset{a}{\operatorname{argmax}}\left(\alpha(q_\pi(s,a)) + b\right)$$

Q3.

Scenario 1

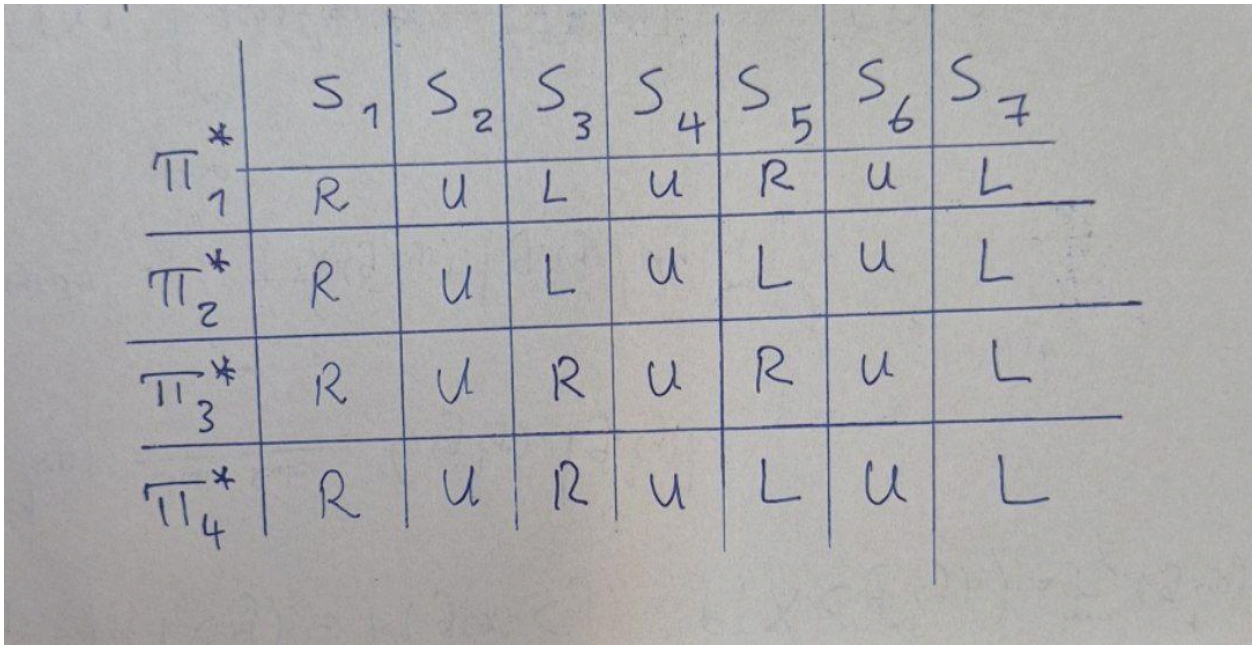
الف)



	L	R	U
S_1	S_1	S_2	S_1
S_2	S_1	S_3	T
S_3	S_2	S_4	S_3
S_4	S_3	S_5	T
S_5	S_4	S_6	S_5
S_6	S_5	S_7	T
S_7	S_6	S_7	S_7

$$\text{reward} = \begin{cases} 1 & \text{if } (a = u \mid s \in \{s_2, s_4, s_6\} \\ 0 & \text{otherwise} \end{cases}$$

ب)
The only states where we have more than 1 optimal option, are S3 and S5. In these states, we can choose to go right or left, and then up. This means there are 4 deterministic optimal policies.



	S_1	S_2	S_3	S_4	S_5	S_6	S_7
π^*_1	R	U	L	U	R	U	L
π^*_2	R	U	L	U	L	U	L
π^*_3	R	U	R	U	R	U	L
π^*_4	R	U	R	U	L	U	L

Scenario 2

In the first scenario, we basically knew everything and there was no need to “learn” anything. The problem was basically a transition problem, where we would try to reach the goal state through the shortest sequence of actions.

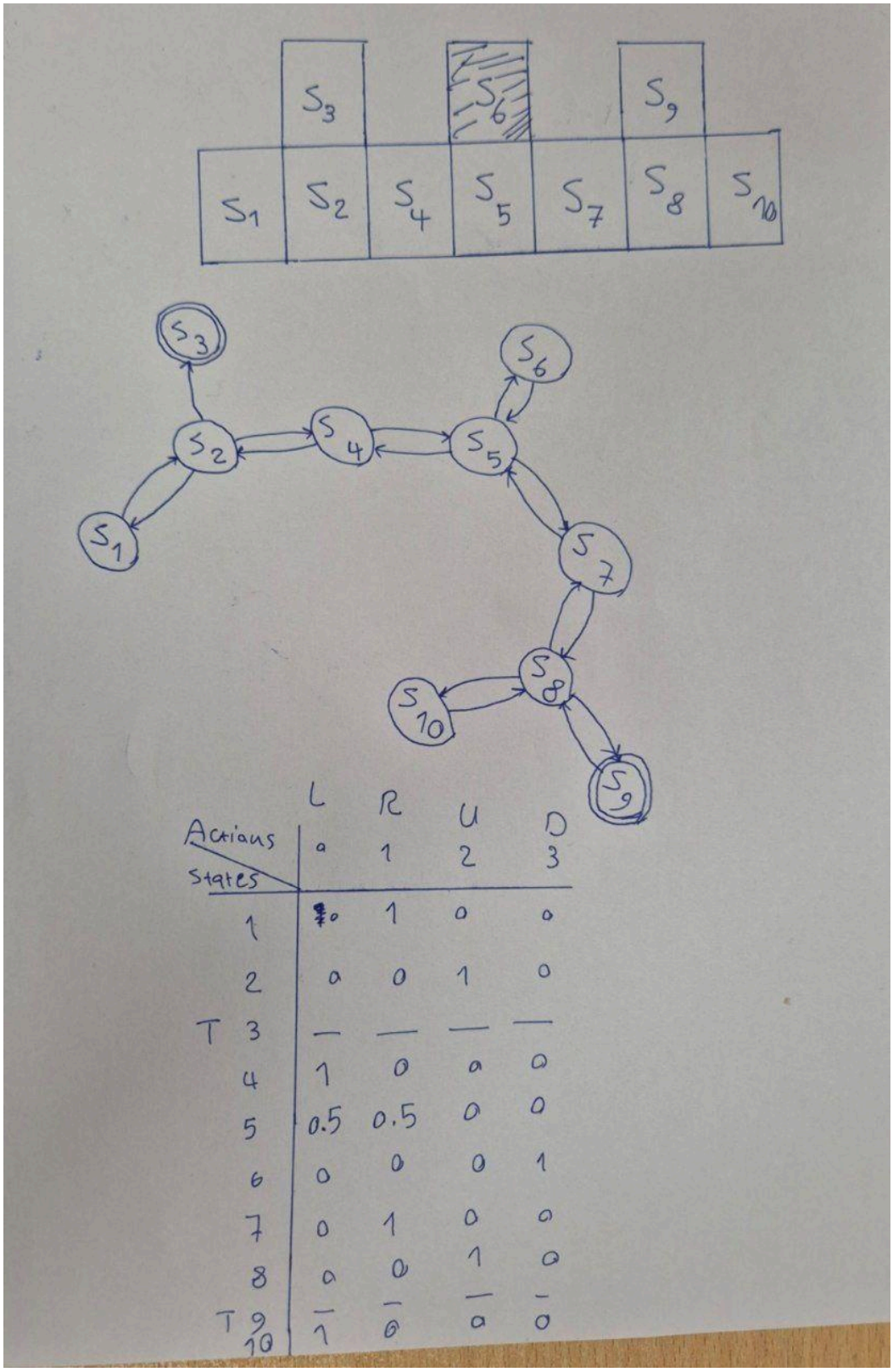
But in this scenario, we do not know where the goal state is, and we have to explore the environment in order to learn what to do in every state.
Here, the optimal policy is the same optimal policy as before, but we need to learn the P(a|s) for every state-action through exploration.

Scenario 3

الف)
This scenario can also be modeled as a MDP problem and there exists an optimal policy which satisfies our goal.

State	S1	S2	S3	S4	S5	S6	S7	Red T
Action	R	U	L	L/R	R	U	L	D

ب)



Actions / States	L 0	R 1	U 2	D 3
1	0	1	0	0
2	0	0	1	0
T 3	—	—	—	—
4	1	0	0	0
5	0.5	0.5	0	0
6	0	0	0	1
7	0	1	0	0
8	0	0	1	0
T 9	—	—	—	—
10	1	0	0	0

Implementation:

The code is available as ```Q3.py```, the agent always finds the optimal policy in the above picture consistently.

Q4.

الف)

For H<4, the agent will always sell.

For 3<H<7, the agent will sell 3 times and after that, it will buy and sell once at a time until the end of the horizon.

For H>6, the agent will buy until it reaches the terminal state and gains 100 reward.

ب)

H>6

ج)

If we want to reach the terminal state asap, at any state the agent will start moving right (Buy).
If we just want to maximize the reward, with an infinite horizon the agent will evade the terminal state, and will start by selling until S=0. And then will do Buy/Sell forever to get an infinite reward.

د)

We must choose the discount factor in a way so it cancels out the +100 reward of entering the terminal state. The minimum number of actions required to enter the terminal state starting from S=3 is 7, and the discount factor has to cancel out the reward in 7 steps.
For this to happen, we probably want to choose the discount factor closer to 0, so we care more about the immediate rewards. If we choose a very small discount factor, the future will be heavily discounted and we'd like to always sell to get +1 reward.

Implementation:

I used this code for the implementation but I did go through it carefully and tried to understand every part.
https://chatgpt.com/share/6738b4a1-728c-8012-8ba9-b79abb83b4a7
The code is attached as Q4.py

Buy = 1
Sell = 0

الف)

H = 10

```
Policy (Policy Iteration): [1 1 1 1 1 1 1 1 1 1 0]
Policy (Value Iteration): [1 1 1 1 1 1 1 1 1 1 0]
```

H = 3
(Only sell starting from S = 3)

```
Policy (Policy Iteration): [0 0 0 0 0 0 0 1 1 1 0]
Policy (Value Iteration): [0 0 0 0 0 0 1 1 1 1 0]
```

ب)

```
Policy (Policy Iteration): [0 0 0 0 0 0 0 0 0 1 0]
Policy (Value Iteration): [0 0 0 0 0 0 0 0 0 1 0]
```

The agent favors immediate reward.

ج)
Discount factor = 0.9

```
Policy (Policy Iteration): [1 1 1 1 1 1 1 1 1 1 0]
Policy (Value Iteration): [1 1 1 1 1 1 1 1 1 1 0]
```

Discount factor = 0.1

```
Policy (Policy Iteration): [0 0 0 0 0 0 0 0 1 1 0]
Policy (Value Iteration): [0 0 0 0 0 0 0 0 1 1 0]
```