LAB 4 THEORETICAL D7047E GROUP 7

Gabriel Röjmyr
gabrjm-0@student.ltu.se
Lulea University of Technology

Benjamin Ådén Fjällman
bendnf-0@student.ltu.se
Lulea University of Technology

Umuthan Ercan
umuerc-2@student.ltu.se
Lulea University of Technology

Aamer Ahmed
aamahm-3@student.ltu.se
Lulea University of Technology

Georgios Savvidis
geosav-2@student.ltu.se
Lulea University of Technology

Sushanta Mohapatra
susmoh-3@student.ltu.se
Lulea University of Technology

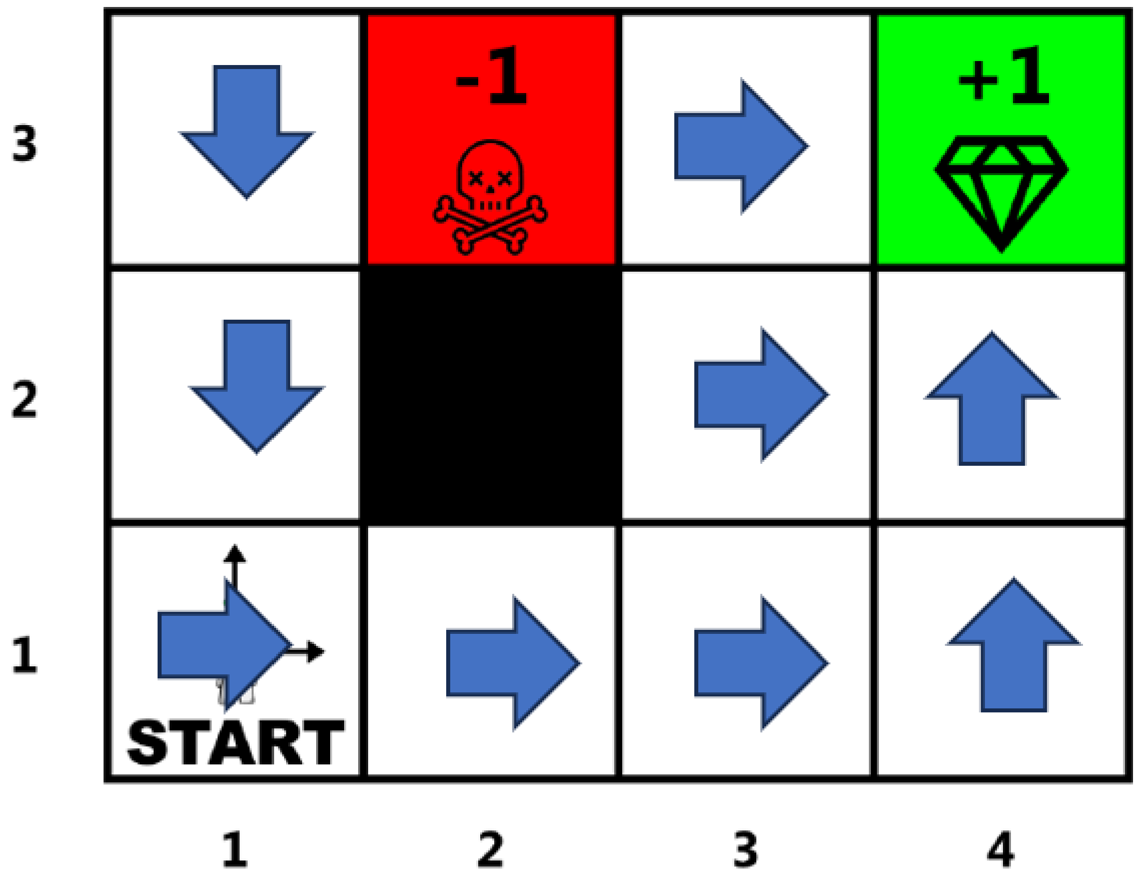
May 24, 2024

Theoretical Tasks

Task 1.1 Optimal policy π*

We consider the following optimal policy. Since the only state that has a positive result is (4,3) the optimal policy will be the one that leads to this state with the minimum number of transitions.

There are multiple policies that are "equally optimal".



Task 1.2.

(4,3) is target state, thus:

$$V(4,3) = Transition\ State\ (Target) = 1$$

Similarly (2,3)

$$V(2,3) = Target\ State = -1$$

(3,3) is one step from target state V(4,3), thus:

$$V(3,3) = \gamma \cdot V(4,3) = 0.9 \cdot 1 = 0.9$$

Similarly, for V(3,1):

$$V(3,1) = \gamma \cdot V(4,1) = \gamma^2 \cdot V(4,2) = \gamma^3 \cdot V(4,3) = 0.9^3 \cdot 1 = 0.729$$

And for (1,1):

$$V(1,1) = \gamma \cdot V(2,1) = \gamma^2 \cdot V(3,1) = 0.9^2 \cdot 0.729 = 0.59049$$

Task 1.3.

Solution 1

```
import numpy as np

g=0.7
p=0.8
H=100

w = np.zeros((3,4))
w[0,1] = -1
w[0,3] = 1
```

```
pi = np.zeros((w.shape[0], w.shape[1], 3,3))

pi[0,0] = np.array([[0., 0., 0.],
                    [0., 0., 1-p],
                    [0., p, 0.]])

pi[1,0] = np.array([[0., 1-p, 0.],
                    [0., 0., 0.],
                    [0., p, 0.]])

pi[2,0] = np.array([[0., 1-p, 0.],
                    [0., 0., p],
                    [0., 0., 0.]])

pi[0,1] = np.array([[0., 0., 0.],
                    [0., 1., 0.],
                    [0., 0., 0.]])

pi[1,1] = np.array([[0., 0., 0.],
                    [0., 0., 0],
                    [0., 0, 0.]])

pi[2,1] = np.array([[0., 0., 0.],
                    [1-p, 0., p],
                    [0., 0., 0.]])

pi[0,2] = np.array([[0., 0., 0.],
                    [(1-p)/2, 0., p],
                    [0., (1-p)/2, 0.]])

pi[1,2] = np.array([[0., (1-p)/2, 0.],
                    [0., 0., p],
                    [0., (1-p)/2, 0.]])

pi[2,2] = np.array([[0., (1-p)/2, 0.],
                    [(1-p)/2, 0., p],
                    [0., 0., 0.]])

pi[0,3] = np.array([[0., 0., 0.],
                    [0., 1., 0.],
                    [0., 0., 0.]])

pi[1,3] = np.array([[0., p, 0.],
                    [(1-p)/2, 0., 0.],
                    [0., (1-p)/2, 0.]])

pi[2,3] = np.array([[0., p, 0.],
                    [1-p, 0., 0.],
                    [0., 0., 0.]])
```

```
w_rows = w.shape[0]
w_columns = w.shape[1]

values0 = np.copy(w)
values0 = np.pad(values0, pad_width=1, mode='constant', constant_values=0)


for step in range(H):
    values1 = np.zeros_like(values0)
    for r in range(w_rows):
        for c in range(w_columns):
            values0_k = values0[r:r+3,c:c+3]

            if pi[r,c][1,1] == 1:
                gamma = 1
            else:
                gamma = g

            v =  gamma * np.sum(values0_k*pi[r,c])
            values1[r+1,c+1] = v

    values0 = np.copy(values1)
```

```
values = np.round(values1[1:4,1:5],4)
values
```

The result values are:

```
array([[-0.1211, -1.    ,  0.5179,  1.    ],
       [ 0.0337,  0.    ,  0.3979,  0.6144],
       [ 0.0905,  0.1532,  0.2509,  0.3792]])
```

And so:

V(4,3) = 1

V(3,3) = 0.5179

Solution 2

(Alternative manual approach with assumption that optimal policy considered and no recursion for reward function)

$$V(4,3) = 0.8 * 1 = 0.8$$

$$V(3,3) = 0.7 * 0.8 * 1 = 0.56$$

Task 2 – Questions

Task 2.1: Explain the exploration vs. exploitation problem in RL

The exploration vs exploitation problem in RL is the basic problem of the agent deciding whether it should try new actions to learn new things from their effects which could lead to better policies (exploration) or if it should continue use the current learning and try to maximize immediate rewards from the currently known actions (exploitation).

In exploration the agent tries to expand its knowledge of the environment, discover new states and though this possibly discover better policies. In doing so there is always the risk of diverging from a currently good strategy to a worse or sub-optimal one.

In exploitation the agent tries to maximize the reward in short term based on the currently developed policy. However, early exploitation might lead to be locked into a suboptimal early policy and failure to discover new ones later on.

Task 2.2: Explain the credit-assignment problem in RL

Credit-assignment problem in RL is the inherent difficulty to attribute the reward (positive or negative) that the agent receives while making repeated, multiple decisions over a period of time to each of those decisions and at what specific time. The first aspect is called the ***Temporal Credit Assignment Problem*** while the second one is called the ***Structural Credit Assignment Problem***.

Task 2.3: What is the Markov property?

Markov Property refers to the property of a process according to which the future state of the system is not dependent on its past states but only on its current one and the actions that the agent in the system can take, i.e. that the state transitions are memoryless.

Task 2.4: Motivate whether or not chess satisfies the Markov property

For the game of chess for any given configuration of the board (the current placement of pieces on the board) and a chosen move by the player whose turn is to play, the next configuration of the board can be unambiguously determined without the need to know how the pieces arrived in that configuration, simply by applying the rules of chess.

However, we need to point out that the set of allowable moves in chess DOES depend on the previous moves. i.e. whether “castling” (roque) is only allowed if the rook(s) and King/Queen have not moved yet, i.e. some kind of memory is required. This means that whether some transitions are allowed or/not depends on the past moves, not their sequence but if an event has occurred in the past. In that respect we should determine that chess **DOES NOT** satisfy Markov property.

If we expand what constituted the “state” of a chess match and include these few Booleans/flags (i.e. the “environment” or in this case the rules of chess will contain information to be able to change its feedback to an action that may not be allowed anymore) then it can be said that **chess does satisfy Markov property.**

Task 2.5: What is the difference between Q-learning and Deep Q-learning?

The main difference between Q-learning and deep Q-learning is the implementation of the Q-table.

Q-Learning consists of mapping each state -action pair to the corresponding Q-Value of the Bellman equation. The values of the Q-table are directly updated in the table during learning.

Deep Q-learning uses a neural network as the function approximator for the estimation of the action-value function Q(s,a values in the Q-Table. The network takes as input states and outputs Q-values for each possible action. It uses Experience Replay, i.e. stored past experiences of state-action-reward-next_state from which samples are taken randomly to overcome correlation issues between consecutive experiences. In Deep-Q-learning it is the weights of the Neural network that are being updated rather than the Q-Values themselves (which are generated from the Neural Network).