Lab3_report

Gustav Wahlquist

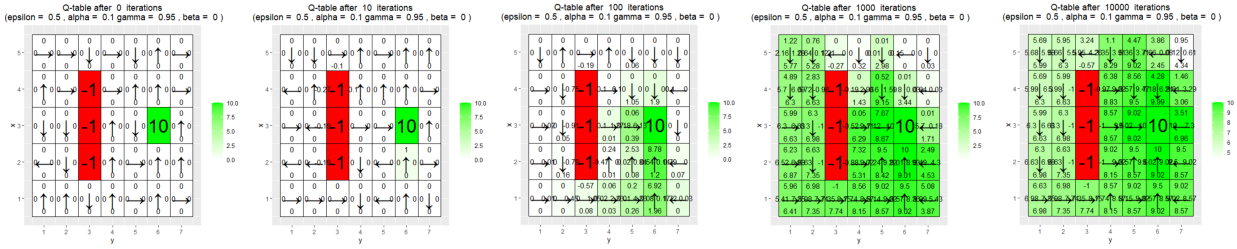
2020-10-05

TASK 1

See code in the annex section for the implemented functions.

Task 2 - Environment A

Run 10000 episodes of Q-learning with $\epsilon = 0.5$, $\beta = 0$, $\alpha = 0.1$ and $\gamma = 0.95$. To do so, simply run the code provided in the file RL Lab1.R. The code visualizes the Q-table and a greedy policy derived from it after episodes 10, 100, 1000 and 10000. Answer the following questions: – What has the agent learned after the first 10 episodes ? – Is the final greedy policy (after 10000 episodes) optimal? Why / Why not ? – Does the agent learn that there are multiple paths to get to the positive reward ? If not, what could be done to make the agent learn this ?



After 10 episodes, the agent has not had time to learn a lot. However, in the plot it is shown that the expected reward on some of the actions running in to the states with negative reward has been updated to negative values. This is since these states are relativly close to the start state for the agent and with only 10 episodes of exploration, these are the only states that the agent has gotten familiar with.

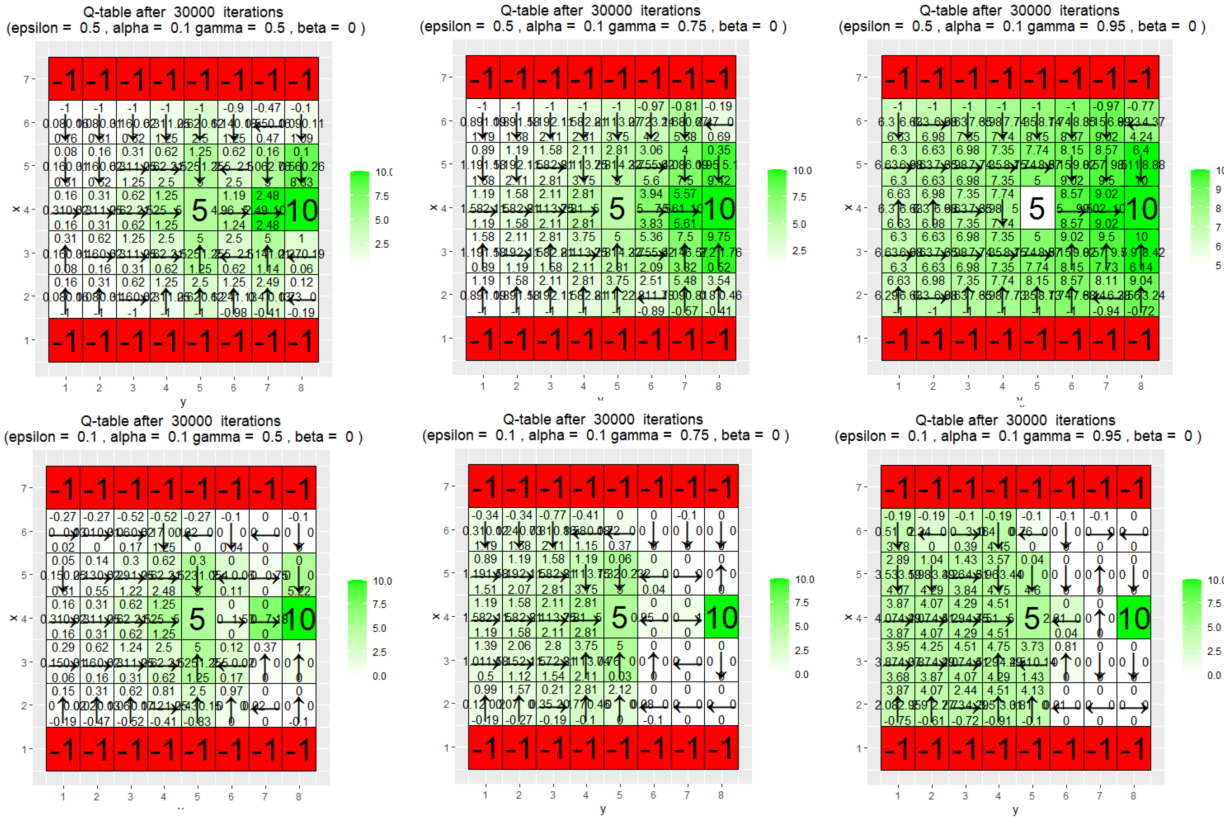
The final greedy policy is good but it could be a little bit better. For the policy to be optimal it should take the closest route to goal state. This is not true for the final greedy policy, we can see this in arrows in the states in the upper left corner of the plot for 10 000 iterations. In an optimal policy, these arrows should lead to the agent going over the red area instead of under since it is a closer route. Since $\beta = 0$ in this environment, the policy does not have to take into account that it could slip into the red area, if $\beta \neq 0$, then the optimal policy should try to take the closest route while still having as big a margin to areas with negative reward.

No really. To enforce the agent to learn this, a higher exploration factor(ϵ) could be used to make the agent try more different routes. Another solution could be to change the starting state for the agent, this could be implemented either by simple randomizing the starting state or by making the agent take a few totally random actions in the beginning of the episode.

Task 3 - Environment B

Your task is to investigate how the ϵ and γ parameters affect the learned policy by running 30000 episodes of Q-learning with $\epsilon = 0.1, 0.5$, $\gamma = 0.5, 0.75, 0.95$, $\beta = 0$ and $\alpha = 0.1$. To do so, simply run the code

provided in the file RL Lab1.R and explain your observations.

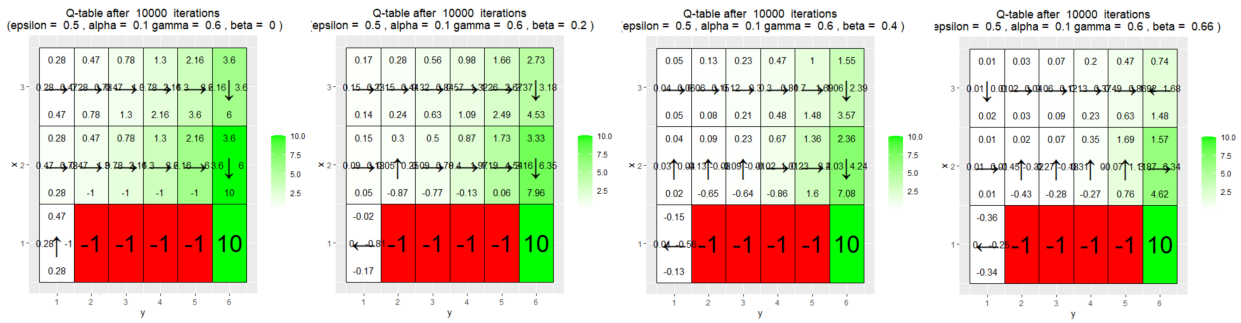


As shown in the three plots in the first row above, when $\epsilon = 0.5$ and $\gamma = 0.5, 0.75$, the agent does not find the reward worth 10 if it does not pass the first reward by chance. This is since with $\gamma = 0.5, 0.75$, a higher reward diminishes quite fast and is not as appealing as with a higher γ.

The runs with $\epsilon = 0.1$ are displayed in the plots on the second row. Looking at them, we can see that the agent never finds the state with the reward worth 10 (except if it passes the first reward by mistake). This is due to the fact that when ϵ is low, the agent will more often choose the action that it think is optimal and not try new routes that potentially could lead to greater rewards.

Task 4 - Environment C

Your task is to investigate how the β parameter affects the learned policy by running 10000 episodes of Q-learning with $\beta = 0, 0.2, 0.4, 0.66$, $\epsilon = 0.5$, $\gamma = 0.6$ and $\alpha = 0.1$.



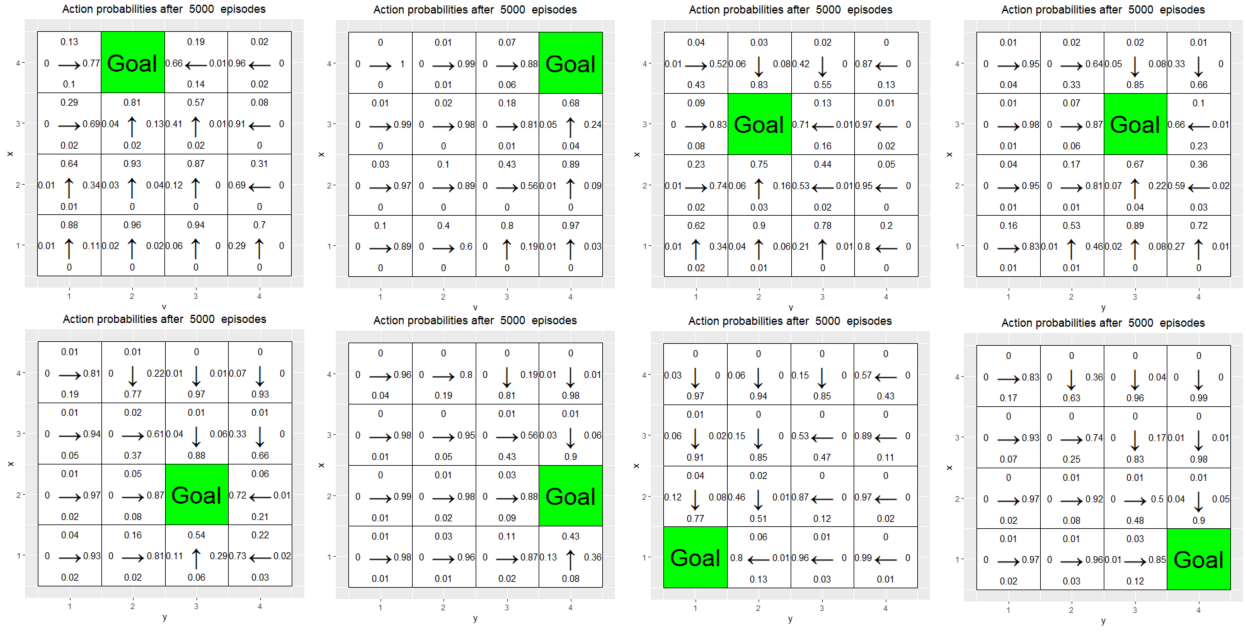
As mentioned in Task 2, the plots above shows that the agent will avoid states close to states with negative

rewards for high beta values. This is to avoid slipping down into the states with negative reward, even though that route is shorter.

Another interesting observation is that in the left most plot where $\beta = 0{,}66$, the optimal policy is to go to the right instead of going down to the reward. This is due to the fact that it is only a 33% chance that the agent will end up in the state with high reward if it goes down, but it also runs the risk of going backwards.

Task 5 - Environment D

- Has the agent learned a good policy? Why / Why not? - Could you have used the Q-learning algorithm to solve this task?

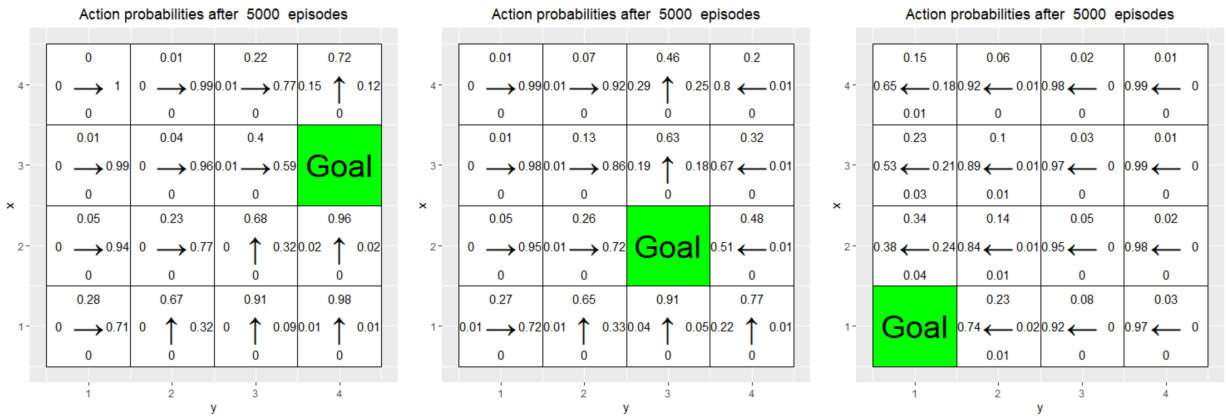


The agent seems to have learned a good policy for each goal. This is fairly easy to observe since the shortest route is measured in manhattan distance and by looking at the plots above, we can see that the arrows always goes towards the goal in either the vertical or the horizontal direction.

A Q-learning algorithm would no be a good choice for this type of problem. This is due to the fact that the agent in Q-learning only can make decisions on what it has previously discovered unlike the the REINFORCE algorithm which generalizes it's solution in a way that it can use the model to find new goals that it has never before experienced.

Task 6 - Environment E

You are now asked to repeat the previous experiments but this time the goals for training are all from the top row of the grid. The validation goals are three positions from the rows below. To solve this task, simply run the code provided in the file RL Lab2.R and answer the following questions: - Has the agent learned a good policy? Why / Why not? - If the results obtained for environments D and E differ, explain why.



In this setting, the agent has not learned an optimal policy. Why this happens is due to the fact that unlike the previous environment, the training data in this environment is heavily biased towards the upper part of the grid and the validation data is only in states in the lower part of the grid. In conclusion, the training data and validation data does not seem to represent the same ground truth distribution and the resulting model from the REINFORCE algorithm does not generalize well.

Appendix for code

In this appendix I have included the code I have modified or added and left out the given code I have not changed.

```
GreedyPolicy <- function(x, y){

  # Get a greedy action for state (x,y) from q_table.
  #
  # Args:
  #   x, y: state coordinates.
  #   q_table (global variable): a HxWx4 array containing Q-values for each state-action pair.
  #
  # Returns:
  #   An action, i.e. integer in {1,2,3,4}.

  # Your code here.
  if (length(which(q_table[x, y,] == max(q_table[x, y,]))) < 2) {
    return(which(q_table[x, y,] == max(q_table[x, y,])))
  } else {
    return(sample(which(q_table[x, y,] == max(q_table[x, y,])),1))
  }

}

EpsilonGreedyPolicy <- function(x, y, epsilon){

  # Get an epsilon-greedy action for state (x,y) from q_table.
  #
  # Args:
  #   x, y: state coordinates.
  #   epsilon: probability of acting randomly.
  #
  # Returns:
  #   An action, i.e. integer in {1,2,3,4}.

  # Your code here.

  if(runif(1) < epsilon) {
    return(sample(c(1,2,3,4),1))
  } else {
    return(GreedyPolicy(x,y))
  }

}

q_learning <- function(start_state, epsilon = 0.5, alpha = 0.1, gamma = 0.95,
                       beta = 0){

  # Perform one episode of Q-learning. The agent should move around in the
  # environment using the given transition model and update the Q-table.
  # The episode ends when the agent reaches a terminal state.
  #
  # Args:
```

```
  #   start_state: array with two entries, describing the starting position of the agent.
  #   epsilon (optional): probability of acting greedily.
  #   alpha (optional): learning rate.
  #   gamma (optional): discount factor.
  #   beta (optional): slipping factor.
  #   reward_map (global variable): a HxW array containing the reward given at each state.
  #   q_table (global variable): a HxWx4 array containing Q-values for each state-action pair.
  #
  # Returns:
  #   reward: reward received in the episode.
  #   correction: sum of the temporal difference correction terms over the episode.
  #   q_table (global variable): Recall that R passes arguments by value. So, q_table being
  #   a global variable can be modified with the superassigment operator <<-.

  # Your code here.

  x = start_state[1]
  y = start_state[2]
  episode_correction = 0

  repeat{
    # Follow policy, execute action, get reward.
    action = EpsilonGreedyPolicy(x, y, epsilon)
    new_state = transition_model(x, y, action, beta)
    reward = reward_map[new_state[1], new_state[2]]

    # Q-table update.
    q_table[x,y,action] <<- q_table[x,y,action] +  alpha*( reward + gamma*max(q_table[new_state[1], new
    episode_correction = episode_correction + reward + gamma*max(q_table[new_state[1], new_state[2],])
    x = new_state[1]
    y = new_state[2]

    if(reward!=0)
      # End episode.
      return (c(reward,episode_correction))
  }

}
```