TDDE15 - Lab 3

Erik Jareman

2022-09-27

Q-Learning implementation

Implementation of the GreedyPolicy and the EpsilonGreedyPolicy functions.

```
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

  possible_q_values = c(
    q_table[x, y, 1], # ^
    q_table[x, y, 2], # >
    q_table[x, y, 3], # v
    q_table[x, y, 4]  # >
  )

  return (which.max(possible_q_values))

}

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 (GreedyPolicy(x, y))
  }

  return (sample(c(1, 2, 3, 4), 1))

}
```

Implementation of the q_learning function

```
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.

  episode_correction = 0

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

    # Get reward
    reward = reward_map[end_state[1], end_state[2]]

    # Calculate temporal difference, store in sum, and update Q-table
    temporal_diff =
      reward +
      gamma * (max(q_table[end_state[1], end_state[2], ])) -
      q_table[start_state[1], start_state[2], policy]

    episode_correction = episode_correction + temporal_diff

    q_table[start_state[1], start_state[2], policy] <<-
```

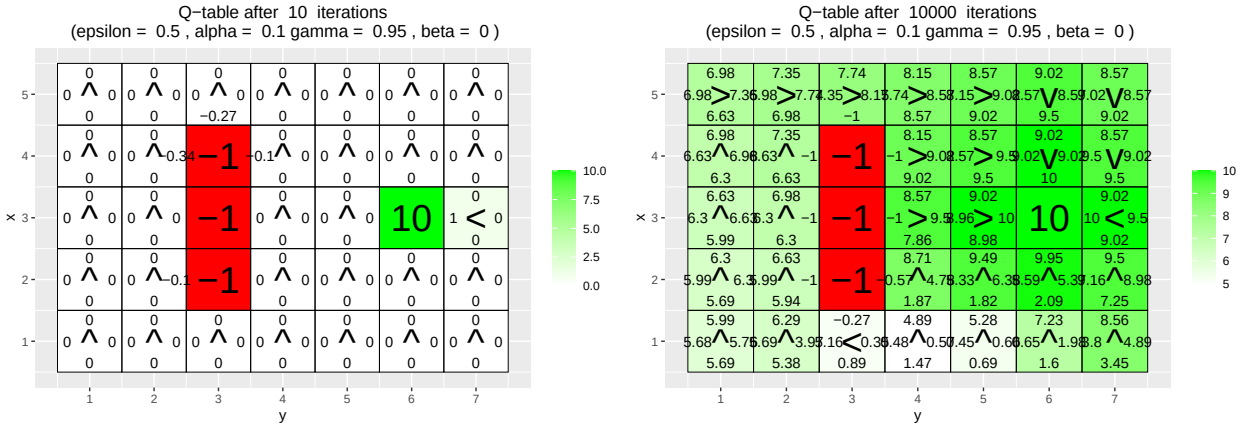
```
      q_table[start_state[1], start_state[2], policy] + alpha*temporal_diff

    # Move agent
    start_state = end_state

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

}
```

Environment A

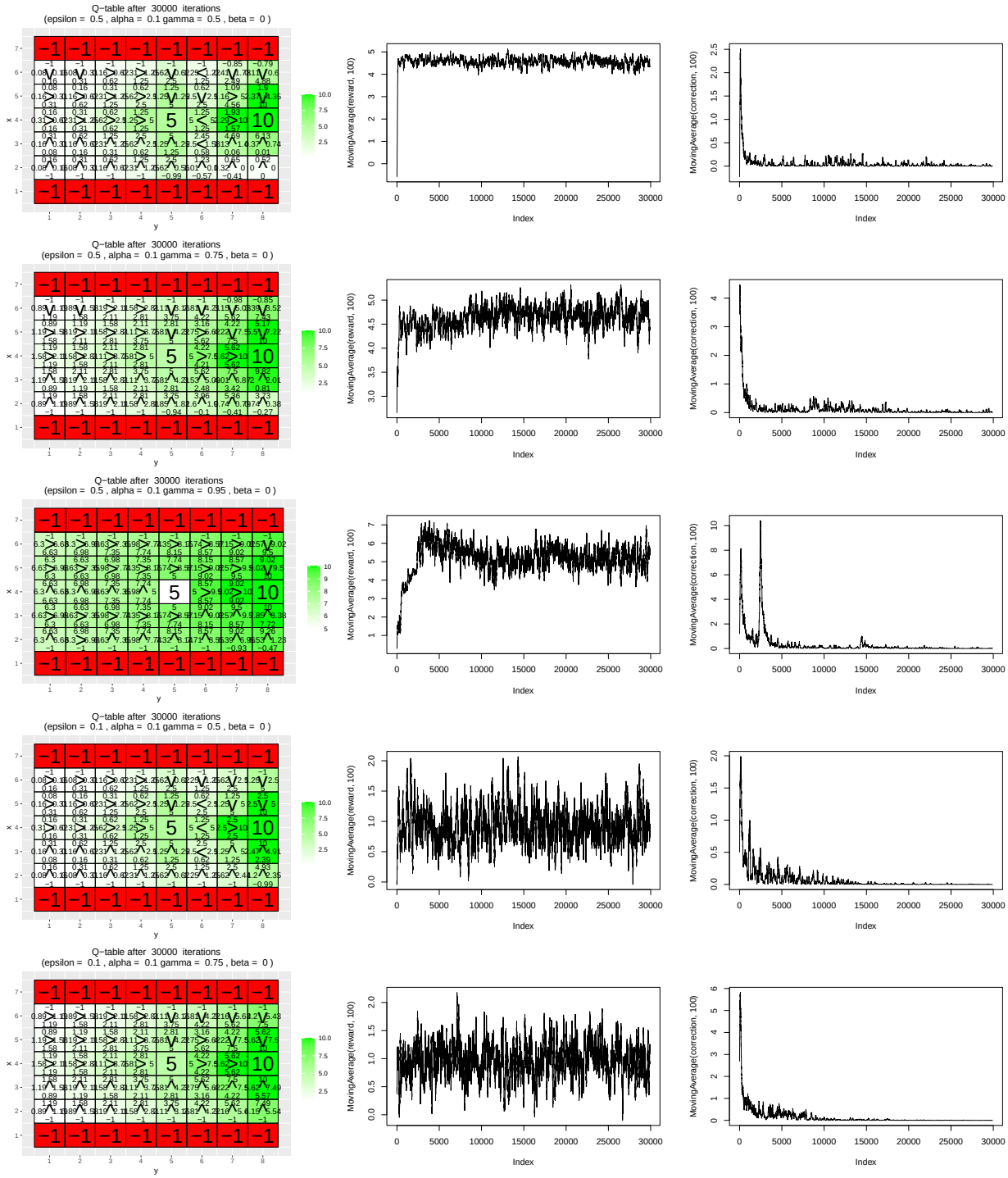


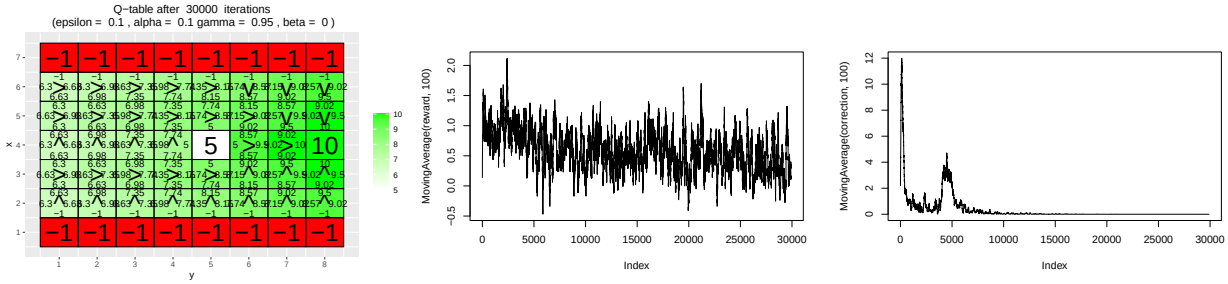
What has the agent learned after the first 10 episodes?
The agent is starting to learn that some of the actions taking the agent to the "-1" states will result in a negative reward. No positive rewards for actions has been set since none of the first 10 episodes has reached the "goal state" with reward 10.

Is the final greedy policy optimal for all states?
The final greedy policy is not optimal since there are states where the agent does not take the shortest possible path to the +10 reward state.

Do the learned values in the Q-table reflect the fact that there are multiple paths (above and below) to get to the possible reward?
The learned Q-values seem to prioritize one of the two possible paths, and thus, do not reflect on this fact. One way to make the agent find both paths could be to increase the probability for exploration.

Environment B





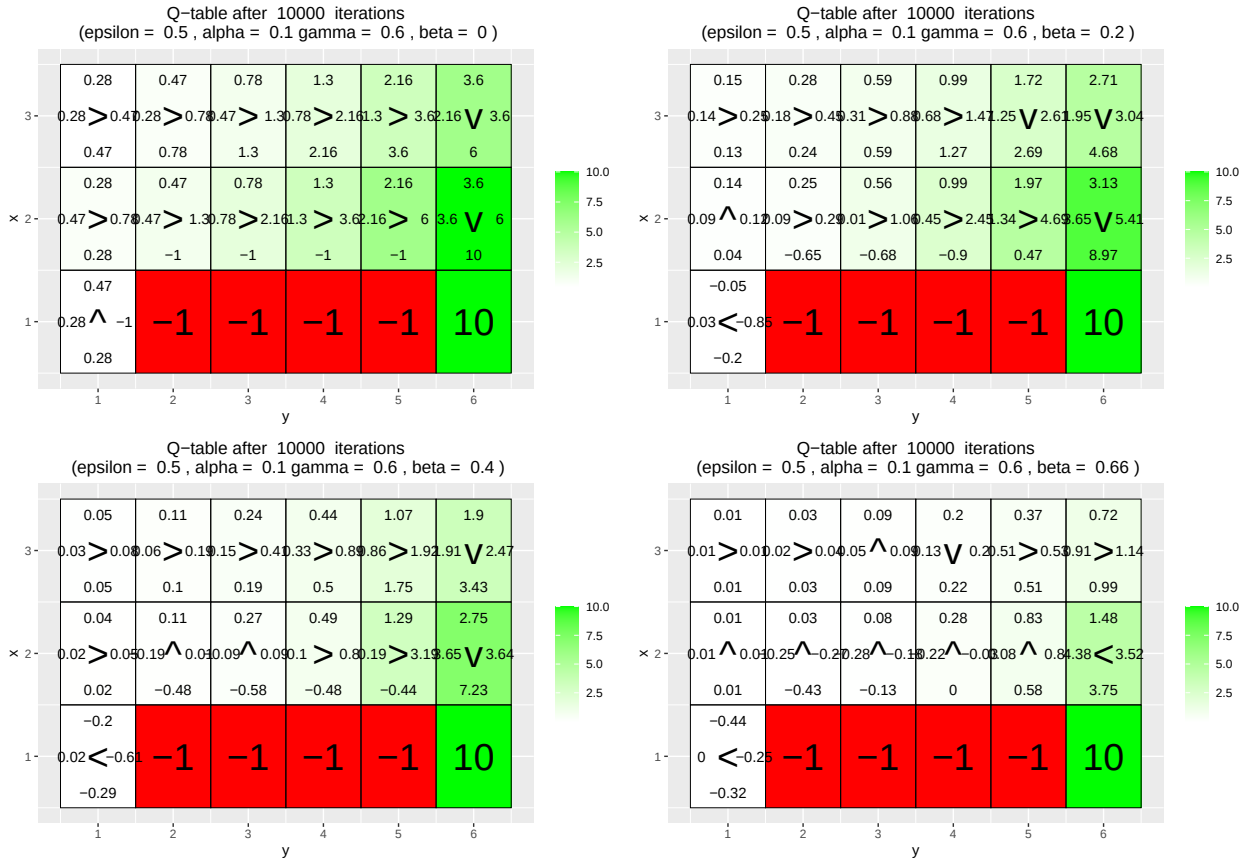
Gamma

A higher gamma leads to the agent finding the "10" reward more consistently. The reason for this is that a higher gamma leads to less punishment when the agent finds rewards after a long time. The agent does not have to find the reward quickly, and can therefore find its way to the higher reward that is farther away.

Epsilon

With a low value of epsilon, the agent will exploit the currently best policy rather than exploring and finding new, possibly better, policies. This means that if the agent finds the "5" reward first, it is likely to exploit that solution rather than exploring and finding its way to the "10" reward.

Environment C



Beta

A higher beta means that the agent has a higher probability to "slip". This means that the agent needs to take the risk of slipping into account more with a higher beta, and thus, take a path farther away from the penalizing states when moving towards the reward.

REINFORCE

Environment D

Has the agent learned a good policy?
The agent has learned a good policy. The greedy policy in most states will result in the agent taking the shortest path possible to the goal state. The model was trained on a large variety of training goals and is therefore able to find a good solution no matter what goal state is set.
train_goals (Environment D): c(4,1), c(4,3), c(3,1), c(3,4), c(2,1), c(2,2), c(1,2), c(1,3)

Could you have used the Q-learning algorithm to solve this task?
The Q-learning algorithm can not be used to solve this task. The reason for this is that the reward state changes position, and the agent in the Q-learning algorithm will make its decisions based on the Q-values for each state, and these are decided based on a specific reward state.

Environment E

Has the agent learned a good policy, and how does the results from environments D and E differ?
Since the goal states in the training data are all in the top position (x=4) in environment E, the agent is not taught to move down (downwards action never gives any benefit based on the training data). The validation data does not share this similarity, and the agent can not find goal states where downward motion is necessary. In environment D, the training data has more variety than in E, and the agent can learn to adapt to more situations.
train_goals (Environment E): c(4,1), c(4,2), c(4,3), c(4,4)