
Machine Learning (CS 391L) : Spring 2013
Homework 5: Reinforcement Learning

Vineet Keshari **EID**: vk3226 **mailto**:vkeshari@cs.utexas.edu

---

1. Problem Description

In Reinforcement Learning, an agent learns by interacting with its environment. The agent maintains an internal representation of the environment and its options for interaction in the form of states and actions respectively, along with a set of values associated with each of them. In Q-learning these values, called Q-values, are assigned to each {state, action} pair. The agent can learn the Q-values through an iterative algorithm known as Q-learning. By continually interacting with the environment, the agent is able to learn which {state, action} pairs are favorable to achieve a certain goal. It learns this by receiving a positive or negative reward for being in certain states. In this experiment, we will make an agent learn how to navigate a 3x25 grid using Q-learning. The agent starts at one end of the grid and has to reach the other end in as few steps as possible, avoiding any obstacles it may encounter on its way. We do so by training the agent on two separate modules:

- The Approach module requires the agent to get to the other end of the grid in as few steps as possible.
- The Obstacle module requires the agent to avoid running into walls or obstacles.

At any time step, an agent can take one of four actions that allow it to move: {Up, Down, Left, Right}. We experiment with various representations of the agent's state. The agent's actions are determined by taking a combination of the values from the two modules. We then test the agent on grids with randomly generated obstacles and compare our results.

2. Approach

2.1 Grid generation

For each episode of training and testing, a 3x25 grid with randomly placed obstacles is generated. Each column in the grid can have zero, one or two obstacles. Care is taken to ensure that there exists at least one path from the start to the end column. This is done by generating obstacles in the columns sequentially and determining at each step whether or not at least one cell in the current column is "reachable" from the first column. Figure 2.1 shows a grid generated for this experiment.

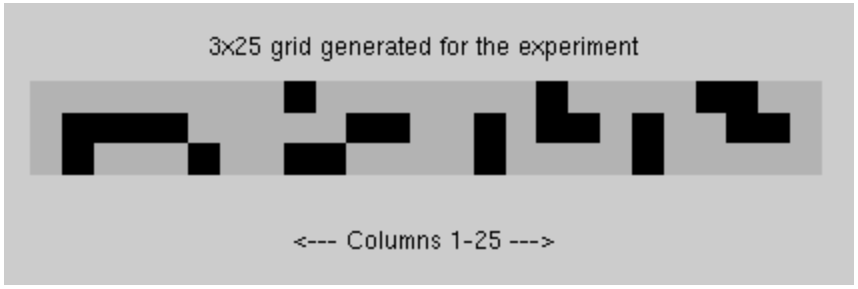

Figure 2.1: Example of a 3x25 grid with randomly generated obstacles. Obstacles are in black. The agent starts in Column 1 (left) and has to reach Column 25 (right). Note that there are no obstacles in these two columns. We also ensure that an obstacle-free path exists between the two ends.

2.2 Q-Learning

At each time step we choose between exploit and explore using an epsilon-greedy strategy as follows:

```
r = rand();
if r < epsilon     % explore
      action = randi(4);
      while action == approachPolicy(state)
            action = randi(4);
      end
else               % exploit
      action = approachPolicy(state);
end
```

After taking the action and determining the new state and reward, we update the Q-values as follows:

```
maxVal = max(qValues(newState,:));
qValues(state, action) = (1-learnRate).*(qValues(state, action)) +
                                 learnRate.*(reward + discount.*maxVal);
[val bestAction] = max(qValues(state,:));
approachPolicy(state) = bestAction;
```

In this example for the approach module, **`state`** and **`newState`** are integers. **`qValues`** is a two-dimensional matrix such that entry **`qValues(s,a)`** contains the Q-value for action **`a`** in state **`s`**, while **`approachPolicy(s)`** contains the optimal action for state **`s`**. More details about the training parameters, rewards and state representations for the approach module are in Section 3, while those for the obstacle module are in Section 4.

3. Approach Module

3.1 State Representation

Since the goal of the agent is to reach the other end of the grid, we represent the state as the column the agent is in. Since states are not ordered, this can also be seen as the horizontal distance between the agent's current column and goal column.

3.2 Rewards

When the agent reaches the goal state (column 25), it receives a reward of +1. For every other step the agent takes, it gets a reward of -0.01. The agent therefore has to minimize the number of steps taken to reach the goal, in order to maximize its reward.

3.3 Training

The module is trained with a learning rate of 0.1 and discount factor of 0.9. In each episode, the agent starts at one of the cells in column 0 and the goal state is to be in column 25. Each episode ends when the agent reaches the goal state. The values converge in less than 1000 episodes. Naturally, the optimal policy learned for this module is to go right in each state, as seen in Figure 3.1.

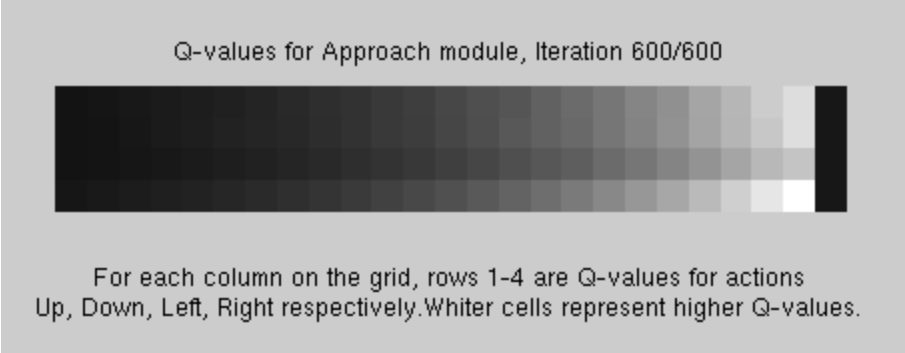

Figure 3.1: Q-values for a trained Approach module. Columns 1-25 are left to right, and actions {Up, Down, Left, Right} are top to bottom for each column. It can be seen that the action Right (row 4) has the highest value in each column, while Left (row 3) has the lowest. We can also see how the Q-values gradually decrease as we move away from the goal due to the discount factor.

4. Obstacle Module

4.1 State Representation

It is infeasible to represent the configuration of the entire grid as the state, since that results in an exponentially growing number of states. At the same time, we want to represent the state relative to the current position of the agent. We therefore represent the configuration of obstacles in the cells surrounding the agent's current position. We try the four different approaches to do this:

- **Four cells -** Obstacles in the four cells the agent can move to (Up, Down, Left, Right).
- **Eight cells -** Obstacles in the eight cells surrounding the agent.
- **Four direction scans -** Distance of the closest obstacle along each of the four directions.
- **Six horizontal scans -** Horizontal distance of the closest obstacle in the agent's current row and the rows above and below, in both directions.

Figure 4.1 shows the four state representations, along with the total number of states for each. We will compare the performance of each of these representations in section 5.3.

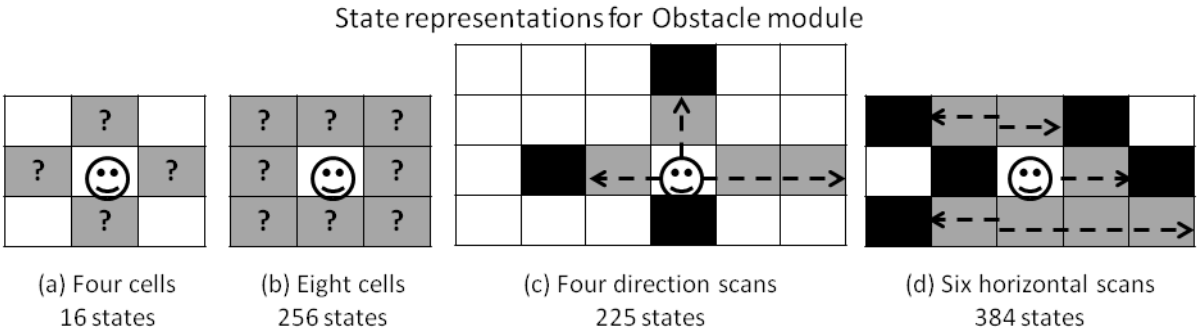

Figure 4.1: State Representations for Obstacle module: (a) Binary values for four cells (gray) indicating whether or not the cell has an obstacle, (b) Same as (a) but for eight surrounding cells, (c) Integer distance of the closest obstacle in four directions (maximum 5 horizontal, 3 vertical), (d) Horizontal integer distance, in both directions, of the closest obstacle in three rows centered at agent (maximum 4 forward, 2 back). All states are determined relative to the agent's current position. All four representations make the agent smile with joy.

4.2 Grid Boundaries

We treat a grid boundary (or "wall") as an obstacle for the following reason: Running into a wall does not result in movement, and as a result the agent's state doesn't change. If in its current state the agent chooses to hit a wall as the best available action as per its policy, it will continue to do so infinitely. Hitting the wall is therefore undesirable, and we will represent it as an obstacle so that the agent learns to avoid boundaries as well.

4.3 Rewards

The agent gets a reward of -1 every time it steps on an obstacle or hits a wall. All other actions do not generate any reward.

4.4 Avoiding cycles

In the absence of a specific position on the grid to reach as a goal in this module, the agent can end up moving in cycles, mostly consisting of only two cells. An example of such a cycle is shown in Figure 4.2. To avoid such cycles, we make the agent leave a marker at every cell it visits. While this does not change the configuration of the grid, the marker will appear as an obstacle in the agent's state representation thereafter, and generate negative reward if the agent visits the cell again. If an action eventually leads the agent to a blocked path, the negative reward when backtracking will reduce the Q-value for that action, and due to discounting the agent will gradually learn to not take that action. The effect of avoiding cycles on successful traversals in discussed in Section 5.5.

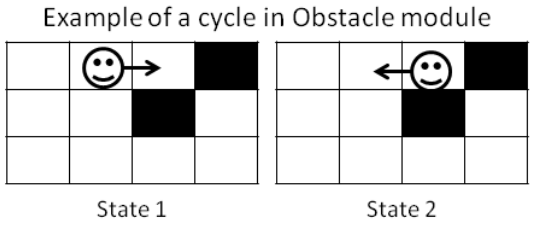

Figure 4.2: An infinite cycle is caused when states representing two adjacent positions have optimal actions leading to one another. This particular case of a two-cell cycle is the most common one during testing, where the approach module increases the Q-value for moving right in State 1.

4.5 Training

Figure 4.3 shows the sum of absolute changes in Q-values for 10000 iterations for the obstacle module. These values are for the 'Six horizontal scans' state representation, with learning rate 0.1 and discount factor 0.9. It should be noted that the states do not represent unique grid configurations, since the agent is only aware of its surroundings. So the Q-values do not converge like they did for the approach module in Section 3.3. Instead, the total change in Q-values reduces over time. This is because more {state, action} pairs converge through the iterations, but some never do. We therefore stop training when the sum of absolute changes in Q-values is below a threshold.

5. Results

5.1 Combining Q-values

For testing, we combine the Q-values for the two modules by taking their weighted sum. The parameter **`obstacleWeight`** $\varepsilon\,[0, 1]$ represents the weight of the Q-value from the obstacle module in the sum, with the rest contributed by the approach module. The agent's policy is the action with the highest resulting weighted sum of Q-values.

5.2 Testing

We test the agent on randomly generated grids for each value of **`obstacleWeight`** between 0.1 and 0.9, in increments of 0.01. We measure the performance of the agent on three parameters. The results are averaged over 100 episodes for each data point. The parameters are:

- The average no. of steps taken to reach the last column (performance of Approach module).
- The average no. of obstacles hit on the way (performance of Obstacle module).
- The Success% of the agent, i.e., the percentage of episodes where the agent reaches the last column.

It should be noted that the Q-values for moving right in the approach module get overwhelmingly large as we get closer to the last column (+1 for column 24 as per figure 3.1), resulting in the approach module dominating in the last few columns for almost all values of **`obstacleWeight`** < 0.9. We can therefore expect the agent to step on one obstacle, if present, toward the end in an otherwise perfectly navigated grid.

5.3 Obstacle module performance

We compare the no. of obstacles hit by the agent for each of the four representations in figure 5.1. It should be noted that the no. of obstacles hit is higher for both low and high values of **`obstacleWeight`**, with an optimum in between. This is because for low values, the agent has a greater tendency to follow the approach module and can step on obstacles. For high values, the agent gives lesser preference to reach the last column, and may hit more obstacles by taking a longer path. As per the data, the 'Six Horizontal scans' representation performs the best since it has the lowest minima. We will therefore use the data for this representation of the obstacle module for subsequent results.

5.4 Approach module performance

The average no. of steps taken by the agent to reach the last column for each value of **`obstacleWeight`** is shown in figure 5.2. Note that the minimum no. of steps possible depends on the shortest path available on the grid, as determined by the randomly placed obstacles. Averaging over 100 iterations reduces the effect of this variance though. As expected, the number of steps increases as we increase the weight of the obstacle module, because the agent has a lower preference for reaching the last column in fewer steps and instead tries to avoid obstacles.

5.4 Optimal weightage of modules

As per figures 5.1 and 5.2, a value of 0.38 is optimal for **`obstacleWeight`**. The obstacle module has a minima at this point, while the approach module has a point of inflection in the roughly flat range 0.2 to 0.5. We get the following performance at this value:

- Average no. of steps taken: 30
- Average no. of obstacles hit: 0.7

5.5 Avoiding cycles

If we do not avoid cycles as described in section 4.4, less than 50% of the episodes result in success. The change in Success% as the value of **`obstacleWeight`** changes in shown in figure 5.3. For small values, the obstacle module has less contribution, so the agent frequently reaches the last column, at the cost of stepping on obstacles. On the other hand, for larger values of **`obstacleWeight`** the agent almost never reaches the last column due to low contribution of the approach module, and we therefore get a Success% of near zero. If we do avoid cycles, however, the agent has a 100% success rate for all values of **`obstacleWeight`**.

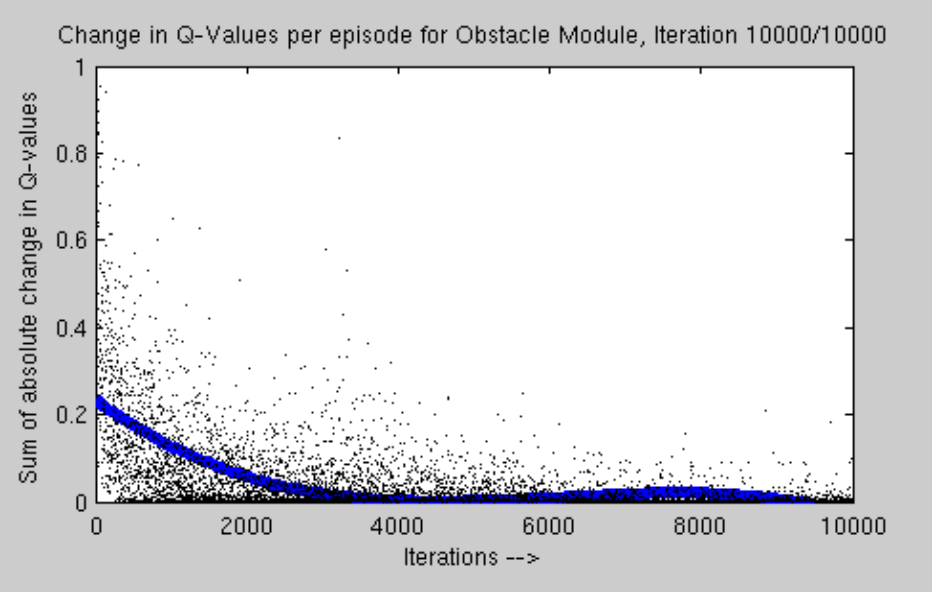

Figure 4.3: Convergence of Obstacle module. As mentioned in section 4.5, the training never converges for some states, but the change per episode is less than a threshold of 0.01 beyond about 7000 episodes.

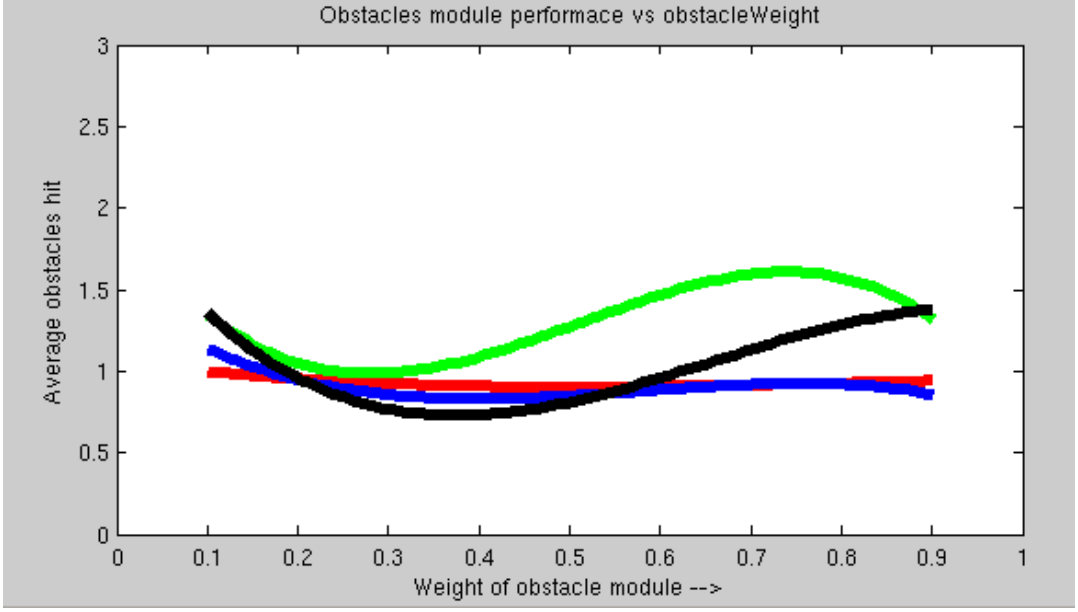

Figure 5.1: Performance of various state representations for obstacle module as the value of obstacleWeight changes: Four cells (blue), Eight cells (green), Four direction scans (red) and Six horizontal scans (black). All values are averaged over 100 episodes.

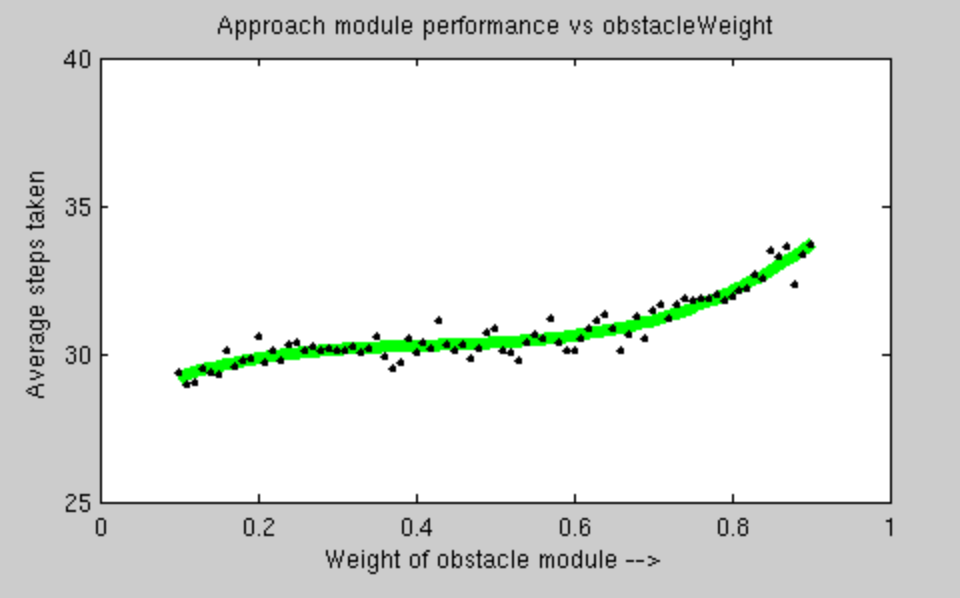

Figure 5.2: Performance of approach module as obstacleWeight changes. Values are averaged over 100 episodes for each data point.

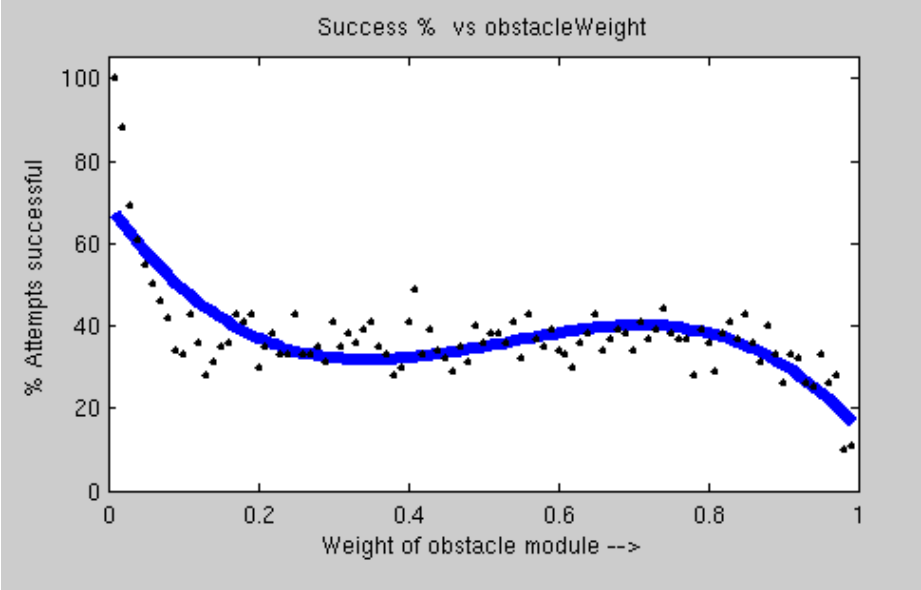

Figure 5.3: Testing Success% without cycle avoidance as described in section 4.4. Each data point is averaged over 100 episodes. With cycle avoidance, the agent has 100% success for all values of obstacleWeight.