CS391L Machine Learning
Assignment 6 Reinforcement Learning

Srinath Tankasala,
st34546

Abstract

In this assignment, an overview of the Q Learning method is presented. This reinforcement learning method is applied to guide an agent through an environment and perform certain tasks. The state space search for the Q learning problem was simplified by limiting how many cells the agent can see. The tasks were then separated into different modules and the individual Q matrices obtained after training were combined with a simple weighted sum. The final agent was able to successfully navigate a randomly generated environment and perform all tasks. Further work can be done to improve robustness of the Q learning approach.

I. Introduction

In this assignment the environment consists of objects, litter and a sidewalk. The agent is incentivized to pick up litter and must avoid obstacles. The agent must also keep moving forward along the sidewalk. Given these above tasks, we can approach this problem using Reinforcement learning. The "world" is defined as a 7x25 ($W = 7,\ L = 25$) grid with the middle 3 rows (green) as the sidewalk. Litter and debris/objects can be randomly scattered along these grid points.

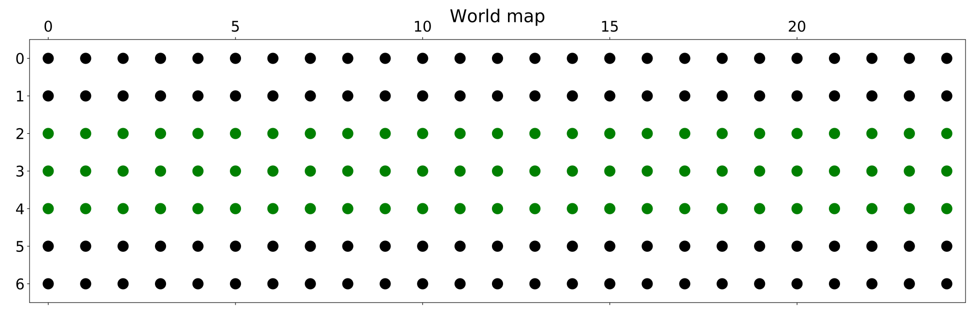

Fig. 1: Map of environment, green = sidewalk; black = outside sidewalk

To solve the full scale RL problem, we can have 3 states for each gridpoint, i.e. litter, obstacle or empty. This would lead to 3^{175} possible states for the whole system. Also a point can either be on the sidewalk or not $\rightarrow 2^{175}$ states. Thus the problem has a maximum state space size of 6^{175}. Clearly doing a complete state space search using an RL method would take a lot of computational time. The problem of large state space is handled by limiting what the agent can see. It is assumed that the agent can only move to the East, North-East or South-East neighbouring cell. Given that the agent is always moving forward, the 3 separate modules are implemented using a Q learning approach that is explained in the next section.

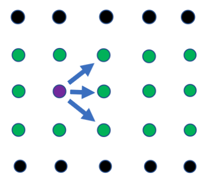

Fig. 2: Possible moves the agent can make from it's position (purple)

A. *ϵ-greedy algorithm:*

The ϵ-greedy algorithm is a probabilistic decision making algorithm. It is a classic decision making problem that allows us to control the exploration vs exploitation tendency by varying the ϵ value. Given 3 or more different possible actions, the epsilon greedy algorithm can be implemented as:

- For each trial, select a random sample θ from a uniform distribution $\mathcal{U}(0, 1)$
- if $\theta < \epsilon$ then choose a random action.
- else if $\theta \geq \epsilon$ choose the action that gives maximum reward.

Typically ϵ value is taken to be 0.2 for many applications and is tuned based on the need. Note that high values can cause instability and the algorithm to not converge to the optimal policy.

II. Method

A. *Sidewalk module:*

If a grid point is inside the sidewalk, then it has value 1 and it has value 0 if it's outside. Given this, there are 6 possible states for representing the 3 gridpoints ahead of the agent. The states 010 and 101 are not possible. A hash table (dictionary in python) can be used to map the states to their positions in the table. Since there are 3 possible actions, the Q matrix has 6x3 state action pairs. The goal of the sidewalk agent is to learn the appropriate Q values for each state action pair $Q(s, a)$. The agent gets a penalty, i.e. negative reward, for an action that takes it to a black cell. An action taking the agent to a green point (sidewalk) is given a small positive reward.

$$R(s, a) = -50(\text{penalty for moving to black}) \qquad R(s, a) = 5(\text{reward for staying on sidewalk}) \tag{1}$$

For the terminal cells, we assume a hypothetical column of entirely green points to determine s' in the Q learning algorithm. During training, the epsilon greedy 'exploration-exploitation' policy was used to explore the grid. Also since the 7x25 grid will never have the state (000), the agent was trained on a 9x25 grid to learn all the Q values in the matrix. Another thing to note is that we cannot determine the state for points along the top and bottom edge as they only have 2 neighbouring grid points. To avoid this any action that takes the agent to the top or bottom cell is prohibited/ignored. So when the agent is in row 2 or W-1, then action 1 and 3 are ignored respectively and we do a redraw for the ϵ-greedy algorithm. Thus the final Q learning algorithm for training the sidewalk agent module is:

1) Initialize 6x3 Q matrix with small values in each cell and initialize a 9x25 grid with middle 3 rows as sidewalk (1's)
2) For each episode, start from a random row in the 1st column
3) determine the state of the agent, s from the neighbouring 3 cells.
4) use ϵ-greedy algorithm on the Q matrix row for state s and determine next action, a.
5) Calculate reward, $R(s, a)$, for the chosen action and determine the corresponding state the action takes you to, s'
6) update the Q value for the state action pair (s, a) as:
$$Q(s, a) = (1 - \alpha)Q(s, a) + \alpha \left(R(s, a) + \gamma \max_{act} (Q(s', act)) \right); \tag{2}$$
where, γ is the discount factor and α is the learning rate
7) update state to s'
8) repeat steps 3-7 till agent reaches the terminal column/state
9) Iterate over multiple such episodes and monitor $\Delta Q = Q_k - Q_{k-1}$
10) when the Frobenius norm of ΔQ falls below a certain threshold, stop training the agent

Note that in the above, when the agent reaches the last column, i.e. terminal state, then we can just use $R(s, a)$ and not use $Q(s', a)$. If we think of the above algorithm as solving a system of equations iteratively (25 equations for 25 steps), then if $\gamma = 1$, the system of eqs. has infinite sols. Since we are using a $\gamma < 1$, the algorithm will converge to a certain Q but it is always better to not assume a s' for

the terminal gridpoints.
Also the algorithm above may terminate in the initial few episodes if there isn't enough exploration. So as a workaround, check for the Frobenius norm condition only after a certain minimum number of episodes have elapsed. Using $\epsilon = 0.1; \gamma = 0.8; \alpha = 0.2$, and after 500 episodes, the Q matrix converged. Finally all the rows of the Q matrix were normalized using the L1 norm for each row. The final policy used by the agent when testing is the policy of maximizing Q. Thus:

$$a^* = \Pi(s) = \max_a Q(s,a) \tag{3}$$

The performance of the agent in a 7x25 grid with the maximising policy is shown in the figure below:

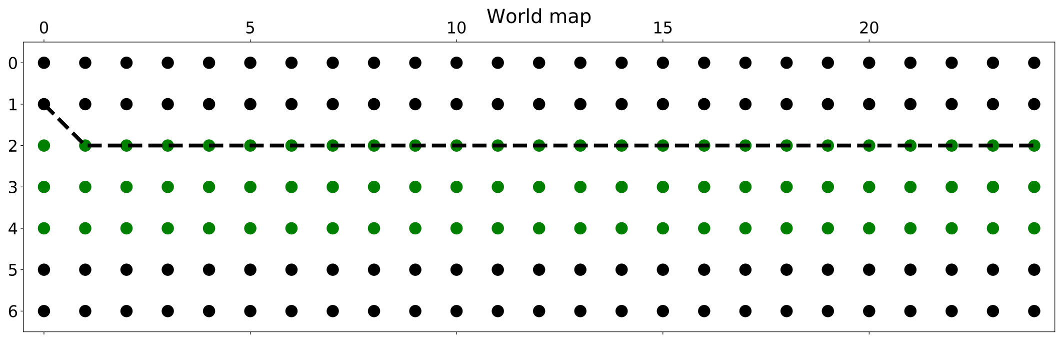

Fig. 3: Performance of sidewalk module

As can be seen the agent sticks to the sidewalk. It should be noted that the action of going to the 1st and last rows are prohibited, and the maximum Q of the remaining 2 actions is chosen.

B. *Obstacle module:*

If a grid point has an obstacle, then it has value 1 and it has value 0 if it doesn't. Given this, there are 8 possible states for representing the 3 gridpoints ahead of the agent. A hash table (dictionary in python) can be used to map the states to their positions in the table. Since there are 3 possible actions, the Q matrix has 8x3 state action pairs. The goal of the obstacle agent is to learn the appropriate Q values for each state action pair $Q(s,a)$. The agent gets a penalty, i.e. negative reward, for an action that steers it into an obstacle. An action taking the agent to any other point has 0 reward. Also between equal reward states (ex. 000), there is a slight reward (10) for the East action compared to NE and SE. This is to make sure the agent takes the path with least time.

$$R(s,a) = -100(\text{penalty for hitting obstacle}) \qquad R(s,a) = 0(\text{avoiding obstacle}) \tag{4}$$

For the terminal cells, we assume a hypothetical column of 0's to determine s' in the Q learning algorithm. During training, the ϵ-greedy policy was used to explore the grid. The agent was again forbidden from going to the extreme rows.
The training method used was the same as the one used for the sidewalk module. In this case since all states are encountered in a 7x25 grid, the same was used for each episode. A smaller grid would also be fine as long as it explores all state action pairs. For each episode a new grid with randomly placed obstacles was generated.
Using $\epsilon = 0.1; \gamma = 0.8; \alpha = 0.2$, and after 150 episodes, the Q matrix converged. Finally all the rows of the Q matrix were normalized using the L1 norm for each row. The final policy used by the agent when testing is the policy of maximizing Q. The performance of the agent in a 7x25 grid with the maximising policy is shown in the figure below:

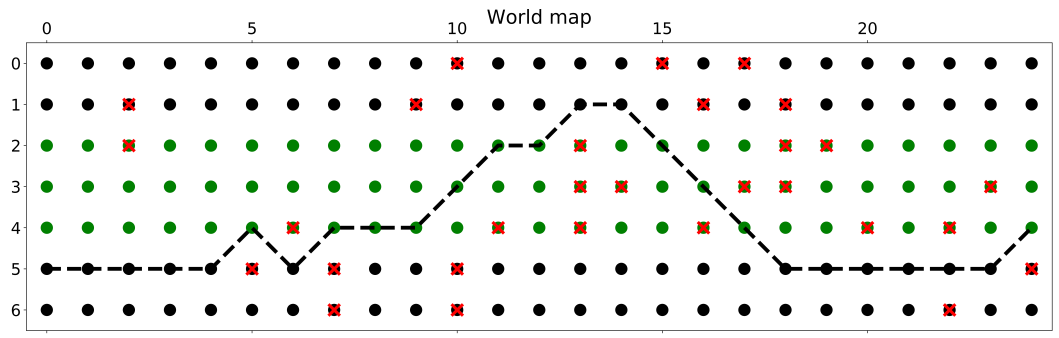

Fig. 4: Performance of obstacle module, red cross is an obstacle

As can be seen the obstacle agent module avoids any obstacle along the path and ignores the constraint of staying on the sidewalk.

C. Litter module:

If a grid point has any litter, then it has value 1 and it has value 0 if it doesn't. Given this, there are 8 possible states for representing the 3 gridpoints ahead of the agent. A hash table (dictionary in python) can be used to map the states to their positions in the table. Since there are 3 possible actions, the Q matrix has 8x3 state action pairs. The agent gets a reward, for an action that steers it towards litter. An action taking the agent to any other point has 0 reward. Also between equal reward states (ex. 000), there is a slight reward (40) for the East action compared to NE and SE. This is to make sure the agent takes the path with least time.

$$R(s,a) = +100(\text{reward for litter}) \qquad\qquad R(s,a) = 0(\text{going to empty cell}) \tag{5}$$

For the terminal cells, we assume a hypothetical column of 1's to determine s' in the Q learning algorithm. During training, the ϵ-greedy policy was used to explore the grid. The agent was again forbidden from going to the extreme rows.
The training method used was the same as the one used for the obstacle module. Again, a smaller grid would also be fine as it would explore all state action pairs. For each episode a new grid with randomly placed litter was generated.
Using $\epsilon = 0.1; \gamma = 0.5; \alpha = 0.4$, and after 100 episodes, the Q matrix converged. Finally all the rows of the Q matrix were normalized using the L1 norm for each row. The final policy used by the agent when testing is the policy of maximizing Q. The performance of the agent in a 7x25 grid with the maximising policy is shown in the figure below:

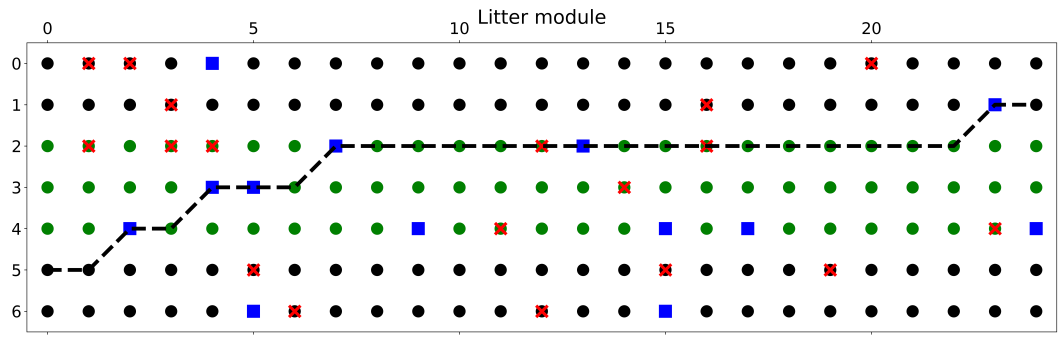

Fig. 5: Performance of litter module, blue box is litter

As can be seen the litter agent module picks up litter but goes into obstacles along the path and also ignores the constraint of staying on the sidewalk.

III. Results and Discussion

The normalized Q values of the 3 different modules were combined using a simple weighted sum. The weights for the different modules were w_o for obstacle module, w_l for litter module, and w_s for sidewalk module. The states of the agent for each module were determined at each step, i.e. s_o, for obstacle, s_l for litter, and s_s for sidewalk. Then for each action, a, the $Q_{weighted}$ values were generated as:

$$Q_{weighted}(a) = w_o \cdot Q_o(s_o, a) + w_l \cdot Q_l(s_l, a) + w_s \cdot Q_s(s_s, a) \tag{6}$$

The final policy is to choose the action that maximizes the weighted Q value, i.e.

$$a^* = \max_a Q_{weighted}(a) \tag{7}$$

The weights chosen for the different modules were, $w_s = 5; w_o = 10; w_l = 8;$. The performance of the final agent is shown below:

The weighted sum approach is able to successfully combine the 3 different modules to give a well

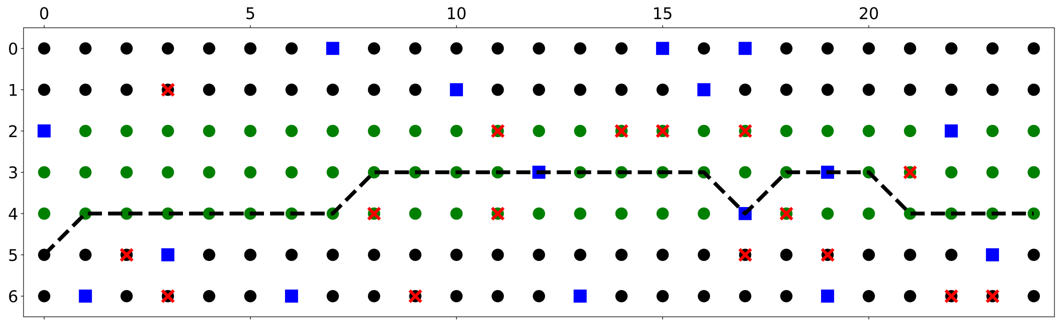

Fig. 6: Performance of combined modules

performing agent. The agent is able to stay in the sidewalk (green), pick up litter and avoid obstacles. Also note that the agent avoids litter that is not on the sidewalk, for example in the second last column and 4^{th} column of figure 6. This is desirable and if litter requirement is greater then it's weight can be increased so that agent picks litter away from sidewalk.

IV. Conclusion

In conclusion, an RL agent was trained by splitting each task into a different module. The size of the state space was reduced by limiting what the agent can see. This converts the problem into a POMDP (Partially Observable Markov Desicion Process). Hence the obtained agent is not close to the globally optimal solution. This can be seen in the fact that the agent does not end up picking the maximum possible amount of trash in the environment (fig 6). Further work can be done on improving the agent by increasing it's degrees of freedom, i.e. allowing it to move backwards/vertically to pick up litter or avoid obstacles. Also the visibility of the agent can be increased to arrive at a more optimal solution.