Report of Lab 4

for the course Neural Network and Learning system

Hao Chi Kiang (haoki222) Yumeng Li (yumli241)

March 8, 2017

Contents

Question 1 and 2: V and Q functions

Given an optimal policy, the V function is defined as

$$V^*(s_t) = \sum_{k=0}^{\infty} r_{t+k}$$

where r_{t+k} are the reward of the state $\mathrm{s_{t+k}}$. However this is generally, as the optimal policy is unknown, and we need to estimate it in practice.

The $Q(s,a)$ function is the expected future reward of doing an action a at state s and then following the optimal policy. $Q(s,a)$ is given by

$$Q(s_k, a) = r(s_k, a) + \gamma V^*(s_{k+1}),$$

and it follows that

$$V^*(s) = max_a Q(s,a)$$

Again, $Q(s_k, a)$ is unknown.

We could, however, use a loop to estimate $Q(s_k, a)$, and therefore $V^*(s_k)$. The loop can be defined by

$$\begin{aligned}\hat{Q}(s_k, a_j) &= (1-\alpha)\hat{Q}(s_k, a_j) + \alpha(r(s_k, a_j) + \gamma \hat{V}(s_{k+1})) \\ &= (1-\alpha)\hat{Q}(s_k, a_j) + \alpha(r(s_k, a_j) + \gamma \max_a Q(s_{k+1}, a))\end{aligned}$$

Question 3: Implementation details

The following is the core of our implementation. As seen, it is just the iterative estimation in the previous section coded in Matlab. The function 'my$\backslash_{\mathrm{choose}}\backslash_{\mathrm{action}}$' chooses an new random action with probability ‘eps‘, and with probability ‘1 - eps‘ it chooses the best action to take according to the ‘Q‘ matrix.

```
gwinit(mapno);
world = gwstate();
Q = rand( world.xsize, world.ysize, 4 ) * (0.3) - 0.4;
Q(1,:,2) = -Inf;
Q(world.xsize,:,1) = -Inf;
Q(:,1,4) = -Inf;
Q(:,world.ysize,3) = -Inf;

start_eps = 0.8
eps = start_eps;
died = 1;

%... Other chores for plotting, etc ...

figure('DoubleBuffer', 'on');
drawnow();

while died <= maxloop
  world = gwstate();
  pos = world.pos;

  while world.isterminal == 0
    act = my_choose_action(Q, pos(1), pos(2), allactions, eps);
    newworld = gwaction(act);
    newpos = newworld.pos; % Less reference for speed
    if newworld.isvalid == 1
      Q(pos(1), pos(2), act) = (1 - alpha) * Q(pos(1), pos(2), act) +...
                               alpha * (newworld.feedback + ...
                               gamma * max(Q(newpos(1), newpos(2), :)));
    end
    world = newworld;
    pos = world.pos;
  end
  Q(pos(1), pos(2), :) = 0;

  if mod(died, 50) == 0
    % ... Blah... Draw some animation...
    eps = max(0.2, start_eps + (0.2 - start_eps)/(0.8 * (maxloop - 1)) * died);
  end

  died = died + 1;
  gwinit(mapno);
end
```

During the loop, eps is linearly decreased every 50 episodes. Starting with 0.8, when the loop arrives its 80% of the maximal number of episodes, eps should be at 0.2; then it stops decreasing. In other words, the last 20% of the training proccess mainly focus on optimizing the already learnt strategy.

We stops the robot from going off the border by initialising the weights of the go-off-the-border actions to $-Inf$, as seen in the above code. This does not literally stop the learning process from choosing the action, but it would make the Q values of these wrong actions stay -Inf during the entire training proccess, thus they are guaranteed not the be included in learned strategy.

Question 4: Differences between the worlds explored by the robots

The first and the third world is static, meaning the robot will not be forced to move in a random direction; in other words, there is no 'random wind' in the map. This makes the training much easier, and the learned Q function converges very quickly. On the other hand, the randomness in the second and fourth world would confuses the learning processes: this forces us to use a larger number of episodes and generally smaller learning rate, so that the algorithm does not get 'confused' by the randomness too much.

Question 5, 6 and 8: Plot of the learning result

Figure 1, 2, 3 and 4 are the learned V functions and strategies for Map 1, 2, 3, and 4 respectively.

As seen, in the first map as shown in Figure 1, the learned strategy is quite good and we have used relatively few number of loops in the learning. As the intuition suggests, the robot would move away from the 'pool' if it is initialised into the 'pool', and it would avoid marching through the 'pool' if it is located in the left part of the map.

The shape of the second map is exactly the same as the first one, though random force is added to the map. Also, the penalty of 'falling' into the pool is changed to -10. As seen, the learnt result is still pretty much the same as previous. When the robot is on the left edge of the pool, it does not intentionally move away from the pool, but instead, it chooses to move along the edge for shorter path.

Map three is a static map which contains a narrow path to the goal. When the map is static, the robot chooses simply the shortest path to the goal, which a lot of time passes through this narrow path. However, in as shown in Figure 4, the robot tends to avoid this narrow path as much as possible: even if it is in the right entry of the path, it chooses to get out of the path as soon as possible to avoid being randomly blown into the pool. Since the path is very narrow, this probability quite high, and the penalty of falling into the pool is -0.5, which is equivalent to five extra 'safe steps' outside.

Question 7: What if Dijkstra shortest path

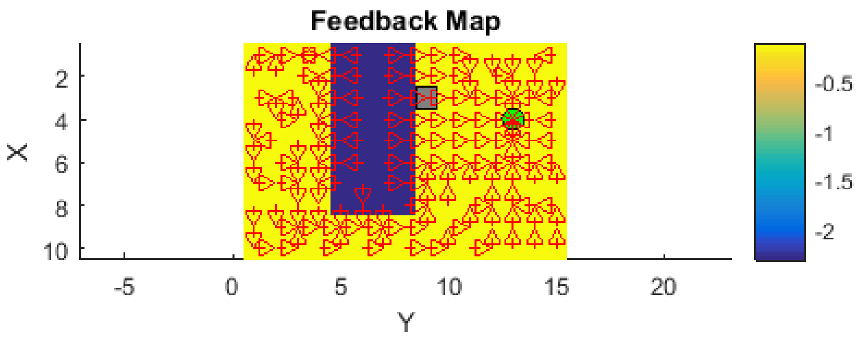
Dijkstra shortest path does not account for the randomness of the movement. Thus the robot will always tend to move along the pool, which is probably not the best thing to do when there is a random wind. However, in the static irritating world it would give a perfect result.

Question 8: Possible application of Q learning

We can probably use Q-learning to program a real robot to maybe clear the floor in some simple indoor environment, or trim grass in a football court. Other variants of reinforcement learning algorithms are used widely in some difficult-to-model problems, such as artificial intelligence.

Question 9: Conclusion

According to our experiment result, an higher discount factor will make the robot more likely to choose a shorter path despite higher risk in a randomised environment; and an lower alpha would make the training algorithm more insistent on what have been already learnt, according to the formula for iterative estimation of Q. Thus in a highly randomized environment, we do not want the alpha to be too high, because we do not want the Q matrix to be swayed too much by bad experience caused by randomness.



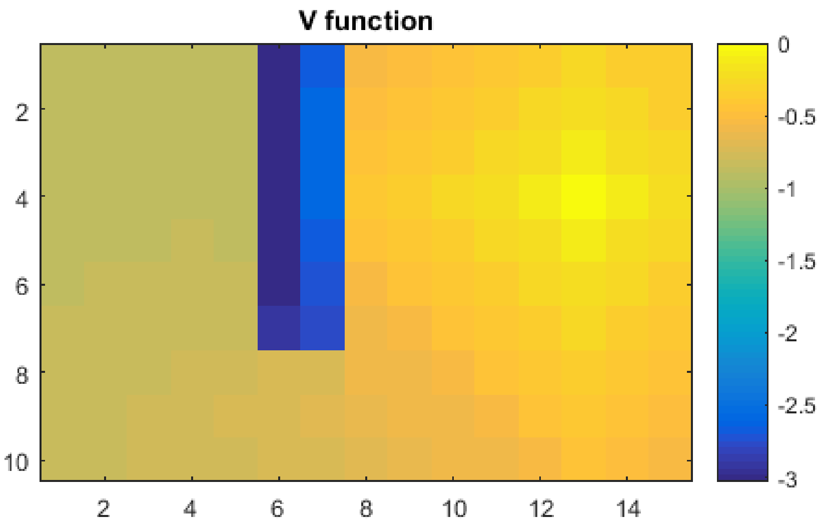
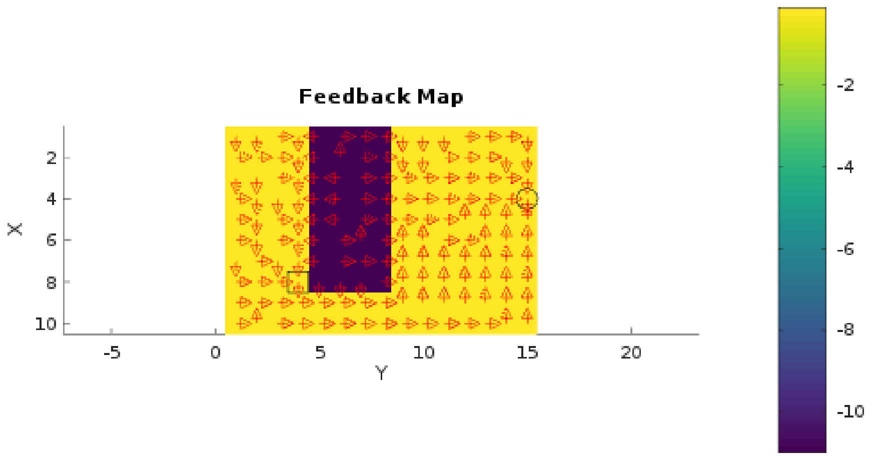


Figure 1: V function and strategy of map 1, with learning rate = 0.1, discount factor = 0.9, 1500 episodes



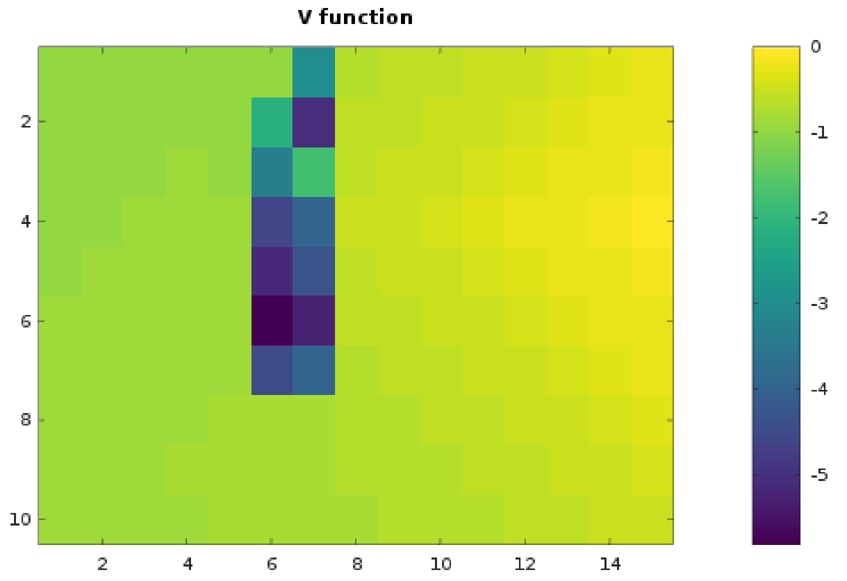


Figure 2: V function and strategy of map 2, with learning rate = 0.3, discount factor = 0.9, 2500 episodes

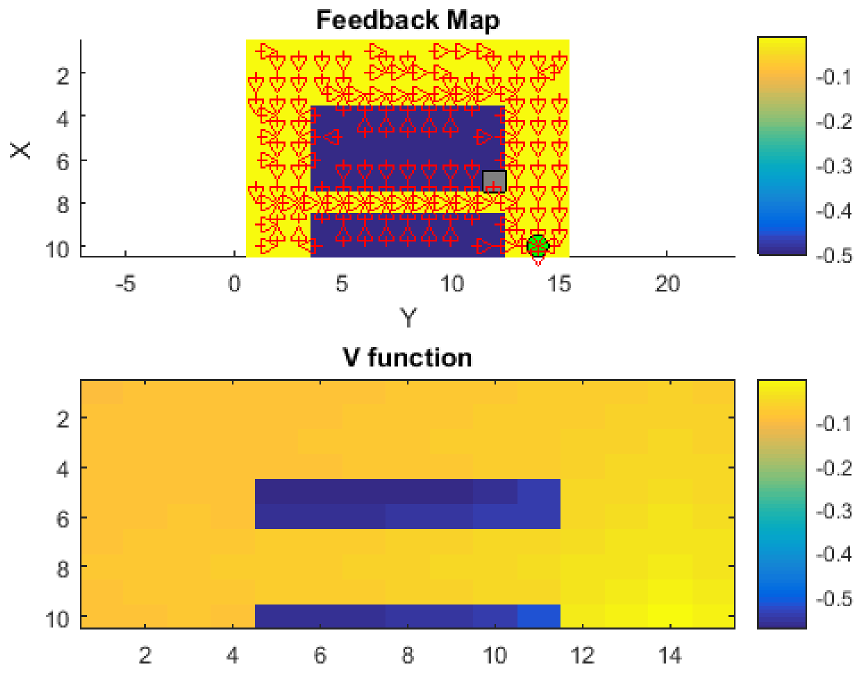


Figure 3: V function and strategy of map 3, with learning rate = 0.5, discount factor = 0.9, 3000 episodes

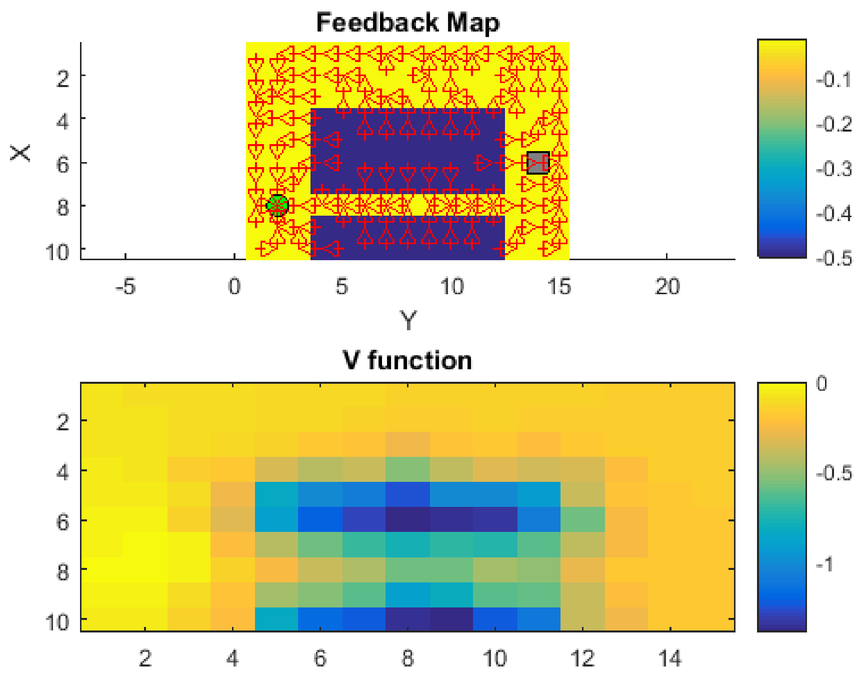


Figure 4: V function and strategy of map 4, with learning rate = 0.05, discount function = 0.95, 5000 episodes