Homework 5: Reinforcement Learning

Shun Zhang

0 Codebase

I used the codebase for CS343 Aritificial Intelligence[1], offered by Prof. Peter Stone in Spring 2012. It provides a graphic view of the gridworld, which is presented in the figures in this report. I also used the Q-Learning function I implemented at that time.

1 Q-Learning

Q-Learning is an off-policy temporal difference learning algorithm. It keeps a Q-table which evaluates each state, action pair. The idea is to update Q-values with the observed sample, with the learning rate of α, shown in the equation below.

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha(r + \gamma Q(s', a'))$$

In this assignment, I need to extract features in the domains. For example, I represent the states by the distance to the nearest obstacle in the obstacle domain (will be described in the experiment section). Let f_i be the i-th feature, and w_i be the weight of the i-th feature, the Q function can be represented as

$$Q(s,a) = \sum w_i f_i(s,a)$$

To update $Q(s,a)$, w_i needs to be updated. As $\frac{\partial Q(s,a)}{\partial w_i} = f_i(s,a)$, the update mechanism can be changed to be as follows.

$$\begin{gathered} correction = r + \gamma Q(s', a') - Q(s,a) \\ w_i = w_i + \alpha[correction] f_i(s,a) \end{gathered}$$

[1]http://www.cs.utexas.edu/~pstone/Courses/343spring12/

2 Modular RL

The task is to combine some sub-MDPs, each of which has some particular behavior, to a hybrid MDP.

2.1 Linear Combination

One easy way to mix sub-MPDs is representing the Q-value of the hybrid MDP as linear combination as sub-MDPs. That is,

$$Q(s,a) = \sum \beta_i Q_i(s,a)$$

where β_i is the weight on the i-th sub-MDP. Q_i is the Q-value of the i-th sub-MDP.

We need to hand-tune the parameter β to make Q weight the sub-MDPs reasonably.

2.2 Gibbs Softmax

The motivation of using Gibbs softmax function is to normalize the Q-values over the actions given a state.

There could be a bias in the Q-values in one sub-MDP. When such bias increase, the Q-values increase so that such sub-MDP will unnecessarily dominates. So I normalize the Q-values by Gibbs softmax function, defined as below.

$$Q'_i(s,a) = \frac{e^{Q_i(s,a)}}{\sum\limits_{a \in A(s)} e^{Q_i(s,a)}}$$

$$Q(s,a) = \sum \beta_i Q'_i(s,a)$$

where $A(s)$ is the set of available actions in the state s. Q'_i is the normalized Q-value.

3 Experiments

Modular RL is evaluated in a grid world domain, in which the agent needs to reach the targets while avoiding the obstacles. First, I train two MDPs for walking towards the goal and obstacle avoiding, respectively. Then, I combine these two MDPs for the desired performance.

3.1 Approaching-the-Goal Module

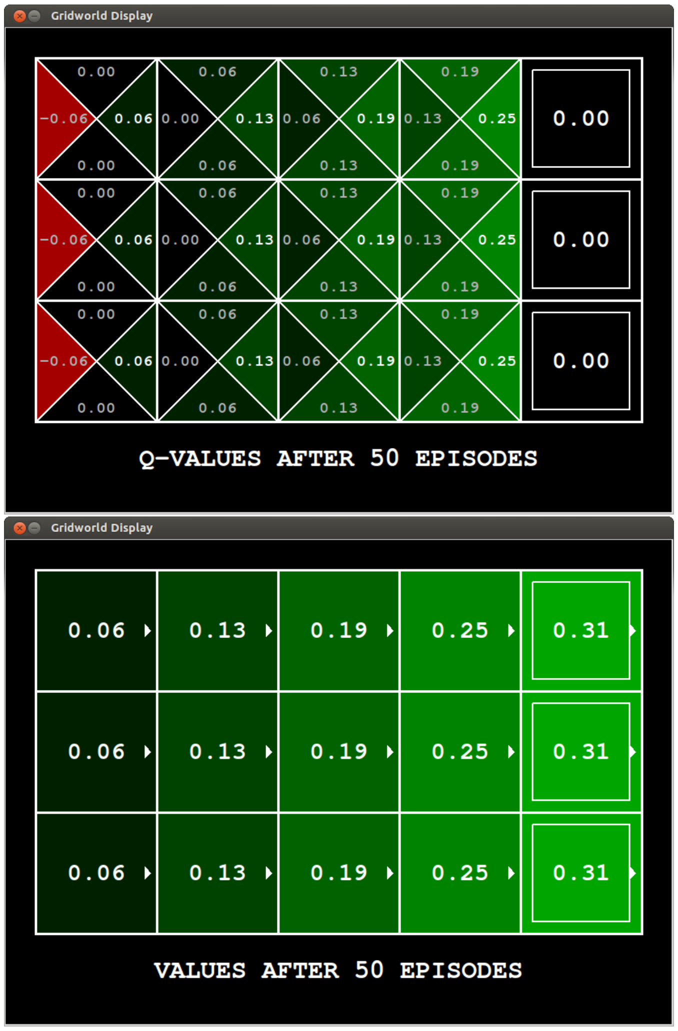


Figure 1: Approaching-the-goal module.

Figure **??** shows the learned Q-values (the upper graph), values and policy (the lower graph) for approaching the goal. There is reward of +1 of reaching the goals, which are the right-most column. $\epsilon = 0.3, \gamma = 0.5$. I keep an exploration rate larger than 0.1, so that it can explore the whole space. The discount factor γ is small here, so the agent will aim at the shortest path, instead of wasting time in the grids. I used one feature x to represent each state, which is the horizontal coordinate of the position of the agent. The learned parameter is

```
'x': 0.06250000371801567
```

3.2 Obstacle-Avoiding Module

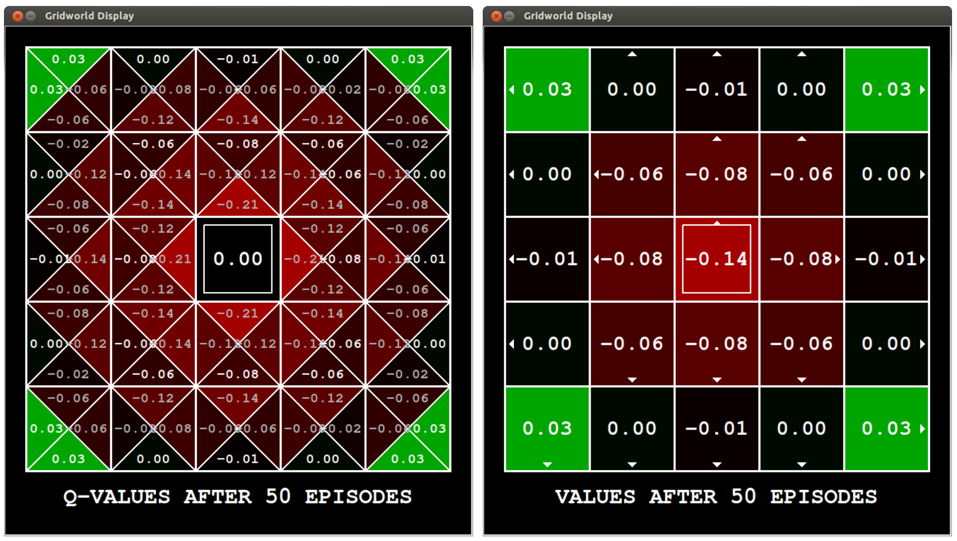


Figure 2: Obstacle-avoiding module.

In Figure **??**, similarly, I trained an obstacle-avoiding agent. There is a -1 reward in the center of the grid world. $\epsilon = 0.3, \gamma = 0.9$. There are two features, *bias*, which is always 1, and *dis* which is the distance to the obstacle. The learned parameters are

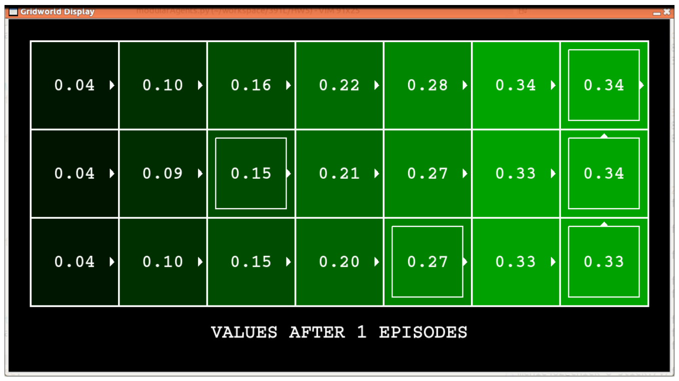
```
'bias': -0.20931133310480204, 'dis': 0.06742681562641269
```

Note that I only put one obstacle (the grid in the center). As the agent only consider the closest obstacle, so there should be no difference between using one and multiple obstacles.

3.3 Hybrid MDP: Linear Combination

For the Hybrid domain, I keep the goals at the right-most column. I randomly set some obstacles in the domain.

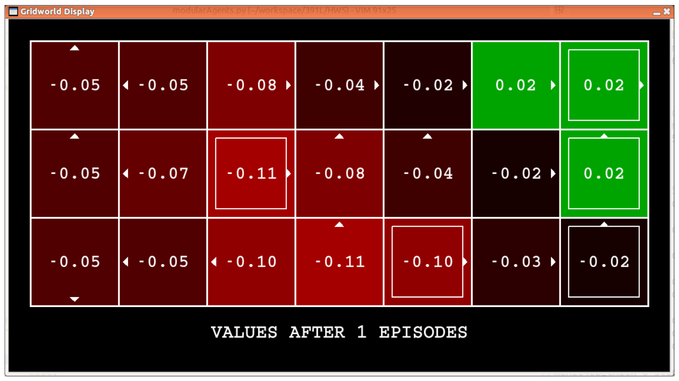
I tried some linear combination of sub-MDPs. If I overweight one module, then the behavior of that module would dominate (Figure **??**). Setting the weights to be 0.4 for approaching module and 0.6 for avoiding module



0.04 ▸	0.10 ▸	0.16 ▸	0.22 ▸	0.28 ▸	0.34 ▸	0.34 ▸
0.04 ▸	0.09 ▸	0.15 ▸	0.21 ▸	0.27 ▸	0.33 ▸	▴ 0.34
0.04 ▸	0.10 ▸	0.15 ▸	0.20 ▸	0.27 ▸	0.33 ▸	▴ 0.33

VALUES AFTER 1 EPISODES

(a) 0.9 approaching + 0.1 avoiding



▴ -0.05	◂ -0.05	-0.08 ▸	-0.04 ▸	-0.02 ▸	0.02 ▸	0.02 ▸
▴ -0.05	◂ -0.07	-0.11 ▸	▴ -0.08	▴ -0.04	-0.02 ▸	▴ 0.02
-0.05 ▾	◂ -0.05	◂ -0.10	▴ -0.11	-0.10 ▸	-0.03 ▸	▴ -0.02

VALUES AFTER 1 EPISODES

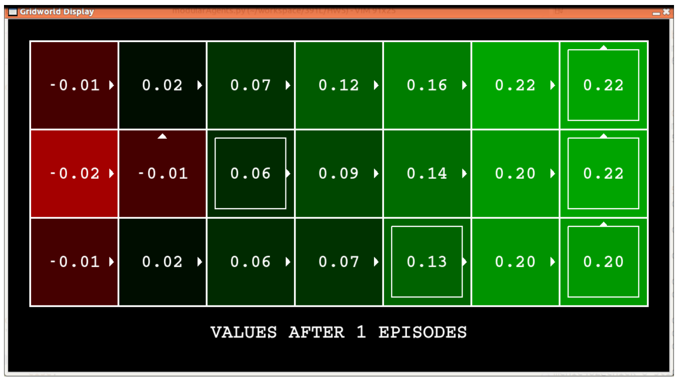
(b) 0.1 approaching + 0.9 avoiding

Figure 3: Overweighting one module makes the behavior of that module dominate.

achieves the best solution in my trials (Figure **??**). Now I observe the problem described in the instruction – the agent will walk over the obstacle near the goal, because the Q-value of approaching the goal is much higher. So I normalize them in the next section.

3.4 Hybird MDP: Gibbs Softmax Function

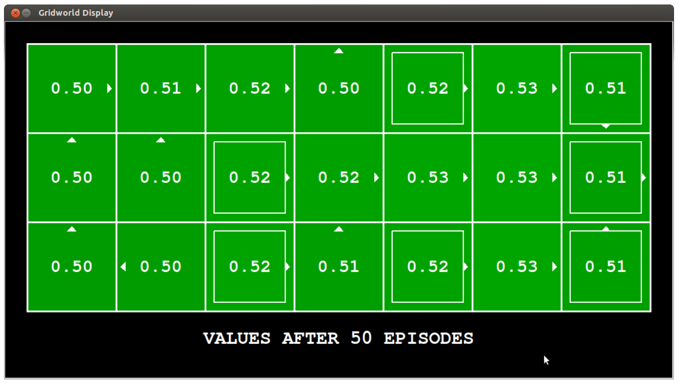
Using the method described in Section **??**, I find the results in Figure **??**. For the grid (3, 0), the Q-values of going up and down are the same, so the agent should randomly choose one. Therefore this solution is correct.



-0.01 ▸	0.02 ▸	0.07 ▸	0.12 ▸	0.16 ▸	0.22 ▸	▴ 0.22
-0.02 ▸	▴ -0.01	0.06 ▸	0.09 ▸	0.14 ▸	0.20 ▸	▴ 0.22
-0.01 ▸	0.02 ▸	0.06 ▸	0.07 ▸	0.13 ▸	0.20 ▸	▴ 0.20

VALUES AFTER 1 EPISODES

Figure 4: 0.4 approaching + 0.6 avoiding



0.50 ▸	0.51 ▸	0.52 ▸	▴ 0.50	0.52 ▸	0.53 ▸	0.51 ▾
▴ 0.50	▴ 0.50	0.52 ▸	0.52 ▸	0.53 ▸	0.53 ▸	0.51 ▸
▴ 0.50	◂ 0.50	0.52 ▸	▴ 0.51	0.52 ▸	0.53 ▸	▴ 0.51

VALUES AFTER 50 EPISODES

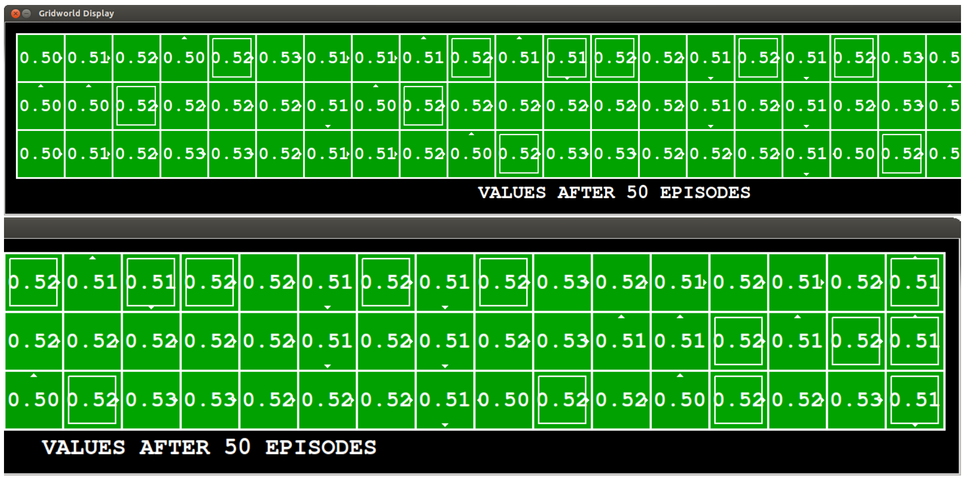
Figure 5: Using Gibbs softmax function.

3.5 On Larger Grid

I further tested this Modular MDP using Gibbs Softmax Function in a 3 x 25 grid world (Figure **??**). The policy are mostly correct.

4 Discussion and Conclusion

In my view, modular RL is one type of transfer learning. It apply the results in some predefined source tasks to the target task. Also, some learning could be allowed after combining the source tasks, rather than our hand-tuning.



0.50	0.51	0.52	0.50	0.52	0.53	0.51	0.51	0.51	0.52	0.51	0.51	0.52	0.52	0.51	0.52	0.51	0.52	0.53	0.5
0.50	0.50	0.52	0.52	0.52	0.52	0.51	0.50	0.52	0.52	0.52	0.52	0.52	0.52	0.51	0.52	0.51	0.52	0.53	0.5
0.50	0.51	0.52	0.53	0.53	0.52	0.51	0.51	0.52	0.50	0.52	0.53	0.53	0.52	0.52	0.52	0.51	0.50	0.52	0.5

0.52	0.51	0.51	0.52	0.52	0.51	0.52	0.51	0.52	0.53	0.52	0.51	0.52	0.51	0.52	0.51
0.52	0.52	0.52	0.52	0.52	0.51	0.52	0.51	0.52	0.53	0.51	0.51	0.52	0.51	0.52	0.51
0.50	0.52	0.53	0.53	0.52	0.52	0.52	0.51	0.50	0.52	0.52	0.50	0.52	0.52	0.53	0.51

Figure 6: Evaluation on 3x25 gridworld. It is one gridworld that broken into two figures.