EX0 Report

Ziyi Yang

September 2020

1 Q1

Class gridworld defined in gridworld.py, with basic features, such as map, start/goal state, get legal actions, and also all the agents of required policies (random, manual, better, worse, learning). It doesn't have user interface for now.
In order to run all scripts successfully, unzip all scripts under one folder.

2 Q2

An agent which run with manual policy.
Usage: **python Q2.py**. After run the script, input string 'u', 'd', 'l', 'r' to make the agent move up, down, left, and right.

```
choose act from ['u', 'd', 'l', 'r']
u
Pos [0 1]
reward 0

choose act from ['u', 'd', 'l', 'r']
r
Pos [1 1]
reward 0

choose act from ['u', 'd', 'l', 'r']
r
Pos [2 1]
reward 0

choose act from ['u', 'd', 'l', 'r']
u
Pos [2 2]
reward 0
```

Figure 1: Sample input&output

3 Q3

The cumulative reward of random policy is around 8-10. If we compare performance of random policy to manual policy then the manual one will significantly outperform the random one if the person who control the manual agent is an 'expert', because the performance of human expert is the ultimate goal that we are pursuing (whatever learning approach we take), and an expert would use the information of the environment perfectly. By contrast, a random agent could only pick a random action blindly, without utilize any information from the environment.

Usage: **python Q3.py**

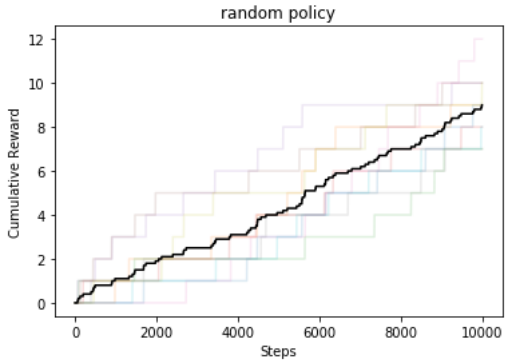

Figure 2: Cumulative rewards of random policy

4 Q4

Better policy: With probability of 0.2 choose action 'up', with probability of 0.2 choose action 'right', with probability of 0.6 choose a random action. As a result, 'up' and 'right' action will have probability of 0.35 to be choose, while 'left' and 'down' will have 0.15.
This makes the agent tend to go to the top-right corner, and thus it'll reach the goal faster than a random policy, and random action will help the agent escape if it gets 'trapped'. Even if the goal is random generated, since the agent is initialized at [0,0] which located at the bottom-left, this policy will still have better performance than a random one.
Usage:**python Q4_better.py**

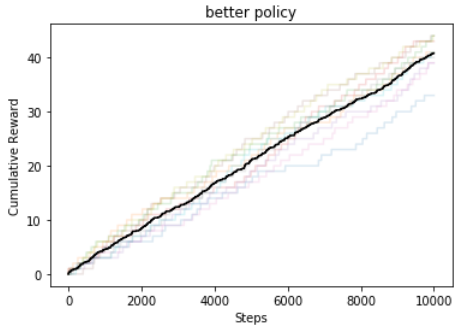

Figure 3: Cumulative rewards of better policy

Worse policy: Goes all the way to the right, which result in basically 0 reward all the time.
Usage: **python Q4_worse.py**

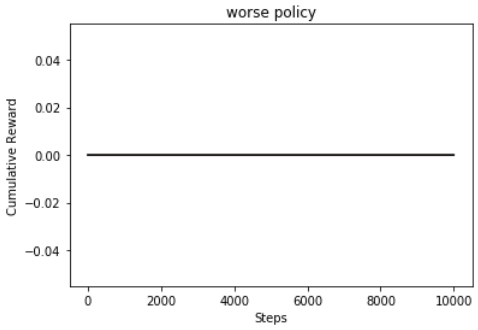

Figure 4: Cumulative rewards of worse policy

5 Q5

I implemented Q learning as my learning policy. Each action-value pairs are stored in the self.qTable and get updated every step when it receives reward or moves on to a new state with non-zero action-value pairs, which is the learning process happening here. In evaluation, the performance of learning policy varies regrading to the goal position: When goal position is [10,10], the performance of learning policy is even worse than the 'better policy', but in general the learning policy outperform all other policies, and eventually the learning policy will outperform them if given longer steps per trial.
Here I use one example which initialized goal position at [6,8]. We can see that the curve rises drastically after first 4000 steps, and the step per episode also decrease to a very low level at 30th episode. It finally achieves the highest reward among all the policies.
Usage: **python Q5.py** then run each section separately.

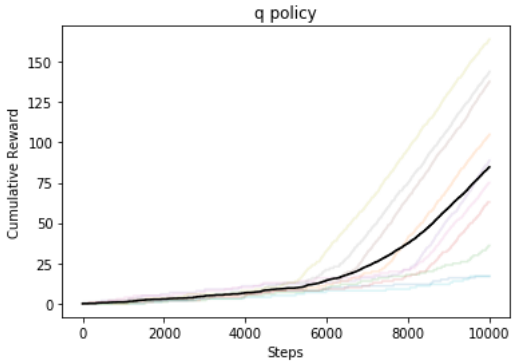

Figure 5: Cumulative rewards of Learning policy

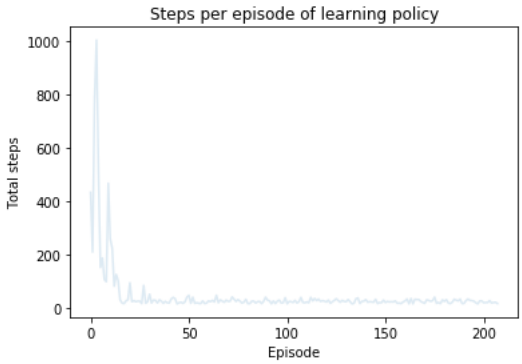

Figure 6: Steps per episode

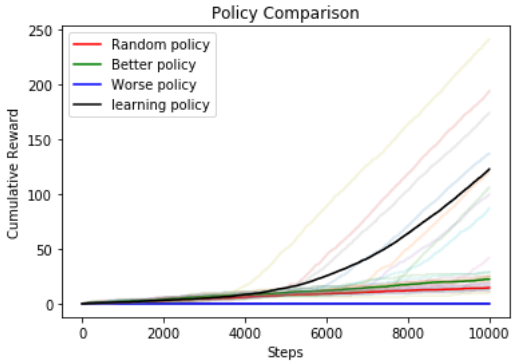

Figure 7: Policy comparison