CS4341

Assignment #4: Reinforcement learning

Due date: February 28, 2022 @ 11:59 pm

This assignment will make a gridworld. You will apply Q-learning to determine the best path to the goals from various start locations.

The world

Your agent will navigate a gridworld. Most squares have a value of 0. Squares with non-zero values are terminal states. Terminals can have a positive or a negative reward. When your agent reaches a terminal state the trial ends and your agent receives the reward value for that state as well as the reward for each action the agent took. Each action receives a constant reward (the value of this reward will be passed into the program).

Your agent can move Up, Right, Down, and Left – the same gridworld actions that we talked about in class. Similar to what we discussed in class, your agent has a chance of moving in the correct direction, and a chance of deflecting and moving 90 degrees to the left/right of the desired direction.

There is no predefined starting location; the agent will start in a random location.

What your agent knows

Your agent does not have access to the values on the map and does not know what its actions will do! It does not know that moving up from 1,1 will take it (usually) to 1,2. It does not know there is a chance its actions will not take it to the correct location.

All it has access to is its current x,y coordinates. It also implicitly knows how large the map is (i.e., you can use a fixed size array to represent the value of various actions from different states).

Your Q(s,a) function should just be a lookup table (3-dimensional array or hashmap). For each location on the map (the states), it should have an estimated value for taking each of the four possible actions.

How your agent will act

Your agent will make decisions by consulting a table that estimates the value of taking different actions from different states. This table should be updated according to the Q-learning update rule we discussed in class. Keep in mind that Q-learning means the agent does not know the environment. It does **not** know that going Right from 1,1 will usually result in winding up in 2,1. It just knows the *expected future reward for going right from 1,1*.

Since your agent does not know where the goals are, or how many there are, it will need to *explore* its environment. You should use an ε-greedy exploration with ε=0.1. Just be sure to update your Q values based on the **best** next action, not the one that was taken.

When your agent reaches a terminal node, if there is still time remaining it will be placed in a random (non-terminal) square and begin a new trial from there.

What your agent learns in one trial will be used in later trials. For example, if it learns that going right from 1,1 leads to a good terminal state and it later goes down from 1,2, it should have a high initial value for going down. Note: the agent will have a high initial value because the maximum Q(s,a) action from 1,1 has a high value, not because it has any knowledge that it is close to the goal.

Running your program

Your program should run similarly to prior assignments. It will take several command line parameters:

1. The name of the file to read in representing the map
2. How long to learn (in seconds)
3. The probability of moving in the correct direction upon taking an action
4. The constant reward for each action. For example:

```
qlearn board.txt 1.3 0.9 -0.05
```

Would run your program with board.txt for 1.3 seconds. Every time the agent moves there is a 90% chance of moving in the correct direction, with a 5% chance of deflecting 90 degrees to the left and a 5% chance of deflecting 90 degrees to the right. Each action the learner takes is worth -0.05.

Hint: we absolutely will test your program with a large negative movement penalty (similar to the demo in class), so be sure your code can handle that case. It shouldn't require any special-case processing; if your code is properly designed it should work without a problem.

You should output the policy in a human-readable fashion. One way to do so is to output each square on the grid and what action the agent should take from there. You don't need anything nearly as fancy as the demo in class.

Writeup

None this time :-)