CS 440 / ECE 448: Homework 3

Intro to Artificial Intelligence – Spring 2016

Preliminary version of code due on: Feb. 24th 11:59PM via Compass

Final version of code and written sections due on: Feb. 29th 11:59PM via Compass

---

Problem Statement

You have been assigned to operate a new pallet transport robot on the floor of a warehouse. Although the robot is capable of being operated manually from the control center overlooking the warehouse, you prefer to automate the process to spend your time on less mundane tasks. To its credit, the robot is able to automatically pick up and stack multiple loads as long as it's in the right position, but it tends to lock up in the presence of forklifts or other obstacles to prevent accidents, requiring you to manually unlock it (from the control center). You decide to train the new robot, where the robot is rewarded when it successfully transports all of its loads, and it is penalized when it locks up.

		F		S
	O		O	
S				
	O		O	
		L		

Table 1: Example warehouse floor layout

Table 1 gives an example (5×5 grid) warehouse floor layout where the robot must navigate. Instructions to the robot for movement are north (0), south (1), east (2), west (3), or just wait (4). That is, from a given location on the grid, the robot may stay put or try to move to one adjacent location. Actions taken to move in an illegal direction (e.g. into the wall) will not work and the robot will stay put in the same location (as though it waited).

Location (L) is the loading dock, where the robot picks up new pallets for stacking. The locations where pallets should be stacked are marked by (S). The robot first picks up as many pallets as there are stacking locations from the loading dock. From there, the robot may visit the stacking locations in any order, but it needs to eventually make its way to each of them (one pallet per stack). The robot is only rewarded when the last pallet it's carrying is stacked (when it makes its way to the remaining stack location with the remaining pallet). After visiting each of the stacking locations, and the robot is no longer carrying any pallets, the robot should return to the loading dock for more pallets. Returning to the loading dock before the current load of pallets are finished transporting will do nothing (except possibly make your boss suspicious). The robot initially begins from the loading dock.

There is a forklift (F) that is simultaneously operating on the warehouse floor according to its own policy, and should be avoided to prevent lockup (when their locations overlap). Locations marked by (O) contain various obstacles, and should also be avoided if possible. Unlike the forklift, where lockup is inevitable, actions to move through obstacles have a probability of locking up between 0.0 and 1.0 (again, when their locations overlap). Leaving the presence of a forklift or an obstacle does not cause lockup. Note that unlocking the robot does not count as an instruction (more as just an inconvenience on your part). Rather, upon taking an action that causes a lock up (and receiving a penalty), the robot can be treated as being immediately unlocked and ready for the next instruction.

Problem 1 : Markov Decision Processes (written)

In order to automate the robot, you decide to find the policy for the robot. For the simplified scenario shown in table 2, formulate the problem as a Markov Decision Process (MDP). For this scenario, assume that the forklift (F) does not move, and the probability of locking up when in the presence of the obstacle is p_o. The reward for successfully stacking the pallet is r_s and the penalization for locking up is r_l.

O		S
L		F

Table 2: Really simple warehouse

Specify the:

- *set of states*: $S = \{s\}$
- *set of actions*: $A = \{a\}$
- *initial distribution over states*: $S_0(s)\ ,\ s \in S$
- *transition model*: $T(s, a, s')\ ,\ s, s' \in S,\ a \in A$
- *reward function*: $R(s)\ ,\ s \in S$

Regarding the states, be clear on your indexing system. How many states are there?
Hint: there are more states than there are locations.

Problem 2 : Q-Learning (coding and written)

Having worked out the scenario in Problem 1, you realize that even in the simplified model of the warehouse, laying out (much less solving) the MDP can be tedious. Expanding the model to what is shown in Table 1 is even more daunting, as you would have to take into account multiple stacking locations, multiple obstacles, and a forklift that may visit more than just one location. Using reinforcement learning by way of temporal differences, you decide to do away with the direct modeling and attempt to make the robot learn the optimal policy as it goes.
Relevant code is provided on the course website.

Implementation

Implement a reinforcement learning agent using an ϵ-greedy Q-Learning algorithm in a simulated environment. Specifically, write a Java class that implements the `Agent` interface below:

```
public interface Agent {
  public void initialize(int numOfStates, int numOfActions);
  public int chooseAction(int state);
  public void updatePolicy(double reward, int action, int oldState, int newState);
  public Policy getPolicy();
}
```

- The `initialize` method will be called only once at the beginning. It tells the agent the number of states and the number of actions in the world. The states are identified by integers from 0 to (`numOfStates` -1). Similarly, the actions are from 0 to (`numOfActions` -1). In all cases, `numOfActions` should be equal to 5.
- The `chooseAction` method returns your selected action (an integer in $\{0, 1, 2, 3, 4\}$) given the state as an argument.
- The `updatePolicy` method processes the reward received by executing an action which transitions the agent from `oldState` to `newState`.
- The `getPolicy` method should return a `Policy` object, which specifies the action for each state. More details can be found in `Policy.java`.

We have provided a random agent (`RandomAgent.java`) as an example.

Simulation

We provide a simulation platform to run your agent. Add your agent code (e.g. `QLearningAgent.java`) into `Platform/` and compile all the java files using the command: `javac *.java` (or feel free to use an IDE).

After successful compilation, you can run the following command:

```
java Simulator <world> <agent> <steps> <episodes> <policy_output> <episode_output>
```

The simulation of a given trial consists of a sequence of episodes, given by `<episodes>`, each of which contains a number of actions the robot will perform, given by `<steps>`. The argument `<world>` chooses the scenario (e.g. `MondayMorning` or `MondayAfternoon`, described below) that the robot will be operating and learning in. The algorithm the robot uses to learn is specified by `<agent>`. At the beginning of each episode, the world is reset as:

- The robot is moved back to the loading dock to reload.
- The total reward accumulated is set to 0.
- If there is a forklift, it will be moved to a random legal position.

Problem 2 continued on next page...

To clarify, for each new trial, learning should start afresh, whereas within a trial, the learned policy should be maintained between episodes. To give an analogy, you may consider different trials as occurring in parallel worlds, and multiple episodes as consecutive days within a given world; what was learned the previous day can be used to maximize the reward received on the current day/episode. At the end of a trial, the learned policy is written to `<policy_output>`, and the records of the accumulated reward per episode is written to `<episode_output>` (you will need this for plotting).

An example command with arguments might then be:

```
java Simulator MondayMorning QLearningAgent 5000 1000 policy.txt episodes.txt
```

We also provide a simulator (with simple graphical output) for your resulting policy, named `PolicySimulator`. This robot takes a learned `<policy_file>` (from simulating an agent) and takes `<steps>` number of actions according to this policy in the specified `<world>`. You can run it using the following command:

```
java PolicySimulator <world> <policy_file> <steps>
```

For example:

```
java PolicySimulator MondayAfternoon policy.txt 200
```

Scenario and Testing

The following are the two different scenarios that you will be training the robot in.
Unless noted otherwise, set the parameters of the agent as:

- Discount factor: $\gamma = 0.95$
- Learning rate: $\alpha = 0.1$
- Exploration: $\epsilon = 0.1$

a) Scenario 1: Monday Morning

There's a mess on the warehouse floor, and the forklift operator hasn't shown up yet.

		O	O	S
	O			
S		O		O
O	O	O		
		L		

Table 3: Monday Morning

i) With the default parameters, simulate 10 trials of 1000 episodes of 5000 steps each. Plot the total reward accumulated at the end of each episode (averaged over the 10 trials) showing how the agent improves with training.

ii) Change the discount factor to $\gamma = 0.9$. Again, simulate 10 trials of 1000 episodes of 5000 steps each, and plot the accumulated reward. What differences do you observe in the learning with respect to the default parameters?

Problem 2 continued on next page...

b) Scenario 2: Monday Afternoon

The mess has been cleaned up (a bit). The forklift operator is now here and randomly moves left and right along the middle row.

S			O	S
		O		
F	↔	F	↔	F
O	O			
		L		

Table 4: Monday Afternoon

i) With the default parameters, simulate 10 trials of 1000 episodes of 5000 steps each. Plot the total reward accumulated at the end of each episode (averaged over the 10 trials) showing how the agent improves with training. What are the differences you observe in the learning between the two scenarios (when using the default parameters)?

ii) Change the learning rate to $\alpha = 0.2$. Again, simulate 10 trials of 1000 episodes of 5000 steps each, and plot the accumulated reward. What differences do you observe in the learning with respect to the default parameters?

iii) Return the learning rate to its default value, and change the exploration to $\epsilon = 0.05$. Again, simulate 10 trials of 1000 episodes of 5000 steps each, and plot the accumulated reward. What differences do you observe in the learning with respect to the default parameters?

Submission

For the coding portion, submit the Java source code for the implemented agent (just the one file) with the parameters set to their default values. Please submit a preliminary version of the code to compass by Feb. 24^{th}, and the final version of the code by Feb. 29^{th}.

For the written portion, provide the five plots (label them), alongside your answers to the questions.