Homework 3 - Reinforcement Learning

Introduction

This exercise continues with the drone delivery company, but this time the drone must navigate an unknown environment under RL settings. There is no client, only a stationary place to drop the packages off.

Environment Settings and Dynamics

Input parameters

The environment is initialized with a parameter dictionary with: ‘map’, ‘drone_location’, ‘target_location’, ‘packages’ and ‘success_rate’ as keys.

Map

- The world map is represented as a N x M grid (tuple of tuples) containing strings that describe cell attributes.
- Each square in the grid is passable (marked ‘P’) or impassable (marked ‘I’).
- Each square in the grid may contain wind which may blow, pushing the drone away.
    - This is marked by ‘WU’ (wind up), ‘WD’ (wind down), ‘WR’ (wind right), ‘WL’ (wind left) or ‘NW’ (no wind)
- For example, for a 2 x 2 map we have:
  `map = (('P_NW', 'I_NW',), ('P_WU', 'P_WL',))`

Drone Location

- The initial location of the drone is described by a 2-tuple holding the x, y location of the drone or the string ‘random’ which marks that the initial location of the drone can be in any passable location in the map.
- For example:
  `drone_location = (0,0)`
  `drone_location = ‘random’`

Target location

- In the map, a fixed location where packages should be delivered.
- For example:
  `target_location = (1,1)`

Packages

- A list of tuples, each tuple contains: package name (A,B,C,D…) and a location (a square in the map or inside the drone).
- For example:
  `packages = [('A', (0, 1)), ('B', (0, 1)), (‘C’, ‘drone’)]`

Success Rate

- A float in [0,1] which represents the probability of an action drone to succeed. For details see section below.
- For example:
  `success_rate = .9`

Environment Dynamics

Each turn, the environment outputs an observation (obs0) which is passed to the drone agent who in turn plots an action. The action is passed back to the environment to process and output a result: new observation and a reward.

Observation

- A dictionary with: ‘drone_location’, ‘packages’, ‘target_location’ as keys.
- The drone is to decide on its next action, reasoning over its experience and current observation, i.e. it doesn’t have direct access to the actual environment map or any other environment parameter.

Action

The available actions are: reset, wait, pick, move_up, move_down, move_left, move_right and deliver. Each action nature is basically described by its name. See code for more details.

Environment Step

Here we briefly describe the environment dynamics. A more precise description is available in the DroneEnv.step method.

Step Process

1. Basic sanity check (the input action is the action space)
2. Action step:
   In case action is ‘reset’:
   reset the environment and return the corresponding reward
   Else: execute a digital coin toss to decide whether the action will succeed:
   a. Successful action (with Pr = success_rate):
      action is executed (if action conditions apply)
      i. Moving to an inaccessible location results in nothing

   b. Unsuccessful action (with Pr = 1 - success_rate):
      a random action is drawn and executed.
3. Reward collection:
   based on the state and action. Basically -10 for ‘reset’, +100 for each successful delivery, +1 for successful pickup and -1 for all the rest. See drone_env.py for more details.
4. Wind effect:
   If the drone is in a windy cell, it may be pushed 0, 1 or 2 places toward the wind direction with different probabilities.

Step Result

The step results with a new observation, a reward and a ‘done’ parameter which signals an environment reset.

Trainer

The trainer class is implemented in trainer.py. It’s a simple class that combines both the environment and the agent. It’s main method is ‘run(nr_episodes, train)’ which its process are briefly described below:

1. Initialize the environment and get the first observation.
2. Repeat X number of episodes:
   a. action = drone.select_action(..)
   b. Execute action and get obs1, reward, done parameters.
   c. Update drone experience and add reward to sum
   d. If episode is over:
      i. Report sum of rewards
      ii. Reset the episode variables
      iii. (note that there’s no automatic environment reset)

For the precise implementation please review trainer.py in the provided code.

Execution

The execution in ‘check.py’ consists of the following steps:

1. For each input do:
   1.1. Initialize environment (already implemented)
   1.2. Initialize Agent(n,m) where n,m represent the size of the map (To be implemented)
   1.3. Initialize trainer (already implemented)
   1.4. Train for 200K episodes
   (Here, the drone explores its surroundings freely for a fixed number of episodes - each episode consists of 30 steps)
   1.5. Evaluate 10K episodes and save score
   (Here, the drone tries to collect as many rewards as possible)
   1.6. Return to 1.

The Task

Code-wise, your task is to implement an agent (class DroneAgent in ex.py file only) with the following methods. Your agent should achieve as many rewards as possible in each episode.

- __init__(self, n, m)
  A constructor to be written per your design decisions.
  Do not change the self.mode field!
- select_action(self, obs0)
  As the name suggests. See observation section. Returns a string from the action space defined in the actions section.
- update(self, obs0, action, obs1, reward)
  This method is called for each "obs-action-obs+reward" step. You should use the input to learn about the environment model.
- train(self) and eval(self) are already implemented and should not be modified.
- Other methods can be written per your design decisions, just make sure you don't exceed the (pretty short) episode time limit.

Your code needs to meet the following timeouts:

- 4e-3 seconds per episode
- 1 second for agent constructor

Any run that will exceed these timeouts will result with a score of -50 (see grading).

Code Handout

Code that you receive has 3 files:

1. ex3.py - the only file that you should modify, implements your agent
2. check.py - the file that implements the check process.
   * This is the file that you should run.
3. trainer.py - here the Trainer class is implemented.
4. drone_env.py - here the DroneEnv class is implemented.
5. inputs.py - list of inputs to check your code.

Submission and Grading

You are to submit **only** the file named ex3.py as a python file (**not zip, rar, etc.**). We will run check.py with our own inputs, and your ex3.py. The check is fully automated, so it is important to be careful with the names of functions and classes. The grades will be assigned as follows:

- **80%** - if your code finishes solving the test inputs score > -10
  - Note that in case of timeout, the resulted score is -50
- **20%** - Grading on the relative performance of the algorithm, based on points
- There is a possibility to earn bonus points by reporting bugs in the checking code.
  The bonus will be distributed to the first reporter.

- The submission is due on 20.1.22, at 23:59
- Submission in pairs/singles only.
- Write your ID numbers in the appropriate field (‘ids’ in ex3.py) as strings. If you submit alone, leave only one string.
- The name of the submitted file should be “ex3.py”. Do not change it.

Important Notes

- You are free to add your own functions and classes as needed, keep them in the ex3.py file.
- Make sure your code doesn’t hit the timeout limitation!
- We encourage you to start by implementing a simple working system - add the fancy stuff later.
- We encourage you to double-check the syntax of the actions as the check is automated.
- You may use any package that appears in the standard library and the Anaconda package list, however, the exercise is built in a way that most packages will be useless.