MSIT6000F 2019 Fall Semester Assignment #1

Date assigned: Friday, Sep 13, 2019

Due time: 23:59pm on Sunday, Sep 29, 2019

How to submit it: Submit your written answers as a pdf file on canvas.ust.hk. Submit your code for the last three programming questions as a zip file named YourStudentID.zip

Penalties on late papers: 20% off each day (anytime after the due time is considered late by one day)

Problem 1. (10%) Consider a 10x10 grid without any obstacles, and a robot with the same specification as our boundary following robot: eight sensors and four actions. Design a reactive production system to control the robot to go to one of the four corners, wherever its initial position is. Write a production system just for this task without calling the boundary following production system in the lecture note.

Problem 2. (10%) Which boolean function does the following TLU implement? The TLU has five inputs. Its weight vector is $(1.1, 3.1, -1, -2, 0.5)$, and the threshold is 1.

Problem 3. (Programming) (30%) Design and implement a genetic programming system to evolve some perceptrons that match well with a given training set. A training set is a collection of tuples of the form $(x_1, ..., x_n, l)$, where x_i's are real numbers and l is either 1 (positive example) or 0 (negative example). So for your genetic programming system, a "program" is just a tuple $(w_1, ..., w_n, \theta)$ of numbers (weights and the threshold). Answer the following questions:

1. What's your fitness function?
2. What's your crossover operator?
3. What's your copy operator?
4. What's your mutation operator, if you use any?
5. What's the size of the initial generation, and how are programs generated?
6. When do you stop the evolution? Evolve it up to a fixed iteration, when it satisfies a condition on the fitness function, or a combination of the two?
7. What's the output of your system for the provided training set `gp-training-set.csv` in `project1.zip`?

Problem 4 (Programming) (20%)

This is a simple exercise to implement our boundary following production system in the Pacman game from UC Berkeley. The goal of the Pacman game is simple: control the Pacman (yellow circle) to eat all the foods (white dots) in the map, as shown in Figure 1.

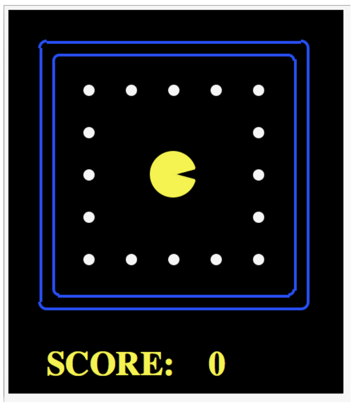

Figure 1: The `smallMap` map

0.1 Environment Setup

Download `project1.zip` on Canvas and unzip it. Open your terminal, navigate to the `pacman` folder and run `python pacman.py`. You will see the graphical interface of the Pacman game pops out, and you can play the game with your keyboard. If the game works normally, you can proceed to the next step.

0.2 Perception and Movement

Now let's learn how to design a Pacman agent. We have implemented a `NaiveAgent` for your reference, which is an agent that goes west until it can't. Run `python pacman.py --layout testMap --pacman NaiveAgent` to how the `NaiveAgent` acts. Note that the `--layout` parameter specifies the map, and the `--pacman` parameter specifies the agent. You will see the `NaiveAgent` goes from east to west in the `testMap` map (Figure 2), eats a food, and wins the game.

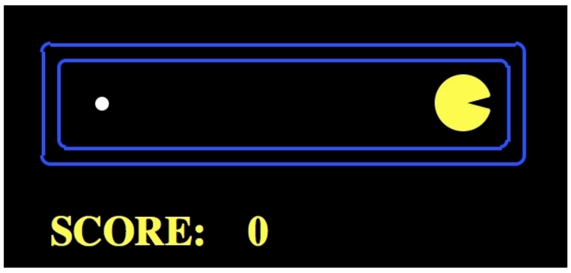

Figure 2: The `testMap` map

The Pacman has eight sensors in eight directions: northwest, north, northeast, east, southeast, south, southwest, and west. The sensor will return 1 if there is a wall in its direction, and 0 otherwise. You can use the `getPacmanSensor` function of the `GameState` class to get the list of return values of the sensors in the order described above. For example, if the `getPacmanSensor` function returns `[0, 0, 1, 1, 1, 0, 0, 0]`, it means that there are three walls in the northeast, east and southeast of the Pacman.

The movement of an agent is controlled by the `getAction` function, which is supposed to return one of the five actions at each step: NORTH, SOUTH, EAST, WEST and STOP. Open `reactiveAgents.py` to see the implementation of the `NaiveAgent`: it always returns WEST until it sensed that there is a wall on its west.

0.3 Task

Below the `NaiveAgent` class in `reactiveAgents.py` you will see the `PSAgent` class, which is incomplete. All you need to do in this problem is to modify `reactiveAgents.py`, specifically the `getAction` function of the `PSAgent` class, to make the `PSAgent` works correctly. You can run your agent in the `smallMap` map for testing.

0.4 Marking Scheme

We will test your agent with four maps (5 points each). We say that your agent passes a test case if it wins the game, i.e. has eaten all the foods in the map. We will place foods along the boundary of each map, so if your agent is able to follow the boundary, it should also be able to eat all the foods and win the game.

0.5 Notes

- If you have no experience on UNIX or Python before, please go through this tutorial first: `http://ai.berkeley.edu/tutorial.html`.
- The Python version is Python 2.7.
- There will be no tight spaces (see lecture slides for definition) in the test cases.
- The order that your agent visits the foods does not matter.
- If your agent made an invalid move (e.g. move towards the wall), the program will throw an exception and exit. Be careful!

Problem 5 (Programming) (30%)

This problem also requires you designing a boundary-following agent, but you need to use the error-correction procedure to learn the action functions this time.

0.6 Task

We have prepared four training sets for learning when to move north, east, south, and west, respectively. Each vector in the training set is in the form of $(s_1, ..., s_8, d)$, where the first

8 elements are the Pacman's sensor readings, and d is the label of this input, with $d = 1$ meaning a positive, and 0 a negative example.

You are required to learn a perceptron for each of the actions using the error-procedure. You can use any programming language for doing this.

1. Give the boolean expression that corresponds to your perceptron for the move north action.
2. Use your learned perceptrons to implement the `ECAgent` in `reactiveAgent.py` in this way: if exactly one of them outputs 1, then do the respective action; if more than one of them output 1, then use the following order: prefer going north, followed by east, south, and west; and finally if none of them outputs 1, then go north. We'll test your controller implemented in `reactiveAgent.py` on four maps, like for Problem 4.

You can again find the training sets named `north.csv`, `east.csv`, `south.csv`, and `west.csv` in `project1.zip`.