**Virtual Pi2Go Programming: Introduction to Machine Learning**

**AIM:** After completing this worksheet you should be able to explain the basics of simple Machine Learning using rewards and program a reward table which matches states and actions to average rewards.

**You Need:** To complete this worksheet you need to have a virtual Pi2Go simulator (see WS1), understand how to control the robot’s motors and sensors (WS3&WS4), be able to use files to store Programs (WS5), use control structures (WS7-10), Data Types (WS12-14), functions (WS16) and the time module (WS6).

**If the simulator isn’t already running: Start the Simulator, Select the Pi2Go Simulation and oval.xml, then start IDLE (open a *new IDLE window* if you have used IDLE to start the simulator).**

In the past few worksheets you have been programming the virtual Pi2Go to take a random action and then you have been giving it a reward. This concept of actions and rewards is the basis of one form of machine learning. In this type of learning, the robot slowly learns over time which actions give it the best rewards and then uses this information to select actions.

Over the next few worksheets we are going to attempt to program the virtual Pi2Go so that it learns how to drive around the edge of the oval in **oval.xml** world.

So far, we have only given rewards to actions based on how many of the line sensors can detect black beneath the sensor.

**Question 1:**Do you think just knowing which action gets the best reward in any situation is the best way to calculate what action should be taken? Explain your answer.

**States:** We are giving rewards based on the values returned by the two infra-red line sensors. At the end of WS21, we represented this as a tuple containing two values. We will refer to this tuple as the *state*. We have the following table of states and rewards using a dictionary:

|  |  |
| --- | --- |
| State | Reward |
| (1, 1) | 1 |
| (1, 0) | 2 |
| (0, 1) | 0 |
| (0, 0) | 1 |

We also want to associate our reward calculations with states. That is, we want to know what is the best action to take in some state. We therefore want to start out our program with some default reward dictionary like the following:

reward\_dictionary = {((1, 1), "forward"):0, ((1, 1), "backward"):0, ((1, 1), "left"):0, ((1, 1), "right"):0, ((1, 0), "forward"):0, ((1, 0), "backward"):0, ((1, 0), "left"):0, ((1, 0), "right"):0, ((0, 1), "forward"):0, ((0, 1), "backward"):0, ((0, 1), "left"):0, ((0, 1), "right"):0, ((0, 0), "forward"):0, ((0, 0), "backward"):0, ((0, 0), "left"):0, ((0, 0), "right"):0}

However it is quite tedious to create a dictionary like this by hand, so we will use a function to do it.

**Exercise 1:** Write a function that will return the action-reward dictionary above by using nested loops to generate each entry.

**Hint:** The function range(a, b) will return a list of all the numbers between a and b including a and not including b. So range(0, 2) returns the list [0, 1].

**Exercise 2:** Modify the function so it will take a list of actions and the default starting reward as input and return the dictionary that pairs each those actions with a state pair.

**Exercise 3:** Write a program that takes actions at random. It records the state of the system before the action was taken and after the action (actions should run for 3 seconds) was taken. It then calculates the total reward earned over time for each action in each state. After 50 actions it stops and prints out the average reward for each action in each state.

**Watch Out:** Watch out for division by zero when calculating averages. Your program should print out a “never attempt” message if it has never attempted some action in some state.

**Question 2:** Have you got any state-action pairs that have not been attempted? Y/N

Try re-running your program where you move the robot so it starts a long way outside the oval, and so it start in the black near the edge of the oval.

**Question 3:** Does the initial placement of the robot make a difference to the number of state-action pairs that have not been attempted?



University of Liverpool, 2019

This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/).