Mission 1

Adam Morris – Computational Social Cognition Bootcamp, July 2017

Your RL agent enters the casino. It is sparse and dimly lit. In the center of the room, under a white spotlight, there is a single slot machine.

In this mission, you will simulate an RL agent playing a simple slot machine with two arms. The left arm always wins you a dollar (a reward of 1), and the right arm always loses you a dollar (a reward of -1). Unfortunately, the RL agent does not know this, and must learn which arm to choose from experience.

Part A

Simulate a Q-learning agent playing this slot machine for 100 rounds. On each round, record the choice it makes and the reward it gets. (To generate a policy from the learned Q values, use a softmax decision function. You should set the agent's learning rate to 0.01, and the inverse temperature of the softmax function to 10.) Plot its choices and rewards over the 100 rounds.

To assist you with plotting, I've written a function called `plotResults`. You can download it here: http://mprlab327.webfactional.com/amorris/topsecret/plotResults.m. To use it, store your results in a 100 x 2 matrix, where each row is a different round, the first column is the agent's choice on that round (left arm = 1, right arm = 2), and the second column is the reward received on that round. Then, call: `plotResults(<name of your results matrix>, <mission number>)`. For example, if your results matrix is called `results`, then you would call: `plotResults(results, 1)`. This will automatically generate the plots for you. (If you want to plot the results yourself you can, but this function will save you time and allow you to focus on the important content of your missions.)

How quickly does the agent learn to choose the correct arm? Experiment with different learning rates and inverse temperatures. What effect does each parameter have on the agent's performance? How do you think people would perform in this task?

Part B

In part A, the rewards were deterministic – the left arm gave exactly 1, and the right arm exactly -1. Now, make the rewards stochastic. There's different ways to do this (feel free to experiment!), but the easiest way is to add Gaussian noise to the reward. Each time the agent pulls one of the arms, add a random normally distributed variable to the reward (with standard deviation of 0.2). How does this affect the agent's performance? Experiment with different standard deviations.