AI Programming (IT-3105) Project # 3: Reinforcement Learning for the Mountain Car Problem using Coarse Coding

Due date: Thursday, May 6 12:00 (noon)

Purpose:

- Tailor your existing Reinforcement Learning (RL) system to handle coarse coding.
- Use that system to solve the classic Mountain Car control problem.
- Gain experience producing a short, educational video that documents your system and its results.

This is strictly a group project, with groups of size 2 to 4.

1 Assignment Overview

You will extend your RL system to handle a more intricate representation for system states known as *coarse coding*. This state encoding proves particularly useful for control problems involving real-valued variables. One classic in the RL literature is the *Mountain Car* problem, which has a simple two-variable state and only 3 possible actions but which requires a relatively sophisticated strategy (i.e. policy) involving a fine resolution of each of the two state variables.

This project includes both a) the improvement of your RL system and b) the presentation of what you've learned, what you've built, and what your system has achieved, all in a 5-minute educational video.

2 The Mountain Car Problem

This is a challenging (yet *toy*) control problem involving a cart on a simple landscape, As shown in Figure 1, the cart sits at the bottom of a one-dimensional valley, and the goal is to move the cart to the top of the right side of that valley. Unfortunately, the cart has very limited power and cannot simply drive up the steep slope. It must use the left and right slopes to help build enough momentum to make it all the way up the right side. Hence, the solution involves a *sloshing* motion whereby the cart slides back and forth between the two slopes while supplying just enough power to get a little higher with each oscillation.

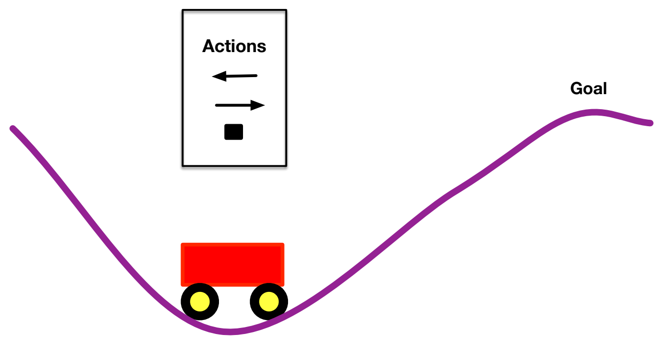

Figure 1: The Mountain Car problem.

At each timestep, your RL system must choose between three actions: 1) apply a small force (-1) that accelerates the cart to the left, 2) apply a small force (+1) that accelerates it to the right, and 3) apply no force (0).

The state of the cart is defined by two variables, location (x) and velocity ($\dot{x}$). The dynamics of the system involve the following update equation for velocity:

$$\dot{x} \leftarrow \dot{x} + (.001)F - (.0025)cos(3x) \tag{1}$$

with the lower and upper bounds on velocity set at -0.07 and 0.07, respectively, and with F being the most recent action (-1, 1 or 0). Location is then updated as:

$$x \leftarrow x + \dot{x} \tag{2}$$

with the lower and upper bounds for x being -1.2 and 0.6, respectively.

Although not needed for computing the dynamics, the relationship between x and height (depth) is given by:

$$y = cos(3(x + \frac{\pi}{2})) \tag{3}$$

This equation comes in handy when plotting the mountain landscape (i.e. y) as a function of x.

In the classic version of the problem, each episode begins with x randomly chosen in the range [-0.6, -0.4] and velocity initialized to zero. An episode should halt when either a) the cart successfully reaches the rightmost point, where x = 0.6, or b) 1000 actions have been taken. You must use this classic version in your movie, although you are free to use other versions as additional material.

3 The Reinforcement Learning System

Your RL system should use the basic SARSA algorithm, with an action-value function, Q(s,a), implemented as a simple neural network. A network with no hidden layers actually works for this problem. You are not required to implement eligibility traces in these networks, although you are welcome to do so.

A typical solution involves mean squared error as the loss function for your critic network, which takes the state-action pair (s,a) as input and produces the evaluation, Q(s,a), while the target value is $r + \gamma Q(s', a')$, where:

s' = the state attained when doing action a in state s.

a' = the action selected (by the current policy) to perform in state s'.

r = the reward received in going from state s to s'

γ = the standard RL discount factor

Since this is a non-trivial, continuous-valued control problem, it requires a more complex encoding of system states. In this project, you must use *coarse coding*, which is briefly discussed in Sutton's 1996 paper (available on the IT-3105 *Materials* web page under *Coarse Coding for RL*). A more complete description of coarse coding and its use in solving the Mountain Car problem can be found in Sutton and Barto's Reinforcement Learning book (2nd edition).

You can restrict your coarse codes to regular rectangular grids, but you need to employ multiple tilings as a vital aspect of this assignment, even though their clear advantages may not come forth in this relatively simple problem.

You will use on-policy RL for this assignment, with the policy based directly on the Q(s,a) values from the neural network and making an ϵ-greedy choice among the three possible actions during each timestep.

4 Diagrams

It is **strongly** recommended that the majority of your visual material is in pictoral form. Images often convey more useful information, more efficiently, than text, and this becomes very important with the 5-minute time restriction. You may decide to draw these images freehand on an online whiteboard, or you may choose more conventional diagramming tools, or a combination. Any of these approaches can serve your purpose. Many educational blogs get excellent results using freehand whiteboard sessions with a few accompanying PDF diagrams.

It is explicitly **forbidden** to use diagrams that are not your own in your video. Violation of this rule will incur a score of ZERO for the project. Copying of images is a matter that is taken very seriously in the media – in many cases it cannot be done without formal permission from the original source. In this project, part of your challenge is to produce diagrams that deliver your own message in the clearest and simplest form.

5 Movie Contents

Your movie must not exceed 5 minutes in duration. Violations of this constraint will incur a major point loss. A key challenge of this project is to concisely and accurately present the content described below.

Your movie must include the following:

1. A section describing how the techniques used in this project compare to those used in both the first and second projects.
2. A description of the mountain-car task.
3. An explanation of coarse coding, its basic role in RL, and how your group used it for the mountain car problem.
4. An overview of your RL system. Diagrams are much preferable to code.
5. The results of a few runs of your system. Feel free to run mountain-car animations at high speed, which matplotlib's FuncAnimation (see below) supports !

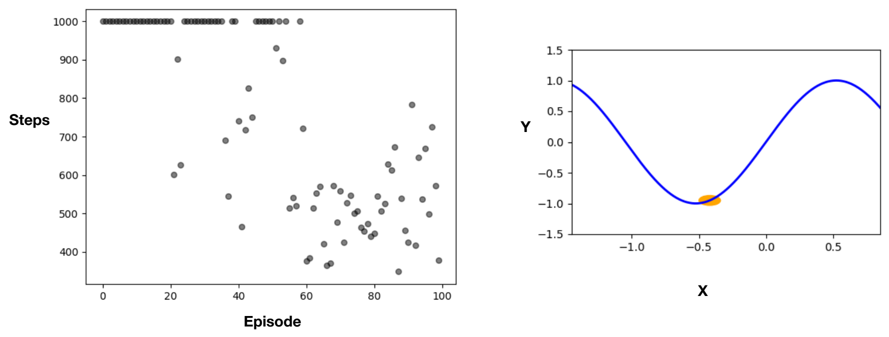

Figure 2: (Left) Plot of the results of 100 episodes of mountain-car simulations, with a maximum of 1000 steps allowed per episode. (Right) Illustration of mountain-car simulation status (normally visualized as a movie) with the curved line depicting the landscape and the oval denoting the mountain car.

6. A **credits** page listing the full names of all team members and a few buzzwords about their main contributions. This page should not be on the screen for more than a few seconds, but we need it for doing our own *credit assignment* to all members of your team. Everyone will get the same number of points.

It is **not** important that every group member speaks in the video. Too many voices could easily detract from the message. So think carefully about how many to use. We assume that all group members make significant contributions to the project. They don't all have to appear in prime time.

Video of the speaker's face or that of other group members should be limited or completely avoided. The 5-minute time restriction entails that anything shown on the screen should be providing important information; a *talking head* might be the best way to convey certain types of information, but most of the required content is quite technical and probably best transmitted via diagrams, equations and short animations (of a sloshing mountain car).

6 Visualization of Results

In addition to the diagrams that you use to explain your system, two others are mandatory: a) progress plots of steps-needed per episode, and b) short movies of simulations. The former simply plots the number of steps needed to solve the problem as a function of the episode, while the latter is a simple movie showing the back-and-forth movement of the mountain car for an entire episode. You may want to use several of these short animations to illustrate learning in your system, with simulations from early episodes showing unsuccessful behavior while later episodes show success. Figure 2 provides an example of each diagram.

To generate simple animations in matplotlib, you might want to use *FuncAnimation*, as nicely described in this blog post by Ken Hughes:

https://brushingupscience.com/2016/06/21/matplotlib-animations-the-easy-way/

On most laptops, the mountain car task with RL runs far too slowly to be recorded live, so you will need to do all of your runs ahead of time and just summarize the results (with plots and short animations) in the 5-minute movie.

7 Deliverables

The movie will be evaluated section by section, with a maximum of 3 points for each of the 5 main topics listed in the *Movie Contents* section above. The final 5 points will be based on the general quality of the production.

There is no written report, nor demonstration for this project. Simply upload a ZIPPED version of your movie to Blackboard by NOON on the delivery date. In the text associated with your file, include the names of every group member.

The 20 points for this project are 20 of the total (100 points) for this course.