CMPUT 397 Capstone Project

October 24, 2020

Project Overview

Great news! You may either choose to complete the standard Capstone Project on Coursera (recommended), or instead work on a more challenging research project.

1. In the capstone project on Coursera, you will tackle the Lunar Lander problem! You will begin by studying the task and implementing a reward function. Then you will need to choose the right algorithm, key performance parameters, and implement an RL agent using Expected Sarsa with Softmax action selection and Neural Networks function approximation. In the final part of the project, you will write code to conduct a careful scientific study of the learning rate parameter, and investigate the results. It will take approximately 20 hours to complete the project. This project will be conducted entirely on Coursera, and no separate submission is needed.

2. Two more-open ended projects specified below.

1 Project 1: Recreating and releasing code for a recent paper

We have recently completed a thorough empirical study (see here: `https://arxiv.org/abs/1906.07865`) with a notebook released to reproduce the results: `jair.adaptingbehavior.com`. The notebook, however, is only for one of the problems. Your job is to create a new notebook, recreating the second or third experiment. The purpose of this project is to both (a) show that you can reproduce an existing result and (b) build a portfolio, with your name associated with this notebook and a research paper.

Project 2: Expected Sarsa on Mountain Car Project

This is a project for individuals or groups of two. Write your own code. Write your own report. No sharing code between groups. No sharing plots or data between groups. Submit a single report if in groups of two

You will implement the Expected Sarsa Control algorithm, and tile coding function approximation. This algorithm is very similar to the Semi-gradient Sarsa algorithm described on page 214 of the textbook—your job is to figure out Episodic Semi-gradient Expected Sarsa.

Part 1: The first step is to implement the mountain car environment described in Example 10.1 (page 244-246). The three actions are: full throttle reverse (-1), zero throttle (0), and full throttle forward) (+1). The states are represented by a numpy array of doubles corresponding to the position ($x_t \in [-1.2, 0.5]$) and velocity ($\dot{x_t} \in [-0.07, 0.07]$) of the car.

Every episode begins with the car at the bottom of the hill (at random position from $[-0.6, -0.4]$), with zero velocity. The episode ends when the agent reaches the top of the hill (position ≥ 0.5). The reward is -1 per step (even on the terminal transition); there are no positive rewards ever. If the car reaches the left side of the track (position < -1.2) the position is reset to -1.2 and the velocity (if less than zero) is set to zero. The textbook describes the state update equations you must implement to simulate the Mountain Car problem.

Set up your code so that you can run many episodes (e.g., 200), and average the performance over independent runs. In every episode the car is reset to a random position at the bottom of the hill. In every run the value function is reset. Runs and episodes are different, do not confuse them. The notebooks on Coursera use rl-glue. You are free to use code from the notebooks if you like, but you must submit stand-alone code that does not require the Coursera Jupiter notebooks to run.

For part one submit all your environment code, and all the code required to run the experiment. Your code must run in python3 on a linux/unix platform. If your code does not compile and run you will lose marks.

Part 2: Implement the Expected Sarsa Control agent. Your agent will use the tile coding software (tiles3.py) to generate features. The software along with examples is available here: http://www.incompleteideas.net/tiles/tiles3.html

Your agent should use the following parameter settings:

memorySize = 4096 (for the tile coder)
numTilings = 8
shape/size of tilings = 8x8
$\alpha = 0.1/(\text{numTilings})$
$\epsilon = 0.0$
initial weights = random numbers between 0 and -0.001
$\gamma = 1$

For part two submit all your agent code.

Part 3: Produce a learning curve for your Expected Sarsa agent. In this part you will use your agent and environment to produce a learning curve, similar to Figure 10.2 in the book (except your plot will only contain 1 line, for the performance of your Expected Sarsa agent). Plot number of steps per episode (y-axis), over 200 episodes (x-axis), averaged over 50 independent runs. We will check that your code produces the plots you submit. If they do not, there will be major deductions.

Part 4: Find a better setting of the parameters for your Expected Sarsa agent on the Mountain

Car problem. Experiment with the parameters listed above, try finding a better setting (except γ, which must be equal to 1.0). You can also change the tile-coding strategy (number of tilings, size and shape of the tiles). Your job is to outperform the performance obtained with the original parameter settings. Please submit that plot.

As an overall measure of performance on a run, use the sum of all the rewards received in the first 200 episodes, averaged over runs. Find a set of parameters that improves this performance measure by two standard errors. To show the improvement, you must do many runs with the standard parameters and then many runs with your parameters, and measure the mean performance and standard error in each case (a standard error is the standard deviation of the performance number divided by the square root of the number of runs). If the difference between the means is greater than 2.5 times the larger of the two standard errors, then you have shown that your parameters are significantly better. It is permissible to use any number of runs greater or equal to 50 (note that larger numbers of runs will tend to make the standard errors smaller).

What to submit: (1) Report the alternate parameter settings (or other variations) that you found, the means and standard errors you obtained for the two sets of parameters, and the number of runs you used. (2) Once you have found your favourite set of parameters, make a learning curve based on 50 runs of 200 episodes with them, like you did in Part 3. Please submit that plot. We will check that your code produces the plots you submit, and the statistics you report. If they do not, there will be major deductions.

We will be checking all code submissions with software to detect plagiarism and copying. If you are found to have submitted code that is similar to another group/classmate, you will be investigated and reported to the Faculty of Science.

Mountain Car Project Submission Checklist
Compress into a single zip file and submit. Only one submission is needed per group.

1. Environment code, Agent code, and Experiment code.
2. Two plots - Learning curve of your Expected Sarsa agent: with default parameter setting and alternate parameter setting. (Axes, title should be properly labeled)
3. Short README.txt that contains:
   - Full names of group members, including Student ID.
   - Short description of each of your code file.
   - Alternate parameter setting that you found.
   - Means and standard errors for the two sets of parameters, and number of runs used.
   - Command to run your code and generate the two plots you submitted.

Project 3: Expected Sarsa with n-step on Lunar Lander

You will take the same agent as for the MOOC project, and you will try a new variant with n-steps. The idea is to take exactly the same parameters as the agent in the capstone, and instead try it with multi-step updates and compare performance. This project is the most open-ended, as it has yet to be tried. You may find your agent fails. The goal is to ask: can we benefit by using more of the observed reward, like Monte Carlo, but still benefit from boostrapping, as in Expected Sarsa.