CS 4180/5180: Reinforcement Learning and Sequential Decision Making (Fall 2024) Lawson L.S. Wong
Northeastern University Due February 23, 2024

Exercise 5: Off-Policy Monte-Carlo & Temporal-Difference Learning

Please remember the following policies:

- Exercises are due at **11:59 PM** Boston time (ET).
- **Submissions should be made electronically on Canvas.**
  Please ensure that your solutions for both the written and programming parts are present.
  Upload all files in a single submission, keeping the files individual (do not zip them).
  You can make as many submissions as you wish, but only the latest one will be considered, and late days will be computed based on the latest submission.
- Each exercise may be handed in up to two days late (24-hour period), penalized by 5% per day.
  Submissions later than this will not be accepted. There is no limit on the total number of late days used over the course of the semester.
- Written solutions may be handwritten or typeset. For the former, please ensure handwriting is legible. If you write your answers on paper and submit images of them, that is fine, but please put and order them correctly in a single .pdf file. One way to do this is putting them in a Word document and saving as a PDF file.
- Programming solutions should be in Python 3, either as `.py` or `.ipynb` files.
  For the latter (notebook format), please export the output as a PDF file and submit that together. To do so, first export the notebook as an HTML file (File: Save and Export Notebook as: HTML), then open the HTML file in a brower, and finally print it as a PDF file.
- You are welcome to discuss these problems with other students in the class, but you must understand and write up the solution **and code** yourself, *and* indicate who you discussed with (if any). If you are collaborating with a large language model (LLM), acknowledge this and include your entire interaction with this system. We strongly encourage you to formulate your own answers first before consulting other students / LLMs.
- Typically, each assignment contains questions designated [CS 5180 only.], which are required for CS 5180 students. Students in CS 4180 can complete these questions for extra credit, for the same point values.
  Occasionally, there will also be [Extra credit.] questions, which can be completed by everyone. Point values for these questions depend on the depth of the extra-credit investigation.
- Contact the teaching staff if there are *extenuating* circumstances.

1. **1 point.** (RL2e 5.10, 5.11) *Off-policy methods.*
   Written:
   (a) Derive the weighted-average update rule (Equation 5.8) from (Equation 5.7). Follow the pattern of the derivation of the unweighted rule (Equation 2.3).
   (b) In the boxed algorithm for off-policy MC control, you may have been expecting the W update to have involved the importance-sampling ratio $\frac{\pi(A_t|S_t)}{b(A_t|S_t)}$, but instead it involves $\frac{1}{b(A_t|S_t)}$. Why is this correct?
2. **1 point.** (RL2e 6.2) *Temporal difference vs. Monte-Carlo.*
   Written: Read and understand Example 6.1, then answer the following:
   (a) This is an exercise to help develop your intuition about why TD methods are often more efficient than Monte-Carlo methods. Consider the driving home example and how it is addressed by TD and Monte-Carlo methods. Can you imagine a scenario in which a TD update would be better on average than a Monte-Carlo update? Give an example scenario – a description of past experience and a current state – in which you would expect the TD update to be better.
   Hint: Suppose you have lots of experience driving home from work. Then you move to a new building and a new parking lot (but you still enter the highway at the same place). Now you are starting to learn predictions for the new building. Can you see why TD updates are likely to be much better, at least initially, in this case? Might the same sort of thing happen in the original scenario?
   (b) Is there any situation (not necessarily related to this example) where the Monte-Carlo approach might be better than TD? Explain with an example, or explain why not.

3. **3 points.** (RL2e 6.3, 6.4, 6.5) *Random-walk task.*
Code/plot/written: Read and understand Example 6.2.
Implement TD(0) on the random-walk task, then answer the following, providing empirical evidence for each:

   (a) From the results shown in the left graph of the random-walk example it appears that the first episode results in a change in only $V(A)$. What does this tell you about what happened on the first episode? Why was only the estimate for this one state changed? By exactly how much was it changed?

   (b) The specific results shown in the right graph of the random walk example are dependent on the value of the step-size parameter, α. Do you think the conclusions about which algorithm is better would be affected if a wider range of α values were used? Is there a different, fixed value of α at which either algorithm would have performed significantly better than shown? Why or why not?

   (c) In the right graph of the random walk example, the RMS error of the TD method seems to go down and then up again, particularly at high α's. What could have caused this? Do you think this always occurs, or might it be a function of how the approximate value function was initialized?

4. **[CS 5180 only.] 3 points.** *Four Rooms, continued.*
We will continue with the Four Rooms environment from Ex4 Q4, and explore off-policy Monte-Carlo prediction (policy evaluation) in this question. Specifically, we will use 3 different sources of behavior data to estimate the Q-values of a greedy target policy:

   - π_{random}: An equiprobable random policy.
   - On-policy Monte-Carlo control: Episodes generated by the gradually changing policy.
   - π_{greedy}: A greedy policy found using on-policy Monte-Carlo control. This also serves as the target policy.

   (a) **Code:** First, we need to collect sample data from each source of behavior data.
      - Run π_{random} for 10^4 episodes and store the sampled episodes.
      - Run the on-policy first-visit Monte-Carlo control you wrote for Ex4 Q4, for $\varepsilon = 0.1$ and 10^4 episodes. Keep the estimated Q-values and sampled episodes; we will use this in the next part.

      In both cases, since we will be using this data for off-policy prediction, you should also store the probabilities of the actions taken under the behavior policy (i.e., $b(A_t|S_t)$).

   (b) **Code/plot/written:** On-policy control only gives us an ε-soft policy, which is suboptimal in Four Rooms. Compute a greedy policy π_{greedy} with respect to the estimated Q-values found in part (a).
   Plot (or print) π_{greedy}. Does the policy make sense? Are there any interesting / unexpected action choices?

   (c) **Code/plot:** We will use the stored episodes generated from part (a) to estimate the Q-values of π_{greedy}. Implement off-policy Monte-Carlo prediction to estimate Q_π using each set of stored episodes.
   Note that you should end up with two Q_π estimates, one estimated from the 10^4 episodes of π_{random}, the second estimated from the 10^4 episodes collected during on-policy Monte-Carlo control.
   Plot (or print out) the estimated V_π (i.e., $Q_\pi(s, \pi(s))$ for each state s), for each source of data.

   (d) **Code/plot:** For comparison, apply on-policy Monte-Carlo prediction to estimate Q_π and plot V_π.
   You will need to generate a fresh set of 10^4 episodes (while following policy π_{greedy}) to do this.

   (e) **Written:** Compare the 3 estimates of V_π (and/or Q_π). Do you notice any interesting trends? If so, try to provide an explanation for what you observe.

   (f) **[Extra credit.] 1 point.**
      i. Use dynamic programming to compute the true q_π values. Compare your Monte-Carlo estimates of Q_π against the ground-truth q_π. Which Monte-Carlo estimate is more accurate, and in what way(s)?
      ii. Use dynamic programming to compute the true optimal q_* values and an optimal policy π_*. Compare the greedy policy π (from part b) and the Monte-Carlo value estimates (from parts c and d) against the ground-truth π_* and q_*. Did on-policy Monte-Carlo control find a good policy?
      iii. Repeat the entire question/comparison several times to see if the trends you observe are robust.

5. **[Project.]** Please fill out the project pre-proposal form (on Canvas) by the Ex5 deadline.

This is the end of the required portion of Ex5. However, two more questions follow:

- Q6 is the same extra-credit question from Ex4, which you may continue to complete as part of Ex5. Your latest submission of this question (in Ex4/Ex5) will be considered.
- Q7 will be formally assigned in Ex6, but we are providing it here in case you wish to start early.

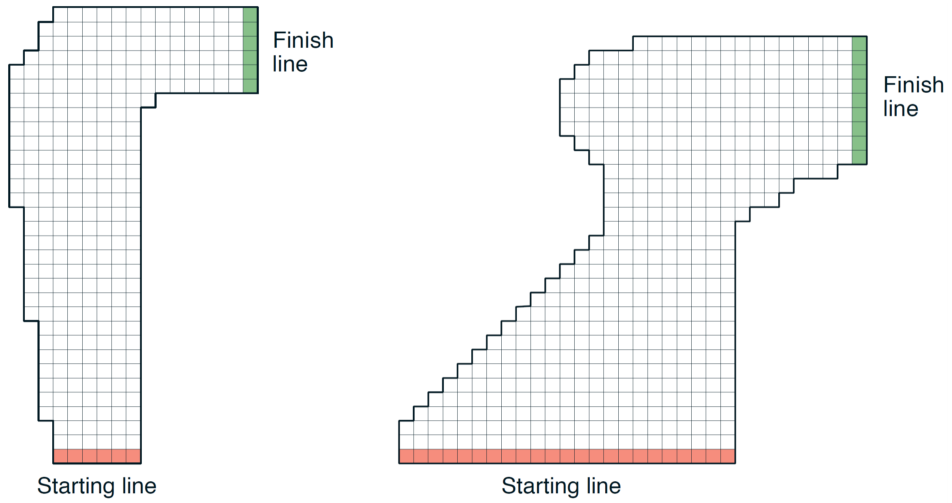

Figure 5.5: A couple of right turns for the racetrack task.

6. **[Extra credit.] 2 points.** (RL2e 5.12) *Racetrack.*
Consider driving a race car around a turn like those shown in Figure 5.5. You want to go as fast as possible, but not so fast as to run off the track. In our simplified racetrack, the car is at one of a discrete set of grid positions, the cells in the diagram. The velocity is also discrete, a number of grid cells moved horizontally and vertically per time step. The actions are increments to the velocity components. Each may be changed by $+1$, -1, or 0 in each step, for a total of nine (3×3) actions. Both velocity components are restricted to be nonnegative and less than 5, and they cannot both be zero except at the starting line. Each episode begins in one of the randomly selected start states with both velocity components zero and ends when the car crosses the finish line. The rewards are -1 for each step until the car crosses the finish line. If the car hits the track boundary, it is moved back to a random position on the starting line, both velocity components are reduced to zero, and the episode continues. Before updating the car's location at each time step, check to see if the projected path of the car intersects the track boundary. If it intersects the finish line, the episode ends; if it intersects anywhere else, the car is considered to have hit the track boundary and is sent back to the starting line. To make the task more challenging, with probability 0.1 at each time step the velocity increments are both zero, independently of the intended increments.

   (a) **Code:** Implement the racetrack domain (both tracks). Apply on-policy first-visit Monte-Carlo control (for ε-soft policies), with $\varepsilon = 0.1$ – ideally, this would be a simple application of the code from Q4(a).
   Plot: For each racetrack, plot the learning curve (multiple trials with confidence bands), similar to Q4(b).

   (b) **Code:** Implement off-policy Monte-Carlo control and apply it to the racetrack domain (both tracks). For the behavior policy, use an ε-greedy action selection method, based on the latest estimate of $Q(s, a)$ – i.e., this is similar to on-policy Monte-Carlo control, except that the target policy is kept as a greedy policy.
   Plot: For each racetrack, plot the learning curve (multiple trials with confidence bands), similar to Q4(b). Show the performance of both the *behavior* and *target* policies; in the latter case, do this by collecting one rollout after each episode of training, which is collected solely for evaluation purposes. (The point of this is to inspect performance on the policy we actually care about, not the one used for data gathering.) Additionally, visualize several rollouts of the optimal policy; consider using: `matplotlib.pyplot.imshow`

   (c) **Written:** Do you observe any significant differences between the on-policy and off-policy methods? Are there any interesting differences between the two racetracks?

   Tip: You can find NumPy arrays containing the racetracks in the Ex4 starter code (`racetracks.py`). Think about which racetrack you expect is easier, and develop your methods in that domain.

7. **[To be formally assigned in Ex6.]** (RL2e 6.9, 6.10) *Windy gridworld.*
Code/plot: In this question, you will implement several TD-learning methods and apply them to the windy gridworld in Example 6.5.

   (a) Implement the windy gridworld domain. Read the description in Example 6.5 carefully to find all details.

   (b) Implement the following methods, to be applied to windy gridworld:

   - On-policy Monte-Carlo control (for ε-soft policies) – consider using your code from Ex4
   - SARSA (on-policy TD control)
   - Expected SARSA
   - n-step SARSA (choose an appropriate n, e.g., $n = 4$)
   - Q-learning (off-policy TD control)
   - *Optional*: Dynamic programming (to provide an upper bound)

   To compare each method, generate line plots similar to that shown in Example 6.5 (do not generate the inset figure of the gridworld). Make sure you understand the axes in the plot, which is not the same as before (why is it different?).
   As in previous exercises, perform at least 10 trials, and show the average performance with confidence bands (1.96× standard error).
   If you implement the optional DP solution, use it to generate and plot an upper bound on performance.
   Note: You may adjust hyperparameters for each method as necessary; for SARSA, use the values provided in the example ($\varepsilon = 0.1, \alpha = 0.5$) so that you can reproduce the plot in the textbook.

   For the following parts, apply at least two of the above TD methods to solve them.

   (c) *Windy gridworld with King's moves*: Re-solve the windy gridworld assuming eight possible actions, including the diagonal moves, rather than four. How much better can you do with the extra actions?
   Can you do even better by including a ninth action that causes no movement at all other than that caused by the wind?

   (d) *Stochastic wind*: Re-solve the windy gridworld task with King's moves, assuming that the effect of the wind, if there is any, is stochastic, sometimes varying by 1 from the mean values given for each column. That is, a third of the time you move exactly according to these values, as you did above, but also a third of the time you move one cell above that, and another third of the time you move one cell below that. For example, if you are one cell to the right of the goal and you move left, then one-third of the time you move one cell above the goal, one-third of the time you move two cells above the goal, and one-third of the time you move to the goal.