CS7180: Special Topic - Artificial Intelligence (Spring 2022) Christopher Amato
Northeastern University Due Mar 7, 2022

Exercise 5: Temporal-Difference Learning

Please remember:

- Exercise due at **11:59 PM EST Mar 7, 2022**.
- Submissions should be made electronically on Canvas. Please ensure that your solutions for both the written and programming parts are present. You can upload multiple files in a single submission, or you can zip them into a single file. You can make as many submissions as you wish, but only the latest one will be considered.
- For **Written** questions, solutions may be handwritten or typeset. If you write your answers by hand and submit images/scans of them, please please ensure legibility and order them correctly in a single PDF file.
- The PDF file should also include the figures from the **Plot** questions.
- For both **Plot** and **Code** questions, submit your source code along with reasonable documentation.
- 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. Also, you *must* list the names of all those (if any) with whom you discussed your answers at the top of your PDF solutions page.
- Each exercise may be handed in up to two days late (24-hour period), penalized by 10% per day late. Submissions later than two days will not be accepted.
- Contact the teaching staff if there are medical or other extenuating circumstances that we should be aware of.

1. **1 point.** (RL2e 6.2) *Temporal difference vs. Monte-Carlo.*
   Written: Read and understand Example 6.1. 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.

2. **1 point.** (RL2e 6.11, 6.12) *Q-learning vs. SARSA.*
   Written:
   (a) Why is Q-learning considered an off-policy control method?
   (b) Suppose action selection is greedy. Is Q-learning then exactly the same algorithm as SARSA? Will they make exactly the same action selections and weight updates?

3. **2 points.** (RL2e 6.3, 6.4, 6.5, 7.3) *Random-walk task.*
Written: Read and understand Example 6.2 and Example 7.1, then answer the following:

   (a) 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?

   (b) 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?

   (c) Why do you think a larger random walk task (19 states instead of 5) was used in Example 7.1?
   Would a smaller walk (fewer states) have shifted the advantage to a different value of n?
   How about the change in left-side outcome from 0 to -1 made in the larger walk?
   Do you think that made any difference in the best value of n?

   (d) **[Extra credit.]** Implement TD(0) on the random-walk task and computationally verify your answers to the above questions.

4. **4 points.** (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 (use $n = 4$)
   - 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.

5. **2 points.** *Bias-variance trade-off.*
In lecture, we discussed that Monte-Carlo methods are unbiased but typically high-variance, whereas TD methods trade off bias to obtain lower-variane estimates. We will investigate this claim empirically in this question, from the perspective of prediction.

The overall experimental setup is as follows.

- We will continue with the deterministic (original) windy gridworld domain.
- A fixed policy π will be specified.
- A certain number of "training" episodes $N \in \{1, 10, 50\}$ will be collected.
- Each method being investigated (TD(0), n-step TD, Monte-Carlo prediction) will estimate the state-value function based on the N episodes.
- We then evaluate the distribution of learning targets each method experiences at a specified state S.
To do so, an additional 100 "evaluation" episodes will be generated. Instead of using these to perform further updates to the state-value function, we will instead evaluate the distribution of learning targets $V(S)$ will effectively experience based on the "evaluation" episodes. For example, TD(0) will experience a set of $\{R + V(S')\}$ targets, whereas Monte-Carlo will experience a set of $\{G\}$ targets.

Note that in practice you should pre-collect both the training and evaluation episodes for efficiency and to ensure consistency while comparing between different methods.

(a) **Code/plot:** Perform the above experiment for the specified methods and training episodes N.
Use a near-optimal stochastic policy π (e.g., found by SARSA or other methods in Q4).
Perform the evaluation for the start state (indicated 'S' in Example 6.5).
For n-step TD, use $n = 4$, but you may consider trying more values of n as well.
Plot the histogram of learning targets experienced in the evaluation episodes for each combination of N and method (i.e., at least 9 histograms total).
Optional: Use dynamic programming or any other appropriate method to compute the true value of $v_\pi(s)$ for comparison purposes, and add this to your plots as well.

(b) **Written:** Describe what you observe from your histograms.
Comment on what they may show about the bias-variance trade-off between the different methods, and how it may depend on the amount of training that has already occurred.

(c) **[Extra credit.]** If we considered the scenario of control (i.e., we would use on-policy action-value methods, iteratively update the policy during training, and use it to generate the next training episode), would that change the results, and how? Implement this to computationally test your hypothesis.