4180/5180: Reinforcement Learning and Sequential Decision Making (Fall 2022) Christopher Amato
Northeastern University Due Nov 9, 2022

Exercise 6: Planning and Learning

Please remember the following policies:

- Exercise due at **11:59 PM EST Nov 9, 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 in Jupyter Notebook (.ipynb file) along with reasonable comments of your implementation. Please make sure the code runs correctly.
- 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.
- **Notations: RL2e is short for the reinforcement learning book 2nd edition. x.x means the Exercise x.x in the book.**

1. **2 point.** (RL2e 8.1) *Planning vs. n-step returns.*
   Written:
   (a) The nonplanning method looks particularly poor in Figure 8.3 because it is a one-step method; a method using multi-step bootstrapping would do better. Do you think one of the multi-step bootstrapping methods from Chapter 7 could do as well as the Dyna method? Explain why or why not.
   (b) See the pseudocode on page 164 in RL2e. Consider using both n-step returns and Dyna (specifically, using n-step returns instead of one-step returns in tabular Dyna-Q). We can clearly do this for the learning phase (d). Can we also do this for the planning phase (f)? What are the advantages and disadvantages of using n-step returns in the planning phase (f)?
   (c) **[Extra credit.] 1 point.** Perform computational experiments to verify your answers to the above questions.
   For part (b), you will have to implement tabular Dyna-Q with n-step returns.

2. **5 points.** *Implementing Dyna-Q and Dyna-Q+.*
   (a) **Written:** For some unknown reason, the textbook is quite vague about Dyna-Q+ and does not provide pseudocode for the modification. Read Section 8.3 carefully (specifically p. 168) and **reconstruct the pseudocode for Dyna-Q+**. In particular, please write your pseudocode that also contains the suggestion from the footnote.
   (b) **Code/plot:** Reproduce the Figures 8.4 and 8.5. Specifically, you are asked to implement both Dyna-Q and **two variants** of Dyna-Q+. Importantly, for the first variant, implement a Dyna-Q+ that does **not** consider the suggestions proposed in the footnote on p. 168 (i.e., only use "bonus rewards" strategy described in the text); For the second variant, implement a Dyna-Q+ that also consider the suggestions proposed in the footnote on p. 168 (i.e., use both "bonus rewards" and "untried actions" described in the footnote). In this case, compared to the original figures in the book, each figure will have three curves for Dyna-Q, Dyna-Q+(no footnote), and Dyna-Q+(with footnote), respectively.
   Here are some tips.

- Note that "cumulative reward" refers to the number of completed episodes, *not* the discounted return.
- We already provide the hyperparameters to run experiments and reproduce the figures (See the Jupyter Notebook for more details). But, feel free to play with other combinations if you are interested.
- As usual, use good experimental practice by performing a reasonable number of trials, and report average performance with confidence bands. Again, you can find the plotting code in the Jupyter Notebook.
- Finally, you should report two figures and the implementation code in the final submission. **Make sure that you also include the figures in the PDF in case that we can not run your implementation.**

(c) **Written:** Did the footnote matter? Why or why not?

3. **2 points.** (RL2e 8.4) *Exploration bonus in rewards vs. action selection.*
Recall that the idea of exploration bonuses appeared in the UCB algorithm for multi-armed bandits as well. However, there is a subtle difference here:

- In UCB, the bonus was applied during *action selection*, but the Q-values (estimated return of arms) were not affected by this bonus.
- In Dyna-Q+, the bonus is applied given to the simulated *rewards*, and therefore the Q-value estimates are affected by the bonus.

This question explores this subtle but important difference.

Written: Think carefully about the implications of applying bonuses to the reward function ("pseudo-reward") versus during action selection. Make some predictions:

- What would generally happen to the policy's behavior during the training under each approach? How do they differ from each other?
- What are the advantages and disadvantages of each approach?

4. **3 points. [5180]** (RL2e 8.5) *Dyna-Q for stochastic environments.*

(a) **Written:** How might the tabular Dyna-Q algorithm be modified to handle stochastic environments? Please write down your pseudocode and explain why your modification is reasonable.

(b) **Code/plot/written:** Implement your modifications to tabular Dyna-Q in part (a), experiment on the *Windy GridWorld* (as defined in the Jupyter Notebook) with stochastic wind. To show your results, you should plot a figure similar to Figure 8.2 and also print out an optimal path. The detailed instructions are in the Jupyter Notebook.

5. **5 points.** *Manual MCTS*

Perform 8 iterations of MCTS on the Random Walk MDP. The MDP is obtained by incorporating actions `left` and `right` to the Markov reward process of Example 6.2. **We perform the first 3 iterations for you and please complete the remaining 5 iterations.** Please refer to the last 3 pages.

- In each state-node, write the MDP state it represents;
- In each action-node, write the total sum of simulated returns as T, and the number of simulations as N;
- As *tree policy*, use the UCB rule with exploration constant $c = \sqrt{2}$, which selects the next action by maximizing

$$\arg\max_a \left\{ \frac{T_a}{N_a} + \sqrt{\frac{2 \ln N_s}{N_a}} \right\} \tag{1}$$

  where N_a is the number of simulations for the proposed action-node, T_a is the total sum of simulated returns for the proposed action-node, and N_s is the number of simulations for the current state-node, i.e. the sum of N_a for both proposed action-nodes. Ties should be broken randomly;
- As *rollout policy*, use the uniform distribution over the two actions;
- To guarantee that we all get the same result, we will seed our randomness by using an online RNG:
  - Go to `https://www.random.org/integers/?mode=advanced`.

  - In Part 1, select however many random integers you want to generate;
  - In Part 1, select that the integers should have a value between 0 and 1;
  - In Part 4, use a pregenerated randomization based on the persistent identifier `rldm_mcts`

  This will give you a sequence of 0s and 1s, which you should respectively interpret as `left` and `right`. Whenever you need to sample an action, use the next symbol in the sequence. To verify that you've got the correct "random" sequence, verify that yours starts with:

  11100000101111100101100011001100000011011011010001110011101111001100010111101100000001

- For each iteration of MCTS, draw the resulting tree, *clearly* indicating the results of each of the four phases:
  - Selection: indicate the actions taken from the root to the selected node.
  - Expansion: indicate the node resulting from expansion.
  - Rollout: write down the entire simulated sequence, i.e. the states, actions, and rewards until the terminal state is reached.
  - Backup: indicate how the values in the tree have changed as a result of the simulation.
- After having run the 8 simulations, indicate which action should be taken for the start state.

As confirmation that you've done everything right:

- Your final tree should have 9 state-nodes.
- You should have queried the RNG 50 times.

Please be as **CLEAR** as possible. This exercise will require you to draw a lot, so make it clear what you're doing. We can only assign points if we understand your answer.

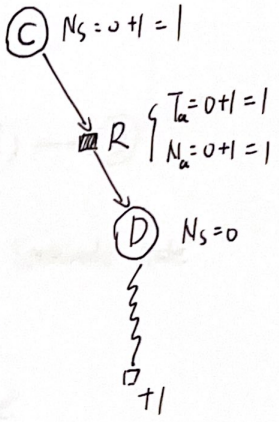
Iteration 1 : Note that : { Right : 1, Left : 0

Selection : (C)

Expansion : 1 (use random list) → (D)

Simulation : (D) —R, 1→ (E) —R, 1→ Terminate State

reward = +1

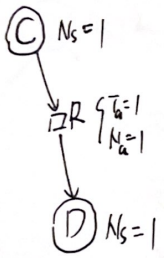


Backup :

Final Tree

Iteration 2: Note that: Right = 1 / Left = 0

Previous tree



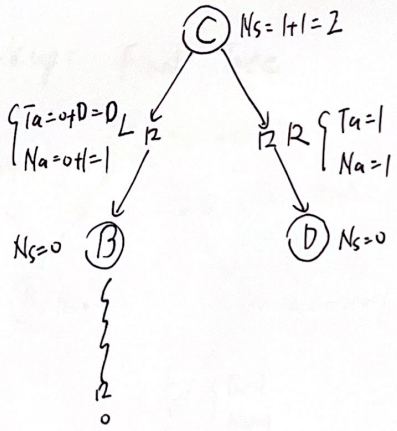
Selection: C

Expansion: D (no use rnd) → B

Simulation: B —L/0→ A —L/0→ Terminate state
reward = 0

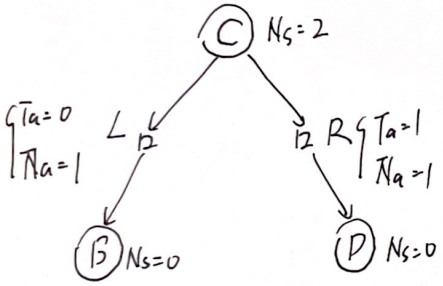
Back up:

Final Tree



Iteration 3 : Note that : Right = 1 : Left = 0

Previous Tree :



Selection : C → D

Expansion : 0 (use the rnd list) → C

Simulation : C —L/0→ B —L/0→ A —R/1→ B —L/0→ A —R/1→ B —R/1→ C —R/1→ D —R/1→ E

Terminate state

reward = +1

Backup : Final Tree