CS 4180/5180: Reinforcement Learning and Sequential Decision Making (Fall 2021) Robert Platt
Northeastern University Due October 29, 2021

Exercise 5: Temporal-Difference Learning

1. **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?

It may not be the same as the original scenario since the original one doesn't have a belief that included all states. TD need more episodes to update every step and bootstrap, but MC can update all states with only one single episode.

(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.

In some environments that the agent rewards only when they reach the goal with 1 point and every other step with 0 reward. Take the maze or puzzle as example, MC will perform better than the TD method since MC will update every state along with the trajectory but TD needs to run for a long time and episodes to have a good result.

2. **1 point.** (RL2e 6.11, 6.12) *Q-learning vs. SARSA.*
Written:

(a) Why is Q-learning considered an off-policy control method?

The reason Q-learning is off-policy is because it updates its value by next state s' and greedy action a regardless the behavior policy that used. Thus, we do not need to consider the behavior policy to update action values. So, Q-learning is 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?

If action-selection using the greedy policy, then the SARSA algorithm will be the same as the Q-learning one. Thus, they will make exactly the action selection and weight updates.

3. **3 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) 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?

From the graph, we could know that the first episode terminated on state A. All state value is initialized to be 0.5. And the reward is all 0 except for state E for 1. $\gamma = 1$

By taking $\alpha = 0.1$, we get

$$RMS(A) = \alpha[R + \gamma V_T - V(A)] = 0.1[0 + 0 - 0.5] = -0.05$$

$$RMS(X) = \alpha[R + \gamma V_{x+1} - V(X)] = 0.1[0 + 1(0.5) - 0.5] = 0$$

So state value of A is changed by -0.05, and all other state remain unchanged.

(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?

No. From the graph, the results of random walk example are affected by α and the results finally converge to an optimal value. TD converges at $\alpha = 0.05$ and MC converges $\alpha = 0.01$. TD are always had better performance than MC even with different value of α, thus, the conclusion about which algorithm is better won't be affected by the range of values of α.

(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?

Different value of α like $\alpha = 0.1$ and $\alpha = 0.15$ may cause this. And initialize state value and function of how to approximate may also cause this since TD method always have a bias.

(d) 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?

In order to compare the effect of different n values on the performance of the TD method. Thus, we need to set the n to a different value for investigating. The original environment use 5 states and it is hard to evaluate the performance.

For a smaller walk problem, the expected value of number of states for travelling is smaller than n. Then the performance is more accurate than a larger n. Therefore, a smaller walk task does affect the results regarding which value of n yields the best performance, the smaller n should be better.

Changing the left-side outcome from 0 to -1 won't make any difference in the best value of n. n only depends on the environment, as long as the state and transition unchanged, the best value of n is unchanged.

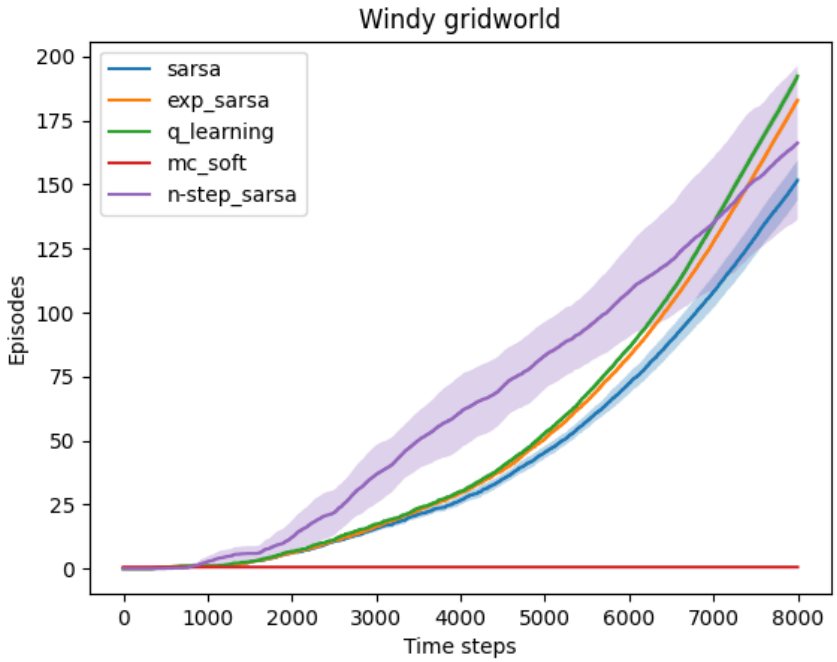
(e) [**Extra credit.**] Implement TD(0) on the random-walk task and computationally verify your answers to the above questions.

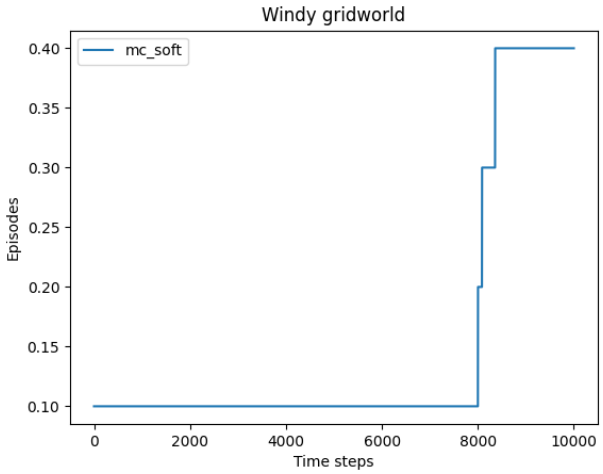
4. **4 points.** (RL2e 6.9, 6.10) *Windy gridworld.*
Code/plot: In this question, you will implement several TD-learning methods and apply them to the windy gridworld in Example 6.5.

(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)

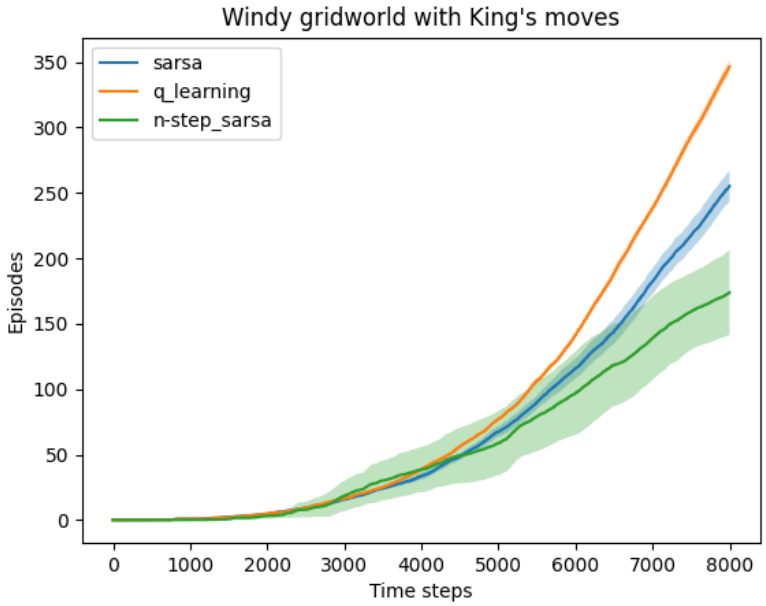




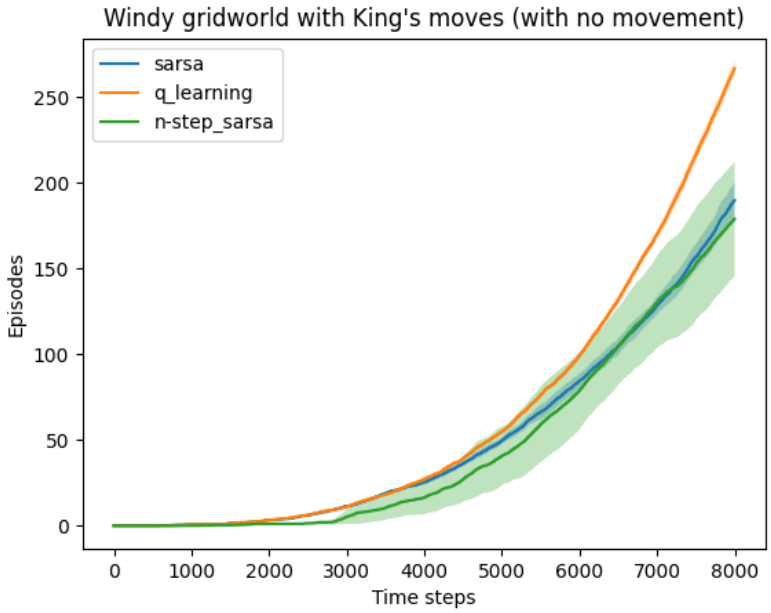
From the graph, Q-learning > Expected Sarsa > N-step Sarsa > Sarsa > First visit Monte-Carlo.

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?

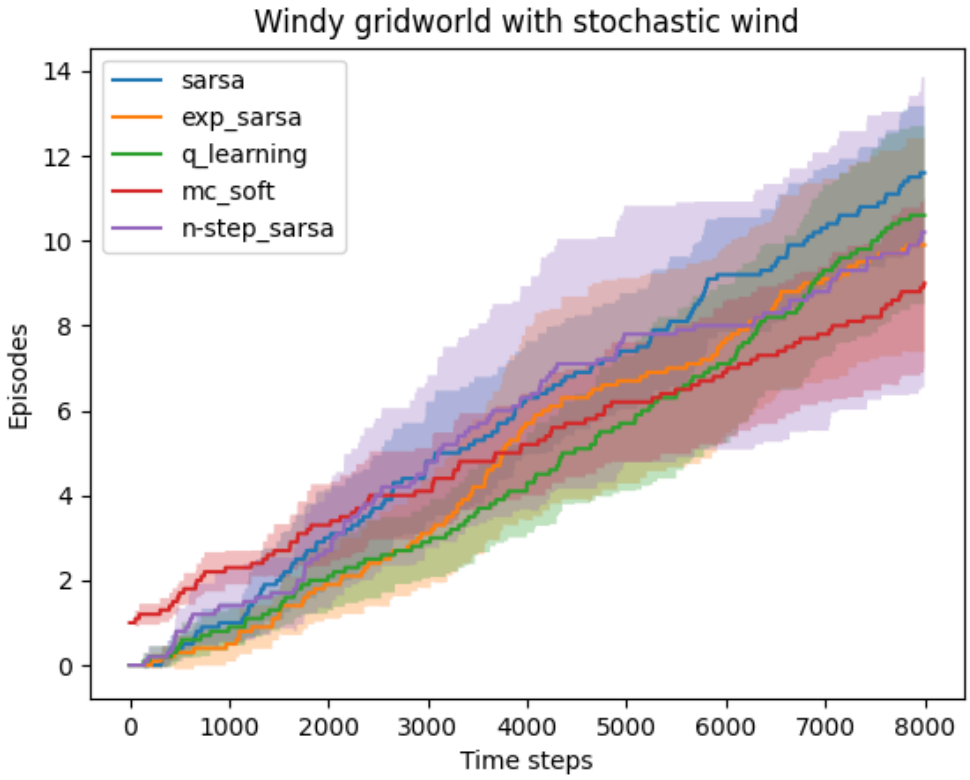


For the eight action king's move Q-learning and Sarsa is much better than the original environment and N-step Sarsa remains approximately same level as the original one.



The nine's step will largely reduce the performance of the algorithms. So, it is not recommended to include the stop action.

(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.



All the algorithms perform worst in the stochastic environment.

5. **[CS 5180 only.] 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.

Monte-Carlo won't affected by the training set since it is not accomplished from state value for bootstrap. But other algorithms will be affected.