CS 4180/5180: Reinforcement Learning and Sequential Decision Making (Fall 2020)– Lawson Wong

Name: [Virender Singh]
Collaborators: [Sunny Shukla]

To run the codes please use main.py in the src folder. All the plots can also be seen in the plt folder.

Question 1. *(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 MonteCarlo 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.*
(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.

Response:
(a) TD methods are better than the Monte Carlo because they update their estimate based in part of other states estimate, this bootstrapping helps TD to converge faster. TD methods compute the MLE estimate of the parameters and thus the certainty-equivalence estimate. Now suppose the resident building is changed in the drive to home example. So we don't know the value estimates of the new states like these new resident building and parking lot. For monte carlo, one has to drive all the way from office to home and compute the state values. However, with TD the estimates of the highways are the same and then we can use their true estimates for finding the estimates of the new states. Hence the TD will converge much faster than the Monte Carlo, further since they will be using the true estimate of the state values of the highway they will eventually converge to the true estimate. With monte carlo even those states which do not need much updates will also be changed and thus Monte Carlo is not practical.
(b) Monte Carlo methods do not carry bias like TD method because TD updates are done based on the estimates of the neighboring states which are not the true estimates before convergence. In monte carlo methods, all states are explored till the end and hence updates are made based on the true estimates of the states thus in this sense monte carlo is better. However with convergence these biases are removed and value estimates converge to true values. For policy evaluation, monte carlo would be much better because of lack of bias like TD methods. For example, in the

example above if we have to find the optimal policy which would help us reach home from office early then we will have to use monte carlo as opposed to TD method.

Question 2. *(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?

Response:

(a) Q learning is considered an off-policy method because it does not stick to the current policy to take the action rather it chooses the action greedily based on the q value of the next state and the greedy action. However, SARSA is an on policy method because it sticks to the current policy and takes the action accordingly no matter if there exist another action which has more q value than the current policy's action.

(b) Suppose action selection is greedy, even then the SARSA and Q-learning are not same. Because SARSA will take the greedy action according to the q values that were set in last iterations. Q-learning on the other hand will take greedy action based on current updated q values. Hence the SARSA is not same as Q learning.

Question 3. *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?
(b) The specific results shown in the right graph of the random walk example are dependent on the value of the step-size parameter, . Do you think the conclusions about which algorithm is better would be affected if a wider range of values were used? Is there a different, fixed value of 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 '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?
(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?

Response:

(a). The first episode must have ended on the left terminal state that's why only the value of state A changed. It would be changed by following formula:
V(A) = V(A) + α(R + γ V(left terminal state) - V(A))
Now V(left terminal state) = 0, γ = 1, R = 0, α = 0.1 so V(A) will be
V(A) = 0.5 + 0.1(0 + 1× 0 - 0.5)
V(A) = 0.5 - 0.05 = 0.45
The value of A thus changed by 0.05.

(b) The α is kept sufficiently small for both the the algorithms to converge thus a larger range of α would not have gotten any better results. For small value of α both the algorithms are reaching the lowest error they can reach. I don't think there is any different value for α for which it algorithms would have performed better since α has already touch the limit for the optimal performance of both the algorithms.

(c) This is the same problem which one faces in the stochastic gradient descent where due to small step size one can get stuck at local minima. Therefore, the error first decreases and and then increases for large α. Hence α should be small. Yes initialization plays an important role, and different ideal initial values might have performed better than this case, although the size

of α will still be an issue.

(d) We used 19 states instead of 5 states because we wanted to prevent early finishing for our random walk. Because if n is very large than the random walk reaches the terminal state using less than the n steps.

Yes, if we had smaller walk i.e. for less number of states, then the number of states would have been less than n and thus performance test would be less accurate for larger values of n and hence it would be more advantageous for smaller n.

No changing in left-side outcome from 0 to -1 won't make any difference in the best value of n. This might however speed up the learning efficiency for n-step TD. As now the reward for achieving the left terminal state (= -1) is different from the reward for achieving any intermediate state (= 0), this difference makes the agent learn know that whether it reaches the left terminal state or not.

Question 4.

Response:

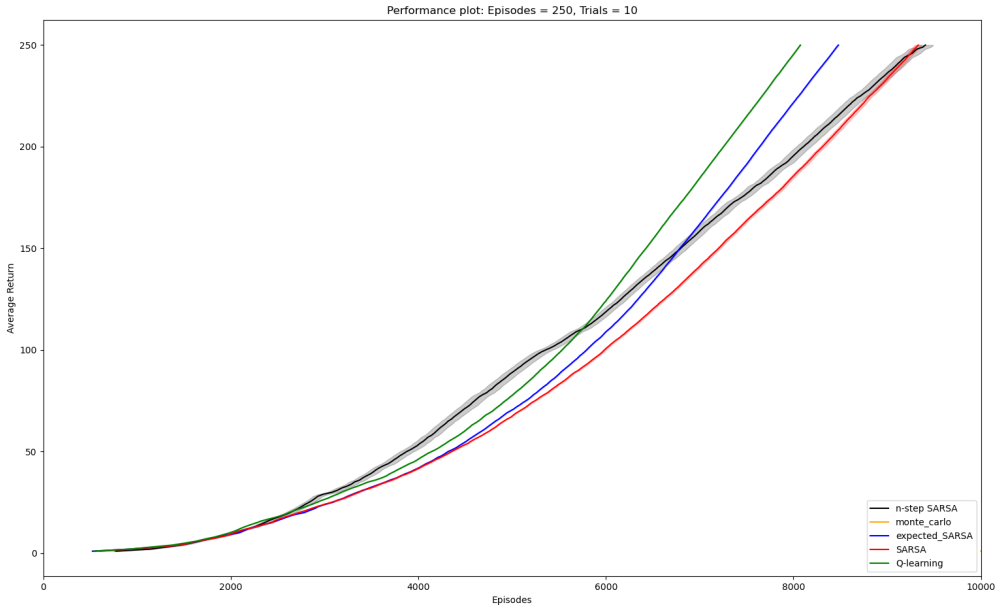

Figure 1: Q4b: Windy gridworld convergence rate of different algorithms

(c) Windy gridworld with king's moves shown in figures 3 and figure 4 In the plots shown, we can see that for the same number of steps in the king's move scene the algorithms (SARSA, Q-Learning and expected SARSA) can complete 50 episodes more(250) as compared to 200 episodes in the same number of steps(8000). Thus the algorithms perform better with the extra action and hence we can say the algorithms learn to converge faster in king's moves. With extra moves, there is not much significant improvement, however we can see that n-step SARSA is a little slower.

(d)

Windy gridworld with king's moves plus one additional action of no move shown in figure 5 and 6 It can be seen from the figure 5 and 6 that algorithms converges slower to true values when stochastic wind is added to the environment as compared to standard windy gridworld with four actions, as well as, w.r.t. the eight and nine action windy gridworld without stochastic wind. There is no significant difference between eight actions and nine actions windy gridworld plots/performance with stochastic wind.

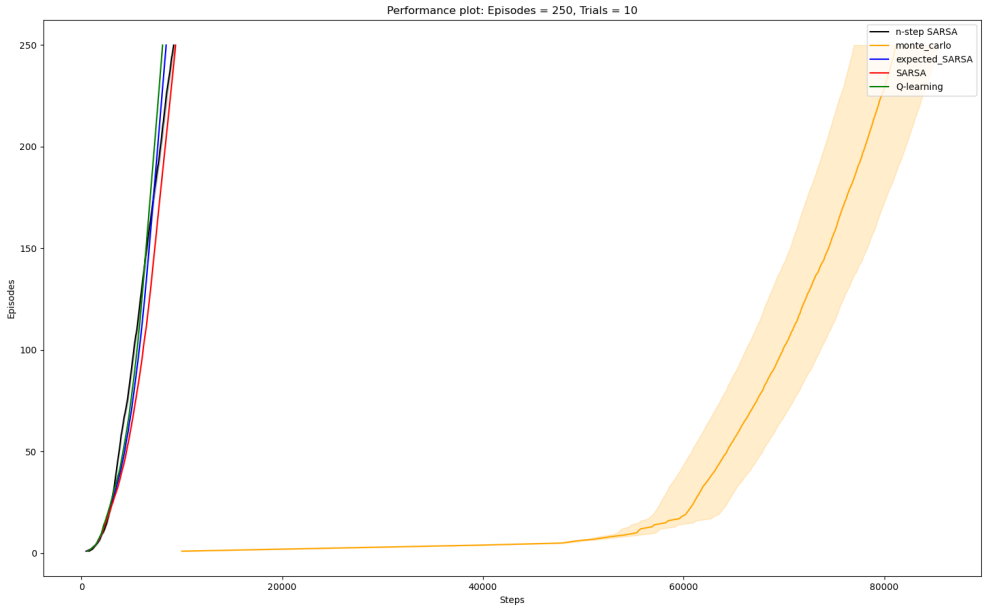

Figure 2: Q4b: Windy gridworld convergence rate of different algorithms including monte carlo

(d) Stochastic Wind for windy gridworld task with king moves shown in figure 7 and 8

Question 5.

Response:

Ans 5(a):

(a)

Histograms for three different training sizes of [1, 10, 50] are shown in figure. Figure 9 shows the histogram for TD(0) method for training size of 1. Figure 10 shows the histogram for TD(0) method for training size of 10 and then Figure 11 shows the histogram for TD(0) for training size of 50 episodes.

Similarly, we have figure 12 shows the histogram for n step TD method for training size of 1. Figure 13 shows the histogram for n step TD method for training size of 10 and then Figure 14 shows the histogram for n step TD for training size of 50 episodes.

Figure 15 shows the histogram for monte carlo method for training size of 1. Figure 16 shows the histogram for monte carlo method for training size of 10 and then Figure 17 shows the histogram for monte carlo method for training size of 50 episodes.

(b) The Monte Carlo method shows relatively more variance than the TD method and this

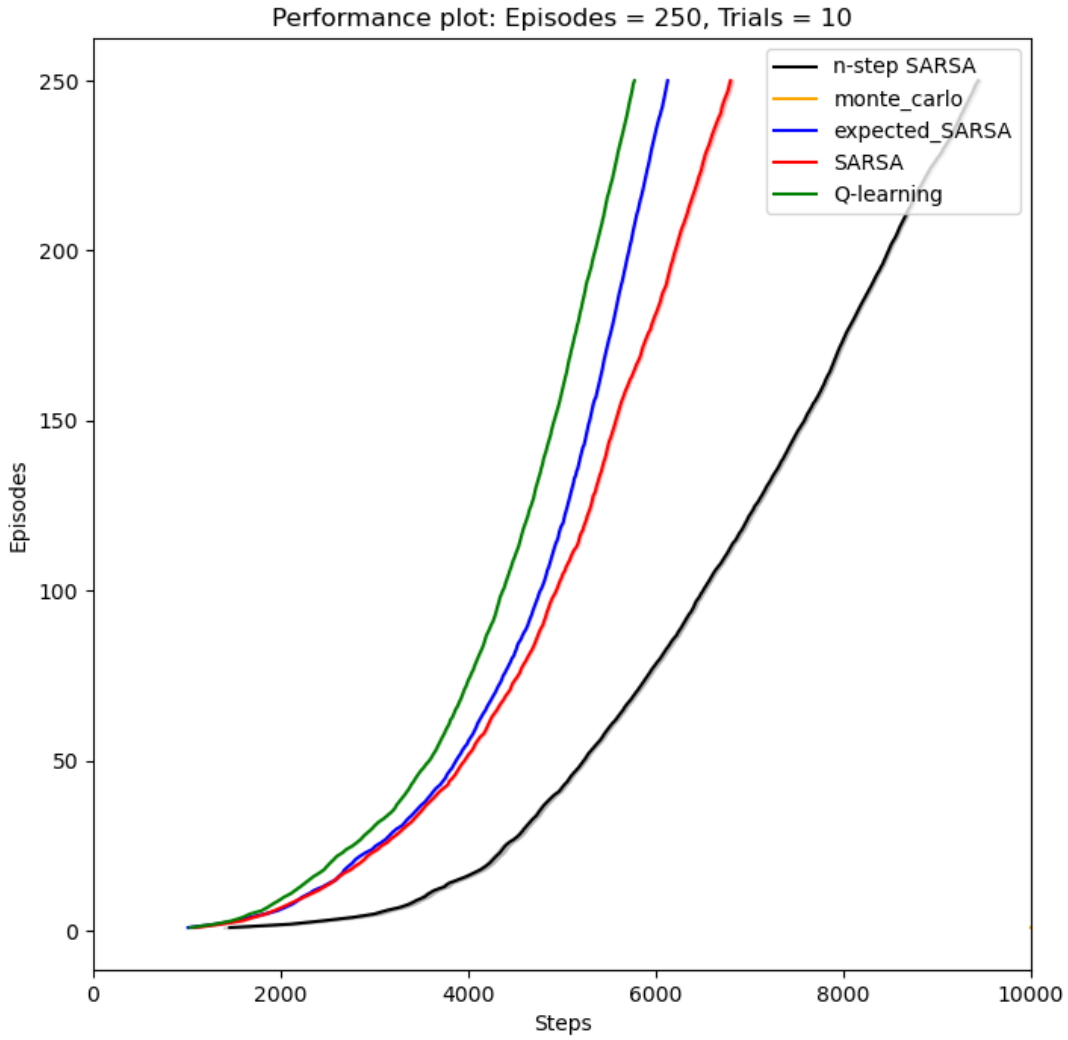

Figure 3: Q4c: Windy gridworld with 8 available actions

is quite evident from the histograms, we can see that the monte carlo histograms shown in figure 15, 16 and 17 have much more variance from the mean value than the plots shown in figures 9, 10, 11, 12, 13 and 14 for TD(0) and n-step TD respectively.

This is the bias-variance tradeoff as the TD methods are highly biased though they show low variance as compared to the Monte Carlo method which show less biased but have high variance. With the increasing amount of training in the TD(0) and the n=step TD, we can see that the bias is reduced. To achieve the same amount of learning monte carlo need more samples to achieve

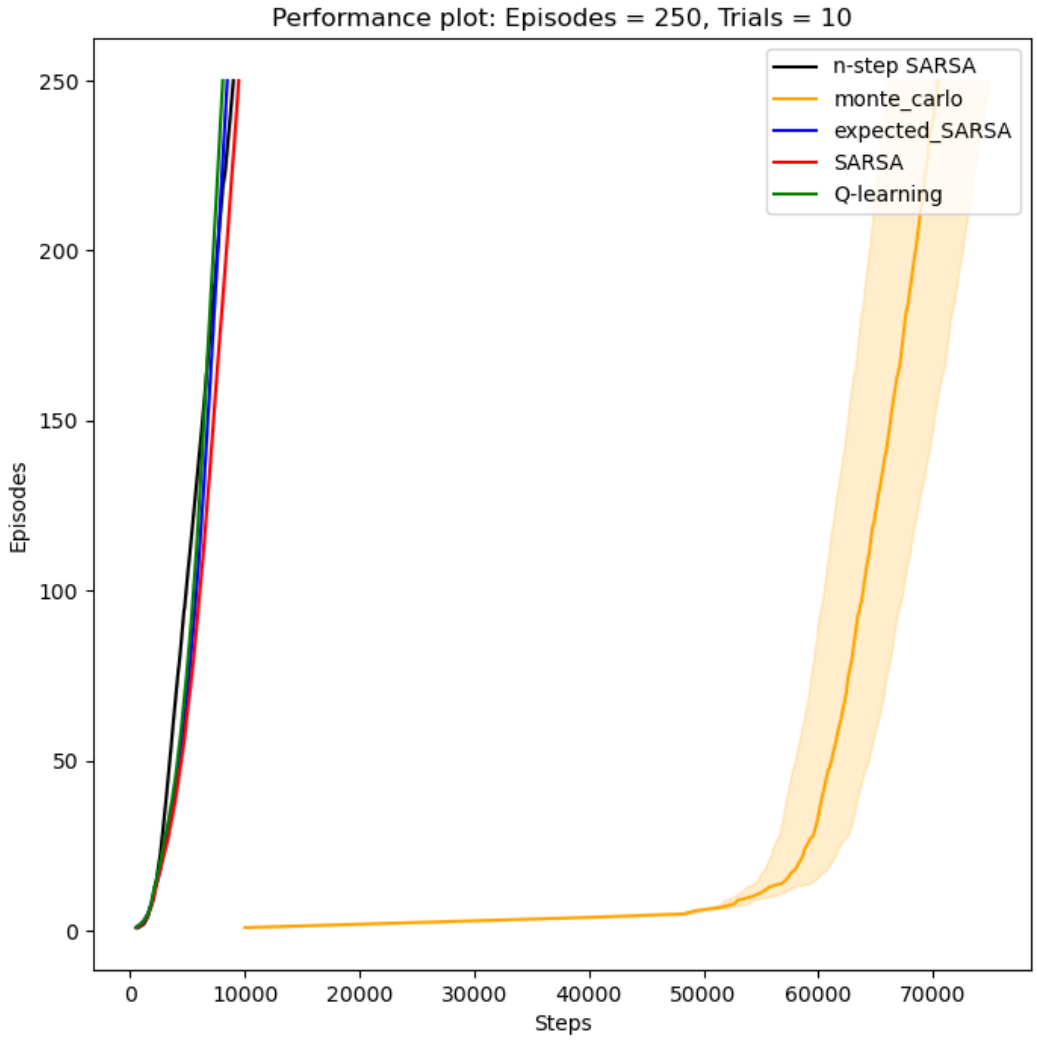

Figure 4: Q4c: Windy gridworld with 8 available actions and monte carlo

the same degree of learning compared to TD.

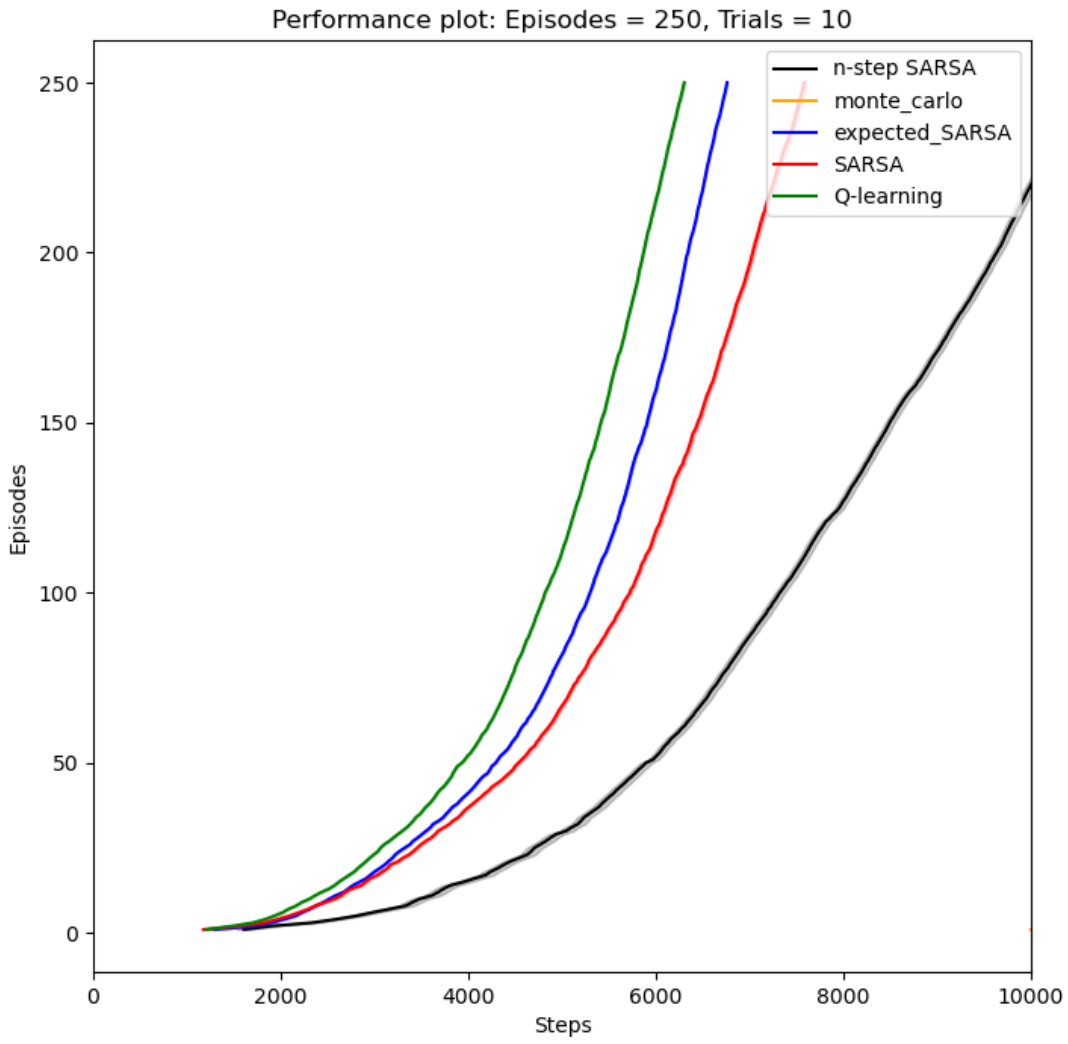

Figure 5: Q4c: Windy gridworld with 8 available actions and additional action

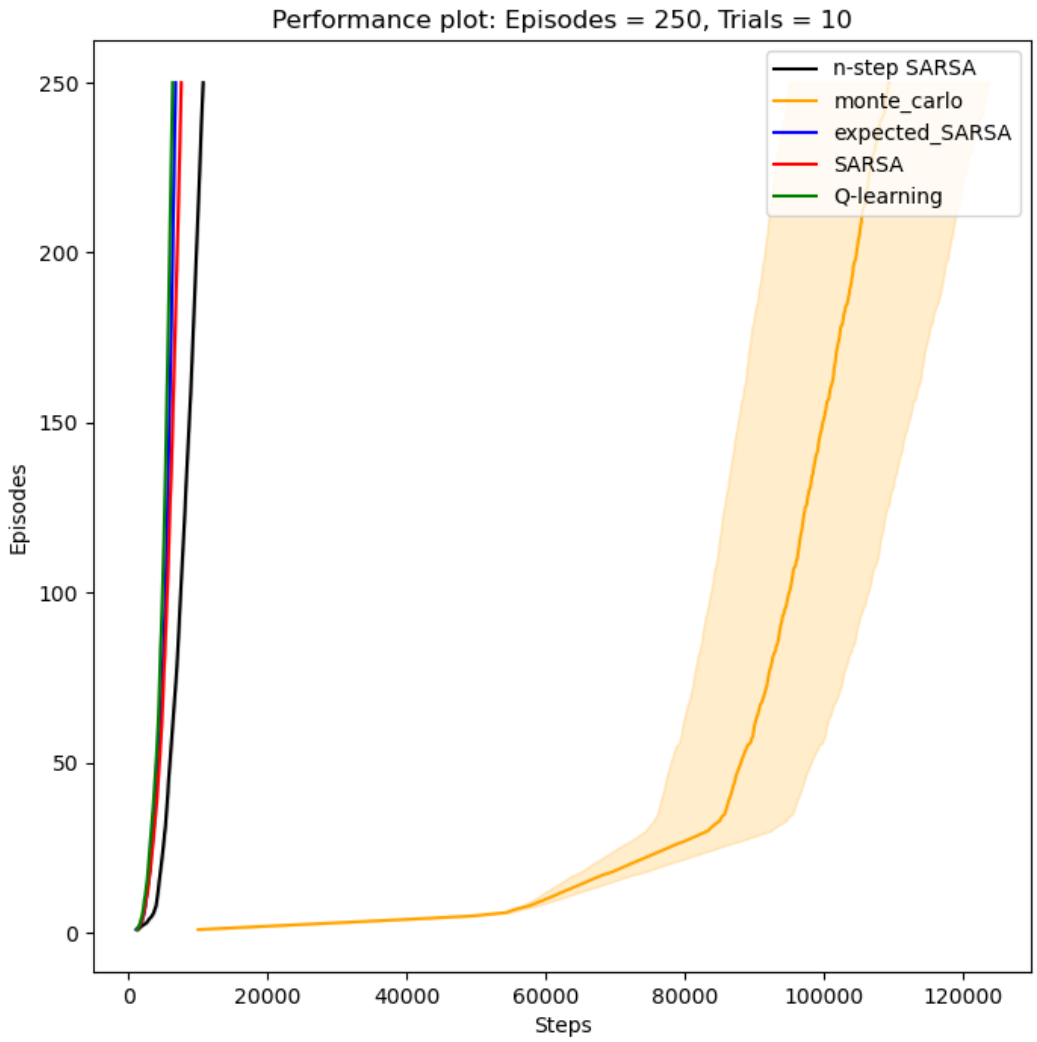

Figure 6: Q4c: Windy gridworld with 8 available actions and additional action with monte carlo

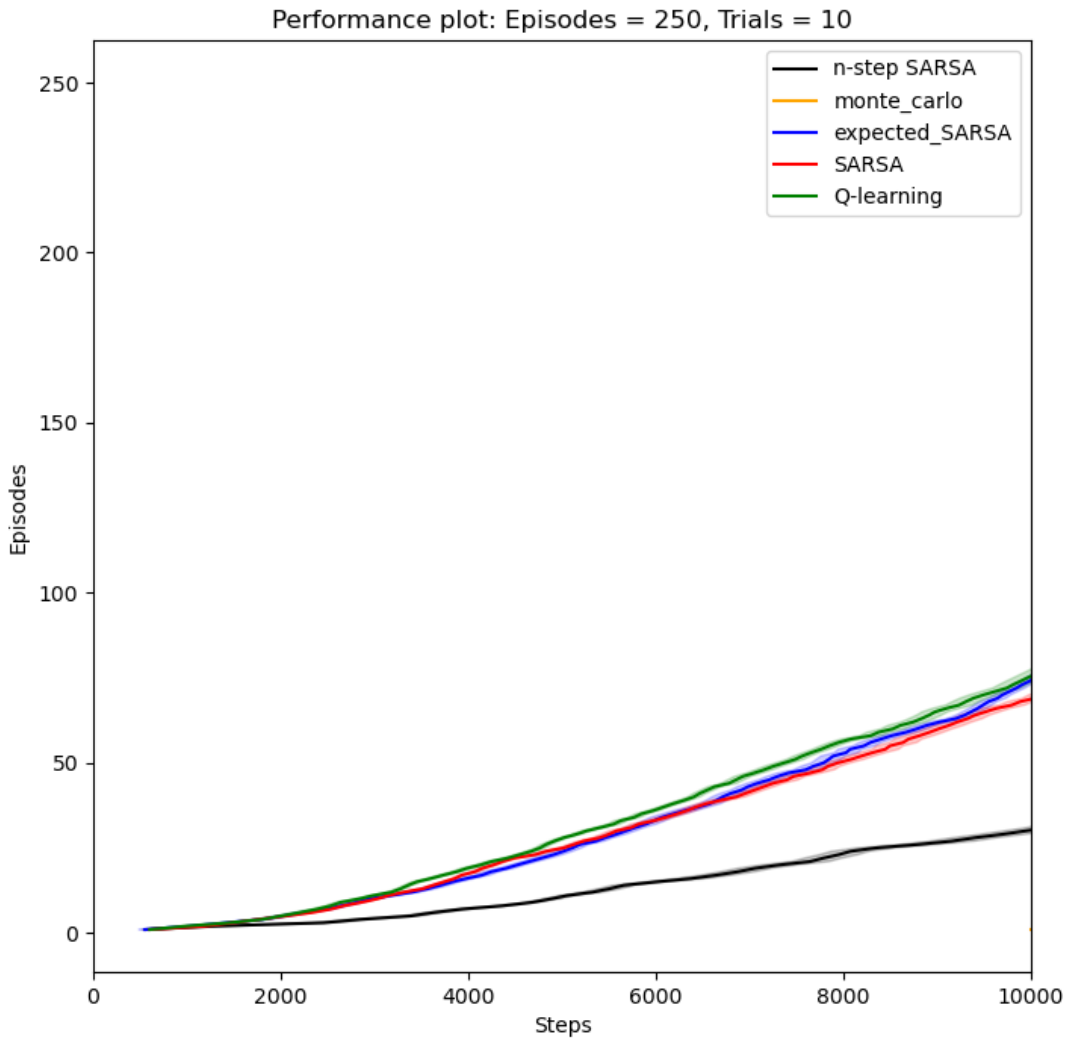

Figure 7: Q4d: Stochastic windy gridworld task with king moves

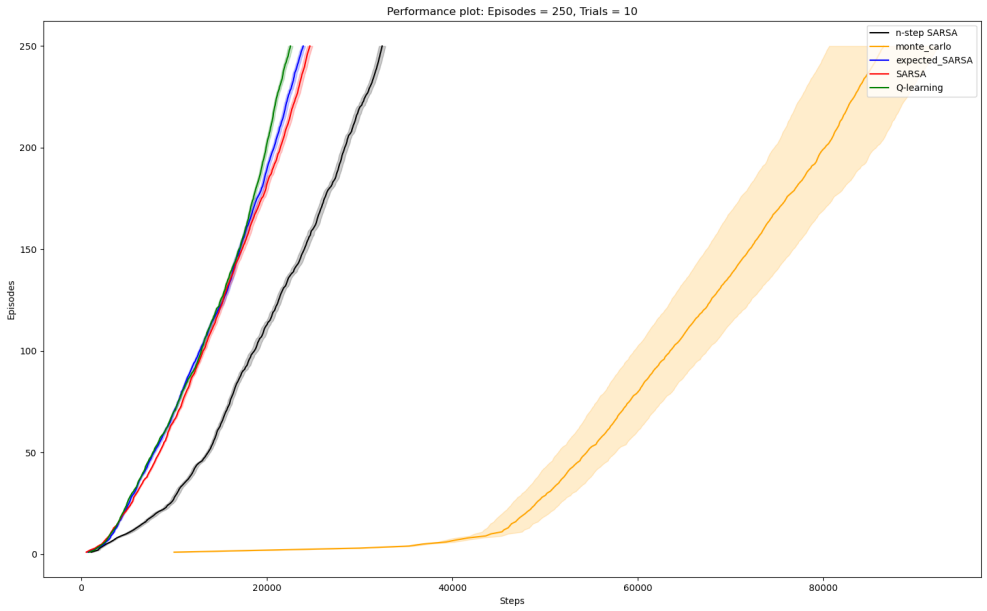

Figure 8: Q4d: Stochastic windy gridworld task with king moves and monte carlo

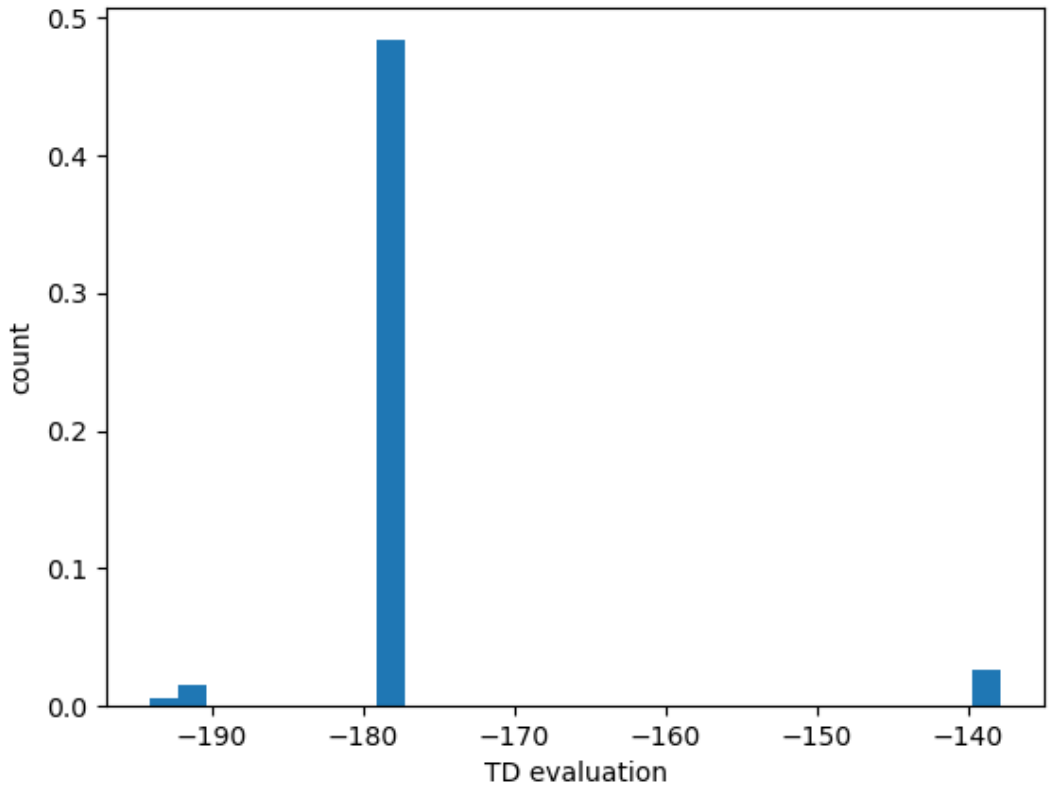

Figure 9: Q5a: Windy gridworld task histogram for TD(0) for 1 episode of training

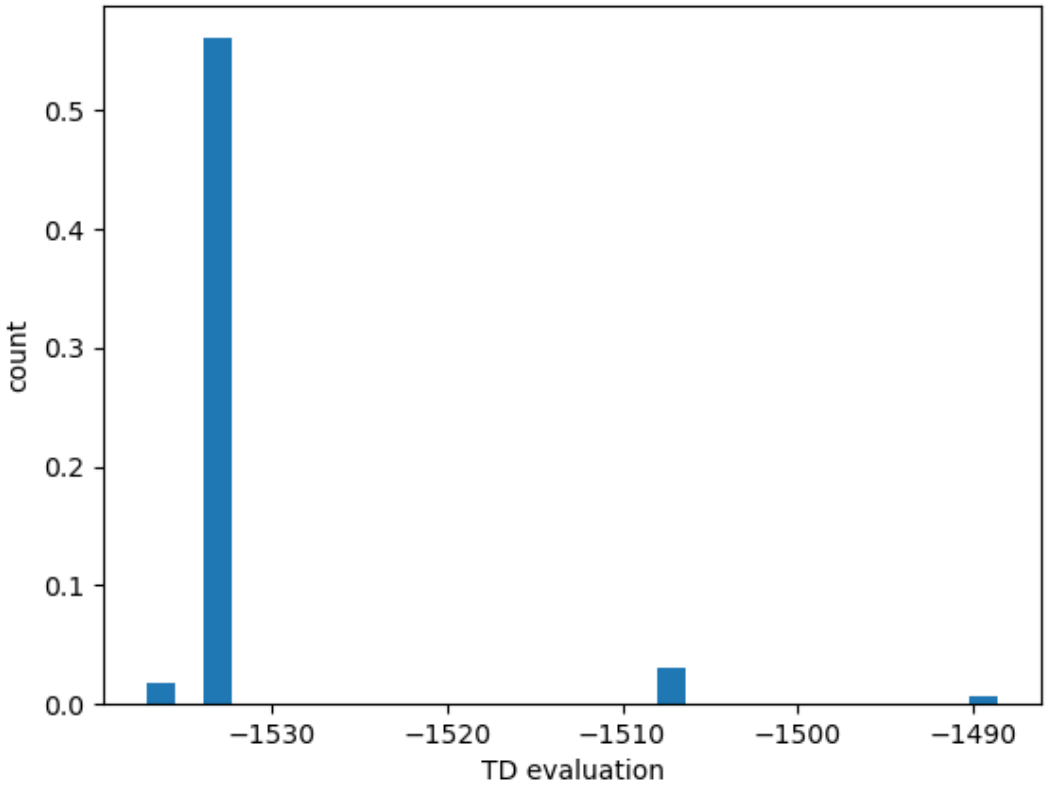

Figure 10: Q5a: Windy gridworld task histogram for TD(0) for 10 episode of training

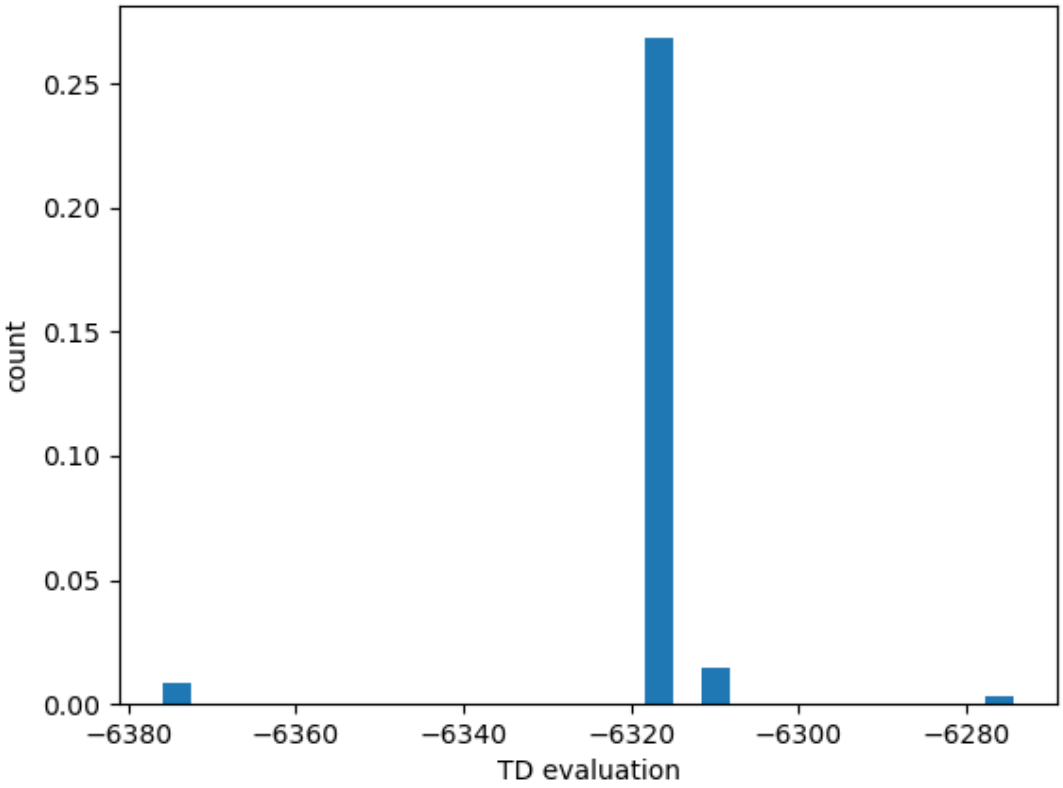

Figure 11: Q5a: Windy gridworld task histogram for TD(0) for 50 episode of training

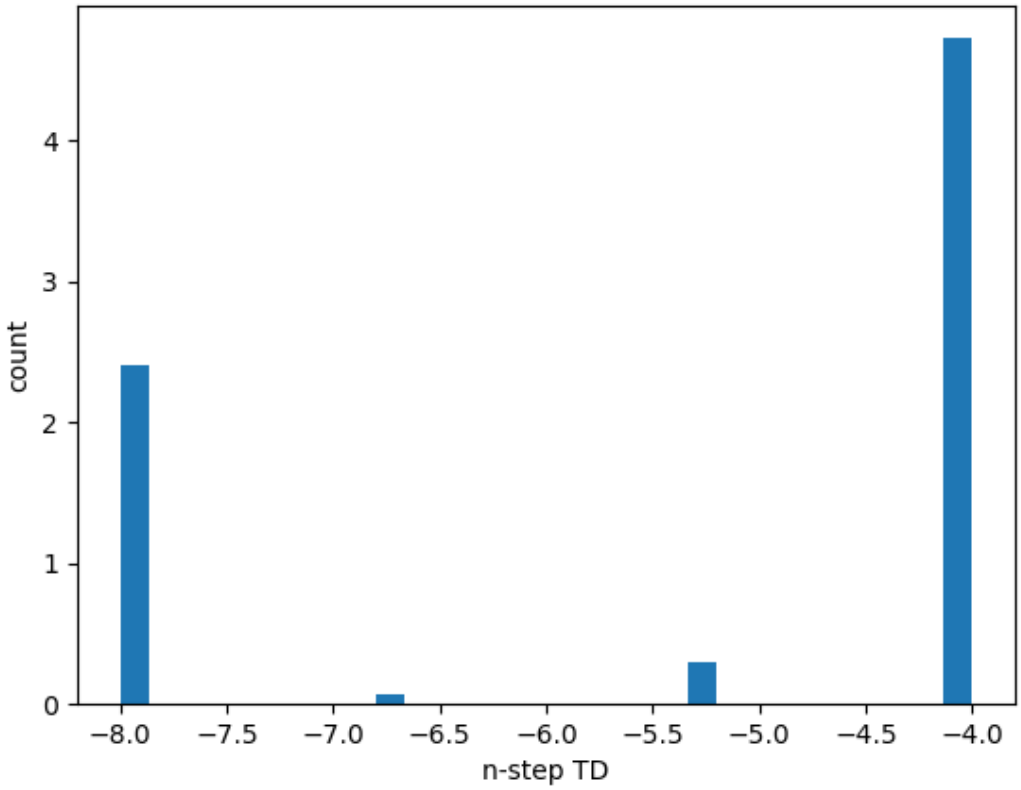

Figure 12: Q5a: Windy gridworld task histogram for n-step TD for 1 episode of training

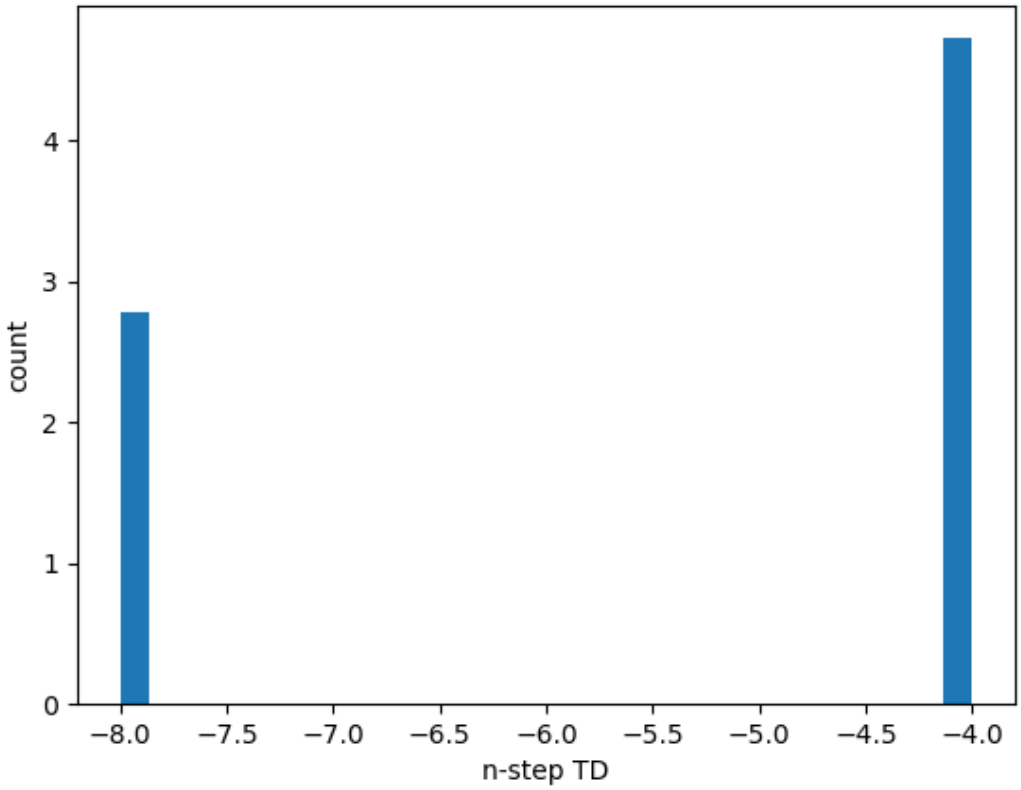

Figure 13: Q5a: Windy gridworld task histogram for n-step TD for 10 episode of training

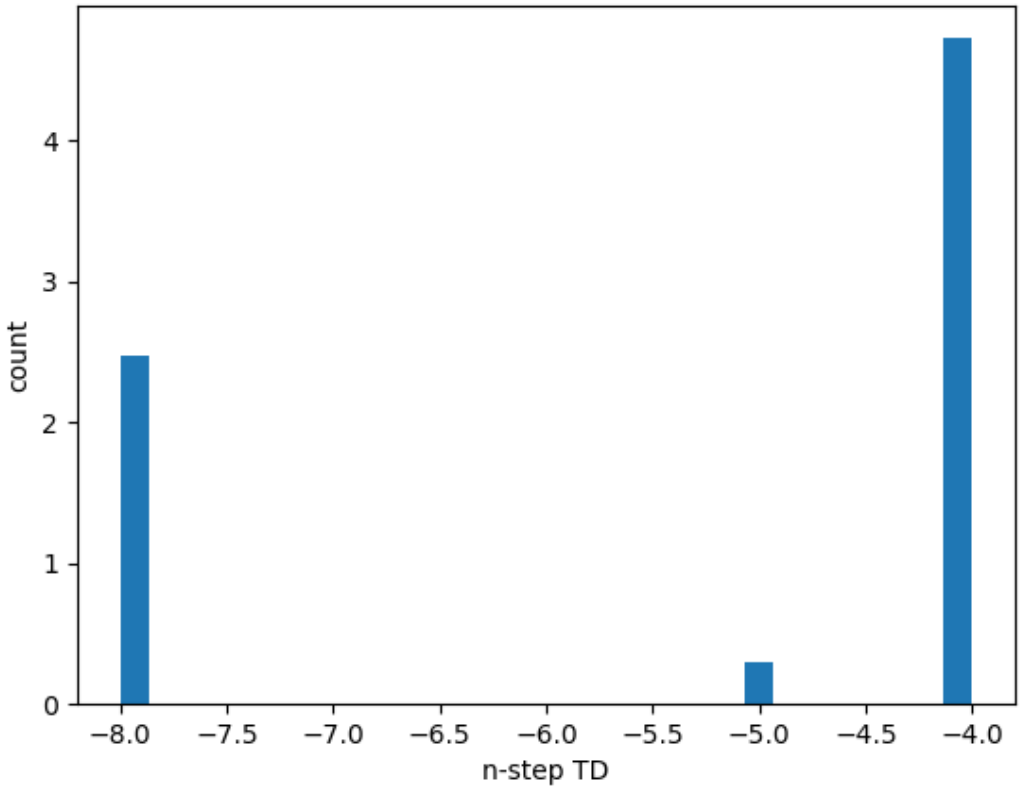

Figure 14: Q5a: Windy gridworld task histogram for n-step TD for 50 episode of training

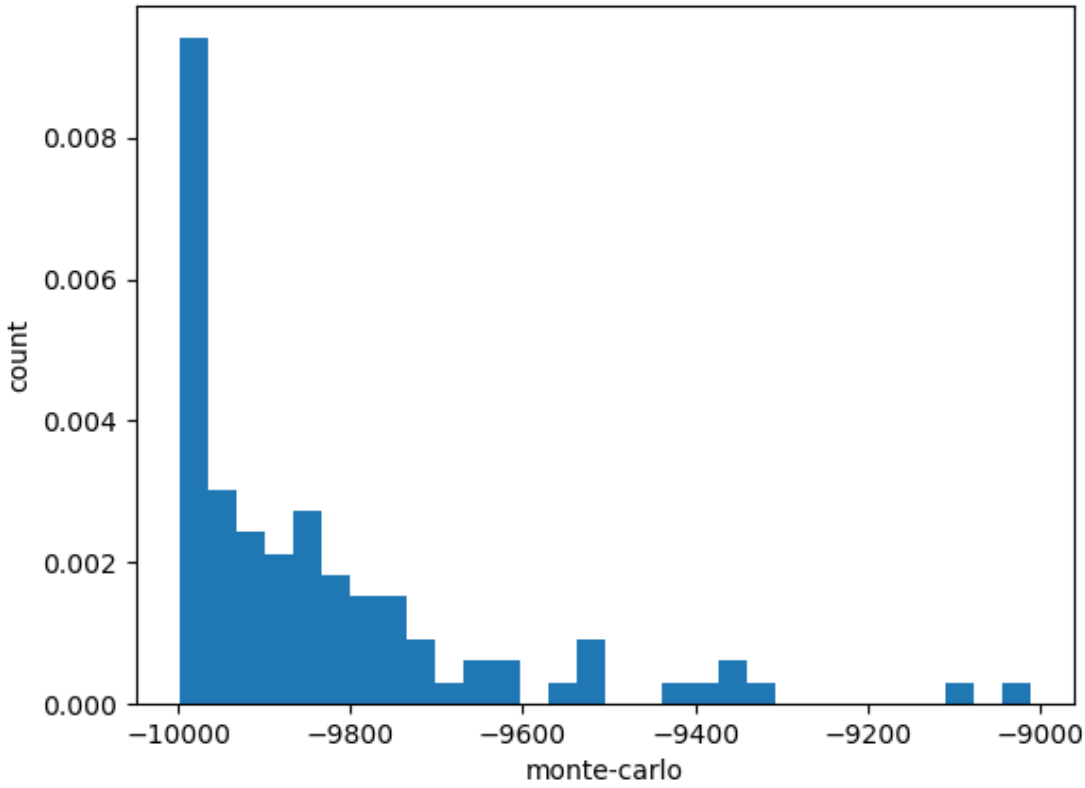

Figure 15: Q5a: Windy gridworld task histogram for monte carlo for 1 episode of training

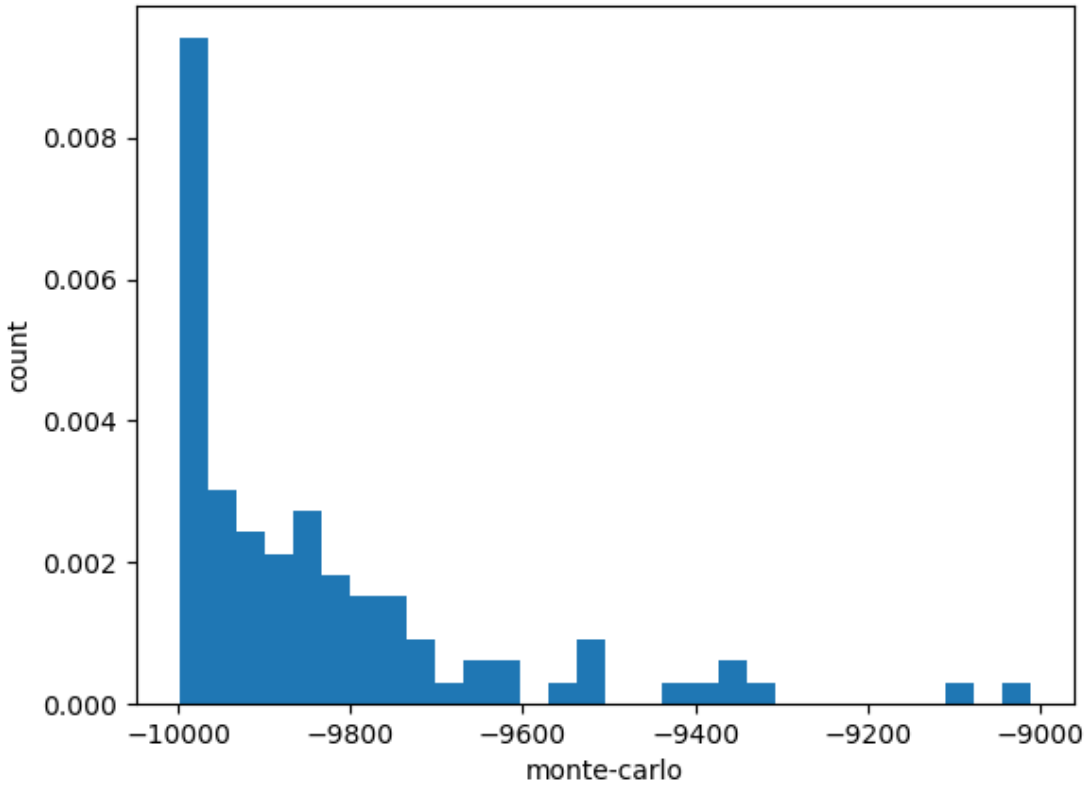

Figure 16: Q5a: Windy gridworld task histogram for monte carlo for 10 episode of training

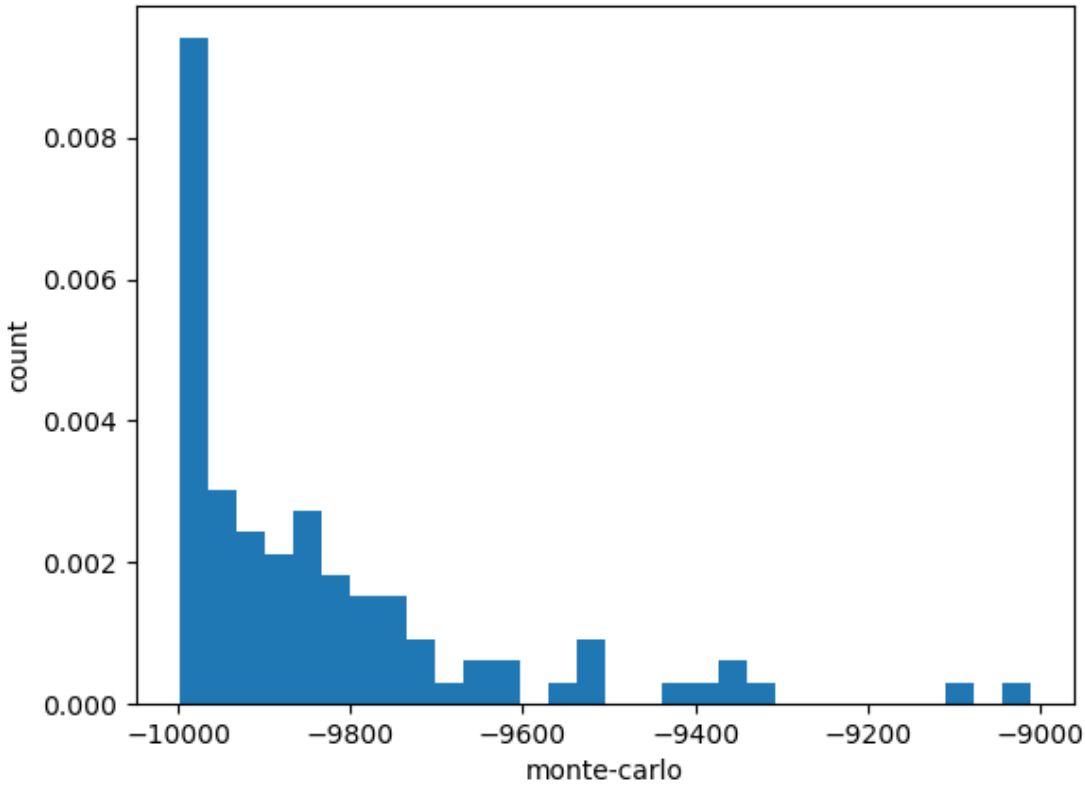

Figure 17: Q5a: Windy gridworld task histogram for monte carlo for 50 episode of training