CMPSCI 687 Homework 5 - Fall 2022

Due November 25, 2022, 11:55pm Eastern Time

1 Instructions

This homework assignment consists only of a programming portion. While you may discuss problems with your peers (e.g., to discuss high-level approaches), you must write your own code from scratch. The exception to this is that you may re-use code that you wrote for the previous homework assignments. **We will run an automated system for detecting plagiarism. Your code will be compared to the code submitted by your colleagues, and also with many implementations available online.** In your submission, do explicitly list all students with whom you discussed this assignment. Submissions must be typed (handwritten and scanned submissions will not be accepted). You must use LaTeX. The assignment should be submitted on Gradescope as a PDF with marked answers via the Gradescope interface. The source code should be submitted via the Gradescope programming assignment as a .zip file. Include with your source code instructions for how to run your code. You **must** use Python 3 for your homework code. You may not use any reinforcement learning or machine learning-specific libraries in your code, e.g., TensorFlow, PyTorch, or scikit-learn. You *may* use libraries like numpy and matplotlib, though. The automated system will not accept assignments after 11:55pm on November 25. The tex file for this homework can be found here.

Before starting this homework, please review this course's policies on plagiarism by reading Section 10 of the syllabus.

Programming Assignment [100 Points, total]

In this assignment, you will implement three algorithms based on Temporal Difference (TD): TD-Learning for policy evaluation; SARSA; and Q-Learning. You will deploy all three algorithms on the 687-Gridworld. You will also deploy SARSA and Q-Learning on the Mountain Car domain. **Notice that you may not use existing RL code for this problem—you must implement the learning algorithms and the environments entirely on your own and from scratch.** The complete definition of the 687-GridWorld domain can be found in Homework 3. The complete definition of the Mountain Car domain can be found in Homework 2.

General instructions:

- You are free to initialize the q-function in any way that you want. More details about this later.
- Whenever displaying values (e.g., the value of a state), use 4 decimal places; e.g, $v(s) = 9.4312$.
- Whenever showing the value function and policy learned by your algorithm, present them in a format resembling that depicted in Figure 1.

1. TD-Learning on the 687-Gridworld [20 Points, total]

First, implement the TD-Learning algorithm for policy evaluation and use it to construct an estimate, $\hat{v}$, of the value function of the optimal policy, π^*, for the 687-Gridworld. **Remember that both the optimal value function and optimal policy for the 687-Gridworld domain were identified in a previous homework by using the Value Iteration algorithm. For reference, both of them are shown in Figure 1.** When sampling trajectories, set d_0 to be a uniform distribution over $\mathcal{S}$ to ensure that you are collecting returns from all states. Do not let the agent be initialized in an Obstacle state. Notice that using this alternative definition of d_0 (instead of the original one) is important because we want to generate trajectories from all states of the MDP, not just states that are visited under the optimal deterministic policy shown in Figure 1.

Value Function

4.0187	4.5548	5.1575	5.8336	6.4553
4.3716	5.0324	5.8013	6.6473	7.3907
3.8672	4.3900	0.0000	7.5769	8.4637
3.4182	3.8319	0.0000	8.5738	9.6946
2.9977	2.9309	6.0733	9.6946	0.0000

Policy

→	→	→	↓	↓
→	→	→	↓	↓
↑	↑		↓	↓
↑	↑		↓	↓
↑	↑	→	→	G

Figure 1: Optimal value function and optimal policy for the 687-GridWorld domain.

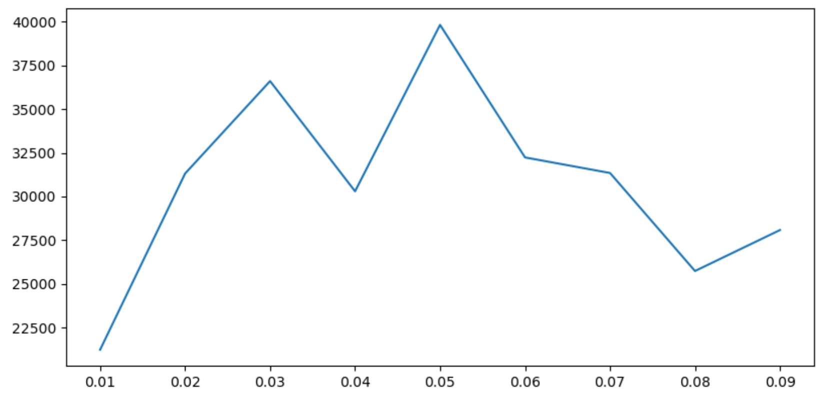


Figure 2: Iterations to Converge vs alpha

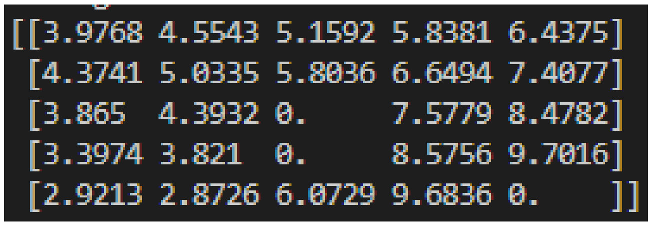
You should run your algorithm until the value function estimate does not change significantly between successive iterations; i.e., whenever the Max-Norm between $\hat{v}_{t-1}$ and $\hat{v}_t$ is at most δ, for some value of δ that you should set empirically. For each run of your algorithm, count how many episodes were necessary for it to converge to an estimate of v^{π^*}. You should run your algorithm, as described above, 50 times and report:

(**Question 1a. 2 Points**) the value of α that was used. You should pick a value of α that causes the algorithm to converge to a solution that is close to v^{π^*}, while not requiring an excessively large number of episodes.

Solution 1a. Hyper-parameter α can be searched by computing the average number of steps it takes for each alpha for the value function to converge to v^{π^*}. I vary α from 0.1 to 0.01 in steps of 0.01. Fig 2 shows the average iterations to converge. Here I run the algorithm 20 times for each value of α and compute the average number of iterations it takes to converge. We can see from the graph that for $\alpha = 0.01$, we get the minimum number of iterations.

$$\alpha = 0.01$$

(**Question 1b. 9 Points**) the average value function estimated by the algorithm—i.e., the average of all 50 value functions learned by the algorithm at the end of its execution. Report, also, the Max-Norm between this average value function and the optimal value function for the 687-Gridworld domain.



```
[[3.9768 4.5543 5.1592 5.8381 6.4375]
 [4.3741 5.0335 5.8036 6.6494 7.4077]
 [3.865  4.3932 0.     7.5779 8.4782]
 [3.3974 3.821  0.     8.5756 9.7016]
 [2.9213 2.8726 6.0729 9.6836 0.    ]]
```

Figure 3: Average value function

Solution 1b. Fig 3 shows the average value function after 50 runs. Max Norm between the average value function and the optimal value function $Maxnorm = 0.076$

(**Question 1c. 9 Points**) The average and the standard deviation across 50 runs of the number of episodes needed for the algorithm to converge.

Solution 1c.

- Average steps to converge = 30723.6
- Std dev of steps to converge = 7896.9025

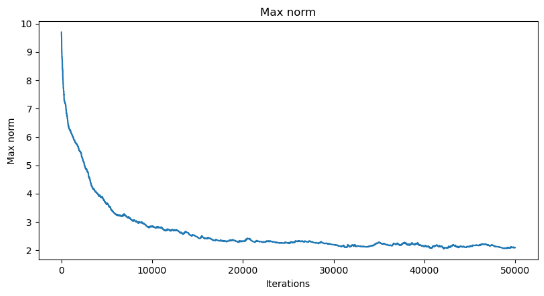
2. SARSA on the 687-Gridworld [20 Points, total]

Implement the SARSA algorithm and use it to construct $\hat{q}$, an estimate of the optimal **action-value function**, q^{π^*}, of the 687-Gridworld domain; and to construct an estimate, $\hat{\pi}$, of its optimal policy, π^*. You will have to make four main design decisions: *(i)* which value of α to use; *(ii)* how to initialize the q-function; you may, e.g., initialize it with zeros, with random values, or optimistically. Remember that $q(s_\infty, \cdot) = 0$ by definition; *(iii)* how to explore; you may use different exploration strategies, such as ϵ-greedy exploration or softmax action selection; and *(iv)* how to control the exploration rate over time; you might choose to keep the exploration parameter (e.g., ϵ) fixed over time, or have it decay as a function of the number of episodes.

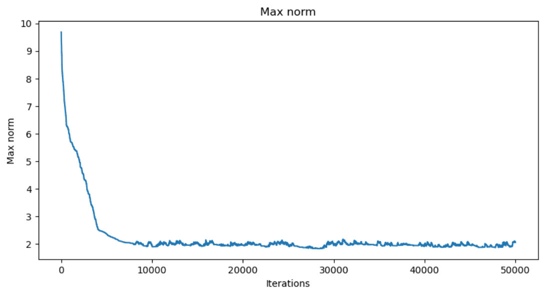
(**Question 2a. 2 Points**) Report the design decisions *(i, ii, iii, and iv)* you made and discuss what was the reasoning behind your choices.

- $\alpha = 0.01$, since α is the learning rate, keeping it low allows the algorithm to converge.
- q initialization to all zeros. Since we are using softmax/eps-greedy policy, we are allowing exploration, q function can be initialized to all zeros.
- Both ϵ- greedy and softmax give similar value function on convergence and similar max-norm between the estimated v and optimal v. Therefore we select softmax policy.
- Fig 4 shows the comparison of the exploration parameter ϵ for constant (Fig 4a) vs decay case (4b), we can see that using ϵ decay converges the algorithm faster. Fig 4c shows constant σ, and Fig 4d shows increasing σ. We can see that increasing σ gives faster convergence.

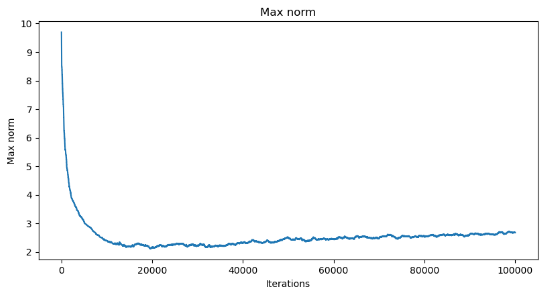
(**Question 2b. 7 Points**) Construct a learning curve where you show, on the x axis, the total number of actions taken by the agent, from the beginning of the training process (i.e., the total number of steps taken across all episodes up to that moment in time). On the y axis, you should show the number of episodes completed up to that point. If learning is successful, this graph should have an increasing slope, indicating that as the agent takes more and more actions (and thus executes more learning updates), it requires fewer actions/timesteps to complete each episode. Your graph should, ideally, look like the figure shown in Example 6.5 of the RL book (2nd edition). To construct this graph, run your algorithm 20 times and show the average curve across those runs.



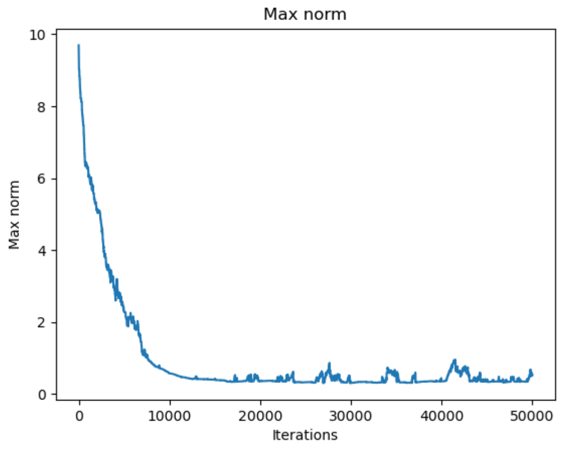
(a) Constant $\epsilon = 0.1$



(b) Decreasing ϵ from 0.1

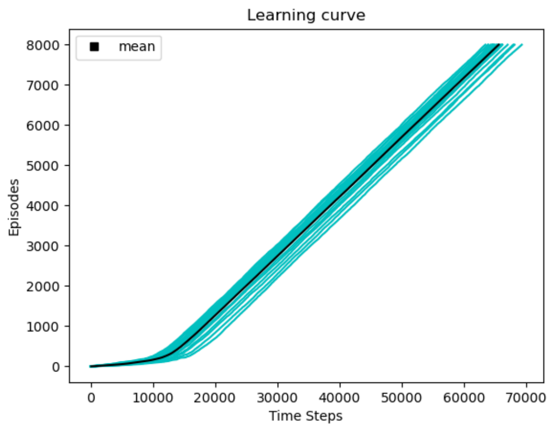


(c) Constant $\sigma = 1$

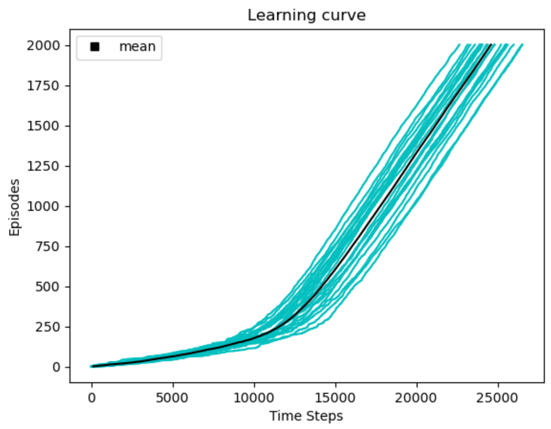


(d) Increasing σ from 1

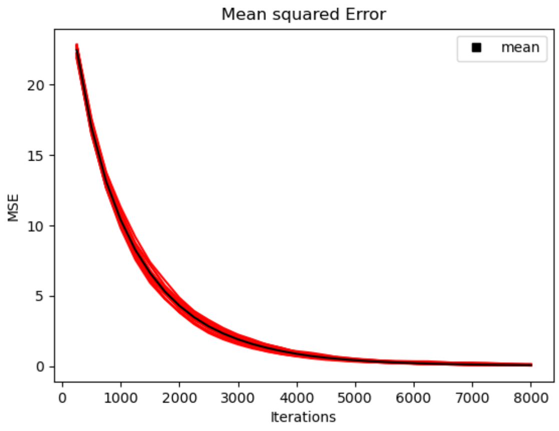
Figure 4: Compare constant vs decreasing/increasing exploration parameters



(a) Action count learning curve



(b) Zoomed in to the curve



(c) MSE

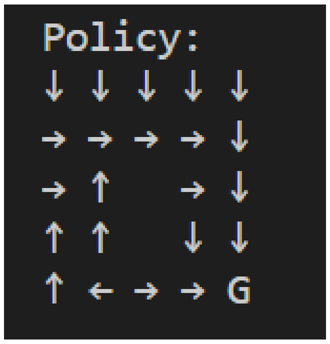
Figure 5: Learning curves for SARSA

Solution 2b. Fig 5a shows the Episodes vs the total number of actions taken by the agent. The ”Cyan” curves shows the learning curves for individual runs, and the ”Black” curve shows the mean. Fig 5b shows the zoomed in view of the same curve, here we can see that the number of actions decreases with the increase in number of episodes.

(Question 2c. 8 Points) Construct another learning curve: one where you show the number of episodes on the x axis, and, on the y axis, the mean squared error between your current estimate of the value function, $\hat{v}$, and the true optimal value function, v^{π^*}. The mean squared error between two value functions, v_1 and v_2, can be computed as $\frac{1}{|\mathcal{S}|}\sum_s (v_1(s) - v_2(s))^2$. Remember that SARSA estimates a q-function ($\hat{q}$), however, not a value function ($\hat{v}$). To compute $\hat{v}$, remember that $\hat{v}(s) = \sum_a \pi(s,a)\hat{q}(s,a)$. Remember, also, that the SARSA policy π, at each given moment, depends on your exploration strategy. Let us say, for example, that you are using ϵ-greedy exploration. Then, in the general case when more than one action may be optimal with respect to $\hat{q}(s,a)$, the ϵ-greedy policy would be as follows:

$$\pi(s,a) = \begin{cases} \frac{1-\epsilon}{|\mathcal{A}^*|} + \frac{\epsilon}{|\mathcal{A}|} & \text{if } a \in \mathcal{A}^* \\ \frac{\epsilon}{|\mathcal{A}|} & \text{otherwise,} \end{cases} \tag{1}$$

where $\mathcal{A}^* = \arg\max_{a \in \mathcal{A}} \hat{q}(s,a)$. To construct the learning curve, run your algorithm 20 times and show the average curve across those runs; i.e., the average mean squared error, computed over 20 runs, as a



```
Policy:
↓ ↓ ↓ ↓ ↓
→ → → → ↓
→ ↑   → ↓
↑ ↑   ↓ ↓
↑ ← → → G
```

Figure 6: Final policy learned by SARSA

function of the number of episodes.

Solution 2c. Fig 5c the Mean squared error for SARSA algorithm. Red curves are for the individual runs and the black curve is the mean. MSE values have been plotted for every 250 iterations. Here we havent shown the MSE for the 0th iteration ($MSE_0 = 32$)

(Question 2d. 3 Points) Show the *greedy policy* with respect to the q-values learned by SARSA.

Solution 2d. Fig 6 shows the learned policy for SARSA. We can observe that this policy is not exactly the same as the optimal policy, nevertheless it still allows the agent to go from any state to the Goal state in less steps.

3. Q-Learning on the 687-Gridworld [20 Points, total]

Answer the same questions as above (*2a, 2b, 2c, and 2d*), but now using the Q-Learning algorithm; i.e., you will be using Q-Learning to solve the 687-Gridworld domain. When computing $\hat{v}$, for question *3c*, you should compute the value function of the greedy policy: $\hat{v}(s) = \max_a \hat{q}(s, a)$. The amount of points associated with questions *3a, 3b, 3c, and 3d*, will be the same as those associated with the corresponding items of Question 2.

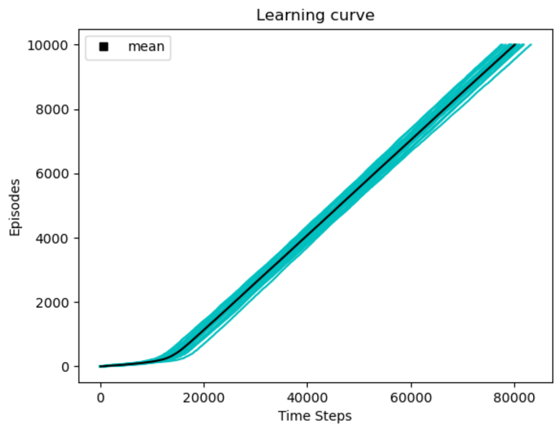
Solution 3a.

- $\alpha = 0.01$, since α is the learning rate, keeping it low allows the algorithm to converge.
- q initialization to all zeros. Since we are using softmax/eps-greedy policy, we are allowing exploration, q function can be initialized to all zeros.
- Both ϵ- greedy and softmax give similar value function on convergence and similar max-norm between the estimated v and optimal v. Therefore we select softmax policy.
- Similar to the logic in SARSA, we keep an increasing sigma.

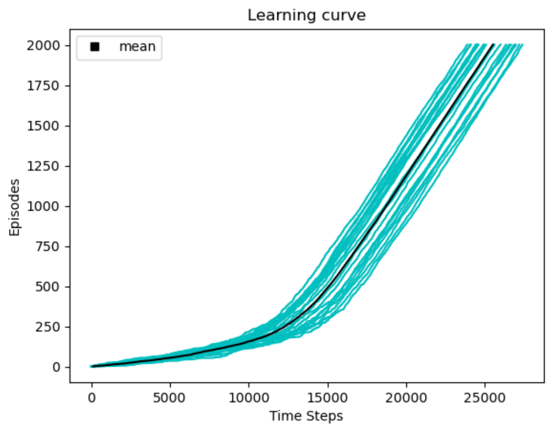
Solution 3b. Fig 7a shows the Episodes vs the total number of actions taken by the agent. The "Cyan" curves shows the learning curves for individual runs, and the "Black" curve shows the mean. Fig 7b shows the zoomed in view of the same curve, here we can see that the number of actions decreases with the increase in number of episodes. Since multiple actions could have same probability, therefore *argmax* of probabilities picks up any state which might not be the same as the optimal policy.

Solution 3c. Fig 7c the Mean squared error for Q Learning algorithm. Red curves are for the individual runs and the black curve is the mean. MSE values have been plotted for every 250 iterations.

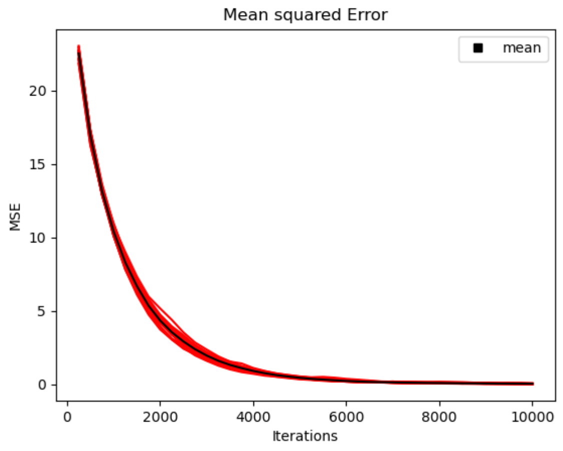
Solution 3d. Fig 8 shows the learned policy for Q Learning. We can observe that this policy is not exactly the same as the optimal policy, nevertheless it still allows the agent to go from any state to the Goal state in less steps.



(a) Learning curve for Q Learning

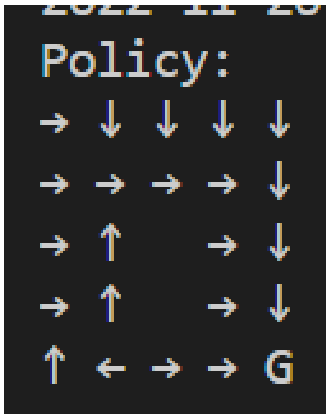


(b) Zoomed in to the curve



(c) MSE

Figure 7: Learning curves for Q Learning



```
Policy:
→ ↓ ↓ ↓ ↓
→ → → → ↓
→ ↑   → ↓
→ ↑   → ↓
↑ ← → → G
```

Figure 8: Final policy learned by Q Learning

4. **SARSA on the Mountain Car domain [20 Points, total]** You will now use the SARSA algorithm to solve the Mountain Car domain. Notice that in the Mountain Car domain, states are continuous. You will first have to discretize them in order to be able to deploy the standard/tabular version of SARSA[1]. You will have to make five main design decisions, here: the same four design decisions as in Question 2, and also an additional design decision (design decision v): the level of discretization to be used; i.e., the number of bins used to discretize the continuous state of the MDP.

(**Question 4a. 2 Points**) Report the design decisions *(i, ii, iii, iv, and v)* you made and discuss what was the reasoning behind your choices.

Solution 4a.

- $\alpha = 0.01$, since α is the learning rate, keeping it low allows the algorithm to converge.
- q initialization to all zeros. Since we are using softmax/eps-greedy policy, we are allowing exploration, q function can be initialized to all zeros.
- Since we do not have the optimal policy in this case, we check the number of iterations does the algorithm converge in for each type of policy. From Fig 9 we can see that ϵ greedy converges faster than softmax policy. Therefore we use ϵ greedy policy.
- We use ϵ decay for ϵ greedy policy. $\epsilon = min(0.01, 1/num\_iterations)$
- We use 5 bins for discretization.

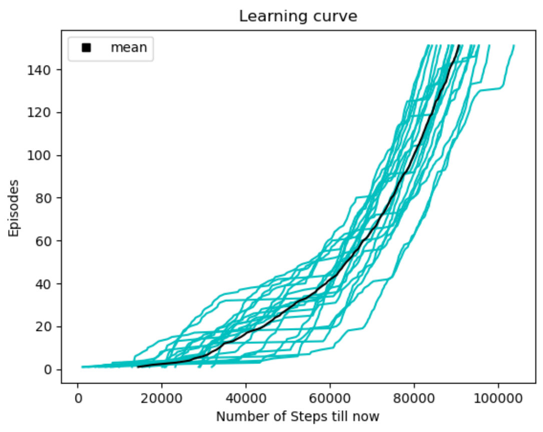
(**Question 4b. 9 Points**) Construct a learning curve where you show, on the x axis, the total number of actions taken by the agent, from the beginning of the training process (i.e., the total number of steps taken across all episodes up to that moment in time). On the y axis, you should show the number of episodes completed up to that point. If learning is successful, this graph should have an increasing slope, indicating that as the agent takes more and more actions (and thus executes more learning updates), it requires fewer actions/timesteps to complete each episode. Your graph should, ideally, look like the figure shown in Example 6.5 of the RL book (2nd edition). To construct this graph, run your algorithm 20 times and show the average curve across those runs.

Solution 4b. Fig 9a shows the learning curve for Episodes vs the total number of actions taken until that episode. Here we can see that as the number of episodes increases it requires fewer actions to complete each episode.

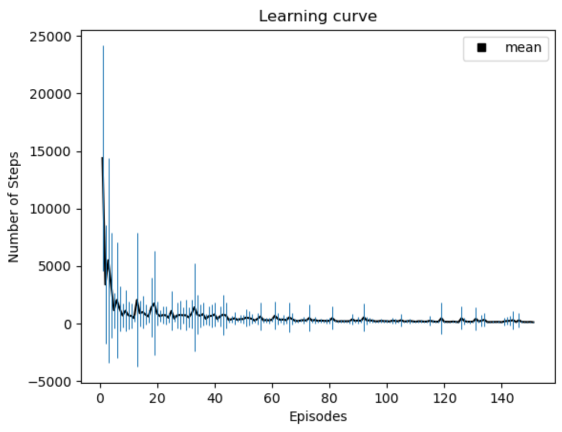
(**Question 4c. 9 Points**) Construct another learning curve: one where you show the number of episodes on the x axis, and, on the y axis, the number of steps needed by the agent to reach the goal state. To construct this graph, run your algorithm 20 times and show the average curve across those runs (i.e., the average number of steps to reach the goal, computed across 20 runs, as a function of the number of episodes). Show, also, the standard deviation associated with each of these points. Your graph should, ideally, look like that in Figure 10.

Solution 4c. Fig 9b shows the graph for the average number of steps per episode vs the Episode number.

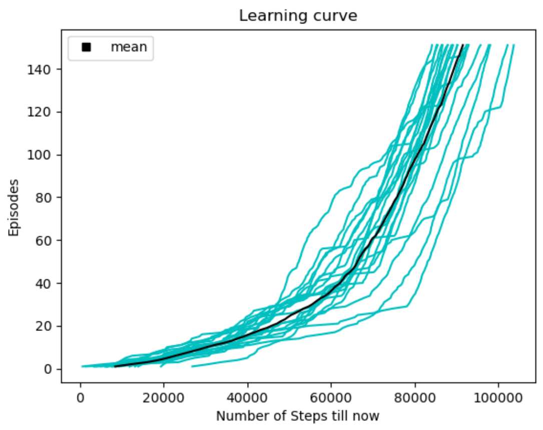
[1] Suppose you are discretizing a continuous state $s = [x, v]$, where the range of x is $[-1.2, 0.5]$ and the range of v is $[-0.07, 0.07]$. Suppose you choose to discretize each component of the state into 3 bins. To do so, you will break down the interval of values of x (from -1.2 to 0.5) into three contiguous bins: the first bin would correspond to values of x from -1.2 to -0.634; the second bin would correspond to values of x from -0.634 to -0.068; and the third bin, to values of x from -0.068 to 0.5. You would perform a similar discretization process over the state variable v. Then, the final discretized version of a continuous state $s = [x, v]$ would correspond to a pair (b_x, b_v) indicating the number of the bin, b_x, on which the x variable fell; and the number of the bin, b_v, where the v variable fell. Notice that if you discretize a 2-dimensional continuous state (such as that of the Mountain Car domain) into N bins, you will end up with N^2 discrete states. Using a coarser discretization results in fewer states, but where each discrete state is a "less accurate" representation of the true underlying continuous state. Using a finer discretization results in a "more accurate" representation of the true underlying continuous state, but generates a larger state space.



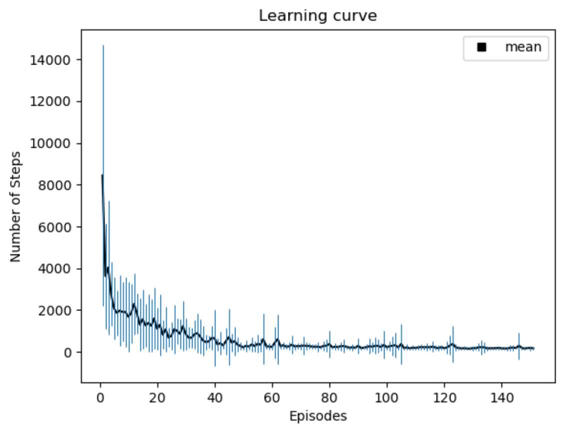
(a) ϵ greedy policy for Mountain Car SARSA



(b) steps per episode vs Episodes for ϵ greedy



(c) Softmax policy for Mountain Car



(d) steps per episode vs Episodes for Softmax Policy

Figure 9: Mountain Car Learning curves for SARSA

5. Q-Learning on the Mountain Car domain [20 Points, total]

Answer the same questions as above (*4a, 4b, and 4c*), but now using the Q-Learning algorithm; i.e., you will be using Q-Learning to solve the Mountain Car domain. The amount of points associated with questions *5a, 5b, and 5c*, will be the same as those associated with the corresponding items of Question 4.

Solution 5a.

- $\alpha = 0.01$, since α is the learning rate, keeping it low allows the algorithm to converge.
- q initialization to all zeros. Since we are using softmax/eps-greedy policy, we are allowing exploration, q function can be initialized to all zeros.
- Since we do not have the optimal policy in this case, we check the number of iterations does the algorithm converge in for each type of policy. From Fig 11 we can see that ϵ greedy converges faster than softmax policy. Therefore we use ϵ greedy policy.
- We use ϵ decay for ϵ greedy policy. $\epsilon = min(0.01, 1/num\_iterations)$
- We use 5 bins for discretization.

Solution 5b. Fig 11a shows the learning curve for Episodes vs the total number of actions taken until that episode. Here we can see that as the number of episodes increases it requires fewer actions to complete each episode.

Solution 5c. Fig 11b shows the graph for the average number of steps per episode vs the Episode number.

---

6. Extra Credit [16 Points, total]

(**Question 6a. 8 Points**) Answer questions *(4b)* and *(4c)* once again, but now using different discretization levels compared to the one you selected when solving Question 4. In particular, you should use the same hyperparameters as in Question 4: same value of α, q-function initialization strategy, exploration strategy, and method for controlling the exploration rate over time. Now, however, you will experiment with *different* discretization levels compared to the one you selected when solving Question 4. You should, in particular, run experiments with 25% fewer bins, 50% fewer bins, 25% more bins, and 50% more bins. Let us say, for example, that in Question 4 you decided to discretize the state features into N bins. In this extra-credit question, you should run the same experiments as in *(4b)* and *(4c)*, but now using $0.5N$ bins, $0.75N$ bins, $1.25N$ bins, and $1.5N$ bins. Discuss the impact that each of these alternative discretization schemes had

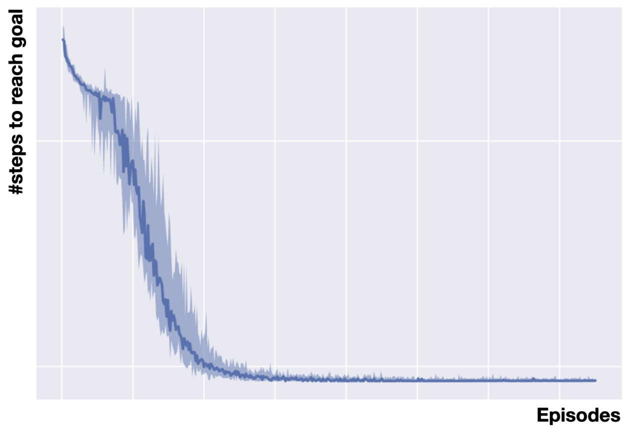
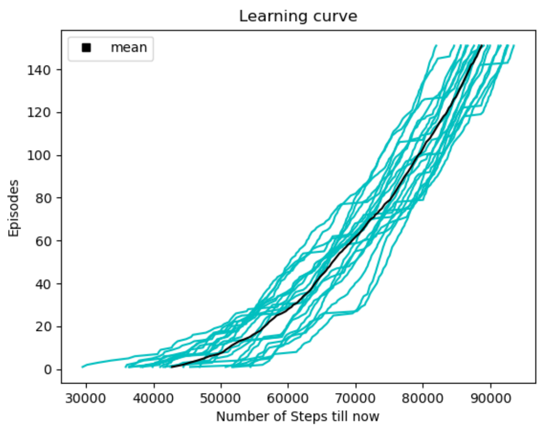
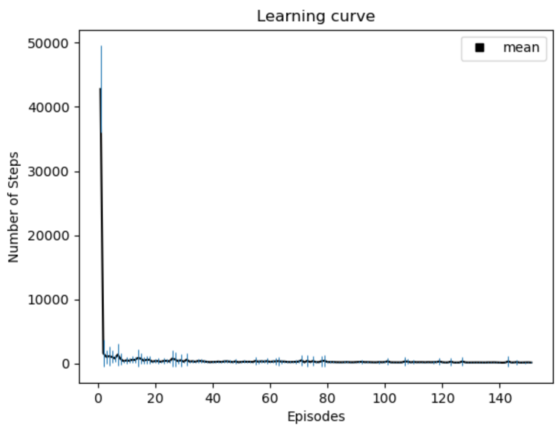


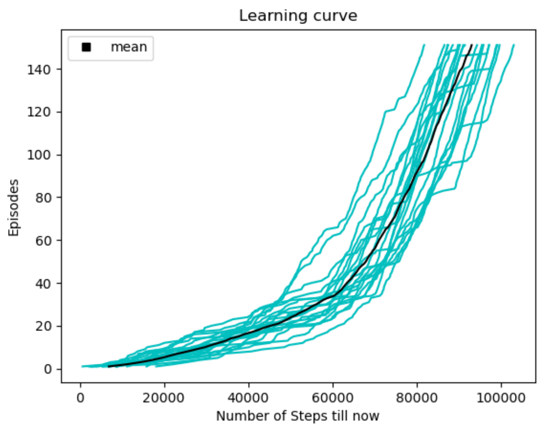
Figure 10: Example of a learning curve: average number of steps to reach the goal (and the associated standard deviation) as a function of the number of episodes.



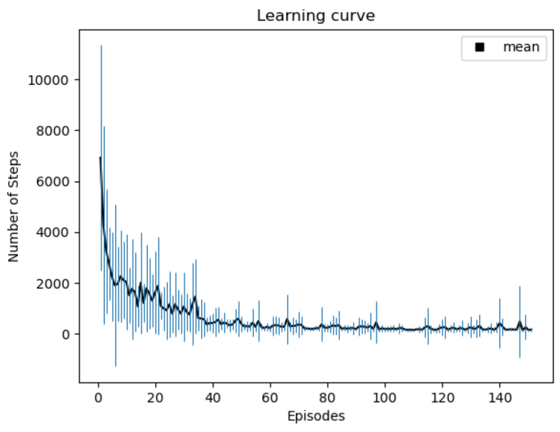
(a) ϵ greedy policy for Mountain Car Q Learning



(b) steps per episode vs Episodes for ϵ greedy



(c) Softmax policy for Mountain Car

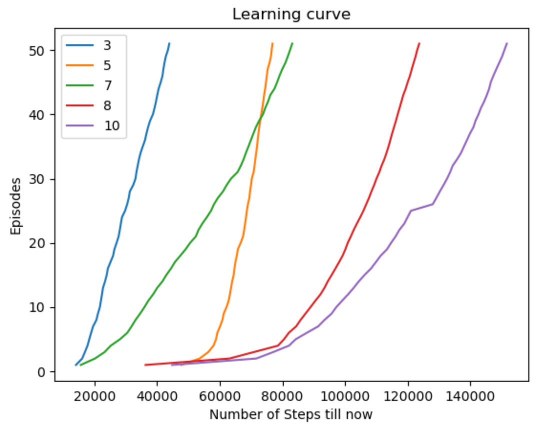


(d) steps per episode vs Episodes for Softmax Policy

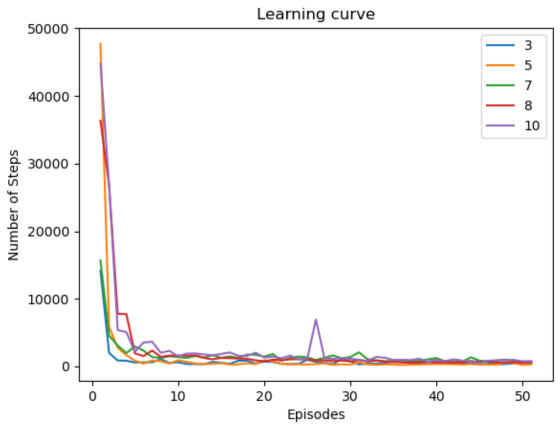
Figure 11: Mountain Car Learning curves for Q Learning

Num Bins	Steps to reach goal state
3	241.18
5	162.99
7	147.28
8	184.73
10	215.32

Table 1: Average number of steps it takes to reach goal state with learned policy



(a) Episodes vs Total steps till that episode for different number of bins



(b) steps per episode vs Episodes for different number of bins

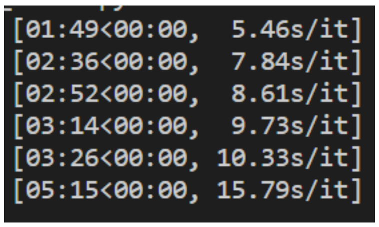
Figure 12: Mountain Car Learning curves for different number of bins

both on the performance of the policy learned by the algorithm, and on the (wall-clock) time needed to complete each of the four experiments, compared to the wall-clock time required to run the corresponding experiment in Question 4.

Solution 6a. From Fig 12b we can see the smaller number of bins converge faster, with 3 being the fastest to converge while training. Table 1 shows the average number of steps it takes the mountain car to reach the goal state for different bins after the policy is learned.

Fig 13 shows the wall clock times for training with different bins. Topmost is the lowest bin ($bin = 3$), Bottom most is the highest bin ($bin = 10$).

(Question 6b. 8 Points) Answer questions *(4b)* and *(4c)* once again, but now using *two* different ways of initializing the q-function optimistically. You should use the same hyperparameters as in Question 4: same value of α and discretization levels. Now, however, you will experiment with *different* ways of optimistically initializing the q-function. Let M be the maximum return achievable in Mountain Car. First, deterministically initialize all entries of the q-function with the optimistic value $2M$. Next, stochastically initialize

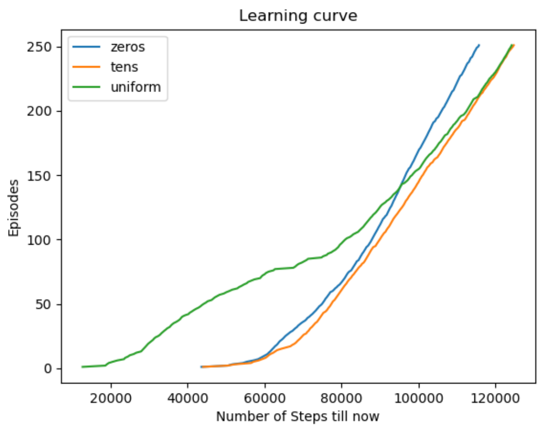


```
[01:49<00:00,  5.46s/it]
[02:36<00:00,  7.84s/it]
[02:52<00:00,  8.61s/it]
[03:14<00:00,  9.73s/it]
[03:26<00:00, 10.33s/it]
[05:15<00:00, 15.79s/it]
```

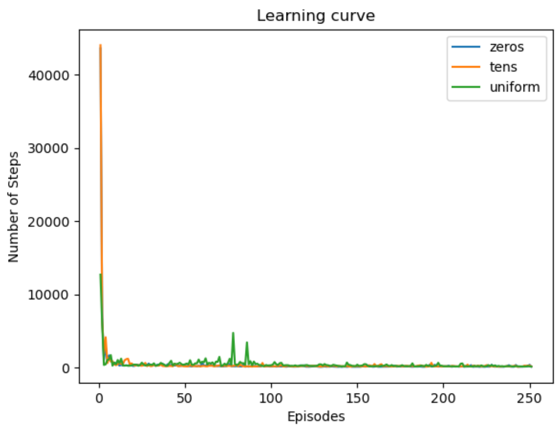
Figure 13: Clock time for different Number of bins (top - lowest bin 3, bottom highest bin - 10)

q initialization	Steps to reach goal state
All zeros	136.02
2M	153.01
$Uniform(1.5M, 3.5M)$	157.34

Table 2: Average number of steps it takes to reach goal state with learned policy



(a) Episodes vs Total steps till that episode for different initialization of q



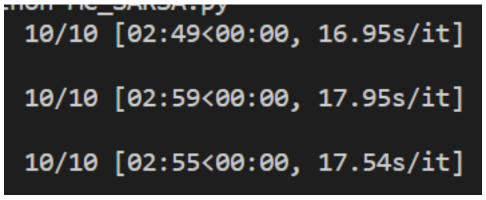
(b) steps per episode vs Episodes for different q initialization

Figure 14: Mountain Car Learning curves for different q initialization

all entries of the q-function with numbers drawn uniformly at random from the interval $[1.5M, 3.5M]$. In this extra-credit question, you should run the same experiments as in *(4b)* and *(4c)*, but now using these alternative ways of optimistically initializing the q-function. **Importantly, when running these experiments you should set the exploration rate to zero (i.e., $\epsilon = 0$, always)**. Discuss the impact that these q-function initialization strategies had both on the performance of the policy learned by the algorithm, and on the (wall-clock) time needed to complete each of the four experiments, compared to the wall-clock time required to run the corresponding experiment in Question 4.

Solution 6b. The mountain car with $\gamma = 0.9$ has $-9 > M > -10$, so we take 2M = 10, and 1.5M = 0 and 3.5M = 40. From Fig 14b we can see the initialization with all zeros gives the most smooth convergence. Table 2 shows the average number of steps it takes the mountain car to reach the goal state for q initialization after the policy is learned. Here also we see that q initialized to all zeros, has the lowest steps to reach goal.

Fig 15 shows the wall clock times for training with different initialization. Topmost is the zeros init, middle one is the initialization to all 2M (10s) Bottom most is the initialization to uniform randomly sampled between 1.5M and 3.5M (0, 40). We see almost the same time to train in all the three cases.



```
10/10 [02:49<00:00, 16.95s/it]

10/10 [02:59<00:00, 17.95s/it]

10/10 [02:55<00:00, 17.54s/it]
```

Figure 15: Clock time for different Number of bins (top - lowest bin 3, bottom highest bin - 10)