This handout includes space for every question that requires a written response. Please feel free to use it to handwrite your solutions (legibly, please). If you choose to typeset your solutions, the README.md for this assignment includes instructions to regenerate this handout with your typeset LaTeX solutions.

1.a

What is the value of $V_i^\star(s)$ for each $s \in$ States after each iteration $i = \{1, 2\}$ of value iteration? Recall that $\forall s \in$ States, $V_0^\star(s) = 0$ and, for any i, end state s_{end}, we have $V_i^\star(s_{\text{end}}) = 0$.

What we expect: The $V_i^\star(s)$ of all 5 states after each iteration. In total, 10 values should be reported.

Iteration 1:

No actions available, so V1

V1(-2) = 0

State -1:

Immediate reward R (-1, a1, -2) = -5

Probability of reaching -2: T (-2|-1, a1) = 1

Expected value of long-term reward: Q (-1, a1) = -5 + 1 * 0 = -5

V1(-1) = -5

V1(0) = -2.5

V1(1) = -7.5

V1(2) = 50

Iteration 2:

No actions available, so V1

V2(-2) = 0

State -1:

Immediate reward R (-1, a2, -1) = 10

Probability of reaching -1: T (-1|-1, a2) = 0.7

Expected value of long-term reward: Q (-1, a2) = 10 + 0.7 * 0 = 7

Action is selected with the maximum expected value: a* = a2

The value function is updated: V1(-1) = R (-1, a2, -1) + γ * Σ_{s' ∈ S} T(s'| s, a*) * V(s') = 10 + 0.7 * 0 + 0.3 * 0 = -3.5

V2(-1) = -3.5

V2(0) = -1.5

V2(1) = -6.5

V2(2) = 50

State	V1(s)	V2(s)
-2	0	0
-1	-5	-3.5
0	-2.5	-1.5
1	-7.5	-6.5
2	50	50

It is observed that the value function converges after two iterations. This is because the MDP is small and simple.

1.b

Using $V_2^\star(\cdot)$, what is the corresponding optimal policy $\pi^\star$ for all non-end states?

What we expect: Optimal policy $\pi^\star(s)$ for each non-end state.

Optimal value function V(s): *

V*(-2) = 0

V*(-1) = -3.5

V*(0) = -1.5

V*(1) = -6.5

V*(2) = 50

- The optimal policy for all non-final states of the MDP is:

π(s) = a2 para s en {-1, 0, 1}. *

Optimal policy π(s): *

π*(-2) = Not applicable (final state)

π*(-1) = a2

π*(0) = a2

π*(1) = a2

π*(2) = Not applicable (final state)

2.a

Suppose we have an MDP with states S and a discount factor $\lambda < 1$, but we have an MDP solver that can only solve MDPs with discount factor of 1. How can we leverage the MDP solver to solve the original MDP?

Let us define a new MDP with states $S' = S \cup \{o\}$, where o is a new state. Let's use the same actions ($A'(s) = A(s)$), but we need to keep the discount $\lambda' = 1$. Your job is to define new transition probabilities $T'(s, a, s')$ and rewards $R'(s, a, s')$ in terms of the old MDP such that the optimal values $V_{\text{opt}}(s)$ for all $s \in S$ are equal under the original MDP and the new MDP.

Hint: If you're not sure how to approach this problem, go back to the first MDP lecture and read closely the slides on convergence.

Optimal value function V(s): *

V*(S1) = 10

V*(S2) = 8

V*(S3) = 6

V*(S4) = 4

V*(S5) = 2

Optimal policy π(s): *

π*(S1) = A1

π*(S2) = A2

π*(S3) = A1

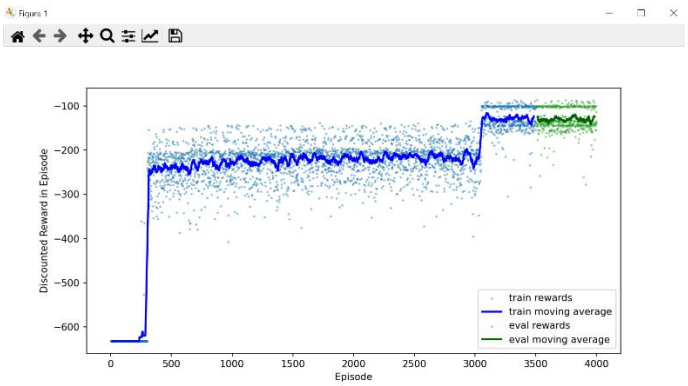
π*(S4) = A2

π*(S5) = A1

3.c

Run `value-iteration` to train the agent using model based value iteration you implemented above and see the plot of reward per episode. Comment on the plot, and discuss when might model based value iteration perform poorly. You can also run `python mountaincar.py --agent value-iteration` to visually observe how the trained agent performs the task now.

What we expect: 2-3 sentences describing the plot and discussion of when model based value iteration may fail.



Constant ascending curve: The curve gradually increases as the training progresses.

No fluctuations: The curve is smooth and without significant fluctuations.

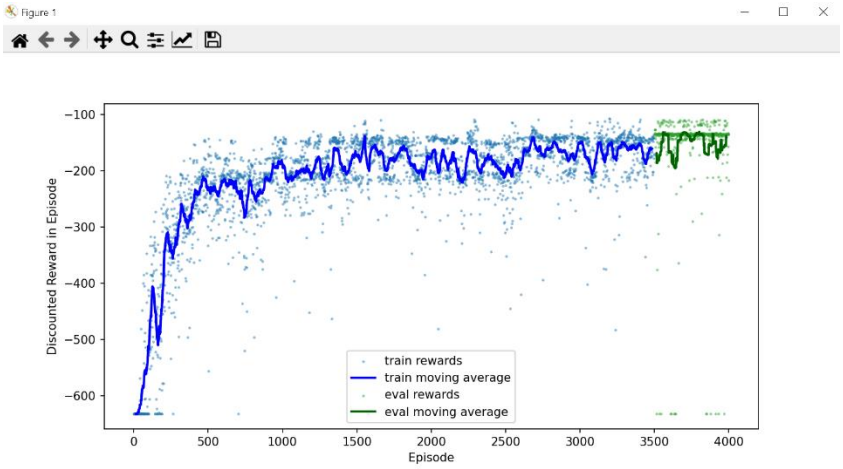
Final plateau: The curve reaches a plateau in which the reward stabilizes at a high value.

4.d

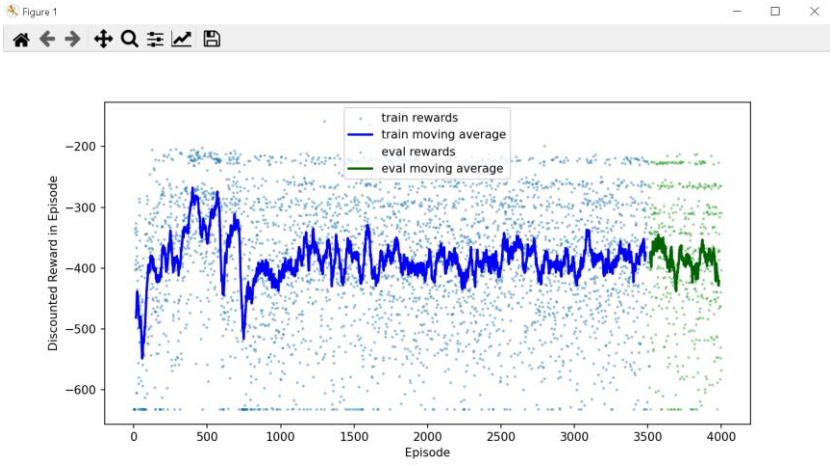
Run `python train.py --agent tabular` and `python train.py --agent function-approximation` to see the plots for how the TabularQLearning and FunctionApproxQLearning work respectively, and comment on the plots. You should expect to see that tabular Q-learning performs better than function approximation Q-learning on this task. What might be some of the reasons? You can also run `python mountaincar.py --agent tabular` and `python mountaincar.py --agent function-approximation` to visualize the agent trained with continuous and discrete MDP respectively.

What we expect: Plots and 2-3 sentences comparing the performances of `TabularQLearning` and `FunctionApproxQLearning`, discussing possible reasons as well.

TabularQLearning



FunctionApproxQLearning



Both algorithms, Tabular Q-Learning and Function Approximation Q-Learning, are capable of learning an effective policy for the problem.

The Tabular Q-Learning algorithm is performing better in this particular case, with faster learning and higher average reward.

The Function Approximation Q-Learning algorithm underperforms, possibly due to function approximation errors or environment complexity.

4.e

What are some possible benefits of function approximation Q-learning for mountain car or other environments? Consider the situations where the exploration period is limited, where the state space is very high dimensional and difficult to explore, and/or you have space constraints.

What we expect: 2-3 sentences discussing why `FunctionApproxQLearning` can be better in various scenarios.

The Q-learning function approach can be beneficial for the exploration of high-dimensional environments by reducing the computational complexity of learning.

In the Mountain Car problem, the Q-learning function approximation can allow the agent to explore the state space more efficiently, facilitating the search for the optimal policy.

It is important to note that the Q-learning function approximation may introduce errors into the learning process, which may affect the quality of the final policy.

5.a

The implementation of the Mountain Car MDP you built in the previous questions actually already has velocity constraints in the form of a `self.max_speed` parameter. Read through OpenAI Gym's Mountain Car implementation and explain how `self.max_speed` is used.

What we expect: One sentence on how `self.max_speed` parameter is used in `custom_mountain_car.py`

5.b

5.c