Problem 3

1 Problem Formulation

We formulate this Mountain Car problem as a finite-state MDP with unknown motion model and cost. With this idea, we are able to formulate the following definitions:

- State space $\mathcal{X} = \mathcal{S} \times \mathcal{V}$, where $s \in \mathcal{S} = \{s| -1.2 \leq s \leq 0.6\}$ is the position of the car and the velocity $v \in \mathcal{V} = \{v| -0.07 \leq v \leq 0.07\}$.

  The terminal state is $\mathbf{x} = (0.5, v)$ for any $v \in \mathcal{V}$.

  The initial state is $\mathbf{x} = (s_0, 0)$, where s_0 is a random position from -0.6 to -0.4.

  In order to solve this problem in a discrete form we can define two discrete steps d_s and d_v such that $\mathcal{S} = \{-1.2, -1.2 + d_s, -1.2 + 2d_s, ..., 0.6\}$, $\mathcal{V} = \{-0.07, -0.07 + d_v, -0.07 + 2d_v, ..., 0.07\}$.

- Control space $\mathcal{U}(x) = \{0 \text{ (push car left)}, 1 \text{ (no push)}, 2 \text{ (push car right)}\}$ for $x_1 \neq 0.5$.

- Motion model $p_f(\mathbf{x}'|\mathbf{x}, u)$: We do not know it but we can somehow learn it from the environment.

- Cost: We have no idea of the cost so we have to depend on the feedback from the environment.

Our goal is find the optimal policy enabling us to reach the terminal state. Mathematically, we can formulate it as an optimization problem for Q-value function, or a model-free reinforcement learning control problem:

$$Q^*(\mathbf{x}, u) = \min_{\pi} l(\mathbf{x}, u) + \gamma \mathbb{E}_{\mathbf{x}' \sim p_f(.|\mathbf{x},u)}[\min_{u \in \mathcal{U}(x)} Q^*(x', u')] \tag{1}$$

$$\pi^*(x) = \arg\min_{u \in \mathcal{U}(x)} Q^*(\mathbf{x}, u) \tag{2}$$

$$s.t.\ \mathbf{x}_0 = (s_0, 0) \tag{3}$$

where s_0 is a randomly drawn from -0.6 to -0.4.
We are asked to implement both on- and off-policy TD (Temporal Differential) Policy Iteration algorithms to solve this problem.

2 Technical Approach

We want to solve this problem by both on- and off-policy TD algorithms, and then compare their difference. For on-policy algorithm we will implement SARSA algorithm; for off-policy, since Q-Learning is easily adapted from SARSA, it is implemented in this report.

2.1 SARSA Algorithm

SARSA, or State-Action-Reward-State-Action, sends an agent to interact with the environment and updates the policy based on action taken, hence it is known as an on-policy algorithm. The main update is shown as

$$Q(\mathbf{x}, u) \leftarrow Q(\mathbf{x}, u) + \alpha[l(\mathbf{x}, u) + \gamma Q(\mathbf{x}', u') - Q(\mathbf{x}, u)], \tag{4}$$

where α is the learning rate, $0 \leq \gamma \leq 1$ is the discount factor. Q-values represent the possible cost in the next time step for taking action u in state $\mathbf{x}$, plus the discounted future cost received from the next state-action observation.
The algorithm is shown in Algorithm 1.

Algorithm 1: SARSA algorithm

Initialize $Q(\mathbf{x}, u) \forall \mathbf{x} \in \mathcal{X}, u \in \mathcal{U}(\mathbf{x})$ except $Q(\mathbf{x}_T, .) = 0$.
for *Each episode* **do**
  Choose u from $\mathbf{x}$ using policy derived from Q(e.g. ϵ-greedy)
  repeat
    Take action u, observe cost l and $\mathbf{x}'$
    Choose next action u' from $\mathbf{x}'$ using policy derived from Q(e.g. ϵ-greedy)
    $Q(\mathbf{x}, u) \leftarrow Q(\mathbf{x}, u) + \alpha[l(\mathbf{x}, u) + \gamma Q(\mathbf{x}', u') - Q(\mathbf{x}, u)]$
    $\mathbf{x} \leftarrow \mathbf{x}'; u \leftarrow u'$
  until $\mathbf{x}$ *is terminal.*;
end

2.2 Q-Learning Algorithm

Q-learning is an off-policy TD control algorithm. Different from SARSA, as an off-policy algorithm Q-learning improves a policy π that is different from the behavior policy, which is used to generate next state data. Then it explores to find optimal policy greedily. See Algorithm 2.

Algorithm 2: Q-Learning algorithm

Initialize $Q(\mathbf{x}, u) \forall \mathbf{x} \in \mathcal{X}, u \in \mathcal{U}(\mathbf{x})$ except $Q(\mathbf{x}_T, .) = 0$.
for *Each episode* **do**
  repeat
    Choose u from $\mathbf{x}$ using policy derived from Q(e.g. ϵ-greedy)
    Take action u, observe cost l and $\mathbf{x}'$
    $Q(\mathbf{x}, u) \leftarrow Q(\mathbf{x}, u) + \alpha[l(\mathbf{x}, u) + \gamma \min_u Q(\mathbf{x}', u') - Q(\mathbf{x}, u)]$
    $\mathbf{x} \leftarrow \mathbf{x}'$
  until $\mathbf{x}$ *is terminal.*;
end

2.3 ϵ-greedy Exploration

To make sure the exploration of the control space $\mathcal{U}(\mathbf{x})$ must encounter all possible controls at state $\mathbf{x}$ with non-zero probability, we need to develop a strategy for choosing actions. Also, we want to take optimal control more. Therefore, we could use ϵ-greedy. From Eq. 5 we can see that as long as we choose ϵ properly we could explore all possible actions while have relatively larger bias towards optimal control.

$$\pi(\mathbf{x}|u) = \mathbb{P}(u_t = u|\mathbf{x}_t = \mathbf{x}) := \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{U}(\mathbf{x})|} & \text{if } u = \arg\min_{u' \in \mathcal{U}(\mathbf{x})} Q(\mathbf{x}, u') \\ \frac{\epsilon}{|\mathcal{U}(\mathbf{x})|} & \text{otherwise} \end{cases} \tag{5}$$

2.4 Conditions for Convergence

One necessary conditions for convergence to Q^* is that, as episodes continue, the length of each episodes will not change too much (See Fig.4), and Q value will be stable around some value (See Fig.1(a)).

3 Results

3.1 Experiments and Parameters Setting

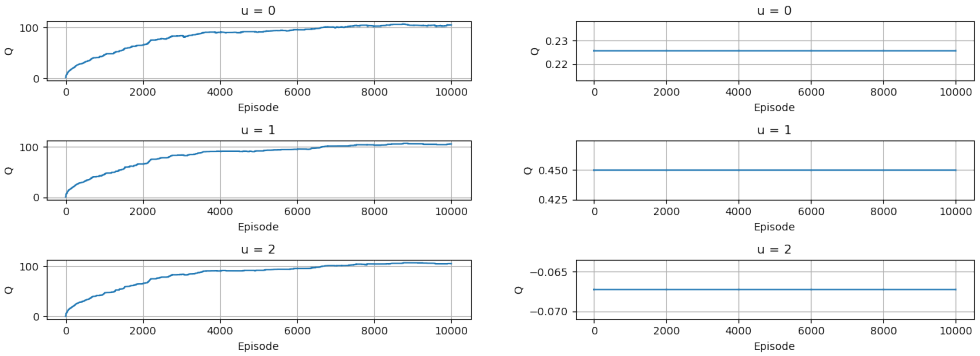
We use `gym` package from OpenAI to construct the Mountain Car problem environment. In particular, there are a few hyper-parameters in this problem: discrete step d_s and d_v, learning rate α, discount factor γ and ϵ for ϵ-greedy. For training, the terminal horizon $T = 5000$ for each episode, and we totally train 10000 episodes for each combination of the parameters; for testing, we consider to use the optimal policy we find to finish the climbing in 200 steps and we will test our optimal policy for 100 different s_0 drawn from -0.6 to -0.4. For initialization, we initialize Q-values by drawing random values from standard normal distribution.

3.2 Results

The quantitative results include Q-value over time and the optimal policy over the state space. Also, we have to try different combinations of hyper-parameters. A qualitative result is included in the zip. file as a video. For learning, the terminal horizon $T = 5000$; for testing, we consider to use our policy to finish the climbing in 200 steps.

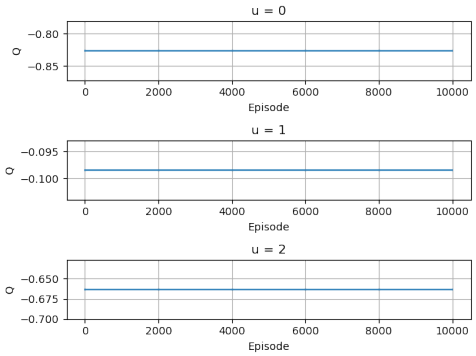
For Q-values, in order to see the processing over time, we choose several representative states to plot the figures. The states are: $\mathbf{x} = (0, 0), (-1.0, 0.05), (0.25, -0.05)$. The figures are shown in Fig.1 and Fig.2 for on- and off-policy algorithms, respectively. The hyper-parameters are as follows: $d_s = 0.1, d_v = 0.01, \gamma = 1.0, \alpha = 0.1, \epsilon = 0.1$. As we can see, the Q-value will rise and then converges to some value. For $Q(-1.0, 0.05)$ and $Q(0.25, -0.05)$, these two states are hardly reach so their values keep the same over the episode.

The optimal policy is shown in Fig.3, with the same hyper-parameters: $d_s = 0.1, d_v = 0.01, \gamma = 1.0, \alpha = 0.1, \epsilon = 0.1$. We can see that, for position $s < 0$ and velocity $v < 0$, the optimal policy is "push left", this is reasonable since while the car are on the left side it is better to push left to get the potential of the gravity; for position $s < 0$ and velocity $v > 0$, the policy is "no push" or "push right", this is because we



(a) Q(0,0)

(b) Q(-1.0, 0.05)



(c) Q(0.25, -0.05)

Figure 1: Q-values over Episode, SARSA

There are also some results with different parameter combinations.

3.3 Parameters Analysis

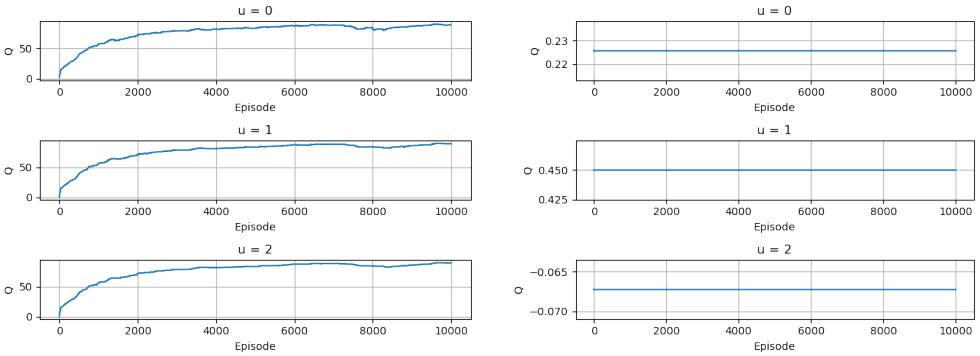
Actually, there are quite a few hyper-parameters in this problem. Since this is a problem with rather small control space, we can discuss the effect of the parameters on the convergence of the Q-values and the number of steps to reach the goal in the test.

For discrete steps d_s and d_v, smaller those two parameters are, larger the state space becomes, which means we need more episodes for training.

For discount factor γ, it is clear that the larger γ is, the further we see in the next state, which will provide a much more accurate policy. A small γ will decrease Q-values. See Table 1. The steps are taken mean values of 100 tests. Except γ, other parameters keep the same, i.e. $d_s = 0.1, d_v = 0.01, \alpha = 0.1, \epsilon = 0.1$.

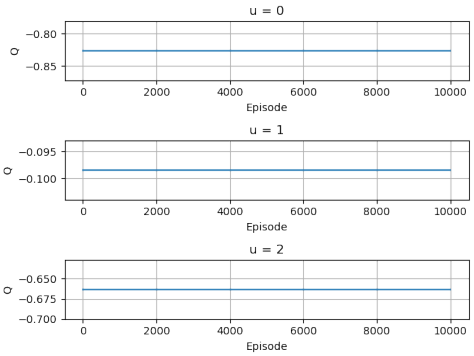
For learning rate α, by my experiment if we want to have fewer steps to reach the goal, we need a smaller α with the same number of training episodes to converge to the "more" optimal point. However, a small α will decrease the speed of convergence. See Table. 2.

For ϵ, it decides the degree how sparse for exploration, see Table 3. From Eq. 5, the smaller ϵ,



(a) Q(0,0)

(b) Q(-1.0, 0.05)



(c) Q(0.25, -0.05)

Figure 2: Q-values over Episode, Q-Learning

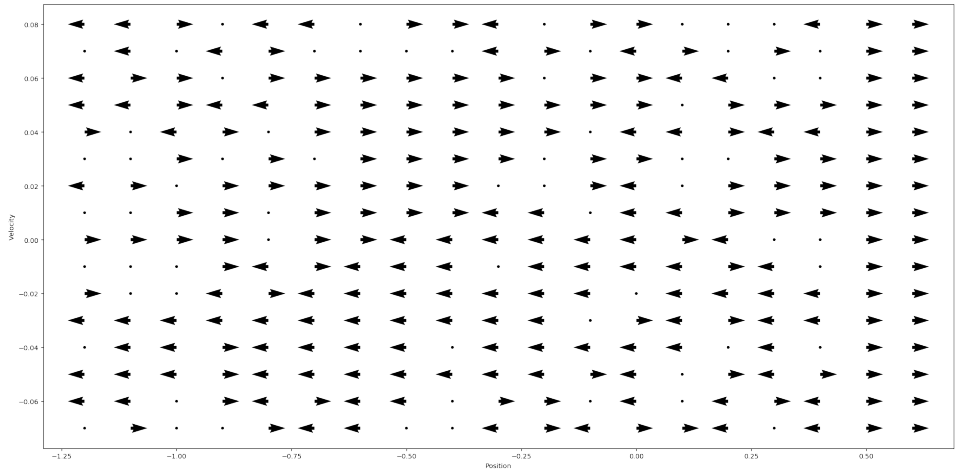
Table 1: Different γ For Testing

γ	1.0	0.9	0.7	0.5
Ave. Steps (SARSA)	143.80	144.96	225.04	241.43
Ave. Steps (Q-Learning)	170.31	213.36	227.44	235.19

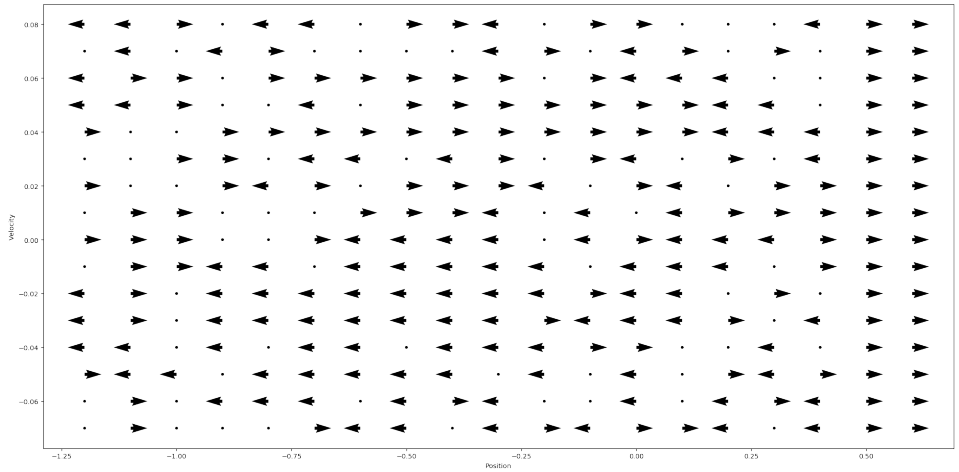
Table 2: Different α For Testing

α	0.1	0.3	0.7	0.9
Ave. Steps (SARSA)	143.80	226.04	239.78	301.0
Ave. Steps (Q-Learning)	177.11	199.59	236.71	300.89

the larger chance we choose the current "optimal" control, which means we trust optimal policy more. However, this would narrow other control options. Since the control space of this problem is not big (only 3 options for each state except for terminal states), this parameter doesn't have big influence on the final results given 10000 episodes in training if we are using SARSA. However, if we are trying to use Q-Learning, since we choose actions greedily, a larger ϵ will decrease the speed



(a) Optimal Policy, SARSA



(b) Optimal Policy, Q-Learning

Figure 3: Optimal Policy, $\rightarrow$ means "Push Right", $\leftarrow$ means "Push left", $\cdot$ means "No Push"

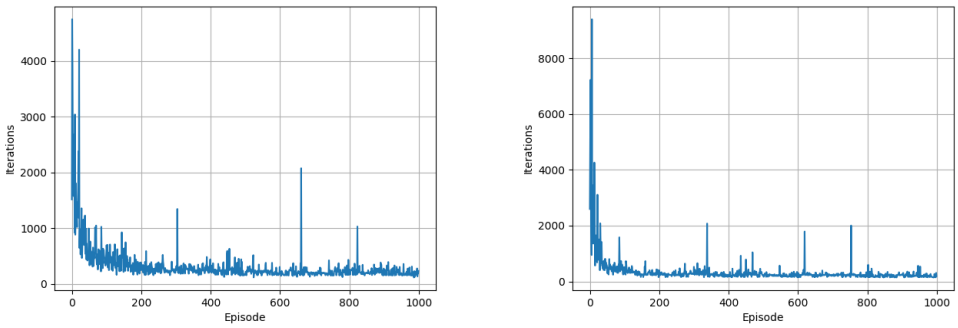
of convergence.

Table 3: Different ϵ For Testing

ϵ	0.1	0.3	0.7	0.9
Ave. Steps (SARSA)	143.80	175.09	165.1	183.53
Ave. Steps (Q-Learning)	177.11	178.15	236.71	300.89

3.4 SARSA and Q-Learning

A difference between two algorithms is that, SARSA explores the control space using ϵ-greedy to update Q while Q-Learning chooses control with ϵ-greedy but greedily updates Q using current best policy. From Fig. 4 we can see, the number of iteration of each episode of Q-Learning is nearly stable after the very beginning, while SARSA still oscillates since it is using ϵ-greedy to choose random policy. However, it is hard to say which one converges faster, since we don't know for a problem, it is better to trust what we have (Q-Learning) or explore more to introduce random action (SARSA). In general, SARSA keeps exploration when it tries to update Q, however, Q-learning has a bias towards the best control for now, which is more greedy.



(a) Training Processing, SARSA

(b) Training Processing, Q-Learning

Figure 4: Training Episodes