OpenAI Gym's Hill Climbing: Project Report

Anmol Popli
A53279430
University of California San Diego
apopli@ucsd.edu

1 Problem Formulation

The objective of this project is to implement Model-free Control in order to solve the Hill Climbing problem. This problem can be summarized as follows: A car is on a track, positioned between two "mountains". The goal is to drive up the mountain on the right; however, the car's engine is not strong enough to scale the mountain in a single pass. Therefore, the only way to succeed is to drive back and forth to build up momentum.

We formulate the problem as a finite-state MDP with unknown motion model and cost function on which I apply on-policy and off-policy TD policy iteration algorithms. As the state space is continuous we first need to apply a disretization so that a finite-state MDP can be formulated.

1.1 Discretization of state space

For this problem, a state x is defined as a tuple (p, v), where p denotes the position of car along the track and v denotes the velocity of the car. From the environment, I found the maximum and minimum values of p and v: $p_{min} = -1.2, p_{max} = 0.6, v_{min} = -0.07, v_{max} = 0.07$. Then I discretized p at precision level 2 and v at precision level 3. So the possible position values will be the set $Pos = \{-1.2, -1.19, -1.18, ..., 0.6\}$ and possible velocity values will be the set $Vel = \{-0.07, -0.069, -0.068, ..., 0.07\}$. And the complete state space will be $X = \{(p, v)|\forall p \in Pos, v \in Vel\}$.

1.2 Control space

There are three controls that can be executed in each state: 0 which indicated *driving backward*, 1 which indicates *not driving*, and 2 which indicates *driving forward*. Thus the control space is $U = \{0, 1, 2\}$.

1.3 Motion model, Stage cost and Terminal cost

The motion model $P_f(x'|x, u)$, stage costs $l(x, u)$ and terminal costs $q(x)$ are unknown for this problem. Hence, we apply Model-free Control algorithms like SARSA and Q-learning.

1.4 Objective function and optimization problem

The objective of this problem is to find a policy π that minimizes the value function $V^\pi(x)$. The optimal value function $V^*(x)$ corresponding to the optimal policy π^* is defined as:

$$V^*(x) = \min_\pi V^\pi(x) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t l(x_t, \pi(x_t))|x_0 = x\right]$$

$$s.t. x_{t+1} \sim p_f(\cdot|x_t, \pi(x_t)),$$
$$x_t \in X, \pi(x_t) \in U$$

Since we do not know the motion model $p_f(x_{t+1}|x_t, u_t)$ and stage costs $l(x_t, u_t)$, we instead optimize the Q-value function $Q^\pi(x, u)$ in Model-free Control algorithms. The Q-value function $Q^\pi(x, u)$ for a policy π and $Q^*(x, u)$ corresponding to the optimal policy π^* satisfy the following equations (which are the basis of TD policy iteration algorithms):

$$Q^\pi(x, u) = l(x, u) + \gamma \mathbb{E}_{x' \sim p_f(\cdot|x,u)}[Q^\pi(x', \pi(x'))]$$

$$Q^*(x, u) = l(x, u) + \gamma \mathbb{E}_{x' \sim p_f(\cdot|x,u)} \left[\min_{u' \in U} Q^*(x', u') \right]$$

2 Technical Approach

2.1 On-policy TD Policy Iteration - SARSA

2.1.1 Model-free Generalized Policy Iteration

The Model-free Generalized Policy Iteration algorithm comprises the following two components:

1. **Policy Evaluation** Given π, Q^π can be estimated by multiple iterations of the TD operator:

$$\mathcal{T}_\pi[Q](x_t, u_t) \approx Q(x_t, u_t) + \alpha[l(x_t, u_t) + \gamma Q(x_{t+!}, u_{t+1}) - Q(x_t, u_t)]$$

2. **Policy Improvement** In the policy evaluation step, we estimate Q^π instead of V^π so that the policy improvement step can be implemented model-free as follows:

$$\pi'(x) \leftarrow \arg\min_u Q^\pi(x, u)$$

By *generalized*, we mean that there can be any arbitrary number of iterations of the TD operator in the policy evaluation step, which in turn implies that for any x, u, $Q(x, u)$ can be updated any arbitrary number of times in one policy evaluation step. Thus we can use either one sampled episode or n sampled episodes to update $Q(x, u)$ in one iteration of model-free policy iteration.

2.1.2 TD Policy Iteration with ϵ-Greedy Improvement (SARSA)

- **SARSA**: We estimate the action-value function Q^π using TD updates after every $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$ transition. Here S indicates state x, A indicates control u and R indicates reward which can be interpreted as negative of stage cost, i.e. -l. Thus the TD update would be as follows:

$$Q(x_t, u_t) \leftarrow Q(x_t, u_t) + \alpha[l(x_t, u_t) + \gamma Q(x_{t+!}, u_{t+1}) - Q(x_t, u_t)]$$

  where $(x_t, u_t, x_{t+1}, u_{t+1})$ are consecutive state-action pairs from a sampled episode.

- To ensure exploration, we use an ϵ-soft policy (stochastic policy) so that every reachable state hasnon-zero probability of being encountered. Accordingly, we select the ϵ-greedy policy in the policy improvement step.

- **ϵ-Greedy Policy**: a stochastic policy that picks the best control according to $Q(x, u)$ in the policy improvement step but ensures that all other controls are selected with a small (non-zero) probability:

$$\pi(u|x) = P(u_t = u|x_t = x) := \begin{cases} 1 - \epsilon + \frac{\epsilon}{|U|} & u = \arg\min_{u' \in U} Q(x, u') \\ \frac{\epsilon}{|U|} & otherwise \end{cases}$$

- **Episode length**: In order for the system to learn to reach the goal, the goal needs to be reached in a sampled episode. Only then, Q updates will be made based on the Q value of goal, which has been initialized as 0. Therefore, I set the maximum length of an episode to 20,000. If the goal is reached the episode ends there, otherwise it continues for 20,000 steps. This results in better exploration earlier on and hence faster convergence.

SARSA

Init: $Q(x, u)$ for all $x \in X$ and all $u \in U$
while *Not converged* **do**
  $\pi \leftarrow \epsilon$-greedy policy derived from Q
  Generate episode $\rho := (\mathrm{x}_{0:T}, u_{0:T-1})$ from π
  for $(x, u, x', u') \in \rho$ **do**
    $Q(x, u) \leftarrow Q(x, u) + \alpha[l(x, u) + \gamma Q(x', u') - Q(x, u)]$
  end
end

2.1.3 Convergence of SARSA

SARSA converges to the optimal action-value function $Q^*(x, u)$ as the number of episodes $k \to \infty$ as long as:

- the sequence of ϵ-greedy policies $\pi_k(u|x)$ is GLIE i.e.,

$$\lim_{k \to \infty} \pi_k(u|x) = \mathbb{1}\{u = \arg\min_{u' \in U} Q(x, u')\}$$

- the sequence of step sizes α_k is Robbins-Monro.

In my implementation, I have tried two different variants of ϵ_k: One where $\epsilon_k = 1/k$ and I increment k after every iteration of the algorithm, the other where $\epsilon_k = 1/k$ but k is incremented after every 5 iterations of the algorithm. Though both these variants satisfy the GLIE condition, the latter performs better. I implemented the latter variant of $_k$ so that the -greedy policy remains exploratory for a larger number of iterations, thus resulting in better initial exploration of the state space.

I implement the step size α as a function of state x and control u:

$$\alpha(x, u) = \frac{1}{N(x, u) + 1}$$

where $N(x, u)$ is the number of updates made to $Q(x, u)$. This satisfies the Robbins-Monro condition for every (x, u) pair.

2.2 Off-Policy TD Policy Iteration - Q-learning

2.2.1 Off-Policy Learning

In off-policy prediction, value function V^π or Q-value function Q^π is estimated usinf experience from a different policy μ. Thus, we evaluate and improve a policy π that is different from the policy μ used to generate the data. This enables us to use an effective exploratory policy μ to generate data while learning about an optimal policy.

2.2.2 Q-Learning

Q-Learning approximates $\mathcal{T}_*[Q](x, u)$ directly using samples instead of approximating $\mathcal{T}_\pi[Q](x, u)$.

$$Q(x_t, u_t) \leftarrow Q(x_t, u_t) + \alpha \left[l(x_t, u_t) + \gamma \min_{u \in U} Q(x_{t+1}, u) - Q(x_t, u_t) \right]$$

This is in a way analogous to Value Iteration where we directly apply the $\mathcal{T}_*[V](x)$ operator. Thus the learned Q function eventually approximates Q^* regardless of the policy being followed. The value function V^* cannot be approximated in a model-free problem because in $\mathcal{T}_*[V](x)$, the *min* operator is applied outside the expectation and thus the motion model is required for computing it.

Even though we could use any exploratory policy to generate episodes in Q-learning, I still use an ϵ-greedy policy derived from Q so that more samples are generated from areas that are promising, and faster convergence of Q function around promising areas is desirable.

Q-Learning

Init: $Q(x, u)$ for all $x \in X$ and all $u \in U$
while *Not converged* **do**
  $\pi \leftarrow \epsilon$-greedy policy derived from Q
  Generate episode $\rho := (\mathrm{x}_{0:T}, u_{0:T-1})$ from π
  for $(x, u, x') \in \rho$ **do**
    $Q(x, u) \leftarrow Q(x, u) + \alpha[l(x, u) + \gamma \min_{u'} Q(x', u') - Q(x, u)]$
  end
end

2.2.3 Convergence of Q-Learning

Q-Learing converges almost surely to Q^* assuming all state-control pairs contnue to be updated and the sequence of step sizes α_k is Robbins-Monro.

here also, I implement the step size α as a function of state x and control u:

$$\alpha(x, u) = \frac{1}{N(x, u) + 1}$$

where $N(x, u)$ is the number of updates made to $Q(x, u)$. This satisfies the Robbins-Monro condition for every (x, u) pair.

3 Results

3.1 On-Policy TD Policy Iteration - SARSA

3.1.1 Hyperparameters

Following are the hyperparameters that perform well: $\gamma = 0.9$, $n_{iters} = 400$ (number of iterations of the SARSA algorithm), $\alpha(x, u) = 1/(N(x, u) + 1)$, $T = 20,000$ (maximum episode length). Here, I present comparison between two variants of ϵ_k:

- $\epsilon_k = 1/k$, where k is incremented after every iteration of the algorithm. Denote by $\epsilon_k(1)$.
- $\epsilon_k = 1/k$, where k is incremented after every 5 iterations of the algorithm. Denote by $\epsilon_k(2)$.

I evaluate the comparison between the above hyperparameters based on the average number of steps required to reach the goal using the optimal policy. So once that optimal policy has been computed, I sample 100 episodes (all of which reach goal) and average the number of steps taken by them. Let's denote this by $STEPS_{avg}$.

3.1.2 Comparison for ϵ_k

The comparison between results of variants of ϵ_k are presented in Table 1.

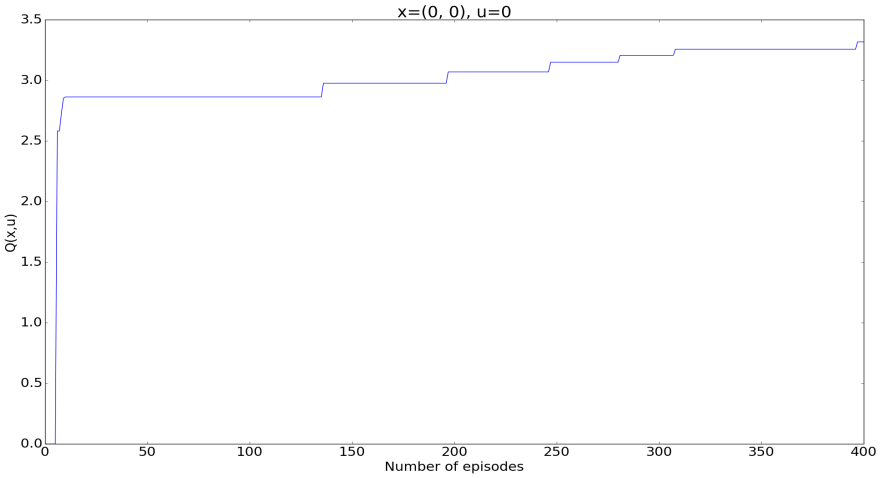
Hyperparameter ϵ_k	$STEPS_{avg}$
$\epsilon_k(1)$	2298
$\epsilon_k(2)$	1440

Table 1: Average steps to goal $STEPS_{avg}$ for different ϵ_k

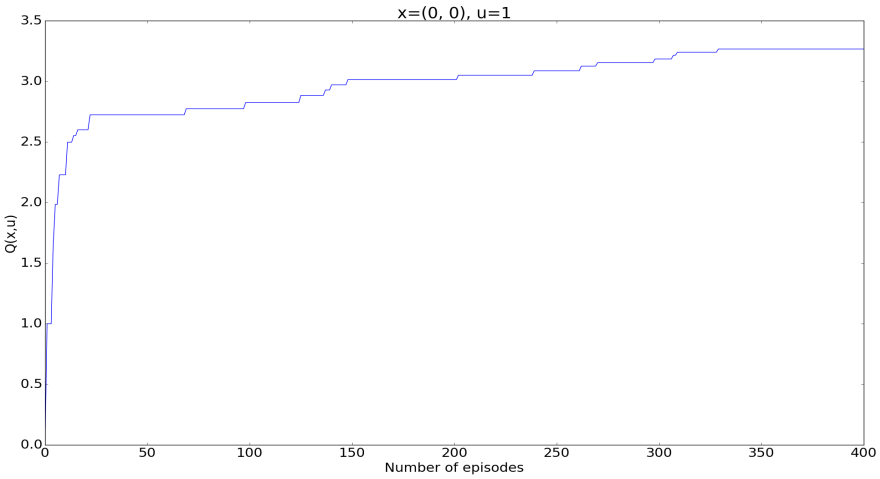
$\epsilon_k(2)$ works better and henceforth, I use it along with the hyperparameters listed in section 3.1.1 for further analysis. I am also attaching a video of the execution in the submission.

3.1.3 Performance Statistics

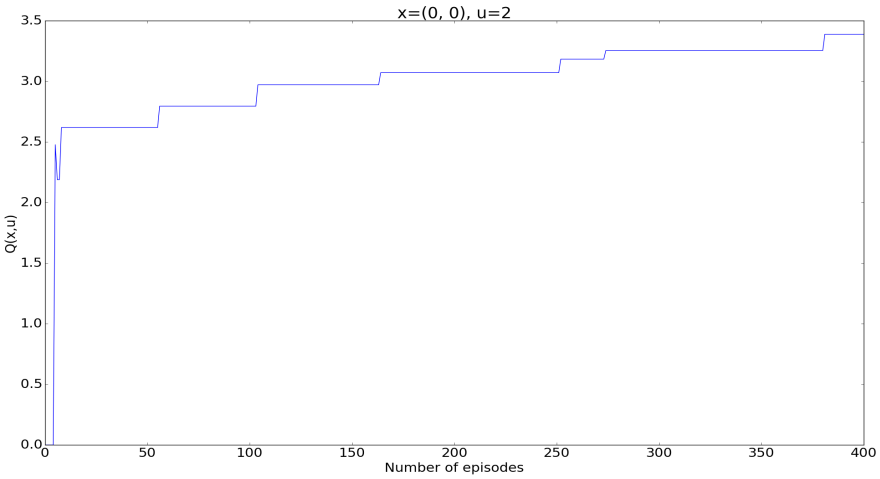
First I plot $Q(x, u)$ for the state-action pairs: $\{((0, 0), u), ((1.0, -0.02), u), ((-0.5, 0.01), u)\}$ for all $u \in U$ in Figure 1.



(a)

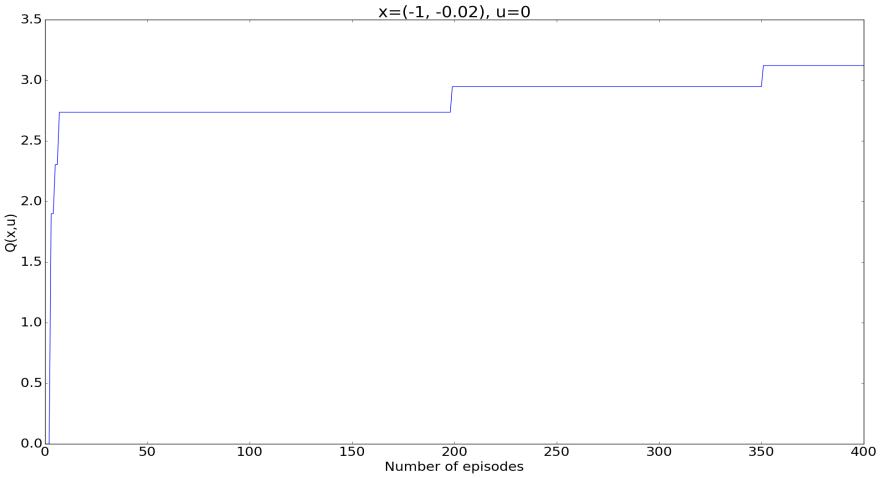


(b)

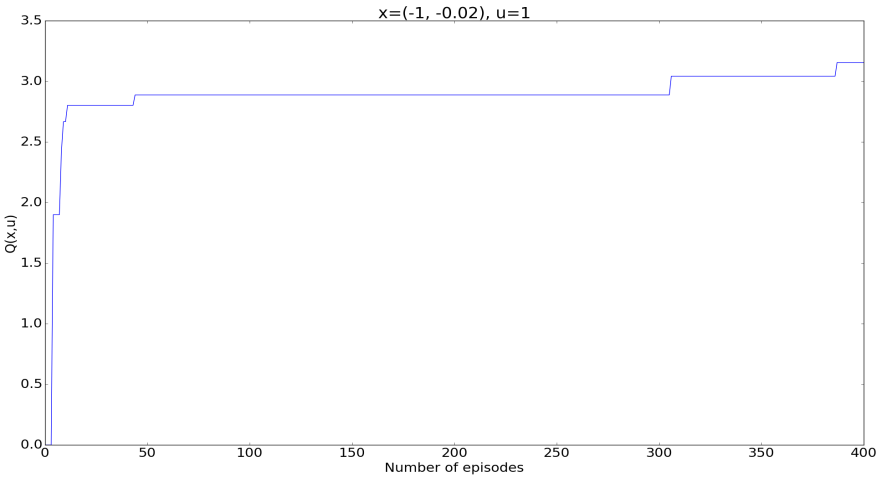


(c)

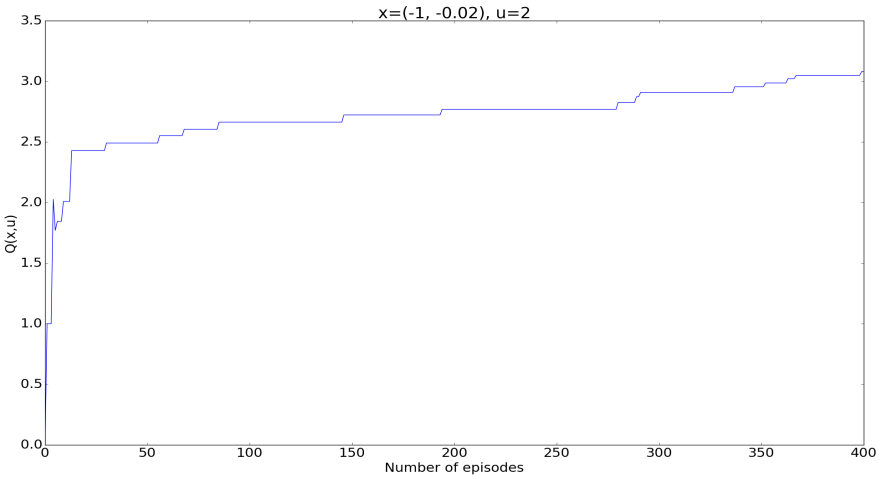
Figure 1: Q(x,u) as a function of number of episodes for a set of states



(d)

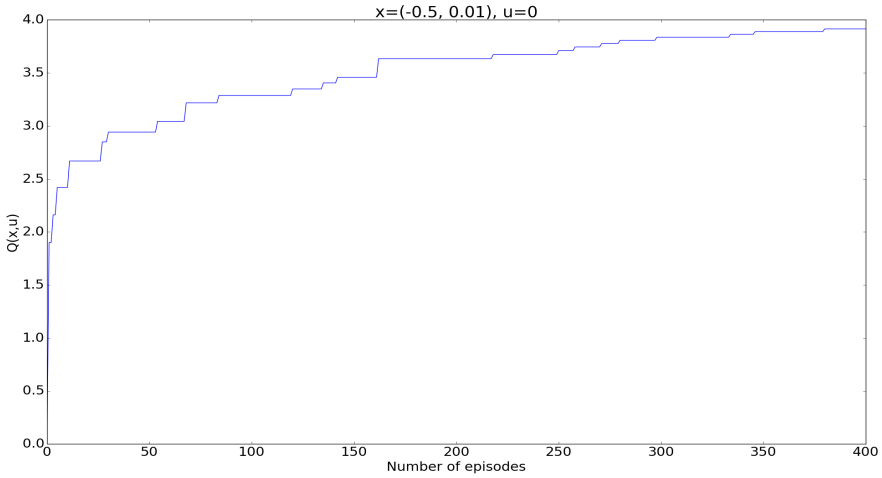


(e)

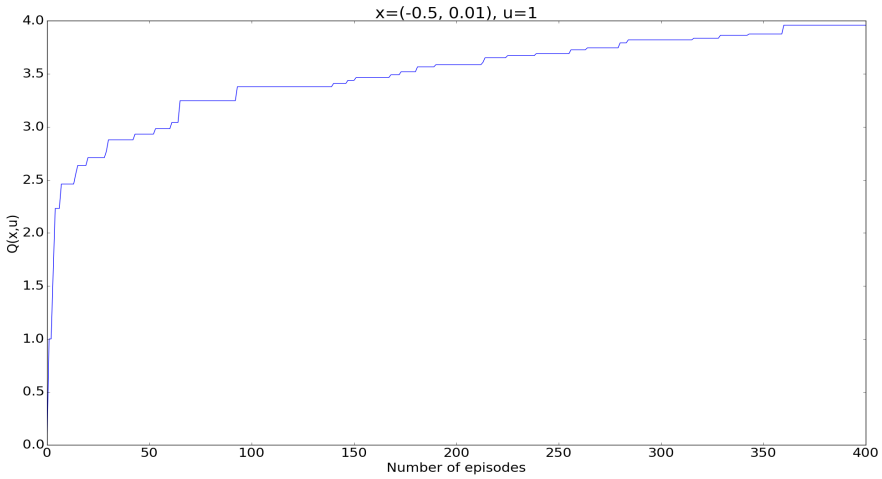


(f)

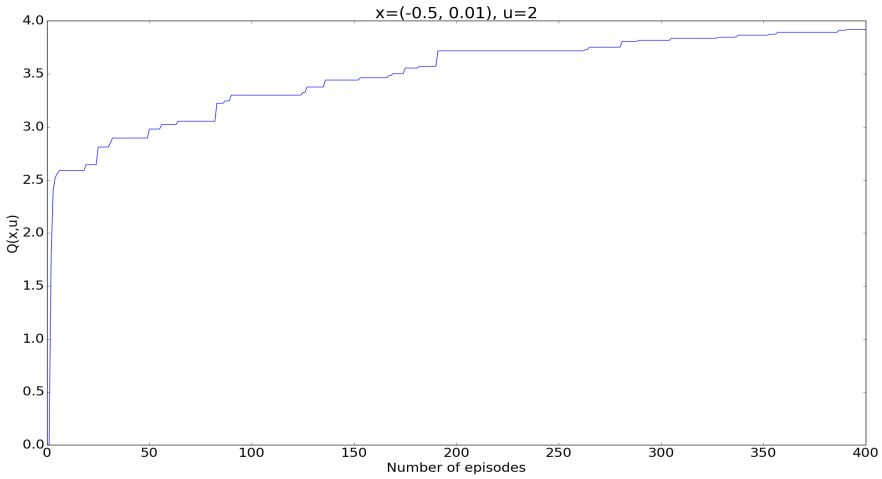
Figure 1: Q(x,u) as a function of number of episodes for a set of states (cont.)



(g)



(h)



(i)

Figure 1: Q(x,u) as a function of number of episodes for a set of states (cont.)

Now I plot the optimized policy $\pi(x)$ over the entire state space X in Figure 2.

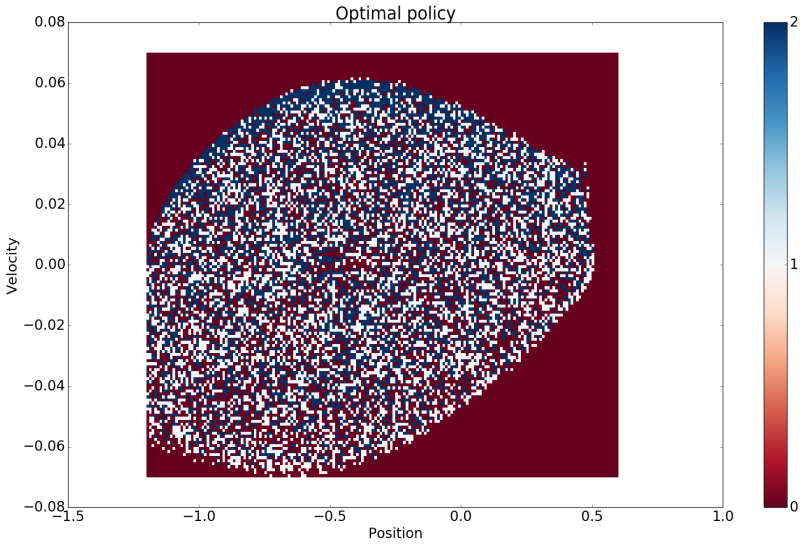


Figure 2: Optimal policy $\pi^*(x)$ over the state space

3.2 Off-Policy TD Policy Iteration - Q-Learning

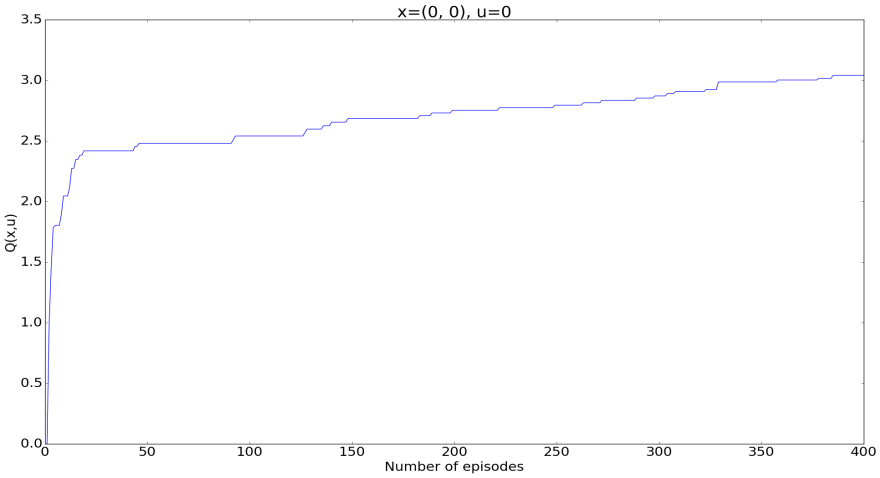
3.2.1 Hyperparameters

Following are the hyperparameters that perform well: $\gamma = 0.9$, $n_{iters} = 400$ (number of iterations of the Q-Learning algorithm), $\alpha(x, u) = 1/(N(x, u) + 1)$, $T = 20,000$ (maximum episode length), $\epsilon_k(2)$ (refer Section 3.1.1).

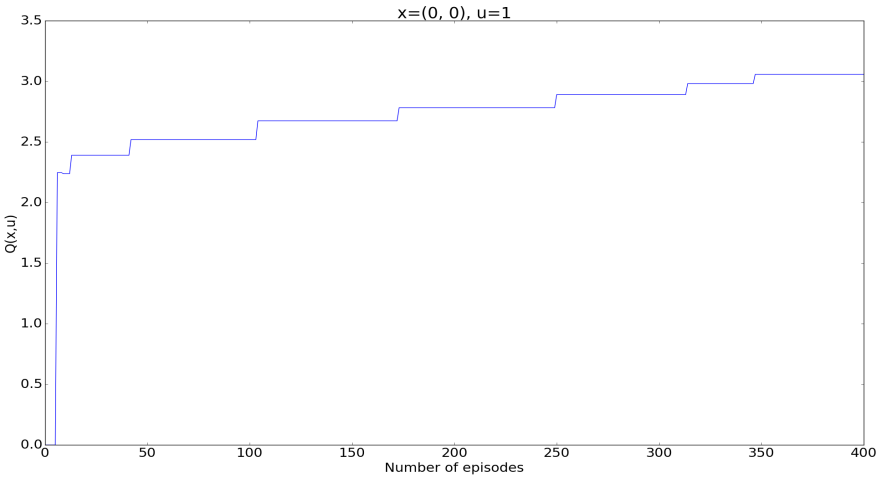
Using these hyperparameters, the average number of steps taken to reach the goal following the optimal policy is: $STEPS_{avg}$=1382.

3.2.2 Performance Statistics

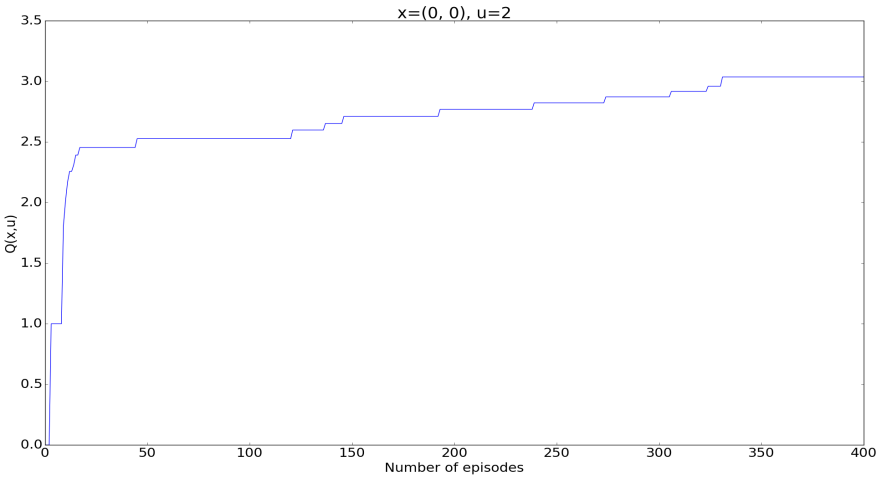
First I plot $Q(x, u)$ for the state-action pairs: $\{((0, 0), u), ((1.0, -0.02), u), ((-0.5, 0.01), u)\}$ for all $u \in U$ in Figure 3.



(a)

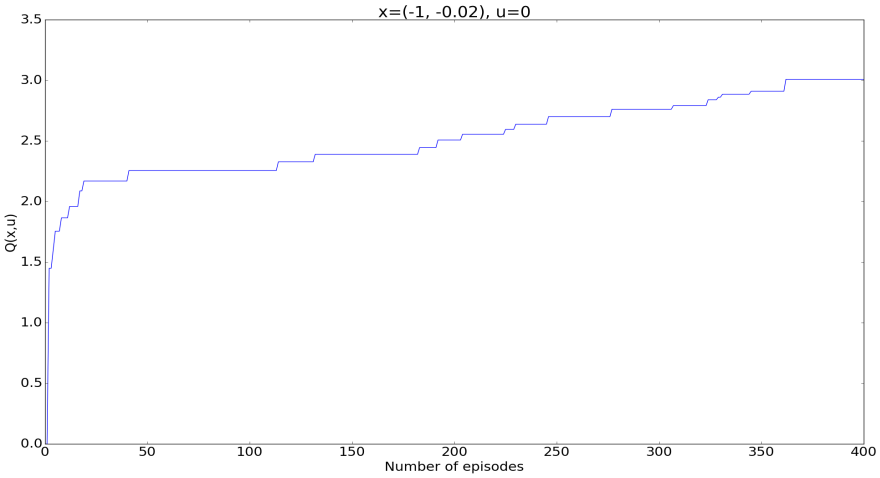


(b)

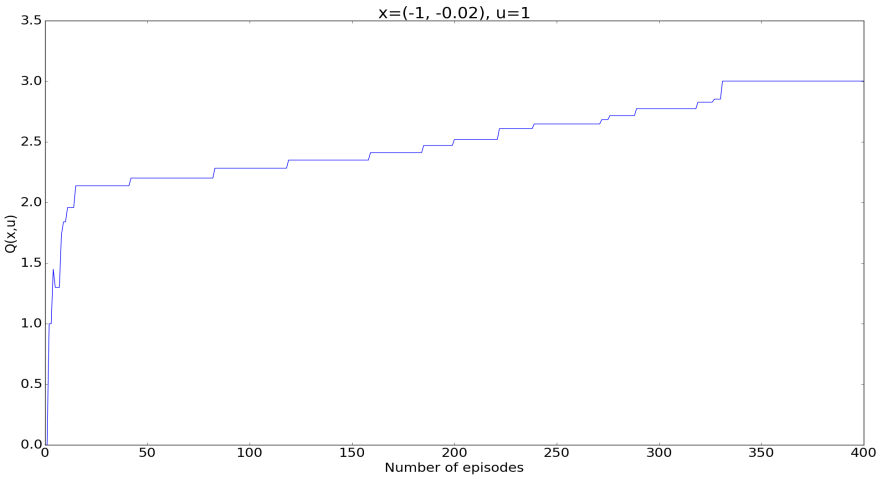


(c)

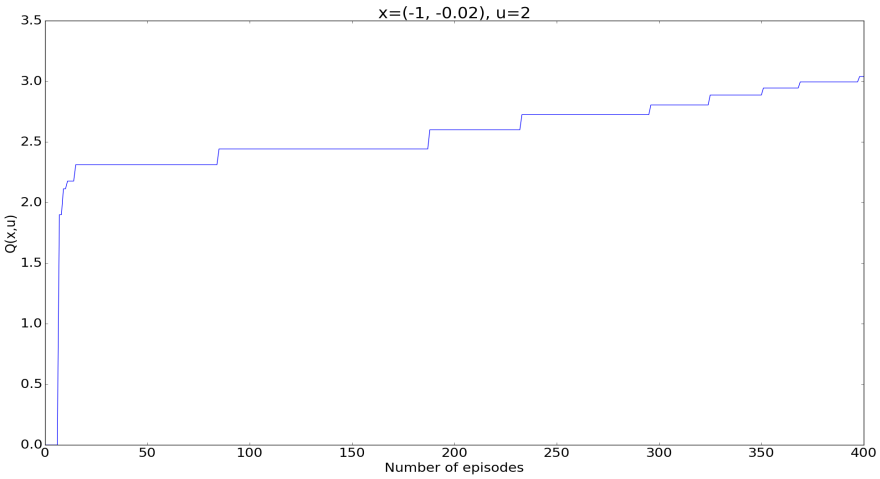
Figure 3: Q(x,u) as a function of number of episodes for a set of states



(d)

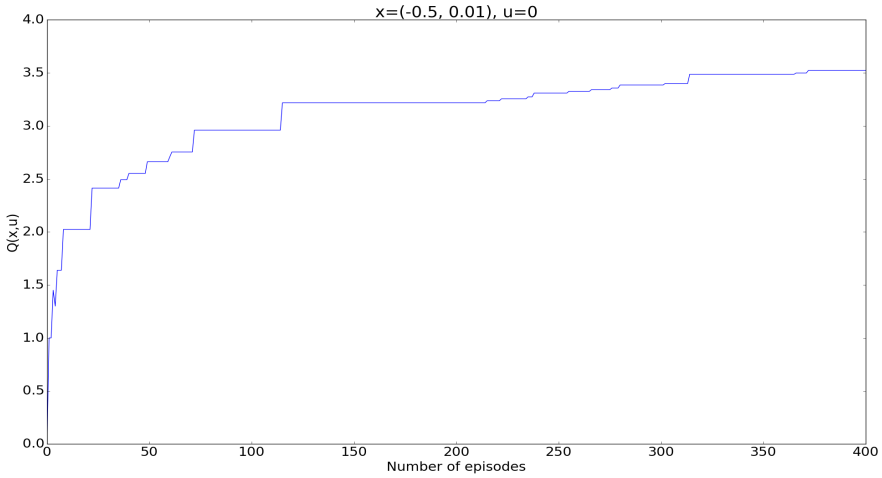


(e)

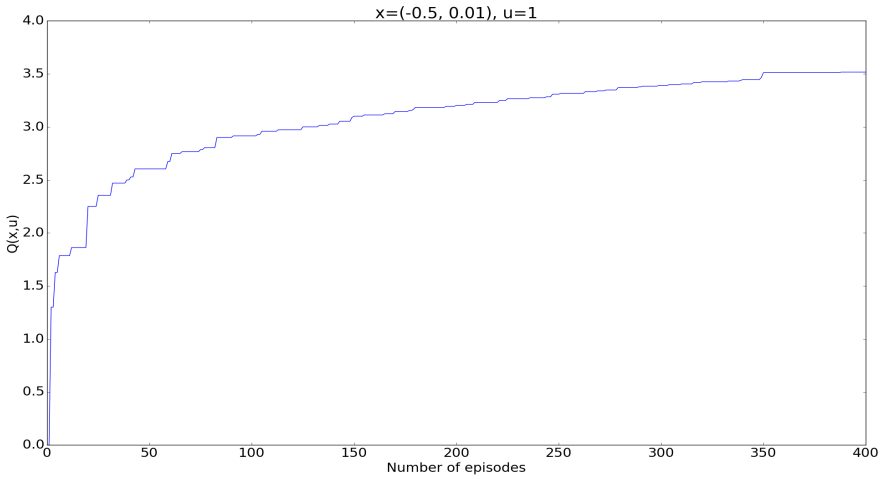


(f)

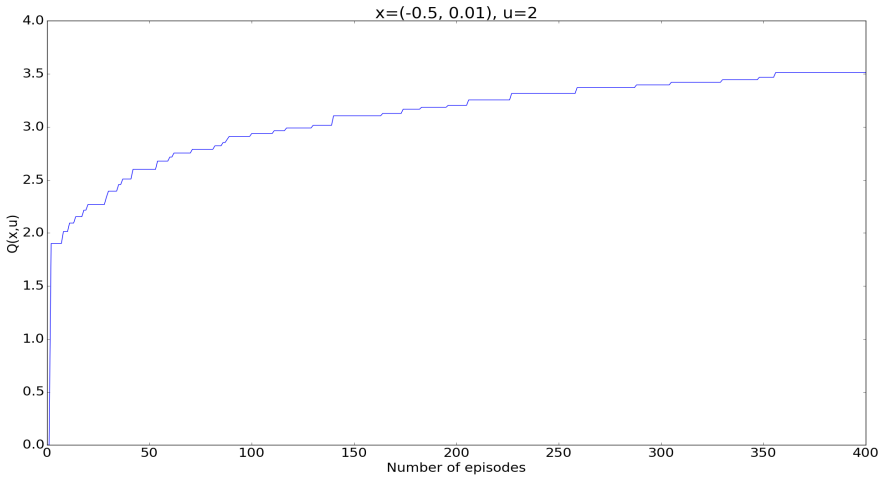
Figure 3: Q(x,u) as a function of number of episodes for a set of states (cont.)



(g)



(h)



(i)

Figure 3: Q(x,u) as a function of number of episodes for a set of states (cont.)

Now I plot the optimized policy $\pi(x)$ over the entire state space X in Figure 4.

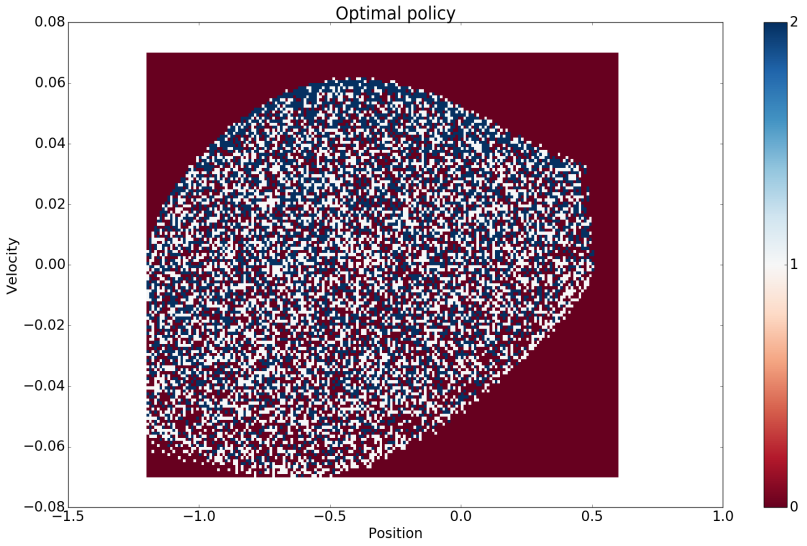


Figure 4: Optimal policy $\pi^*(x)$ over the state space

3.3 Comparison between SARSA and Q-Learning

As can be observed from plots in Figures 1 and 3, Q-Learning converges to smaller values of Q compared to SARSA. The final values of Q in these plots are as follows:

Q(x,u)	SARSA	Q-Learning
Q((0,0),0)	3.31769	3.04142
Q((0,0),1)	3.26742	3.05789
Q((0,0),2)	3.38934	3.03673
Q((-1,-0.02),0)	3.12262	3.00901
Q((-1,-0.02),1)	3.15576	3.00278
Q((-1,-0.02),2)	3.08149	3.04034
Q((-0.5,0.01),0)	3.91433	3.52253
Q((-0.5,0.01),1)	3.95905	3.51647
Q((-0.5,0.01),2)	3.91948	3.51191

Table 2: Final values of $Q(x, u)$ for SARSA and Q-Learning

Consequently, the $STEPS_{avg}$ for Q-Learning is also smaller than that for SARSA. Thus, the Q-value function in Q-Learning has converged faster (to smaller values) in the same number of episodes compared to SARSA.

This might be attributed to the fact that in Q-Learning, we directly approximate $\mathcal{T}_*[Q](x, u)$ using samples from any policy. On the other hand, in SARSA, we estimate Q^π for a policy π using $\mathcal{T}_\pi[Q](x, u)$. But the fact that this just gives an approximation to the true Q^π might cause problems as picking the "best" control π' according to the current estimate Q^π might not be the actual best control. And this could be the reason for relatively slower convergence of SARSA.