**Assignment: Use the Q-learning example given as a basis for your homework and develop a SARSA learning algorithm instead of Q-learning.**

**SARSA Learning Algorithm:**

#!/usr/bin/env/ python

"""

sarsa\_learner.py

An easy-to-follow script to train, test and evaluate a SARSA-learning agent on the Mountain Car

"""

import gym

import numpy as np

MAX\_NUM\_EPISODES = 50000

STEPS\_PER\_EPISODE = 200 # This is specific to MountainCar. May change with env

EPSILON\_MIN = 0.005

max\_num\_steps = MAX\_NUM\_EPISODES \* STEPS\_PER\_EPISODE

EPSILON\_DECAY = 500 \* EPSILON\_MIN / max\_num\_steps

ALPHA = 0.05 # Learning rate

GAMMA = 0.98 # Discount factor

NUM\_DISCRETE\_BINS = 30 # Number of bins to Discretize each observation dim

class SARSA\_Learner(object):

def \_\_init\_\_(self, env):

self.obs\_shape = env.observation\_space.shape

self.obs\_high = env.observation\_space.high

self.obs\_low = env.observation\_space.low

self.obs\_bins = NUM\_DISCRETE\_BINS # Number of bins to Discretize each observation dim

self.bin\_width = (self.obs\_high - self.obs\_low) / self.obs\_bins

self.action\_shape = env.action\_space.n

# Create a multi-dimensional array (aka. Table) to represent the

# Q-values

self.Q = np.zeros((self.obs\_bins + 1, self.obs\_bins + 1,

self.action\_shape)) # (51 x 51 x 3)

self.alpha = ALPHA # Learning rate

self.gamma = GAMMA # Discount factor

self.epsilon = 1.0

def discretize(self, obs):

return tuple(((obs - self.obs\_low) / self.bin\_width).astype(int))

def get\_action(self, obs):

discretized\_obs = self.discretize(obs)

# Epsilon-Greedy action selection

if self.epsilon > EPSILON\_MIN:

self.epsilon -= EPSILON\_DECAY

if np.random.random() > self.epsilon:

return np.argmax(self.Q[discretized\_obs])

else: # Choose a random action

return np.random.choice([a for a in range(self.action\_shape)])

def learn(self, obs, action, reward, next\_obs):

discretized\_obs = self.discretize(obs)

discretized\_next\_obs = self.discretize(next\_obs)

td\_target = reward + self.gamma \* (self.Q[discretized\_next\_obs][action])

td\_error = td\_target - self.Q[discretized\_obs][action]

self.Q[discretized\_obs][action] += self.alpha \* td\_error

def train(agent, env):

best\_reward = -float('inf')

for episode in range(MAX\_NUM\_EPISODES):

done = False

obs = env.reset()

total\_reward = 0.0

while not done:

action = agent.get\_action(obs)

next\_obs, reward, done, info = env.step(action)

agent.learn(obs, action, reward, next\_obs)

obs = next\_obs

total\_reward += reward

if total\_reward > best\_reward:

best\_reward = total\_reward

print("Episode#:{} reward:{} best\_reward:{} eps:{}".format(episode,

total\_reward, best\_reward, agent.epsilon))

# Return the trained policy

return np.argmax(agent.Q, axis=2)

def test(agent, env, policy):

done = False

obs = env.reset()

total\_reward = 0.0

while not done:

action = policy[agent.discretize(obs)]

next\_obs, reward, done, info = env.step(action)

obs = next\_obs

total\_reward += reward

return total\_reward

if \_\_name\_\_ == "\_\_main\_\_":

env = gym.make('MountainCar-v0')

agent = SARSA\_Learner(env)

learned\_policy = train(agent, env)

# Use the Gym Monitor wrapper to evalaute the agent and record video

gym\_monitor\_path = "./gym\_monitor\_output"

env = gym.wrappers.Monitor(env, gym\_monitor\_path, force=True)

for \_ in range(1000):

test(agent, env, learned\_policy)

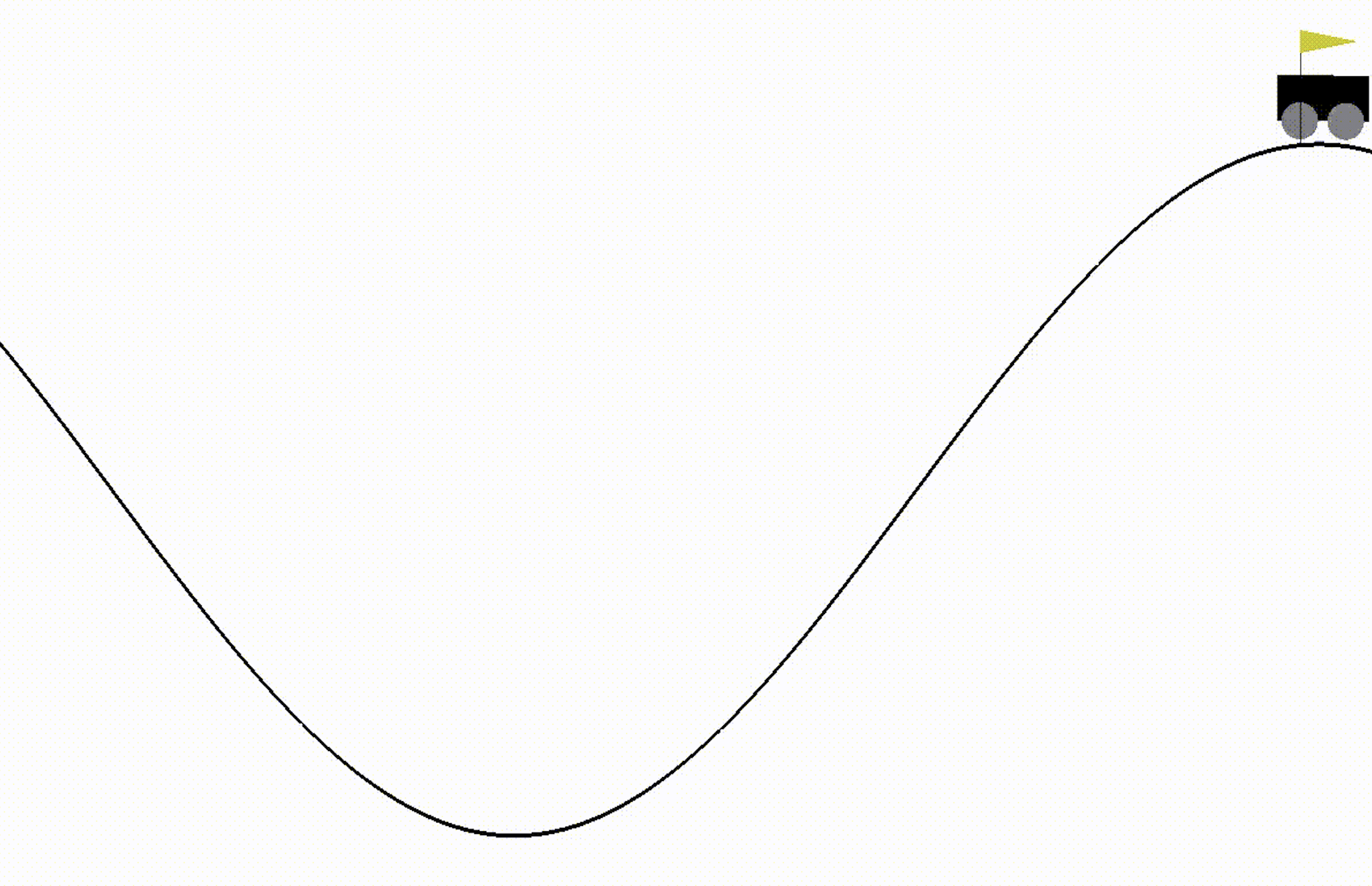
env.close()

SARSA Learning Results:

Total number of Episodes: 50000

Best Reward: -84





**Q-Learner Algorithm**

**#!/usr/bin/env/ python**

**"""**

**q\_learner.py**

**An easy-to-follow script to train, test and evaluate a Q-learning agent on the Mountain Car**

**problem using the OpenAI Gym. |Praveen Palanisamy**

**# Chapter 5, Hands-on Intelligent Agents with OpenAI Gym, 2018**

**"""**

**import gym**

**import numpy as np**

**#MAX\_NUM\_EPISODES = 500**

**MAX\_NUM\_EPISODES = 50000**

**STEPS\_PER\_EPISODE = 200 # This is specific to MountainCar. May change with env**

**EPSILON\_MIN = 0.005**

**max\_num\_steps = MAX\_NUM\_EPISODES \* STEPS\_PER\_EPISODE**

**EPSILON\_DECAY = 500 \* EPSILON\_MIN / max\_num\_steps**

**ALPHA = 0.05 # Learning rate**

**GAMMA = 0.98 # Discount factor**

**NUM\_DISCRETE\_BINS = 30 # Number of bins to Discretize each observation dim**

**class Q\_Learner(object):**

**def \_\_init\_\_(self, env):**

**self.obs\_shape = env.observation\_space.shape**

**self.obs\_high = env.observation\_space.high**

**self.obs\_low = env.observation\_space.low**

**self.obs\_bins = NUM\_DISCRETE\_BINS # Number of bins to Discretize each observation dim**

**self.bin\_width = (self.obs\_high - self.obs\_low) / self.obs\_bins**

**self.action\_shape = env.action\_space.n**

**# Create a multi-dimensional array (aka. Table) to represent the**

**# Q-values**

**self.Q = np.zeros((self.obs\_bins + 1, self.obs\_bins + 1,**

**self.action\_shape)) # (51 x 51 x 3)**

**self.alpha = ALPHA # Learning rate**

**self.gamma = GAMMA # Discount factor**

**self.epsilon = 1.0**

**def discretize(self, obs):**

**return tuple(((obs - self.obs\_low) / self.bin\_width).astype(int))**

**def get\_action(self, obs):**

**discretized\_obs = self.discretize(obs)**

**# Epsilon-Greedy action selection**

**if self.epsilon > EPSILON\_MIN:**

**self.epsilon -= EPSILON\_DECAY**

**if np.random.random() > self.epsilon:**

**return np.argmax(self.Q[discretized\_obs])**

**else: # Choose a random action**

**return np.random.choice([a for a in range(self.action\_shape)])**

**def learn(self, obs, action, reward, next\_obs):**

**discretized\_obs = self.discretize(obs)**

**discretized\_next\_obs = self.discretize(next\_obs)**

**td\_target = reward + self.gamma \* np.max(self.Q[discretized\_next\_obs])**

**td\_error = td\_target - self.Q[discretized\_obs][action]**

**self.Q[discretized\_obs][action] += self.alpha \* td\_error**

**def train(agent, env):**

**best\_reward = -float('inf')**

**for episode in range(MAX\_NUM\_EPISODES):**

**done = False**

**obs = env.reset()**

**total\_reward = 0.0**

**while not done:**

**action = agent.get\_action(obs)**

**next\_obs, reward, done, info = env.step(action)**

**agent.learn(obs, action, reward, next\_obs)**

**obs = next\_obs**

**total\_reward += reward**

**if total\_reward > best\_reward:**

**best\_reward = total\_reward**

**print("Episode#:{} reward:{} best\_reward:{} eps:{}".format(episode,**

**total\_reward, best\_reward, agent.epsilon))**

**# Return the trained policy**

**return np.argmax(agent.Q, axis=2)**

**def test(agent, env, policy):**

**done = False**

**obs = env.reset()**

**total\_reward = 0.0**

**while not done:**

**action = policy[agent.discretize(obs)]**

**next\_obs, reward, done, info = env.step(action)**

**obs = next\_obs**

**total\_reward += reward**

**return total\_reward**

**if \_\_name\_\_ == "\_\_main\_\_":**

**env = gym.make('MountainCar-v0')**

**agent = Q\_Learner(env)**

**learned\_policy = train(agent, env)**

**# Use the Gym Monitor wrapper to evalaute the agent and record video**

**gym\_monitor\_path = "./gym\_monitor\_output"**

**env = gym.wrappers.Monitor(env, gym\_monitor\_path, force=True)**

**for \_ in range(1000):**

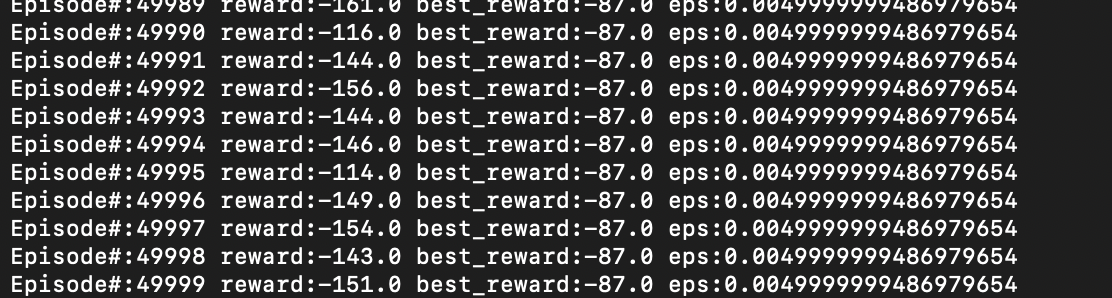
**test(agent, env, learned\_policy)**

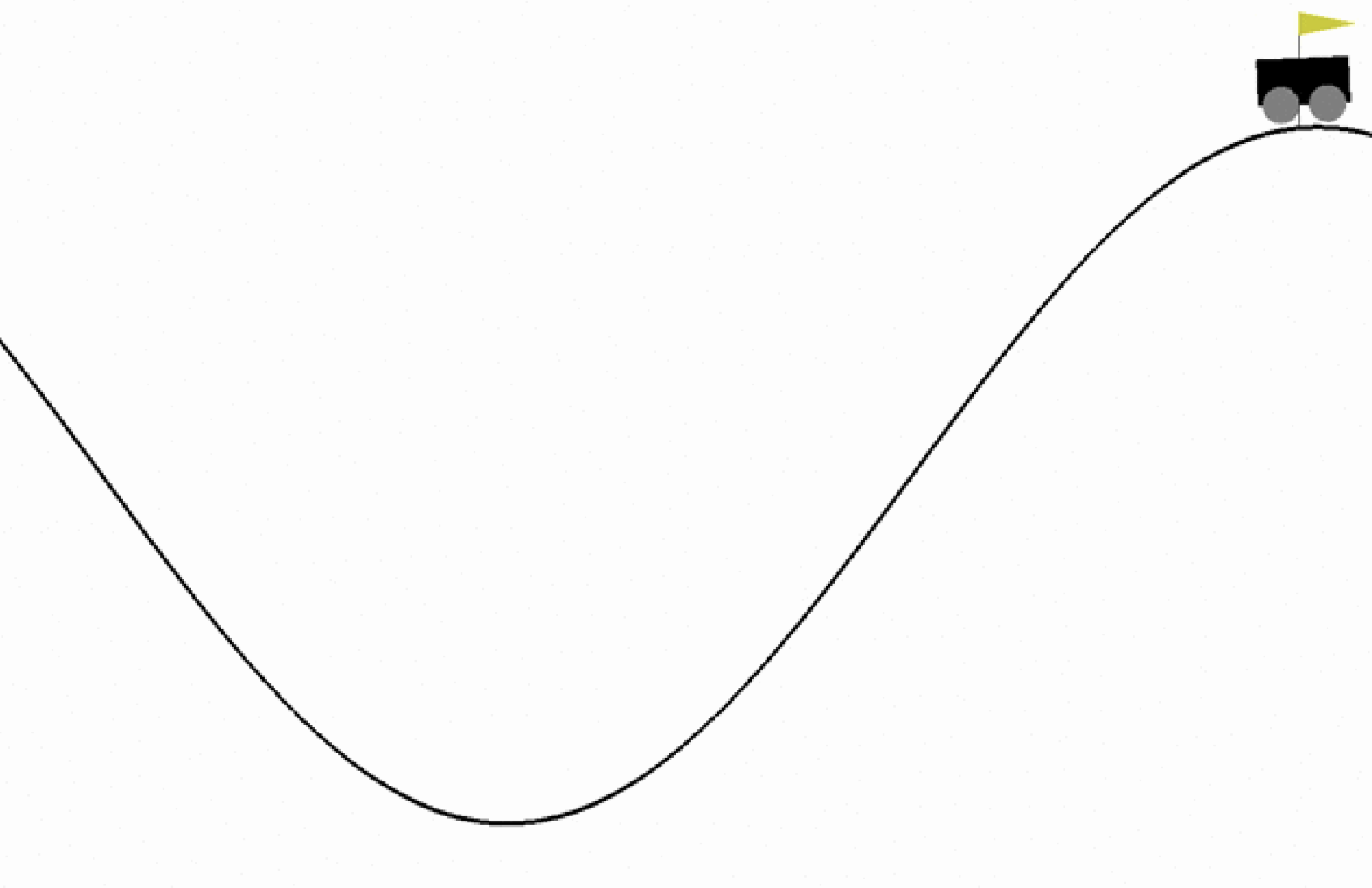
**env.close()**

Q Learning Results:

Total number of Episodes: 50000

Best Reward: -87





**Inference**

As a result of both the algorithms it's clear that the best reward converges to higher value in case of Q-learning (-87) as compared to SARSA(-84) as the episodes increase. This is due to the selection of action in case of Q-learning where we select the maximum Q-value. This results in getting the largest reward in long run.