# EECS 498: Reinforcement Learning
# Homework 1 Responses

Tejas Jha
tjha

October 12, 2018

This document includes my responses to Homework 2 questions. Responses that involved the use of coding will provide references to specific lines of code to provide a better overview of how the problem was approached. The code can either be referenced in the Appendix or in the accompanied python script submitted with this assignment.

## Question 1

(a)

(b)

## Question 2

## Question 3

On the following page is the plot corresponding to the implementation that can be found in the Appendix as well as submitted with this document.

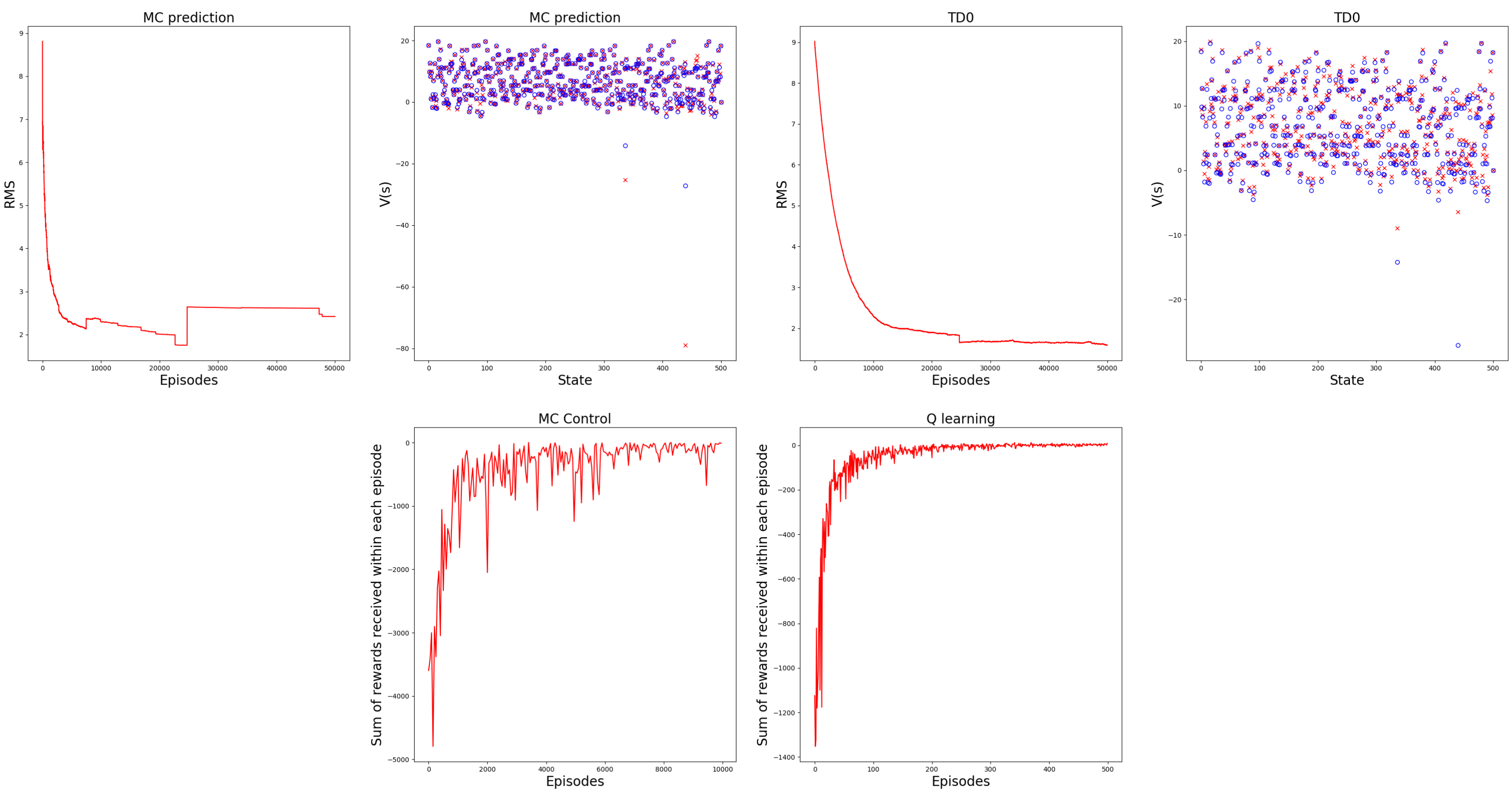


Figure 1: Plot for exploring taxi-v3 By Tejas Jha

## Appendix: Relevant Code - tjha.py

```python
# Tejas Jha
# EECS 498: Reinforcement Learning - Homework 2

import numpy as np
import gym
import copy
import mytaxi
import math
import mdp.mdp as mdp
import matplotlib.pyplot as plt
import randomwalk

##################  Setup in Part (a) ###################
# Part (a): Policy Evaluation
def evaluate_policy(trans_mat, V_init, policy, theta, gamma=1, inplace=True):
    return mdp.policy_eval(trans_mat, V_init, policy, theta, gamma, inplace)

# Global Variables for default actions taken
given_policy = np.load('policy.npy')
# Gather environment details and stored policy
ENV = gym.make('Taxi-v3').unwrapped
TRANS_MAT = ENV.P
V_INIT = np.zeros(len(TRANS_MAT))
ACTIONS = [0,1,2,3,4,5]
#ACTIONS = [0,1]
# Part (a): Evaluate value function of given policy in policy.npy
true_value_fn = evaluate_policy(TRANS_MAT, V_INIT, given_policy, theta=0.01, gamma=1)
##########################################################

# Helper function - Generate the steps in an episode for an environment and given policy
# Returns: (T, 3) numpy array for length T with elements corresponding to each time step for
# states, actions, and Rewards
def generate_episode(env, policy, limit=100000):
    states_visited = list()
    actions_taken = list()
    rewards_received = list()

    # edge case implementation error possible (currently do not check if initial state is final state)
    states_visited.append(env.reset())
```

```
    actions_taken.append(np.random.choice(ACTIONS,p=policy[states_visited[-1]]))
    next_state, reward, done, info = env.step(actions_taken[-1])

    step = 0

    while not done and step < limit:
        rewards_received.append(reward)
        states_visited.append(next_state)
        actions_taken.append(np.random.choice(ACTIONS,p=policy[states_visited[-1]]))
        next_state, reward, done, info = env.step(actions_taken[-1])
        step += 1

    rewards_received.append(reward)

    return states_visited, actions_taken, rewards_received

# Helper function to see if pair of state-action was seen earlier
def pair_appears(states_visited, actions_taken, step):
    state = states_visited[step]
    action = actions_taken[step]
    for idx in range(step):
        if states_visited[idx] == state and actions_taken[idx] == action:
            return True
    return False

# Part (b): Implementation for first-visit Monte Carlo Prediction for estimating state-value
#            functions
#            Returns: rms w.r.t baseline at end of each episode (rms), final value function (V)
def mc_prediction(env, policy=given_policy, baseline=true_value_fn, gamma=1, episodes=50000):
    np.random.seed(3)
    env.seed(5)
    rms = np.zeros(episodes)
    #V = np.random.rand(env.nS)
    V = np.zeros(env.nS)
    #V = np.full(env.nS, -1)
    returns = [[] for _ in range(env.nS)]

    # Loop over each episode run
    for i_episode in range(episodes):
        # Generate an episode following policy
        states_visited, actions_taken, rewards_received = generate_episode(env, policy)
```

```
        G = 0
        # Loop over each step of the episode
        for step in range(len(states_visited)-1, -1, -1):
            G = gamma*G + rewards_received[step]
            if states_visited.index(states_visited[step]) == step:
                returns[states_visited[step]].append(G)
                V[states_visited[step]] = sum(returns[states_visited[step]]) / float(len(returns[states_visited[step]]))
        rms_value = math.sqrt(sum((V - baseline)**2)/float(len(V)))
        rms[i_episode] = rms_value

        # To keep track of progress in loop
        if i_episode % 10000 == 0:
            print("Completed episode: " + str(i_episode))

    return rms, V

# Part (c): Implementation for first-visit Monte Carlo Control for epsilon-soft policies
def mc_control(env, epsilon=0.1, gamma=1, episodes=10000, runs=10, T=1000):
    np.random.seed(3)
    env.seed(5)
    avgrew = np.zeros(episodes)
    # Loop over runs
    for run in range(runs):
        policy = np.full((env.nS,env.nA), float(1/env.nA))
        Q = np.zeros((env.nS,env.nA))
        returns = [[ [] for _ in range(env.nA)] for _ in range(env.nS)]
        # Loop over episodes
        for i_episode in range(episodes):
            states_visited, actions_taken, rewards_received = generate_episode(env, policy, limit=T)
            G = 0
            # Loop over each step of the episode
            for step in range(len(states_visited)-1,-1,-1):
                G = gamma*G + rewards_received[step]
                if not pair_appears(states_visited, actions_taken, step):
                    state = states_visited[step]
                    action = actions_taken[step]
                    returns[state][action].append(G)
                    Q[state][action] = sum(returns[state][action]) / float(len(returns[state][action]))
                    max_action_val = max(Q[state])
                    all_max_idx = [idx for idx, val in enumerate(Q[state]) if val == max_action_val]
                    best_action = all_max_idx[0]
```

```
                    # Break ties randomly
                    if len(all_max_idx) > 1:
                        best_action = np.random.choice(all_max_idx)
                    for a in range(len(policy[state])):
                        if a == best_action:
                            policy[state][a] = 1 - epsilon + epsilon/(len(policy[state]))
                        else:
                            policy[state][a] = epsilon/(len(policy[state]))
            avgrew[i_episode] += sum(rewards_received) / float(runs)
            # To keep track of progress in loop
            if i_episode % 1000 == 0:
                print("Completed run: " + str(run) + " episode: " + str(i_episode))
    return avgrew


# Part (d) TD0
def td0(env, policy=given_policy, baseline=true_value_fn,gamma=1,alpha=0.1,episodes=50000):
    np.random.seed(3)
    env.seed(5)
    rms = np.zeros(episodes)
    V = np.zeros(env.nS)
    for i_episode in range(episodes):
        S = env.reset()
        done = False
        while not done:
            A = np.random.choice(ACTIONS,p=policy[S])
            S_prime, R, done, _ = env.step(A)
            V[S] = V[S] + alpha*(R + gamma*V[S_prime] - V[S])
            S = S_prime
        rms_value = math.sqrt(sum((V - baseline)**2)/float(len(V)))
        rms[i_episode] = rms_value
        # To keep track of progress in loop
        if i_episode % 10000 == 0:
            print("Completed episode: " + str(i_episode))
    return rms, V

# Helper for maybe use
def action_max(arr):
    max_action_val = max(arr)
    all_max_idx = [idx for idx, val in enumerate(arr) if val == max_action_val]
    best_action = all_max_idx[0]
    # Break ties randomly
    if len(all_max_idx) > 1:
```

```
            best_action = np.random.choice(all_max_idx)
    return best_action

# Part (e) qlearn
def qlearn(env,gamma=1,alpha=0.9,epsilon=0.1,runs=10,episodes=500):
    np.random.seed(3)
    env.seed(5)
    avgrew = np.zeros(episodes)
    # Loop over runs
    for run in range(runs):
        #policy = np.full((env.nS,env.nA), float(1/env.nA))
        Q = np.zeros((env.nS,env.nA))
        # Loop over episodes
        for i_episode in range(episodes):
            S = env.reset()
            done = False
            TotalReward = 0
            while not done:
                A = action_max(Q[S])
                if np.random.binomial(1, epsilon) == 1:
                    A = np.random.choice(ACTIONS)
                S_prime, R, done, _ = env.step(A)
                Q[S][A] = Q[S][A] + alpha*(R + gamma*(max(Q[S_prime])) - Q[S][A])
                S = S_prime
                TotalReward += R
            avgrew[i_episode] += TotalReward / float(runs)
            # To keep track of progress in loop
            if i_episode % 50 == 0:
                print("Completed run: " + str(run) + " episode: " + str(i_episode))
    return avgrew


if __name__ == '__main__':

    # Part (b): Utilization of first-visit Monte Carlo Prediction to plot rms vs episodes and
    #           scatter plot of estimated value function (red x) verses the true value function
    #           (blue empty o)
    rms, V = mc_prediction(ENV)
    episodes = np.arange(len(rms))
    states = np.arange(ENV.nS)

    fig = plt.figure(figsize=(40,20))
    txt = 'Figure 1: Plot for exploring taxi-v3 By Tejas Jha'
```

```
    plt.figtext(0.5, 0.01, txt, wrap=True, horizontalalignment='center', fontsize=28)

    # Generate plots for Part(b)
    plt.subplot(241)
    plt.plot(episodes,rms, 'r')
    plt.xlabel("Episodes", fontdict={'fontname':'DejaVu Sans', 'size':'20'})
    plt.ylabel("RMS", fontdict={'fontname':'DejaVu Sans', 'size':'20'})
    plt.title("MC prediction", fontdict={'fontname':'DejaVu Sans', 'size':'20'})

    plt.subplot(242)
    plt.plot(states,V, 'rx')
    plt.plot(states, true_value_fn, 'bo', mfc='none')
    plt.xlabel("State", fontdict={'fontname':'DejaVu Sans', 'size':'20'})
    plt.ylabel("V(s)", fontdict={'fontname':'DejaVu Sans', 'size':'20'})
    plt.title("MC prediction", fontdict={'fontname':'DejaVu Sans', 'size':'20'})

    # Part (c)
    #rdmwlk = randomwalk.RandomWalk()
    avgrew = mc_control(ENV)
    episodes = np.arange(len(avgrew))
    avgrew_subsamples = avgrew[::50]
    episodes_subsample = episodes[::50]

    plt.subplot(246)
    plt.plot(episodes_subsample,avgrew_subsamples, 'r')
    plt.xlabel("Episodes", fontdict={'fontname':'DejaVu Sans', 'size':'20'})
    plt.ylabel("Sum of rewards received within each episode", fontdict={'fontname':'DejaVu Sans', 'size':'20'})
    plt.title("MC Control", fontdict={'fontname':'DejaVu Sans', 'size':'20'})


    # Part (d)
    rms, V = td0(ENV)
    episodes = np.arange(len(rms))
    states = np.arange(ENV.nS)

    # Generate plots for Part(d)
    plt.subplot(243)
    plt.plot(episodes,rms, 'r')
```

```
    plt.xlabel("Episodes", fontdict={'fontname':'DejaVu Sans', 'size':'20'})
    plt.ylabel("RMS", fontdict={'fontname':'DejaVu Sans', 'size':'20'})
    plt.title("TD0", fontdict={'fontname':'DejaVu Sans', 'size':'20'})

    plt.subplot(244)
    plt.plot(states,V, 'rx')
    plt.plot(states, true_value_fn, 'bo', mfc='none')
    plt.xlabel("State", fontdict={'fontname':'DejaVu Sans', 'size':'20'})
    plt.ylabel("V(s)", fontdict={'fontname':'DejaVu Sans', 'size':'20'})
    plt.title("TD0", fontdict={'fontname':'DejaVu Sans', 'size':'20'})

    # Part (e)
    avgrew = qlearn(ENV)
    episodes = np.arange(len(avgrew))
    avgrew_subsamples = avgrew
    episodes_subsample = episodes

    plt.subplot(247)
    plt.plot(episodes_subsample,avgrew_subsamples, 'r')
    plt.xlabel("Episodes", fontdict={'fontname':'DejaVu Sans', 'size':'20'})
    plt.ylabel("Sum of rewards received within each episode", fontdict={'fontname':'DejaVu Sans', 'size':'20'})
    plt.title("Q learning", fontdict={'fontname':'DejaVu Sans', 'size':'20'})


    plt.savefig("Figure3")
```