

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.

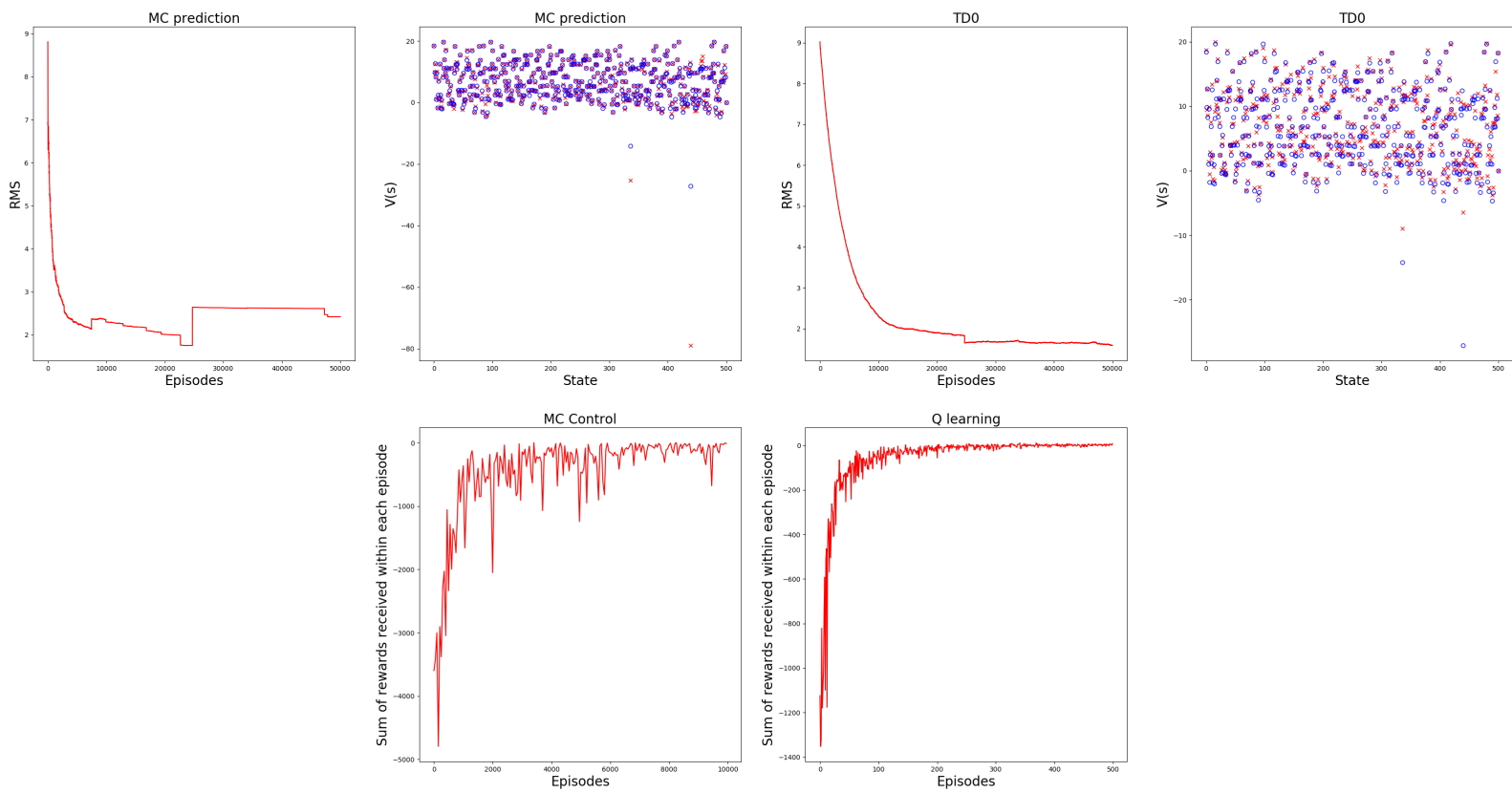


Figure 1: Plot for exploring taxi-v3 By Tejas Jha

Appendix: Relevant Code - tjha.py

```
1 # Tejas Jha
2 # EECS 498: Reinforcement Learning – Homework 2
3
4 import numpy as np
5 import gym
6 import copy
7 import mytaxi
8 import math
9 import mdp.mdp as mdp
10 import matplotlib.pyplot as plt
11 import randomwalk
12
13 ##### Setup in Part (a) #####
14 # Part (a): Policy Evaluation
15 def evaluate_policy(trans_mat, V_init, policy, theta, gamma=1, inplace=
    True):
16     return mdp.policy_eval(trans_mat, V_init, policy, theta, gamma,
        inplace)
17
18 # Global Variables for default actions taken
19 given_policy = np.load('policy.npy')
20 # Gather environment details and stored policy
21 ENV = gym.make('Taxi-v3').unwrapped
22 TRANS_MAT = ENV.P
23 V_INIT = np.zeros(len(TRANS_MAT))
24 ACTIONS = [0,1,2,3,4,5]
25 #ACTIONS = [0,1]
26 # Part (a): Evaluate value function of given policy in policy.npy
27 true_value_fn = evaluate_policy(TRANS_MAT, V_INIT, given_policy, theta
    =0.01, gamma=1)
28 #####
29
30 # Helper function – Generate the steps in an episode for an environment
    and given policy
31 # Returns: (T, 3) numpy array for length T with elements corresponding
    to each time step for
32 # states, actions, and Rewards
33 def generate_episode(env, policy, limit=100000):
34     states_visited = list()
35     actions_taken = list()
36     rewards_received = list()
37
38     # edge case implementation error possible (currently do not check
        if initial state is final state)
39     states_visited.append(env.reset())
```

```

40     actions_taken.append(np.random.choice(ACTIONS,p=policy[
        states_visited[-1]]))
41     next_state, reward, done, info = env.step(actions_taken[-1])
42
43     step = 0
44
45     while not done and step < limit:
46         rewards_received.append(reward)
47         states_visited.append(next_state)
48         actions_taken.append(np.random.choice(ACTIONS,p=policy[
            states_visited[-1]]))
49         next_state, reward, done, info = env.step(actions_taken[-1])
50         step += 1
51
52     rewards_received.append(reward)
53
54     return states_visited, actions_taken, rewards_received
55
56 # Helper function to see if pair of state-action was seen earlier
57 def pair_appears(states_visited, actions_taken, step):
58     state = states_visited[step]
59     action = actions_taken[step]
60     for idx in range(step):
61         if states_visited[idx] == state and actions_taken[idx] ==
            action:
62             return True
63     return False
64
65 # Part (b): Implementation for first-visit Monte Carlo Prediction for
estimating state-value
66 #
67 # Returns: rms w.r.t baseline at end of each episode (rms),
final value function (V)
68 def mc_prediction(env, policy=given_policy, baseline=true_value_fn,
    gamma=1, episodes=50000):
69     np.random.seed(3)
70     env.seed(5)
71     rms = np.zeros(episodes)
72     #V = np.random.rand(env.nS)
73     V = np.zeros(env.nS)
74     #V = np.full(env.nS, -1)
75     returns = [[] for _ in range(env.nS)]
76
77     # Loop over each episode run
78     for i_episode in range(episodes):
79         # Generate an episode following policy
80         states_visited, actions_taken, rewards_received =
            generate_episode(env, policy)

```

```

81     G = 0
82     # Loop over each step of the episode
83     for step in range(len(states_visited)-1, -1, -1):
84         G = gamma*G + rewards_received[step]
85         if states_visited.index(states_visited[step]) == step:
86             returns[states_visited[step]].append(G)
87             V[states_visited[step]] = sum(returns[states_visited[
88                 step]]) / float(len(returns[states_visited[step]]))
89     rms_value = math.sqrt(sum((V - baseline)**2)/float(len(V)))
90     rms[i_episode] = rms_value
91
92     # To keep track of progress in loop
93     if i_episode % 10000 == 0:
94         print("Completed_episode:_" + str(i_episode))
95
96     return rms, V
97
98 # Part (c): Implementation for first-visit Monte Carlo Control for
99 epsilon-soft policies
100 def mc_control(env, epsilon=0.1, gamma=1, episodes=10000, runs=10, T
101     =1000):
102     np.random.seed(3)
103     env.seed(5)
104     avgrew = np.zeros(episodes)
105     # Loop over runs
106     for run in range(runs):
107         policy = np.full((env.nS, env.nA), float(1/env.nA))
108         Q = np.zeros((env.nS, env.nA))
109         returns = [[ [] for _ in range(env.nA)] for _ in range(env.nS)]
110         # Loop over episodes
111         for i_episode in range(episodes):
112             states_visited, actions_taken, rewards_received =
113                 generate_episode(env, policy, limit=T)
114             G = 0
115             # Loop over each step of the episode
116             for step in range(len(states_visited)-1, -1, -1):
117                 G = gamma*G + rewards_received[step]
118                 if not pair_appears(states_visited, actions_taken, step
119                     ):
120                     state = states_visited[step]
121                     action = actions_taken[step]
122                     returns[state][action].append(G)
123                     Q[state][action] = sum(returns[state][action]) /
124                         float(len(returns[state][action]))
125                     max_action_val = max(Q[state])
126                     all_max_idx = [idx for idx, val in enumerate(Q[
127                         state]) if val == max_action_val]
128                     best_action = all_max_idx[0]

```

```

122         # Break ties randomly
123         if len(all_max_idx) > 1:
124             best_action = np.random.choice(all_max_idx)
125         for a in range(len(policy[state])):
126             if a == best_action:
127                 policy[state][a] = 1 - epsilon + epsilon/(
128                     len(policy[state]))
129             else:
130                 policy[state][a] = epsilon/(len(policy[
131                     state]))
132             avgrew[i_episode] += sum(rewards_received) / float(runs)
133         # To keep track of progress in loop
134         if i_episode % 1000 == 0:
135             print("Completed_run:_ " + str(run) + "_episode:_ " + str
136                 (i_episode))
137     return avgrew
138
139 # Part (d) TD0
140 def td0(env, policy=given_policy, baseline=true_value_fn, gamma=1, alpha
141     =0.1, episodes=50000):
142     np.random.seed(3)
143     env.seed(5)
144     rms = np.zeros(episodes)
145     V = np.zeros(env.nS)
146     for i_episode in range(episodes):
147         S = env.reset()
148         done = False
149         while not done:
150             A = np.random.choice(ACTIONS, p=policy[S])
151             S_prime, R, done, _ = env.step(A)
152             V[S] = V[S] + alpha*(R + gamma*V[S_prime] - V[S])
153             S = S_prime
154         rms_value = math.sqrt(sum((V - baseline)**2)/float(len(V)))
155         rms[i_episode] = rms_value
156         # To keep track of progress in loop
157         if i_episode % 10000 == 0:
158             print("Completed_episode:_ " + str(i_episode))
159     return rms, V
160
161 # Helper for maybe use
162 def action_max(arr):
163     max_action_val = max(arr)
164     all_max_idx = [idx for idx, val in enumerate(arr) if val ==
165         max_action_val]
166     best_action = all_max_idx[0]
167     # Break ties randomly
168     if len(all_max_idx) > 1:

```

```

165         best_action = np.random.choice(all_max_idx)
166     return best_action
167
168 # Part (e) qlearn
169 def qlearn(env, gamma=1, alpha=0.9, epsilon=0.1, runs=10, episodes=500):
170     np.random.seed(3)
171     env.seed(5)
172     avgrew = np.zeros(episodes)
173     # Loop over runs
174     for run in range(runs):
175         #policy = np.full((env.nS, env.nA), float(1/env.nA))
176         Q = np.zeros((env.nS, env.nA))
177         # Loop over episodes
178         for i_episode in range(episodes):
179             S = env.reset()
180             done = False
181             TotalReward = 0
182             while not done:
183                 A = action_max(Q[S])
184                 if np.random.binomial(1, epsilon) == 1:
185                     A = np.random.choice(ACTIONS)
186                 S_prime, R, done, _ = env.step(A)
187                 Q[S][A] = Q[S][A] + alpha*(R + gamma*(max(Q[S_prime]))
188                     - Q[S][A])
189                 S = S_prime
190                 TotalReward += R
191             avgrew[i_episode] += TotalReward / float(runs)
192             # To keep track of progress in loop
193             if i_episode % 50 == 0:
194                 print("Completed_run:_ " + str(run) + "_episode:_ " + str
195                     (i_episode))
196
197     return avgrew
198
199 if __name__ == '__main__':
200     # Part (b): Utilization of first-visit Monte Carlo Prediction to
201     # plot rms vs episodes and
202     # scatter plot of estimated value function (red x) verses
203     # the true value function
204     # (blue empty o)
205     rms, V = mc_prediction(ENV)
206     episodes = np.arange(len(rms))
207     states = np.arange(ENV.nS)
208
209     fig = plt.figure(figsize=(40,20))
210     txt = 'Figure 1: Plot for exploring taxi-v3 By Tejas Jha'

```

```

208     plt.figtext(0.5, 0.01, txt, wrap=True, horizontalalignment='center',
209                , fontsize=28)
210
211     # Generate plots for Part(b)
212     plt.subplot(241)
213     plt.plot(episodes, rms, 'r')
214     plt.xlabel("Episodes", fontdict={'fontname': 'DejaVu_Sans', 'size': '20'})
215     plt.ylabel("RMS", fontdict={'fontname': 'DejaVu_Sans', 'size': '20'})
216     plt.title("MC_prediction", fontdict={'fontname': 'DejaVu_Sans', 'size': '20'})
217
218     plt.subplot(242)
219     plt.plot(states, V, 'rx')
220     plt.plot(states, true_value_fn, 'bo', mfc='none')
221     plt.xlabel("State", fontdict={'fontname': 'DejaVu_Sans', 'size': '20'})
222     plt.ylabel("V(s)", fontdict={'fontname': 'DejaVu_Sans', 'size': '20'})
223     plt.title("MC_prediction", fontdict={'fontname': 'DejaVu_Sans', 'size': '20'})
224
225     # Part (c)
226     #rdmwlk = randomwalk.RandomWalk()
227     avgrew = mc_control(ENV)
228     episodes = np.arange(len(avgrew))
229     avgrew_subsamples = avgrew[::50]
230     episodes_subsample = episodes[::50]
231
232     plt.subplot(246)
233     plt.plot(episodes_subsample, avgrew_subsamples, 'r')
234     plt.xlabel("Episodes", fontdict={'fontname': 'DejaVu_Sans', 'size': '20'})
235     plt.ylabel("Sum_of_rewards_received_within_each_episode", fontdict=
236               {'fontname': 'DejaVu_Sans', 'size': '20'})
237     plt.title("MC_Control", fontdict={'fontname': 'DejaVu_Sans', 'size': '20'})
238
239     # Part (d)
240     rms, V = td0(ENV)
241     episodes = np.arange(len(rms))
242     states = np.arange(ENV.nS)
243
244     # Generate plots for Part(d)
245     plt.subplot(243)
246     plt.plot(episodes, rms, 'r')

```



```

246 plt.xlabel("Episodes", fontdict={'fontname': 'DejaVu_Sans', 'size': '
    20'})
247 plt.ylabel("RMS", fontdict={'fontname': 'DejaVu_Sans', 'size': '20'})
248 plt.title("TD0", fontdict={'fontname': 'DejaVu_Sans', 'size': '20'})
249
250 plt.subplot(244)
251 plt.plot(states, V, 'rx')
252 plt.plot(states, true_value_fn, 'bo', mfc='none')
253 plt.xlabel("State", fontdict={'fontname': 'DejaVu_Sans', 'size': '20'
    })
254 plt.ylabel("V(s)", fontdict={'fontname': 'DejaVu_Sans', 'size': '20'
    })
255 plt.title("TD0", fontdict={'fontname': 'DejaVu_Sans', 'size': '20'})
256
257 # Part (e)
258 avgrew = qlearn(ENV)
259 episodes = np.arange(len(avgrew))
260 avgrew_subsamples = avgrew
261 episodes_subsample = episodes
262
263 plt.subplot(247)
264 plt.plot(episodes_subsample, avgrew_subsamples, 'r')
265 plt.xlabel("Episodes", fontdict={'fontname': 'DejaVu_Sans', 'size': '
    20'})
266 plt.ylabel("Sum_of_rewards_received_within_each_episode", fontdict
    ={'fontname': 'DejaVu_Sans', 'size': '20'})
267 plt.title("Q_learning", fontdict={'fontname': 'DejaVu_Sans', 'size':
    '20'})
268
269
270 plt.savefig("Figure3")

```