

EECS 498: Reinforcement Learning

Homework 3 Responses

Tejas Jha
tjha

November 5, 2018

This document includes my responses to Homework 3 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) The dynaq algorithm was implemented following the pseudocode shown on page 164 of the textbook. In the below plot, the red curve corresponds to the use of the Q-learning algorithm and the blue curve represents the use of the dynaq algorithm. As can be seen by the images, the curve for dynaq appears to approach a constant slope earlier than the curve for Q-learning, indicating a faster approach towards an optimal policy with the optimal number of steps required in an episode to reach the terminal state.

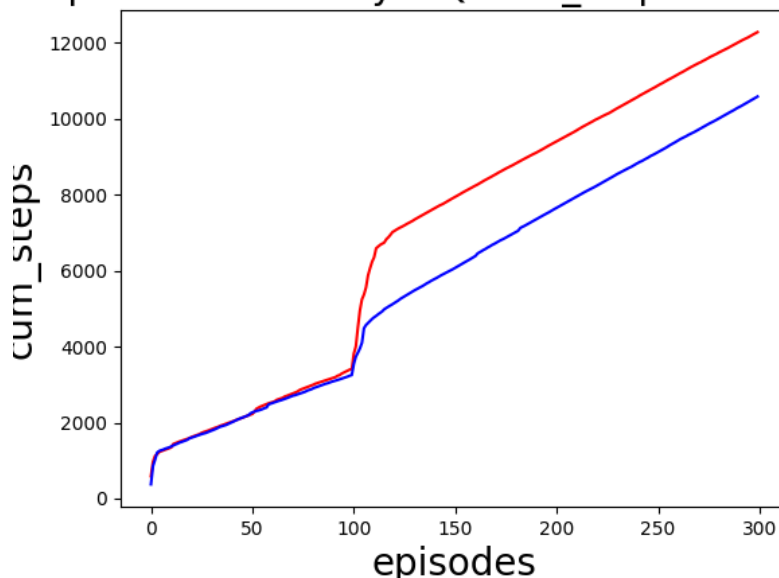


- (b) In the case where the environment switches from "Taxi-v4" to "Taxi-v5" after the initial 100 episodes, the length of each episode drastically increased as the algorithm needed more iterations to adjust the

model and action-values to the new environment. As the algorithm was able to iterate and better learn the new environment, the length of each episode begins to converge to a constant as represented by a straight line. In order to better account for the change in the environment, I have modified the use of the algorithm to use a lower discount rate (γ). In my case, I used a discount factor of 0.5, which would help in a faster adjustment to the new environment by reducing the weight placed on already learned Q values. As a result, the algorithm is affected less by future expected returns and more by the immediate rewards, which greatly boosts immediate learning to a changing environment. Note that a similar change can be seen by decreasing alpha, the learning rate, as well as a combination of both.

The plot below displays this difference with the two curves. The red curve is the original algorithm from part (a) reacting to the change in the environment. The blue curve represents the modified algorithm as described above.

naq and Modified DynaQ cum_steps over episodes



Question 2

Appendix: Relevant Code - tjha.py

```

1 # Tejas Jha
2 # 5 November 2018
3 # EECS 498 – Reinforcement Learning HW 3
4 #
5 #

```

```

6 # The code below implements a modified version of "Tabular Dyna-Q" as
   # well as
7 # implementation modifications to adapt to changes in the environemnt
   # faster.

```

```

8 # This file also contains the code used to implement n-step semi-
   gradient TD estimation
9 #
10 # The functions below expand on the starter code provided that help
   generate the
11 # states , actions , and rewards using Taxi-v4 and Taxi-v5 on openAi gym
12 #

```

```

13
14 import numpy as np
15 import gym
16 import copy
17 import mytaxi
18 import math
19 import matplotlib.pyplot as plt
20
21 # Environment for Taxi-v4
22 ENV4 = gym.make('Taxi-v4').unwrapped
23 # Environment for Taxi-v5
24 ENV5 = gym.make('Taxi-v5').unwrapped
25 # Possible actions that can be taken
26 ACTIONS = [0,1,2,3,4,5]
27
28 # Helpers to choose best action given probability distributions
29 def _greedy(Q,s):
30     qmax = np.max(Q[s])
31     actions = []
32     for i,q in enumerate(Q[s]):
33         if q == qmax:
34             actions.append(i)
35     return actions
36
37 def greedy(Q,s):
38     return np.random.choice(_greedy(Q,s))
39
40 def ep_greedy(Q,s,ep):
41     if np.random.rand() < ep:
42         return np.random.choice(len(Q[s]))
43     else:
44         return greedy(Q,s)
45
46 # Part (a) – Tabular Dyna-Q:
47 # Function implementation of dyna-q to handle the stochastic nature of
   the environment
48 # returns Q and cum_steps – list of the cumulative number of steps
   counted from the
49 # first episode to the end of each episode.

```

```

50 def dynaQ(env, n=10, gamma=1, alpha=1, epsilon=0.1, runs=1, episodes=100):
51     for run in range(runs):
52         # Update kept on cumulative number of steps
53         cum_steps = np.zeros(episodes)
54         Q = np.zeros((env.nS, env.nA))
55         # Using deterministic model
56         Model = {}
57         for s in range(env.nS):
58             Model[s] = {}
59             for a in range(env.nA):
60                 Model[s][a] = (-1, s)
61         # Loop over episodes
62         for idx in range(episodes):
63             visited_states = []
64             taken_actions = {}
65             s = env.reset()
66             visited_states.append(s)
67             done = False
68             counter = 0
69             while not done:
70                 a = ep_greedy(Q, s, epsilon)
71                 if s in taken_actions:
72                     if a not in taken_actions[s]:
73                         taken_actions[s].append(a)
74                 else:
75                     taken_actions[s] = []
76                     taken_actions[s].append(a)
77                 ss, r, done, _ = env.step(a)
78                 Q[s, a] = Q[s, a] + alpha * (r + gamma * np.max(Q[ss]) -
79                                     Q[s, a])
80                 Model[s][a] = (r, ss)
81                 s = ss
82                 for i in range(n):
83                     rand_s = np.random.choice(visited_states, size=1)
84                     rand_s = rand_s[0]
85                     rand_a = np.random.choice(taken_actions[rand_s],
86                                                 size=1)
87                     rand_a = rand_a[0]
88                     tup = Model[rand_s][rand_a]
89                     Q[rand_s, rand_a] = Q[rand_s, rand_a] + alpha * (tup
90                             [0] + gamma * np.max(Q[tup[1]])) - Q[rand_s,
91                             rand_a])
92                 counter += 1
93                 visited_states.append(s)
94             if idx == 0:
95                 cum_steps[idx] = counter
96             else:
97                 cum_steps[idx] = counter + cum_steps[idx - 1]

```

```

94     return Q, cum_steps
95
96 # Part (a) – adaptation of qlearn from hw2 to compare with Tabular Dyna
    -Q
97 # Key difference is 100 episodes instead of 500 by default
98 # Also, steps are now averaged through multiple callings of function
99 def qlearn(env, gamma=1, alpha=0.9, ep=0.05, runs=1, episodes=100):
100     #np.random.seed(3)
101     #env.seed(5)
102     nS = env.nS
103     nA = env.nA
104     rew_alloc = []
105     for run in range(runs):
106         Q = np.zeros((nS, nA))
107         rew_list = np.zeros(episodes)
108         cum_steps = np.zeros(episodes)
109         for idx in range(episodes):
110             s = env.reset()
111             done = False
112             counter = 0
113             cum_rew = 0
114             while not done:
115                 a = ep_greedy(Q, s, ep)
116                 ss, r, done, _ = env.step(a)
117                 Q[s, a] = Q[s, a] + alpha * (r + gamma * np.max(Q[ss]) -
                    Q[s, a])
118                 s = ss
119                 cum_rew += gamma**counter * r
120                 counter += 1.
121             rew_list[idx] = cum_rew
122             if idx == 0:
123                 cum_steps[idx] = counter
124             else:
125                 cum_steps[idx] = counter + cum_steps[idx - 1]
126             rew_alloc.append(rew_list)
127         rew_list = np.mean(np.array(rew_alloc), axis=0)
128     return Q, cum_steps
129
130 # Modified versions of the algorithms above for usage in random change
    to v5 environment after 100 episodes
131 def original_dynaq(env1, env2, n=10, gamma=1, alpha=1, epsilon=0.1, runs=1,
    episodes=300):
132     for run in range(runs):
133         # Update kept on cumulative number of steps
134         cum_steps = np.zeros(episodes)
135         Q = np.zeros((env1.nS, env1.nA))
136         # Using deterministic model
137         Model = {}

```

```

138     for s in range(env1.nS):
139         Model[s] = {}
140         for a in range(env1.nA):
141             Model[s][a] = (-1, s)
142     env = env1
143     # Loop over episodes
144     for idx in range(episodes):
145         if idx == 100:
146             env = env2
147             visited_states = []
148             taken_actions = {}
149             s = env.reset()
150             visited_states.append(s)
151             done = False
152             counter = 0
153             while not done:
154                 a = ep_greedy(Q,s,epsilon)
155                 if s in taken_actions:
156                     if a not in taken_actions[s]:
157                         taken_actions[s].append(a)
158                 else:
159                     taken_actions[s] = []
160                     taken_actions[s].append(a)
161                 ss, r, done, _ = env.step(a)
162                 Q[s,a] = Q[s,a] + alpha * (r + gamma * np.max(Q[ss]) -
                    Q[s,a])
163                 Model[s][a] = (r, ss)
164                 s = ss
165                 for i in range(n):
166                     rand_s = np.random.choice(visited_states, size=1)
167                     rand_s = rand_s[0]
168                     rand_a = np.random.choice(taken_actions[rand_s],
                        size=1)
169                     rand_a = rand_a[0]
170                     tup = Model[rand_s][rand_a]
171                     Q[rand_s,rand_a] = Q[rand_s,rand_a] + alpha * (tup
                        [0] + gamma * np.max(Q[tup[1]]) - Q[rand_s,
                        rand_a])
172                 counter += 1
173                 visited_states.append(s)
174             if idx == 0:
175                 cum_steps[idx] = counter
176             else:
177                 cum_steps[idx] = counter + cum_steps[idx - 1]
178     return Q, cum_steps
179
180 # Modified improvement for dynaq to account for environment change
181 def modified_dynaq(env1, env2, n=10,gamma=0.5,alpha=0.5,epsilon=0.1,

```

```

runs=1, episodes=300):
182     for run in range(runs):
183         # Update kept on cumulative number of steps
184         cum_steps = np.zeros(episodes)
185         Q = np.zeros((env1.nS, env1.nA))
186         # Using deterministic model
187         Model = {}
188         for s in range(env1.nS):
189             Model[s] = {}
190             for a in range(env1.nA):
191                 Model[s][a] = (-1, s)
192     env = env1
193     # Loop over episodes
194     for idx in range(episodes):
195         if idx == 100:
196             env = env2
197             visited_states = []
198             taken_actions = {}
199             s = env.reset()
200             visited_states.append(s)
201             done = False
202             counter = 0
203             while not done:
204                 a = ep_greedy(Q, s, epsilon)
205                 if s in taken_actions:
206                     if a not in taken_actions[s]:
207                         taken_actions[s].append(a)
208                 else:
209                     taken_actions[s] = []
210                     taken_actions[s].append(a)
211                 ss, r, done, _ = env.step(a)
212                 Q[s, a] = Q[s, a] + alpha * (r + gamma * np.max(Q[ss]) -
                    Q[s, a])
213                 Model[s][a] = (r, ss)
214                 s = ss
215                 for i in range(n):
216                     rand_s = np.random.choice(visited_states, size=1)
217                     rand_s = rand_s[0]
218                     rand_a = np.random.choice(taken_actions[rand_s],
                        size=1)
219                     rand_a = rand_a[0]
220                     tup = Model[rand_s][rand_a]
221                     Q[rand_s, rand_a] = Q[rand_s, rand_a] + alpha * (tup
                        [0] + gamma * np.max(Q[tup[1]]) - Q[rand_s,
                            rand_a])
222                 counter += 1
223                 visited_states.append(s)
224             if idx == 0:

```

```

225         cum_steps[idx] = counter
226     else:
227         cum_steps[idx] = counter + cum_steps[idx - 1]
228     return Q, cum_steps
229
230 if __name__ == '__main__':
231
232     # Part (a)
233     print("Training using Tabular Dyna-Q for 100 episodes using Taxi-v4
        ")
234     # Average results over 20 runs
235     #dynaq-Q_avg = 0
236     #qlearn-Q_avg = 0
237
238     # dynaq_cum_steps_avg = np.zeros(shape=(100,))
239     # qlearn_cum_steps_avg = np.zeros(shape=(100,))
240
241     # for i in range(20):
242     #     print("Performing run: " + str(i + 1))
243     #     # Randomize seeds so runs are independent
244     #     np.random.seed(i)
245     #     ENV4.seed(i)
246     #     _, dynaq_cum_steps = dynaq(ENV4)
247     #     _, qlearn_cum_steps = qlearn(ENV4)
248     #     # Update averages
249     #     #dynaq-Q_avg += dynaq-Q / 20.0
250     #     #qlearn-Q_avg += qlearn-Q / 20.0
251     #     dynaq_cum_steps_avg += np.divide(dynaq_cum_steps, 20.0)
252     #     qlearn_cum_steps_avg += np.divide(qlearn_cum_steps, 20.0)
253
254     # # Compare results with Q learning implementation used in hw2 in
        plot
255     # # Generate plots for Question 1 Part(a)
256     # episodes = np.arange(len(qlearn_cum_steps_avg))
257     # plt.plot(episodes, qlearn_cum_steps_avg, 'r')
258     # plt.plot(episodes, dynaq_cum_steps_avg, 'b')
259     # plt.xlabel("episodes", fontdict={'fontname': 'DejaVu Sans', 'size
        ': '20'})
260     # plt.ylabel("cum_steps", fontdict={'fontname': 'DejaVu Sans', 'size
        ': '20'})
261     # plt.title("Dyna-Q and Q-learn cum_steps over episodes", fontdict
        ={'fontname': 'DejaVu Sans', 'size': '20'})
262     # plt.savefig("Figure1")
263
264
265     dynaq_cum_steps_avg1 = np.zeros(shape=(300,))
266     dynaq_cum_steps_avg2 = np.zeros(shape=(300,))
267     for i in range(5):

```



```

268     print("Performing run:_" + str(i + 1))
269     # Randomize seeds so runs are independent
270     np.random.seed(i)
271     ENV4.seed(i)
272     ENV5.seed(i)
273     _, dynaq_cum_steps1 = original_dynaq(ENV4, ENV5)
274     _, dynaq_cum_steps2 = modified_dynaq(ENV4, ENV5)
275     # Update averages
276     #dynaq-Q_avg += dynaq-Q / 20.0
277     #qlearn-Q_avg += qlearn-Q / 20.0
278     dynaq_cum_steps_avg1 += np.divide(dynaq_cum_steps1, 5.0)
279     dynaq_cum_steps_avg2 += np.divide(dynaq_cum_steps2, 5.0)
280
281     episodes = np.arange(len(dynaq_cum_steps_avg1))
282     plt.plot(episodes, dynaq_cum_steps_avg1, 'r')
283     plt.plot(episodes, dynaq_cum_steps_avg2, 'b')
284     plt.xlabel("episodes", fontdict={'fontname': 'DejaVu_Sans', 'size': '20'})
285     plt.ylabel("cum_steps", fontdict={'fontname': 'DejaVu_Sans', 'size': '20'})
286     plt.title("DynaQ_and_Modified_DynaQ_cum_steps_over_episodes",
               fontdict={'fontname': 'DejaVu_Sans', 'size': '20'})
287     plt.savefig("Figure4")

```