EECS 498: Reinforcement Learning
Homework 3 Responses

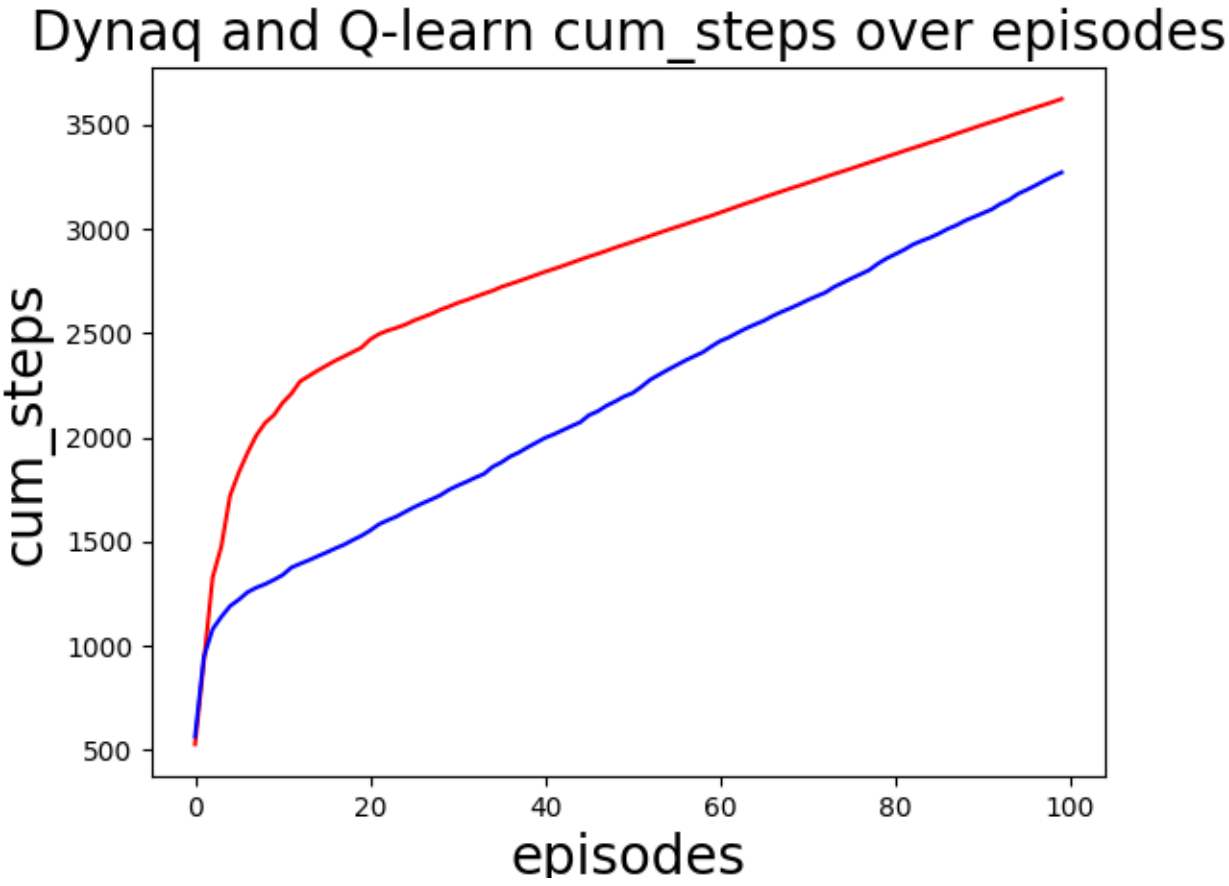
Tejas Jha
tjha

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.

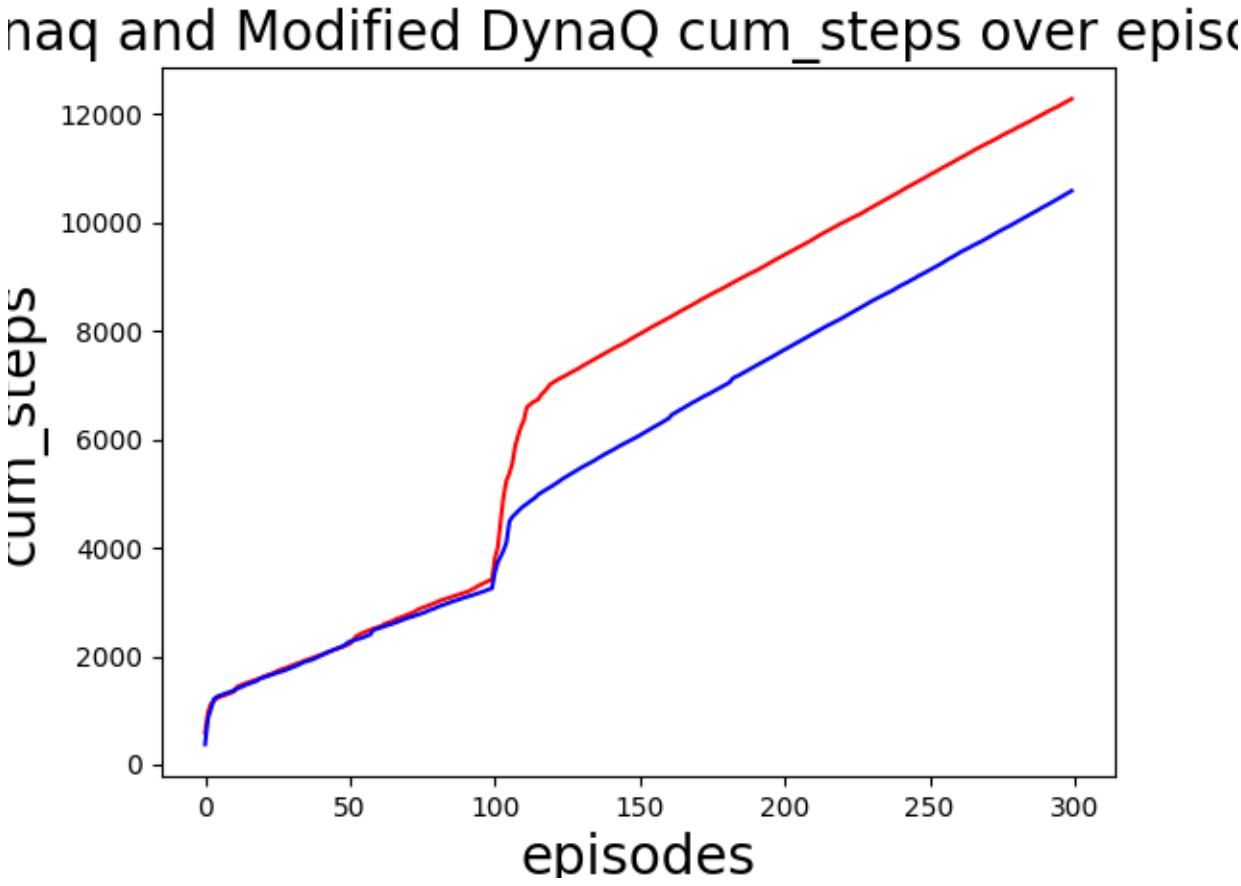
Question 1

(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 the Q-learning algorithm and the blue curve represents the use of dynaq algorithm. As can be seen by the images, there curve for dynaq appears to apprach 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.



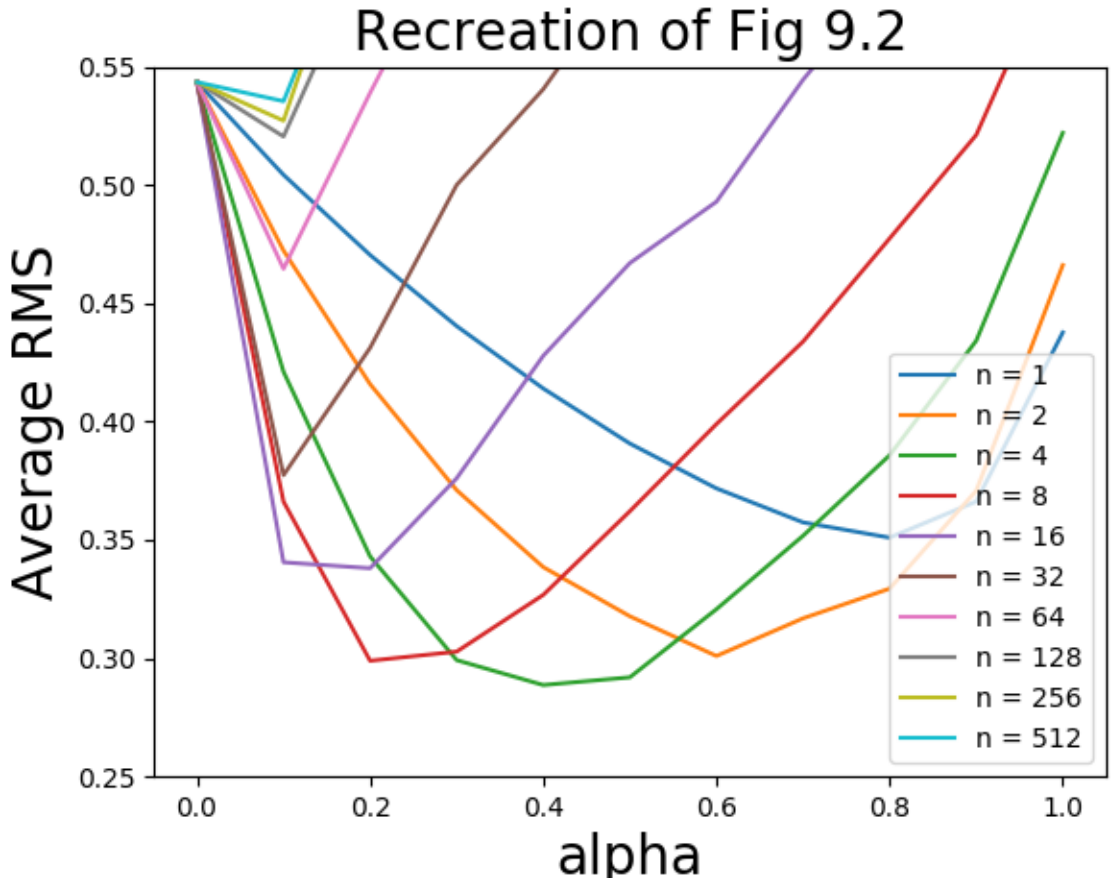
(b) In the case where the environment switches from "Taxi-v4" to "Taxi-v5" after the initial 100 episodes, the length of each episode drastically increased as the algorithm needed more iterations to adjust the model and action-values to the new environment. As the algorithm was able to iterate and better 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 discout 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 step size, as well as a combination of both.

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



Question 2

Below is the plot produced in an attempt to reproduce the right plot in Figure 9.2 of the textbook.



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 # Einvironment 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 dynaq to handle the stochastic nature of
      the environment
48 # returns Q and cum_steps - list of the cumulatative 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]) -
                        Q[s,a])
79                  Model[s][a] = (r, ss)
80                  s = ss
81                  for i in range(n):
82                      rand_s = np.random.choice(visited_states, size=1)
83                      rand_s = rand_s[0]
84                      rand_a = np.random.choice(taken_actions[rand_s],
                            size=1)
85                      rand_a = rand_a[0]
86                      tup = Model[rand_s][rand_a]
87                      Q[rand_s,rand_a] = Q[rand_s,rand_a] + alpha * (tup
                            [0] + gamma * np.max(Q[tup[1]]) - Q[rand_s,
                            rand_a])
88                  counter += 1
89                  visited_states.append(s)
90              if idx == 0:
91                  cum_steps[idx] = counter
92              else:
93                  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 everaged 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=1,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 class ValueFunction:
231     def __init__(self, num_of_groups):
```

```
232         self.num_of_groups = num_of_groups
233         self.group_size = 1000
234         self.params = np.zeros(num_of_groups)
235
236     def value(self, state):
237         if state in [0, 1002]:
238             return 0
239         group_index = (state - 1) // self.group_size
240         return self.params[group_index]
241
242     # update parameters
243     def update(self, delta, state):
244         group_index = (state - 1) // self.group_size
245         self.params[group_index] += delta
246
247 def get_action():
248     if np.random.binomial(1, 0.5) == 1:
249         return 1
250     return -1
251
252 def step(state, action):
253     step = np.random.randint(1, 100 + 1)
254     step *= action
255     state += step
256     state = max(min(state, 1000 + 1), 0)
257     if state == 0:
258         reward = -1
259     elif state == 1000 + 1:
260         reward = 1
261     else:
262         reward = 0
263     return state, reward
264
265 # Random Walk Algorithm
266 def randomWalk(alpha, n, tsv, runs=100, episodes=10):
267
268     rms = 0
269     for run in range(runs):
270         for episode in range(episodes):
271             value_function = ValueFunction(20)
272
273             # initial starting state
274             state = 500
275
276             # arrays to store states and rewards for an episode
277             # space isn't a major consideration, so I didn't use the
                mod trick
278             states = [state]
```

```
279             rewards = [0]
280
281             # track the time
282             time = 0
283
284             # the length of this episode
285             T = float('inf')
286             while True:
287                 # go to next time step
288                 time += 1
289
290                 if time < T:
291                     # choose an action randomly
292                     action = get_action()
293                     next_state, reward = step(state, action)
294
295                     # store new state and new reward
296                     states.append(next_state)
297                     rewards.append(reward)
298
299                     if next_state in [0, 1002]:
300                         T = time
301
302                 # get the time of the state to update
303                 update_time = time - n
304                 if update_time >= 0:
305                     returns = 0.0
306                     # calculate corresponding rewards
307                     for t in range(update_time + 1, min(T, update_time
                            + n) + 1):
308                         returns += rewards[t]
309                     # add state value to the return
310                     if update_time + n <= T:
311                         returns += value_function.value(states[
                                update_time + n])
312                     state_to_update = states[update_time]
313                     # update the value function
314                     if not state_to_update in [0,1002]:
315                         delta = alpha * (returns - value_function.value
                                (state_to_update))
316                         value_function.update(delta, state_to_update)
317                 if update_time == T - 1:
318                     break
319                 state = next_state
320
321             state_value = np.asarray([value_function.value(i) for i in
                    np.arange(1, 1000 + 1)])
322             rms += np.sqrt(np.sum(np.power(state_value - tsv[1: -1], 2)
```

```
                ) / 1000) / runs / episodes
323
324
325
326
327         # if run % 10 == 0:
328         #     print("Finished run: " + str(run))
329
330     print(rms)
331     return rms
332
333 # other
334 def semi_gradient_temporal_difference(value_function, n, alpha):
335
336     # initial starting state
337     state = 500
338
339     # arrays to store states and rewards for an episode
340     # space isn't a major consideration, so I didn't use the mod trick
341     states = [state]
342     rewards = [0]
343
344     # track the time
345     time = 0
346
347     # the length of this episode
348     T = float('inf')
349     while True:
350         # go to next time step
351         time += 1
352
353         if time < T:
354             # choose an action randomly
355             action = get_action()
356             next_state, reward = step(state, action)
357
358             # store new state and new reward
359             states.append(next_state)
360             rewards.append(reward)
361
362             if next_state in [0, 1002]:
363                 T = time
364
365         # get the time of the state to update
366         update_time = time - n
367         if update_time >= 0:
368             returns = 0.0
369             # calculate corresponding rewards
```

```
370             for t in range(update_time + 1, min(T, update_time + n) +
                    1):
371                 returns += rewards[t]
372             # add state value to the return
373             if update_time + n <= T:
374                 returns += value_function.value(states[update_time + n
                        ])
375             state_to_update = states[update_time]
376             # update the value function
377             if not state_to_update in [0,1002]:
378                 delta = alpha * (returns - value_function.value(
                        state_to_update))
379                 value_function.update(delta, state_to_update)
380         if update_time == T - 1:
381             break
382         state = next_state
383
384
385 if __name__ == '__main__':
386
387     # Part (a)
388     print("Training using Tabular Dyna-Q for 100 episodes using Taxi-v4
            ")
389     # Average results over 20 runs
390     #dynaq_Q_avg = 0
391     #qlearn_Q_avg = 0
392
393     # dynaq_cum_steps_avg = np.zeros(shape=(100,))
394     # qlearn_cum_steps_avg = np.zeros(shape=(100,))
395
396     # for i in range(20):
397     #     print("Performing run: " + str(i + 1))
398     #     # Randomize seeds so runs are independent
399     #     np.random.seed(i)
400     #     ENV4.seed(i)
401     #     _, dynaq_cum_steps = dynaq(ENV4)
402     #     _, qlearn_cum_steps = qlearn(ENV4)
403     #     # Update averages
404     #     #dynaq_Q_avg += dynaq_Q / 20.0
405     #     #qlearn_Q_avg += qlearn_Q / 20.0
406     #     dynaq_cum_steps_avg += np.divide(dynaq_cum_steps, 20.0)
407     #     qlearn_cum_steps_avg += np.divide(qlearn_cum_steps, 20.0)
408
409     # # Compare results with Q learning implementation used in hw2 in
            plot
410     # # Generate plots for Question 1 Part(a)
411     # episodes = np.arange(len(qlearn_cum_steps_avg))
412     # plt.plot(episodes,qlearn_cum_steps_avg, 'r')
```

```
413     # plt.plot(episodes, dynaq_cum_steps_avg, 'b')
414     # plt.xlabel("episodes", fontdict={'fontname':'DejaVu Sans', 'size
            ':'20'})
415     # plt.ylabel("cum_steps", fontdict={'fontname':'DejaVu Sans', 'size
            ':'20'})
416     # plt.title("Dynaq and Q-learn cum_steps vs. episodes", fontdict={'
            fontname':'DejaVu Sans', 'size':'20'})
417     # plt.savefig("Figure1")
418
419
420     # dynaq_cum_steps_avg1 = np.zeros(shape=(300,))
421     # dynaq_cum_steps_avg2 = np.zeros(shape=(300,))
422     # for i in range(5):
423     #     print("Performing run: " + str(i + 1))
424     #     # Randomize seeds so runs are independent
425     #     np.random.seed(i)
426     #     ENV4.seed(i)
427     #     ENV5.seed(i)
428     #     _, dynaq_cum_steps1 = original_dynaq(ENV4, ENV5)
429     #     _, dynaq_cum_steps2 = modified_dynaq(ENV4, ENV5)
430     #     # Update averages
431     #     #dynaq_Q_avg += dynaq_Q / 20.0
432     #     #qlearn_Q_avg += qlearn_Q / 20.0
433     #     dynaq_cum_steps_avg1 += np.divide(dynaq_cum_steps1, 5.0)
434     #     dynaq_cum_steps_avg2 += np.divide(dynaq_cum_steps2, 5.0)
435
436     # episodes = np.arange(len(dynaq_cum_steps_avg1))
437     # plt.plot(episodes,dynaq_cum_steps_avg1, 'r')
438     # plt.plot(episodes, dynaq_cum_steps_avg2, 'b')
439     # plt.xlabel("episodes", fontdict={'fontname':'DejaVu Sans', 'size
            ':'20'})
440     # plt.ylabel("cum_steps", fontdict={'fontname':'DejaVu Sans', 'size
            ':'20'})
441     # plt.title("Original/Modified DynaQ cum_steps vs. episodes",
            fontdict={'fontname':'DejaVu Sans', 'size':'20'})
442     # plt.savefig("Figure2")
443
444     plt.figure(figsize=(20,10))
445     alphas = np.arange(0,1.1, 0.1)
446     ns = np.power(2,np.arange(0,10))
447     tsv = np.load('trueStateValue.npy')
448
449     print(ns)
450     print(alphas)
451
452     plt.xlabel("alpha", fontdict={'fontname':'DejaVu Sans', 'size':'20'
            })
453     plt.ylabel("Average \nRMS error \nover 1000 states \nand first 10\
```

```
        nepisodes", fontdict={'fontname': 'DejaVu Sans', 'size':'20'})
454     plt.title("Original/Modified DynaQ cum_steps vs. episodes",
            fontdict={'fontname': 'DejaVu Sans', 'size':'20'})
455
456     # dynaq_cum_steps_avg = np.zeros(shape=(100,))
457     # episodes = np.arange(len(dynaq_cum_steps_avg))
458     # for n in ns:
459     #     print(n)
460     #     lab = "n=" + str(n)
461     #     errorset = []
462     #     for alph in alpha:
463     #         print(alph)
464     #         rms = randomWalk(alph, n, tsv)
465     #         errorset.append(rms)
466     #     plt.plot(alpha,errorset, label=lab)
467     #     break
468     # plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
469     # plt.savefig("FigureX")
470
471
472     # track the errors for each (step, alpha) combination
473     errors = np.zeros((len(ns), len(alphas)))
474     for run in range(10):
475         for step_ind, n in zip(range(len(ns)), ns):
476             for alpha_ind, alpha in zip(range(len(alphas)), alphas):
477                 # we have 20 aggregations in this example
478                 value_function = ValueFunction(20)
479                 for ep in range(0, 10):
480                     semi_gradient_temporal_difference(value_function, n
                            , alpha)
481                     # calculate the RMS error
482                     state_value = np.asarray([value_function.value(i)
                            for i in np.arange(1, 1000 + 1)])
483                     errors[step_ind, alpha_ind] += np.sqrt(np.sum(np.
                            power(state_value - tsv[1: -1], 2)) / 1000)
484         print(run)
485     # take average
486     errors /= 10 * 10
487     # truncate the error
488     for i in range(len(ns)):
489         plt.plot(alphas, errors[i, :], label='n = ' + str(ns[i]))
490     plt.legend()
491     plt.savefig("FigureY")
```