**Programming Assignment 4**

1. An advantage to bootstrapping in Easy 21 is that we can update in any step of the algorithm, which allows us to adapt to the values given to us. Monte Carlo, instead updates at the end of each episode and does not allow for updating after each step.
2. I would expect bootstrapping to help more in blackjack than easy 21. In the implementation of easy 21 that we have, the value of the card is generated randomly, whereas with blackjack it is much more predictable since we are using a deck. This, along with the fact that we can update in between each step, makes it advantageous for blackjack.
3. With function approximation we could have a guess at what the outcome of the algorithm will be and use this to make our decision to maximize reward.
4. We could learn an exploration and step-size to maximize our potential reward.

CODE:

1. from \_\_future\_\_ import print\_function
2. #!/usr/bin/env python2
3. # -\*- coding: utf-8 -\*-
4. """
5. Created on Tue Nov 14 10:07:17 2017
7. @author: rditljtd
8. """
10. from random import randint
11. import operator
12. import sys
13. import random
14. import time
16. hit = False
17. currentPlayerScore = 0
18. currentDealerScore = 0
19. initialDealerScore = 0
20. bust = False
21. playGame = True
22. win = False
23. possibleDealerInitialScores = [1,2,3,4,5,6,7,8,9,10]
24. possiblePlayerScores = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21]
25. possibleHitValues = [True, False]
26. states = {}
27. **for** dis in possibleDealerInitialScores:
28. **for** ps in possiblePlayerScores:
29. **for** hv in possibleHitValues:
30. states[dis, ps, hv] = 0
31. #states = [possibleDealerInitialScores, possiblePlayerScores, possibleHitValues]
32. #[DealerInitialScore, PlayerCurrentScore, hit]
33. reward = states
34. stepsTaken = []
35. gamesPlayed = 0

38. def getCard(isFirstCard = False):
39. **if** (isFirstCard):
40. color = 2
41. **else**:
42. color = randint(1, 3)
43. number = randint(1,10)
44. **if** color == 1:
45. #print "red " + str(number)
46. number = -number
47. #if color == 2 or color == 3:
48. #print "black " + str(number)
49. **return** number


53. #Draw first card for both dealer and player
54. def initializeGame():
55. #print
56. #print ("Dealer first card: ")
57. global currentDealerScore
58. global initialDealerScore
59. currentDealerScore = getCard(True)
60. initialDealerScore = currentDealerScore
61. #print ("Score: " + str(currentDealerScore))
62. #print
63. #print "--------------PLAYER---------------------"
64. #print
65. #print ("Player first card: ")
66. global currentPlayerScore
67. currentPlayerScore = getCard(True)
68. #print ("Score: " + str(currentPlayerScore))
70. def endGame():
71. firstTimeDealer = True
72. firstTimePlayer = True
73. hit = "Y"
74. global currentPlayerScore
75. currentPlayerScore = 0
76. global initialDealerScore
77. initialDealerScore = 0
78. global currentDealerScore
79. currentDealerScore = 0
80. global win
81. win = False
82. global stepsTaken
83. stepsTaken = []
85. #For every action taken during a game, if the game ends in a win add 1 to that state.
87. **while** playGame == True:
88. initializeGame()
89. #hit = raw\_input("Hit: Y/N\t")[0].upper()
90. hit = random.choice(["Y", "N"])
91. stepsTaken.append([initialDealerScore, currentPlayerScore, hit=="Y"])
92. **while** hit == "Y":
93. global currentPlayerScore
94. global currentDealerScore
95. global initialDealerScore
96. currentPlayerScore = sum([currentPlayerScore, getCard()])
97. #print "Score: " + str(currentPlayerScore)
98. **if** currentPlayerScore > 21 or currentPlayerScore < 1:
99. #print "BUST"
100. bust = True
101. hit = "N"
102. win = False
103. **continue**
104. #hit = raw\_input("Hit: Y/N\t")[0].upper()
105. hit = random.choice(["Y", "N"])
106. stepsTaken.append([initialDealerScore, currentPlayerScore, hit=="Y"])

109. #print "-----------------------DEALER-------------------------"
110. #print "Score: " + str(currentDealerScore)
111. **while** currentDealerScore < 17 and currentPlayerScore in range (1, 22) and currentDealerScore in range(1,22):
112. currentDealerScore = sum([currentDealerScore, getCard()])
113. #print "Score: " + str(currentDealerScore)
114. **if** currentDealerScore > 21 or currentDealerScore < 1:
115. #print "BUST"
116. win = True
117. **continue**
118. #time.sleep(2)

121. **if** win == False and currentPlayerScore in range (currentDealerScore, 22):
122. win = True
123. **if** (win == True):
124. #print stepsTaken
125. **for** step in stepsTaken:
126. #print step
127. reward[step[0], step[1], step[2]] += 1
128. #print "YOU WON!"
129. **if** (win == False):
130. **for** step in stepsTaken:
131. #print step
132. reward[step[0], step[1], step[2]] += -1
133. #print "YOU LOST!"
134. endGame()
136. **if** (gamesPlayed == 1000000):
137. playGame = False
138. **if** (gamesPlayed < 1000000):
139. playGame = True#input("Play Again: Y/N\t")
140. **if** (gamesPlayed % 10000 == 0):
141. #print (str(gamesPlayed / 10000) + " percent complete ", end='\r')
142. sys.stdout.write("\r" + str(gamesPlayed/10000) + " percent complete")
143. sys.stdout.flush()
144. gamesPlayed += 1
146. print ("Thanks for Playing!")
147. sorted\_reward = sorted(reward.items(), key=operator.itemgetter(1))
148. **for** indReward in sorted\_reward:
149. print (str(indReward[0]) + " = " + str(reward[indReward[0]]))

152. #Gt = x + x2(v) + x3(v^2) + x4(v^3) ...
153. #Ss = Ss + Gt
154. #Ns = Ns + 1
155. #Vs = Ss/Ns
156. ##I'm confused as to how to implement the time-step to give correct reward values.
157. #I was not able to generate a graph or visualization

**Corrected Portion:**

To fix my solution to work correctly. I had to implement a couple of things that I had left out of the original.

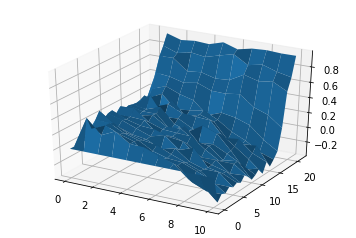
The first of these was the epsilon calculation, which helps determine the action of hit or stay for the player. It uses N0 constant and the number of occurrences of the state ignoring the action to calculate epsilon. Then uses that epsilon value to determine if the algorithm will choose a random action or if it will be greedy.

The second was the reward calculation using the number of appearances of this state N(S) and the actual calculation:

Reward = reward + ((1/N(S)) \* (return – reward))

The return here is 1, -1, or 0 for win, loss, and tie respectively.

Once this was done, I also needed to present the information in a visual form. The plot\_vf function plots a 3d grid of the reward values so that we can visualize the trends within the data.



Corrected Code: (Things that have changed are highlighted with a commented line of dashes, “-“ and a comment stating that it has changed and how it has changed. There were several changes. Many of which are small, such as adding a reward calculation for dealer and player ties)

1. from \_\_future\_\_ import print\_function
2. #!/usr/bin/env python2
3. # -\*- coding: utf-8 -\*-
4. """
5. Created on Tue Nov 14 10:07:17 2017
7. @author: rditljtd
8. """
10. from random import randint
11. import operator
12. import sys
13. import random
14. import time
15. import matplotlib.pyplot as plt
16. from mpl\_toolkits.mplot3d import Axes3D
18. hit = False
19. currentPlayerScore = 0
20. currentDealerScore = 0
21. initialDealerScore = 0
22. bust = False
23. playGame = True
24. win = False
26. #Begin------------------------------------------------------------------------------------------------------------
27. #Changed: commented out portion of code instantiating states. Instead doing it as a numpy array
28. #possibleDealerInitialScores = [1,2,3,4,5,6,7,8,9,10]
29. #possiblePlayerScores = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21]
30. #possibleHitValues = [True, False]
31. #states = {}
32. #for dis in possibleDealerInitialScores:
33. #    for ps in possiblePlayerScores:
34. #        for hv in possibleHitValues:
35. #            states[dis, ps, hv] = 0


39. #states = [possibleDealerInitialScores, possiblePlayerScores, possibleHitValues]
40. #[DealerInitialScore, PlayerCurrentScore, hit]
41. reward = np.zeros((11,22,2))
42. states = np.zeros(reward.shape)
43. #End------------------------------------------------------------------------------------------------------------
45. stepsTaken = []
46. gamesPlayed = 0

49. def getCard(isFirstCard = False):
50. **if** (isFirstCard):
51. color = 2
52. **else**:
53. color = randint(1, 3)
54. number = randint(1,10)
55. **if** color == 1:
56. #print "red " + str(number)
57. number = -number
58. #if color == 2 or color == 3:
59. #print "black " + str(number)
60. **return** number


64. #Draw first card for both dealer and player
65. def initializeGame():
67. global currentDealerScore
68. global initialDealerScore
69. currentDealerScore = getCard(True)
70. initialDealerScore = currentDealerScore
72. global currentPlayerScore
73. currentPlayerScore = getCard(True)
75. #reset variables
76. def endGame():
77. firstTimeDealer = True
78. firstTimePlayer = True
79. hit = "Y"
80. global currentPlayerScore
81. currentPlayerScore = 0
82. global initialDealerScore
83. initialDealerScore = 0
84. global currentDealerScore
85. currentDealerScore = 0
86. global win
87. win = False
88. global stepsTaken
89. stepsTaken = []
91. #Begin------------------------------------------------------------------------------------------------------------
92. #Changed: epsilonGreedy function added
93. def epsilonGreedy(s, Q, epsilon):
94. **if** random.random() < epsilon:
95. **return** random.choice(["Y", "N"])
96. **else**:
97. **return** "N" **if** Q[s[0]][s[1]][0] > Q[s[0]][s[1]][1] **else** "Y"
98. #End------------------------------------------------------------------------------------------------------------
100. #Begin------------------------------------------------------------------------------------------------------------
101. #Changed: getEpsilon Function added
102. def getEpsilon(state, states):
103. N0 = 100
104. NS = states[state[0]][state[1]][0] + states[state[0]][state[1]][1]
105. **return** N0 / (N0 + NS)
106. #End------------------------------------------------------------------------------------------------------------
108. #Begin------------------------------------------------------------------------------------------------------------
109. #Changed: added a function for plotting the data
110. def plot\_vf(Q):
111. #Plot the value function represented by Q
112. fig = plt.figure()
113. ax = fig.gca(projection='3d')
114. xx, yy = np.meshgrid(list(range(11)), list(range(22)))
115. ax.plot\_surface(xx, yy, np.max(Q, axis=2).T)
116. plt.show()
117. #End------------------------------------------------------------------------------------------------------------

120. #For every action taken during a game, if the game ends in a win add 1 to that state.
122. #MCControl
123. **while** playGame == True:
125. #Begin------------------------------------------------------------------------------------------------------------
126. #Changed: Added variables for epsilon
127. global epsilon
128. #End------------------------------------------------------------------------------------------------------------
130. initializeGame()
132. #Begin------------------------------------------------------------------------------------------------------------
133. #Changed: instantiate epsilon
134. epsilon = getEpsilon(state, states)
135. #End------------------------------------------------------------------------------------------------------------
137. #Begin------------------------------------------------------------------------------------------------------------
138. #Changed: changed the way the action is determined
139. hit = epsilonGreedy(state,states,epsilon)#random.choice(["Y", "N"])
140. state = (initialDealerScore, currentPlayerScore)
141. **if** (hit=="Y"):
142. action = 1
143. **else**:
144. action = 0
145. stepsTaken.append((state[0], state[1], action))
146. #End------------------------------------------------------------------------------------------------------------
148. **while** hit == "Y":
149. currentPlayerScore = sum([currentPlayerScore, getCard()])
150. **if** currentPlayerScore > 21 or currentPlayerScore < 1:
151. bust = True
152. hit = "N"
153. win = False
154. **continue**
156. #Begin------------------------------------------------------------------------------------------------------------
157. #Changed: the way the action is determined and adding state to states taken in this episode
158. hit = epsilonGreedy(state,states,epsilon)#random.choice(["Y", "N"])
159. **if** (hit == "Y"):
160. action = 1
161. **else**:
162. action = 0
163. state = (currentDealerScore, currentPlayerScore)
164. stepsTaken.append((state[0], state[1], action))
165. #End------------------------------------------------------------------------------------------------------------
167. **while** currentDealerScore < 17 and currentPlayerScore in range (1, 22) and currentDealerScore in range(1,22):
168. currentDealerScore = sum([currentDealerScore, getCard()])
169. **if** currentDealerScore > 21 or currentDealerScore < 1:
170. win = True
171. **continue**

174. **if** win == False and currentPlayerScore in range (currentDealerScore+1, 22):
175. win = True
177. #Begin------------------------------------------------------------------------------------------------------------
178. #Changed: added in method for ties
179. elif win == False and currentPlayerScore == currentDealerScore:
180. win = None
181. #Changed: added in reward and increment for ties
182. **if** (win == None):
183. **for** step in stepsTaken:
184. #Changed: to increment counter of state
185. states[step[0], step[1], step[2]] += 1
186. #Changed: to add reward for state
187. reward[step[0], step[1], step[2]] += (1/(1.0\*(states[step[0]][step[1]][step[2]])) \* (0 - reward[step[0]][step[1]][step[2]]))
188. #End------------------------------------------------------------------------------------------------------------
190. **if** (win == True):
191. **for** step in stepsTaken:
193. #Begin------------------------------------------------------------------------------------------------------------
194. #Changed: to increment counter of state
195. states[step[0]][step[1]][step[2]] += 1
196. #End------------------------------------------------------------------------------------------------------------
198. #Begin------------------------------------------------------------------------------------------------------------
199. #Changed: to add reward for state
200. reward[step[0]][step[1]][step[2]] += (1/(1.0\*(states[step[0]][step[1]][step[2]])) \* (1 - (reward[step[0], step[1], step[2]])))
201. #End------------------------------------------------------------------------------------------------------------
203. **if** (win == False):
204. **for** step in stepsTaken:
206. #Begin------------------------------------------------------------------------------------------------------------
207. #Changed: to increment counter of state
208. states[step[0]][step[1]][step[2]] += 1
209. #End------------------------------------------------------------------------------------------------------------
211. #Begin------------------------------------------------------------------------------------------------------------
212. #Changed: to add reward for state
213. reward[step[0]][step[1]][step[2]] += (1/(1.0\*(states[step[0]][step[1]][step[2]])) \* (-1 - (reward[step[0]][step[1]][step[2]])))
214. #End------------------------------------------------------------------------------------------------------------
216. endGame()
218. **if** (gamesPlayed == 1000000):#000000):
219. playGame = False
220. **if** (gamesPlayed < 1000000):#000000):
221. playGame = True
222. **if** (gamesPlayed % 10000 == 0):
223. sys.stdout.write("\r" + str(gamesPlayed/10000) + " percent complete")
224. sys.stdout.flush()
225. gamesPlayed += 1
227. #print (reward)
228. plot\_vf(reward)