Machine Learning

CSE 8673

Programming Assignment 4

**Student**: Anh Do

**NetID**: aqd14

1. What are the pros and cons of bootstrapping in Easy21?

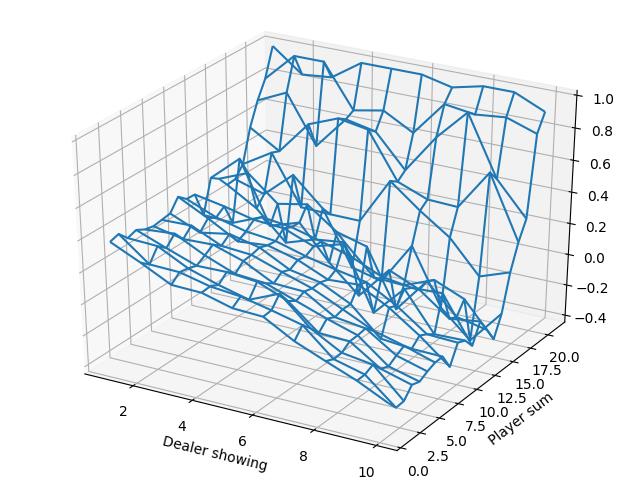
**Answer:** Bootstrapping has less variance, since it depends on the current reward and intermediate steps more than only the end result. However, there may be some bias, while Monte-Carlo is guaranteed to converge given enough examples.

1. Would you expect bootstrapping to help more in blackjack or Easy21? Why?

**Answer:** I expect bootstrapping to help more in blackjack. Bootstrap works by updating current state’s estimates based on the estimates successor states. Therefore, in order for bootstrapping to work well, the

Easy21 has red cards that can move your total sum backwards, so its episodes will last longer overall than blackjack. Therefore, bootstrapping will help more than in blackjack, where Monte-Carlo methods will work well since the episodes are short.

**Monte Carlo Control Plot for Easy21**



# Appendix

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**from** **mpl\_toolkits.mplot3d.axes3d** **import** Axes3D

**class** **Environment**(object):

**def** \_\_init\_\_(self):

self.new\_game()

**def** new\_game(self):

self.dealer\_first\_card = self.dealer\_hand = np.random.randint(1, 11)

self.player\_hand = np.random.randint(1, 11)

self.game\_end = False

**def** deal\_card(self):

color = np.random.randint(3)

value = np.random.randint(1, 11)

**if** color <= 1:

**return** value

**else**:

**return** -value

**def** observe(self):

*# return current state*

**return** [self.dealer\_first\_card, self.player\_hand]

**def** is\_terminal(self):

**return** self.game\_end

**def** step(self, action):

**if** self.is\_terminal():

self.new\_game()

**if** action == 1:

*# Player hits*

self.player\_hand += self.deal\_card()

**if** 1 < self.player\_hand < 22:

*# continue for another action, zero reward*

reward = 0

self.game\_end = False

**else**:

*# Player goes bust*

self.player\_hand = 0

reward = -1

self.game\_end = True

**else**:

*# Player sticks*

**while** self.dealer\_hand < 17:

*# Dealer hits*

self.dealer\_hand += self.deal\_card()

**if** 1 < self.dealer\_hand < 22:

**continue**

**else**:

*# Dealer goes bust*

self.dealer\_hand = 0

reward = 1

self.game\_end = True

**return** [self.dealer\_first\_card, self.player\_hand], reward

*# Dealer sticks*

**if** self.dealer\_hand > self.player\_hand:

reward = -1

**elif** self.dealer\_hand == self.player\_hand:

reward = 0

**else**:

reward = 1

self.game\_end = True

**return** [self.dealer\_first\_card, self.player\_hand], reward

**def** epsilon\_greedy(N0, N, Q, x, y):

*# epsilon-greedy exploration*

e = N0 / (N0 + np.sum(N[x - 1, y - 1, :]))

**if** np.random.uniform(0, 1) > e:

action = np.argmax(Q[x - 1, y - 1, :])

**else**:

action = np.random.randint(0, 2)

**return** action

**def** monte\_carlo(max\_episode, discount, N0):

*# Initialization*

Q = np.zeros([10, 21, 2])

N = np.zeros([10, 21, 2])

**for** i **in** range(max\_episode):

*# initial a new episode*

episode = Environment()

*# the initial state of the episode*

x, y = episode.observe()

*# sample until terminal*

history = []

**while** **not** episode.is\_terminal():

*# decide action*

action = epsilon\_greedy(N0, N, Q, x, y)

N[x - 1, y - 1, action] += 1

*# run one step*

(state, reward) = episode.step(action)

history.append(([x, y], action, reward))

[x, y] = state

*# calculate return Gt for each state in this episode*

Gt = 0

**for** j, (state, action, reward) **in** enumerate(reversed(history)):

[x, y] = state

alpha = 1.0 / N[x - 1, y - 1, action]

Gt = discount \* Gt + reward

Q[x - 1, y - 1, action] += alpha \* (Gt - Q[x - 1, y - 1, action])

**return** Q

**def** plot\_value\_function(V):

*# plot value function*

x = np.arange(1, 11)

y = np.arange(1, 22)

xs, ys = np.meshgrid(x, y)

fig = plt.figure()

ax = Axes3D(fig)

ax.set\_ylabel('Player sum')

ax.set\_xlabel('Dealer showing')

ax.plot\_wireframe(xs, ys, V.T, rstride=1, cstride=1)

plt.show()

**def** get\_value\_function(Q):

**return** np.amax(Q, axis=2)

**def** main():

N0 = 100

discount = 1 *# gamma*

max\_episode = 100000

Q = monte\_carlo(max\_episode, discount, N0)

*# optimal value function*

V\_max = get\_value\_function(Q)

plot\_value\_function(V\_max)

**if** \_\_name\_\_ == "\_\_main\_\_":

main()