Shubham Thakral, 2017193

RL HW3

Q1.

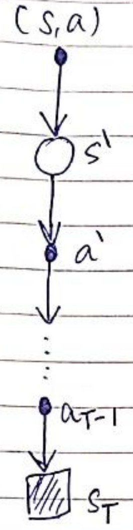
```
Initialise:
    For all s, pi(s) belongs to A(s) randomly
    For all s in states and a in A(s), q(s,a) belongs to R (real numbers) randomly
    **For all s in states and a in A(s), num_r(s,a)=0. These are the number of times the particular (s,a) pair has been sampled in the calculation of average return**
Loop for each episode:
    Choose $s_0$ from all states and $a_0$ from A(s) randomly such that probability of choosing all possible pairs is >0
    Generate an episode starting from $s_o$,$a_0$ following policy pi as defined above:
    $s_0,a_0,r_1,...,s_{T-1},a_{T-1},r_T$
    G=0
    Loop for each step of episode t=T-1,T-2,...,0:
        G=gamma*G+$r_{t+1}$
        Unless the pair $s_t$,$a_t$ appears in $s_0,a_0,s_1,a_1,...,s_{T-1},a_{T-1}$:
            **q($s_t$,$a_t$)=q($s_t$,$a_t$)*num_r($s_t$,$a_t$)**
            **q($s_t$,$a_t$)+=G**
            **num_r($s_t$,$a_t$)+=1**
            **q($s_t$,$a_t$)=q($s_t$,$a_t$)/num_r($s_t$,$a_t$)**
            pi($s_t$)=$\text{argmax}_a$ q($s_t$,a)
```

Explanation - Although the above code is pretty much self explanatory, the changes made by me are highlighted in bold. I've added a variable to keep track of the number of returns on which the average q is calculated. Then when I add the next return to it, I multiply the q with number of times its averaged to get the value of returns, and then add my return to this. Then I increment the number of returns taken by 1 and divide the q with it to get the new averaged value.

Shubham Thakral, 2017193

Q2 and Q3:

Q2: Backup diagram for Monte Carlo estimation of q_π.



Q3: $q_\pi(s,a) = E_\pi[G_t \mid S_t = s, A_t = a]$

We have $E_b[G_t \mid S_t = s, A_t = a]$.

$$= E_b[R_{t+1} + R_{t+2} + \dots + R_T \mid S_t = s, A_t = a]$$

$$= \sum_{(r_{t+1},\dots,r_T)} (r_{t+1} + r_{t+2} + \dots + r_T) \cdot P_b[R_{t+1} = r_{t+1}, \dots R_T = r_T \mid S_t = s, A_t = a]$$

$$= \sum_{(a_{t+1},\dots,a_T)} \sum_{(r_{t+1},\dots r_T)} (r_{t+1} + \dots + r_T) \cdot P_b[R_{t+1} = r_{t+1}, A_{t+1} = a_{t+1}, R_{t+2} = r_{t+2}, \dots, A_{T-1} = a_{T-1}, R_T = r_t \mid S_t = s, A_t = a]$$

Shubham Thakral, 2017193

Now,

$$P_b[R_{t+1}=r_{t+1}, A_{t+1}=a_{t+1}, R_{t+2}=r_{t+2}, \ldots, A_{T-1}=a_{T-1}, R_T=r_T \mid S_t=s, A_t=a]$$

$$= \sum_{s} P_b[R_{t+1}=r_{t+1}, S_{t+1}=s_{t+1}, A_{t+1}=a_{t+1}, \ldots, A_{T-1}=a_{T-1}, R_T=r_T, S_T=s_T \mid S_t=s, A_t=a]$$

$$= \sum_{s} P_b[R_T=r_T, S_T=s_T \mid S_{T-1}=s_{T-1}, A_{T-1}=a_{T-1}] \times P_b[R_{t+1}=r_{t+1}, S_{t+1}=s_{t+1}, A_{t+1}=a_{t+1}, \ldots, R_{T-1}=r_{T-1}, S_{T-1}=s_{T-1}, A_{T-1}=a_{T-1} \mid S_t=s, A_t=a]$$

$$= \sum_{s} P[R_T, S_T \mid S_{T-1}, A_{T-1}] \cdot P_b[A_{T-1} \mid S_{T-1}] \cdot P_b[R_{t+1}=r_{t+1} \ldots R_{T-1}, S_{T-1} \mid S_t=s, A_t=a]$$

$$= \sum_{s} P[R_T, S_T \mid S_{T-1}, A_{T-1}] \cdot P_b[A_{T-1} \mid S_{T-1}] \cdot P[R_{T-1}, S_{T-1} \mid S_{T-2}, A_{T-2}] \cdot P_b[A_{T-2} \mid S_{T-2}] \ldots P_b[S_{t+1}, R_{t+1} \mid S_t=s, A_t=a]$$

$$= \sum_{s} P[S_{t+1}, R_{t+1} \mid S_t, A_t] \cdot b(A_{t+1} \mid S_{t+1}) \cdot P[S_{t+2}, R_{t+2} \mid S_{t+1}, A_{t+1}] \times \ldots \times b(A_{T-1} \mid S_{T-1}) \cdot P[R_T, S_T \mid S_{T-1}, A_{T-1}]$$

$$= \sum_{s} \prod_{k=t+1}^{T-1} b(A_k \mid S_k) \prod_{k=t}^{T-1} P[S_{k+1}, R_{k+1} \mid S_k, A_k]$$

$$\therefore E_b[G_t \mid S_t, A_t] = \sum_{\{(a_k, s_{k+1}, r_{k+1}) : t+1 \le k \le T-1\}} (r_t + \ldots + r_T) \prod_{k=t+1}^{T-1} b(A_k \mid S_k) \cdot \prod_{k=t}^{T-1} P[S_{k+1}, R_{k+1} \mid S_k, A_k]$$

We want $E_\pi[G_t \mid S_t, A_t, \pi] = E_b[\rho_{t:T-1} G_t \mid S_t, A_t, b]$

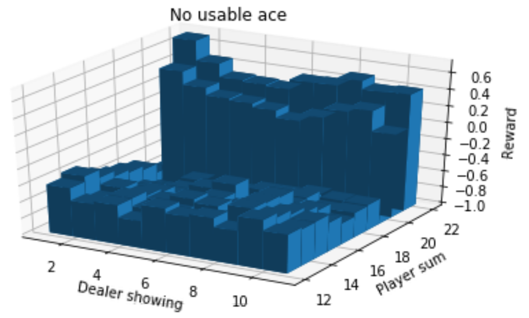
So
$$\rho_{t:T-1} = \frac{\prod_{k=t+1}^{T-1} \pi(a_k \mid s_k) \cdot \cancel{\prod_{k=t}^{T-1} P(s_k, r_{k+1} \mid s_k, a_k)}}{\prod_{k=t+1}^{T-1} b(a_k \mid s_k) \cdot \cancel{\prod_{k=t}^{T-1} P(s_k, r_{k+1} \mid s_k, a_k)}}$$

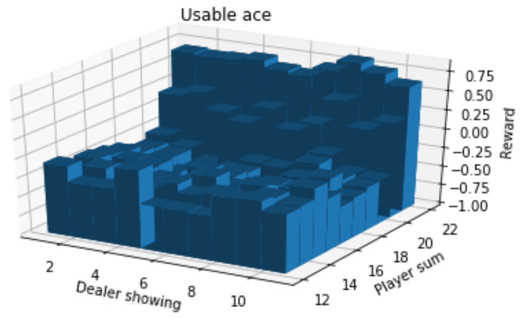
$$\boxed{\rho_{t:T-1} = \prod_{k=t+1}^{T-1} \frac{\pi(a_k \mid s_k)}{b(a_k \mid s_k)}}$$

Shubham Thakral, 2017193

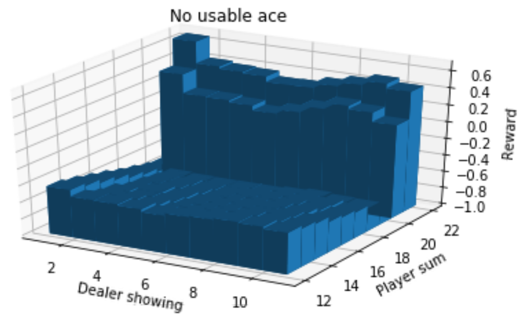
Q4. Done in HW3_Q4.py file

Fig. 5.1
With 10000 iterations:





With 500000 iterations:



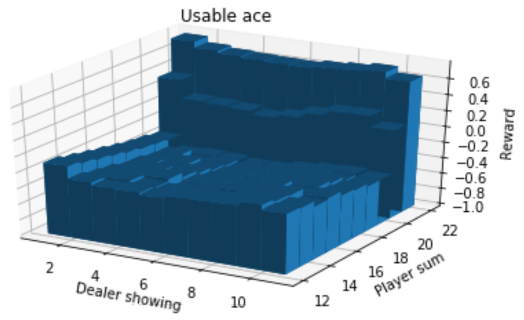
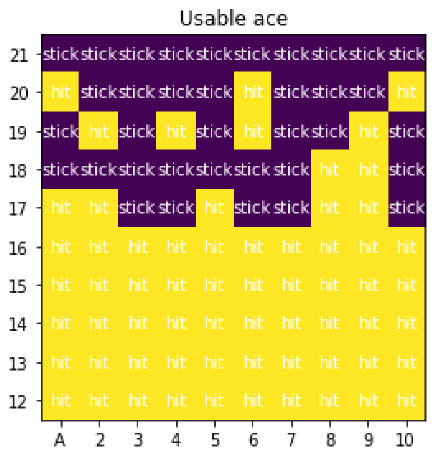
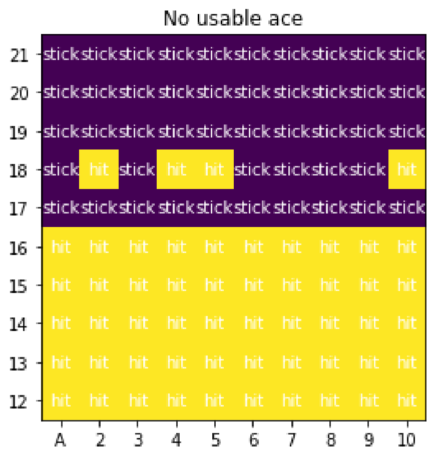


Fig 5.2:



Usable ace

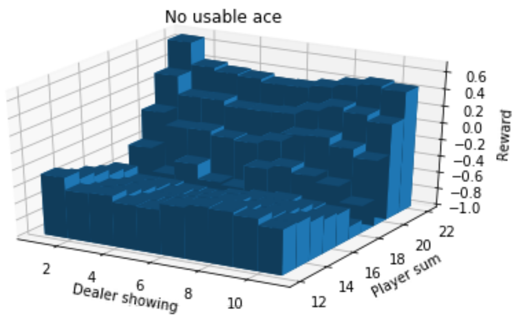
	A	2	3	4	5	6	7	8	9	10
21	stick	stick	stick	stick	stick	stick	stick	stick	stick	stick
20	hit	stick	stick	stick	stick	hit	stick	stick	stick	hit
19	stick	hit	stick	hit	stick	hit	stick	stick	hit	stick
18	stick	stick	stick	stick	stick	stick	stick	hit	hit	stick
17	hit	hit	stick	stick	hit	stick	stick	hit	hit	stick
16	hit	hit	hit	hit	hit	hit	hit	hit	hit	hit
15	hit	hit	hit	hit	hit	hit	hit	hit	hit	hit
14	hit	hit	hit	hit	hit	hit	hit	hit	hit	hit
13	hit	hit	hit	hit	hit	hit	hit	hit	hit	hit
12	hit	hit	hit	hit	hit	hit	hit	hit	hit	hit



No usable ace

	A	2	3	4	5	6	7	8	9	10
21	stick	stick	stick	stick	stick	stick	stick	stick	stick	stick
20	stick	stick	stick	stick	stick	stick	stick	stick	stick	stick
19	stick	stick	stick	stick	stick	stick	stick	stick	stick	stick
18	stick	hit	stick	hit	hit	stick	stick	stick	stick	hit
17	stick	stick	stick	stick	stick	stick	stick	stick	stick	stick
16	hit	hit	hit	hit	hit	hit	hit	hit	hit	hit
15	hit	hit	hit	hit	hit	hit	hit	hit	hit	hit
14	hit	hit	hit	hit	hit	hit	hit	hit	hit	hit
13	hit	hit	hit	hit	hit	hit	hit	hit	hit	hit
12	hit	hit	hit	hit	hit	hit	hit	hit	hit	hit

Shubham Thakral, 2017193



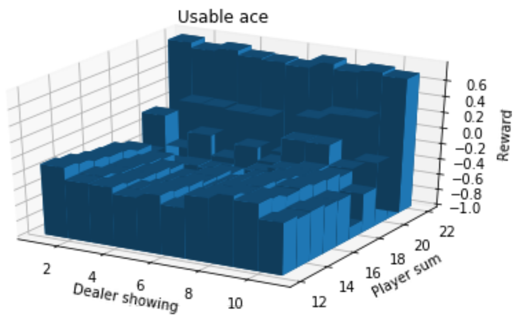
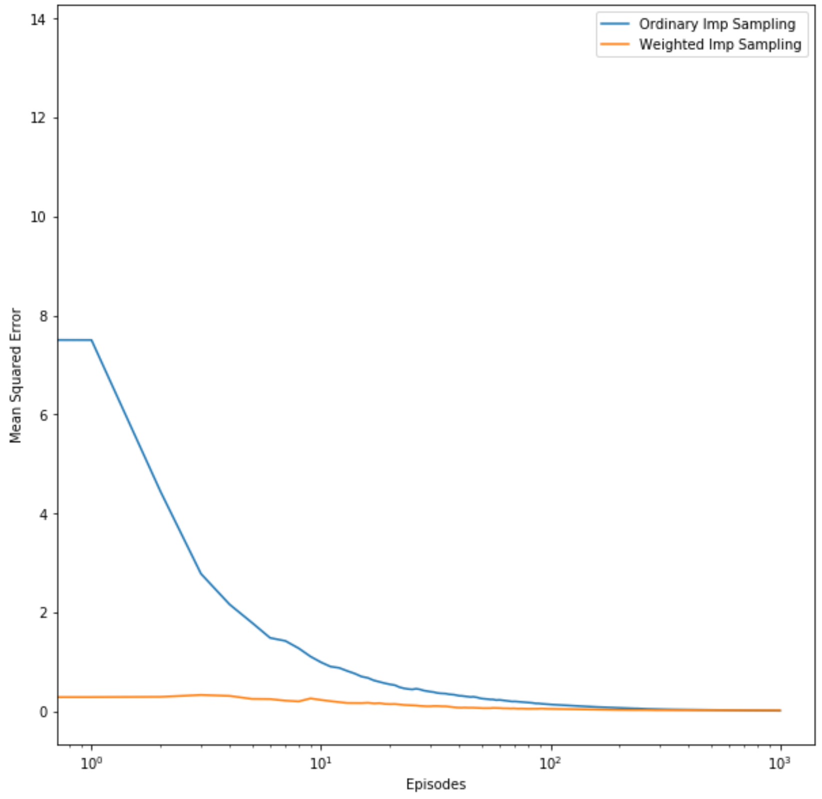


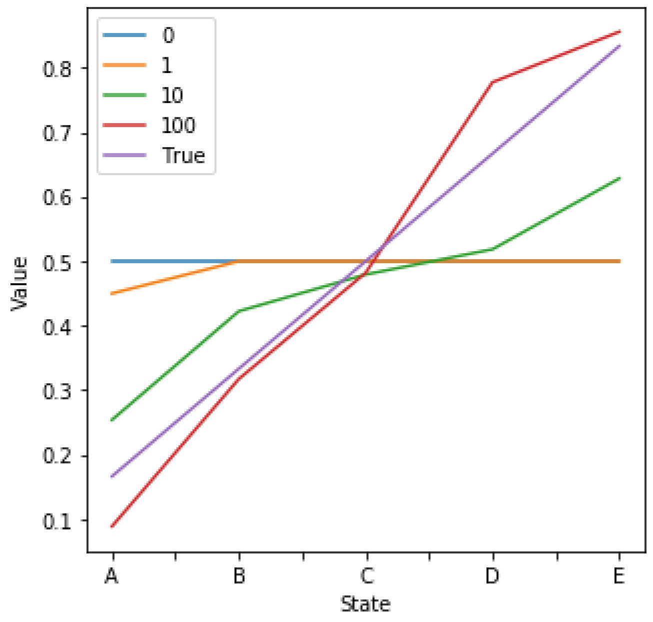
Fig 5.3:

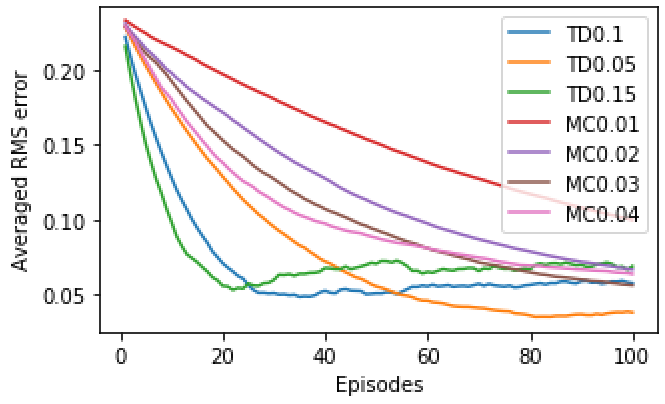


Q5. In such a scenario where I had a lot of driving experience from work to home and then my workplace changes to a nearby building such that some path is still common (say some highway and path after that is common) then I will greatly benefit from my bootstrapped past experience since some of the path is already same. Hence TD update will be better than an MC update on average as TD will use the bootstrapped value function which will remain the same for some of the states as those states occur in both new and old scenario. The same can also happen in the original problem if our initial estimates of value functions are very close to the optimal value functions. This will make convergence faster.

Shubham Thakral, 2017193

Q6. Done in HW3_Q6.py file



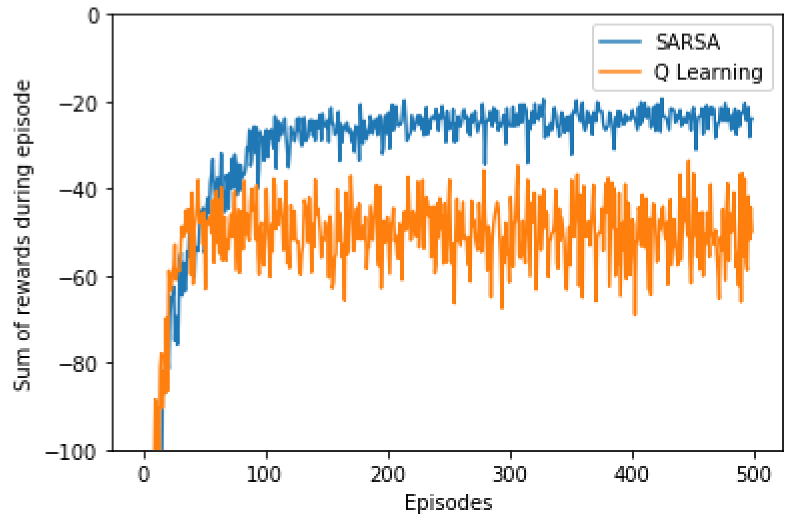


Ex 6.3: In the first episode only V(A) changed. This means that we went to terminal state from state A. All other states aren't updated because TD0 looks at the next reward and then bootstraps. Since for all s except terminal states, V(s) start equal at 0.5, hence for all other states, the TD0 error is 0 (as TD error = r+gamma*v(s')-v(s) and taking gamma=1, v(s')=v(s) for all states for which s,s' aren't terminal as all v(s) except v(terminal) start same ie 0.5 -> and r = 0 (for each step, r=0, we only get final r at terminal) so TD error comes out 0) and for the state A : we get reward 0 and our initial estimate was 0.5 and we go into terminal state whose value estimate was 0 so error is 0.5 using above formula. Now we have alpha = 0.1 so the actual update would be of 0.1*0.5=0.05 (decrease of 0.05). This is what we see from the plot that the value of A is decreased by 0.05 and it becomes 0.45. Similarly, the other possible update that could have happened is of E ie its increase in value by 0.05 as it will get reward=1 on reaching terminal state in episode 1.

Ex 6.4: Alpha denotes the step size or learning rate. It tells by how much amount should we update the value function at each step. If a larger alpha is chosen, this means that we will make high variance updates. These types of updates will be affected more or biased more on the particular steps taken and rewards achieved. On the other hand choosing a small alpha means that we will update value function slowly thus mitigating the bias at some level since each update only contributes very small. Choosing small alpha will also make learning slow.

Ex6.5: For high alphas, as explained in Ex. 6.4 above, the bias is more ie update depends on particular steps more. Hence the error increased due to bias and is increased more at higher alphas as they update more on each step and hence update depends heavily on what action had been taken and what reward was got. I think it will always occur irrespective of the initialisation because the updates are stochastic (as actions are stochastic).

Q7. Done in HW3_Q7.py file



Analysis -> Q learning rewards are more varied than those of SARSA. This is because since our policy is epsilon greedy hence with Q learning, even if we are following the optimal path, we still fell into the cliff because actions are chosen according to epsilon greedy and even if we get it once in a while, our reward gets decreased by 100. On the other hand SARSA's path is very far from cliff so even if its not optimal, it has very less chance of falling into cliff even by epsilon greedy. Hence it does not vary much. Had we chosen a perfectly greedy policy at the end or decreased epsilon with time, then both SARSA and Q learning would be found to converge to optimal policy.

Q8. If action selection is greedy, even then Q learning isn't exactly the same as SARSA. This is because in SARSA, we chose the next action (a') in current step only when we haven't updated the value function but in Q learning we chose the next action after updating the value function. So even if the weight updates will be similar as both will choose the same maximising action greedily but for updating value function and SARSA will continue with this chosen action but Q learning will again take greedy action from the updated policy for the next step. Hence the action selection will be different.