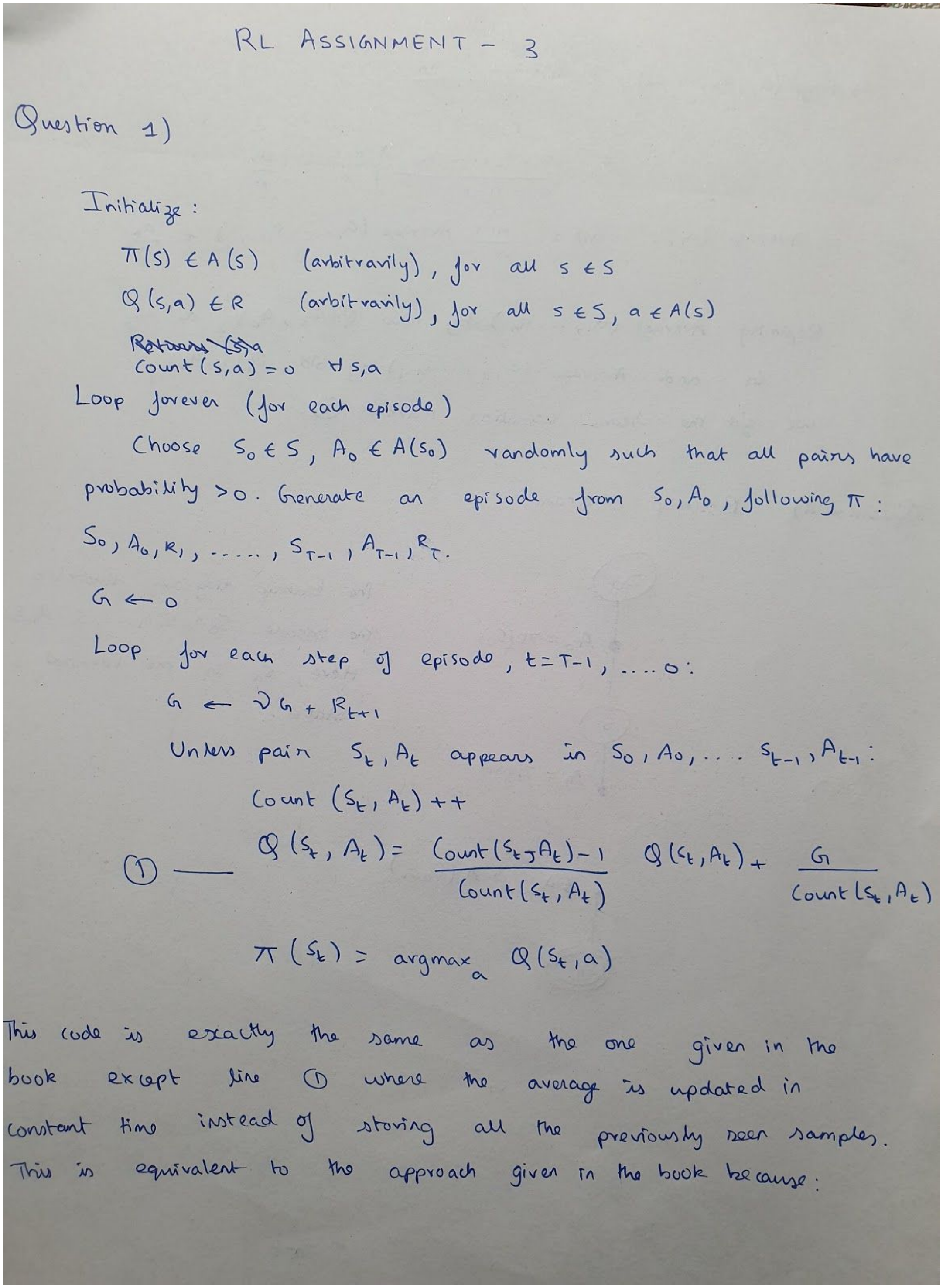
RL Assignment 3

Author

Nihesh Anderson (2016059)



RL ASSIGNMENT - 3

Question 1)

```
Initialize:
    $\pi(s) \in A(s)$   (arbitrarily), for all $s \in S$
    $Q(s,a) \in R$   (arbitrarily), for all $s \in S$, $a \in A(s)$
    ~~Returns (s)~~ a
    $count(s,a) = 0 \quad \forall s,a$
Loop forever (for each episode)
    Choose $S_0 \in S$, $A_0 \in A(S_0)$ randomly such that all pairs have
probability $> 0$. Generate an episode from $S_0, A_0$, following $\pi$:
$S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$.
    $G \leftarrow 0$
    Loop for each step of episode, $t = T-1, \ldots 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        Unless pair $S_t, A_t$ appears in $S_0, A_0, \ldots S_{t-1}, A_{t-1}$:
            $Count(S_t, A_t)$++
(1) ——      $Q(S_t, A_t) = \frac{Count(S_t, A_t) - 1}{Count(S_t, A_t)} Q(S_t, A_t) + \frac{G}{Count(S_t, A_t)}$
            $\pi(S_t) = \text{argmax}_a \; Q(S_t, a)$
```

This code is exactly the same as the one given in the book except line (1) where the average is updated in constant time instead of storing all the previously seen samples. This is equivalent to the approach given in the book because:

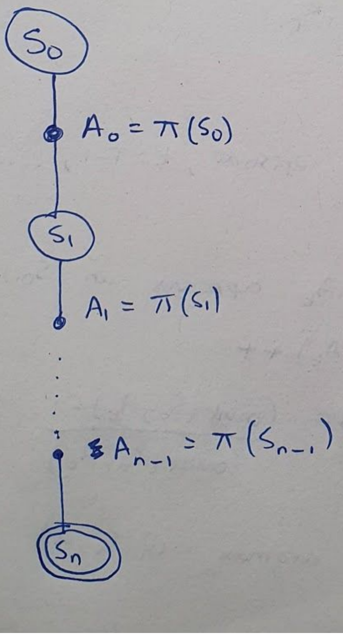
$$\text{Average}(a_1, a_2, \ldots a_n) = \frac{a_1 + a_2 + \ldots a_n}{n}$$

$$= \left(\frac{a_1 + a_2 + \ldots a_{n-1}}{n-1}\right)\frac{n-1}{n} + \frac{a_n}{n}$$

$$\Rightarrow \text{Average}(a_1, \ldots a_n) = \frac{n-1}{n}\,\text{Average}(a_1, \ldots a_{n-1}) + \frac{a_n}{n}$$

Replaing Average $(a_1, \ldots a_n)$ by new $Q(S_t, A_t)$, a_n by G and Average $(a_1, \ldots a_{n-1})$ by old $Q(S_t, A_t)$, we get the same equation marked ①.

Question 2) Backup diagram:



The backup diagram illustrates the episode $S_0 A_0 S_1 A_1 \ldots S_{n-1} A_{n-1} S_n$. Here, S_n is the terminal state.

Q3)

$$Q(s,a) = E_{\pi}\left[R_{t+1} + \gamma \cancel{Q} \sum_{a' \in A(s')} \pi(a'|s') Q(s',a')\right]$$

$$= \sum_{\text{actions}} \sum_{\text{rewards}} \sum_{\text{states}} G_t \; P\left[S_{t+1}, A_{t+1}, R_{t+1} \ldots\ldots \mid S_t, A_t\right]$$

derived in clas for v(s) →

$$= \sum_{\text{actions}} \sum_{\text{rewardl}} \sum_{\text{states}} G_t \prod_{i=t+1}^{T-1} b \frac{(A_i|S_i)}{\pi(A_i|S_i)} \times \text{other common terms.}$$

$\Rightarrow$ Importance sampling ratio $= \displaystyle\prod_{i=t+1}^{T-1} \frac{b(A_i|S_i)}{\pi(A_i|S_i)}$

Note that the only difference between V(s) & Q(s, a) is that the sampling ratio product starts from $i = t+1$, not t! This is because in case of Q(s,a), A_t is fixed.

$$\Rightarrow Q(s,a) = \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t+1:T(t)-1} \, G_t}{\sum_{t \in \mathcal{T}(s)} \rho_{t+1:T(t)-1}}$$

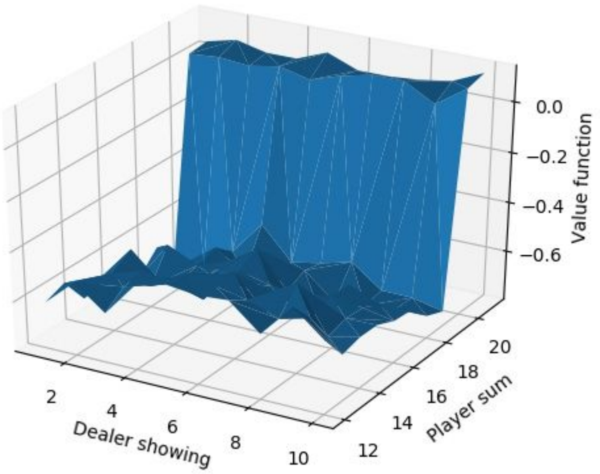
Q5) Suppose you have a lot of experience driving home from work. When you move to a new parking lot, but enter highway at the same place, we can assert that the estimates from highway are reasonably good, in case of TD 0. Therefore, this information can be reused and converge to the actual estimates for parking lot. However, in case of monte carlo, the entire episode is regenerated and the randomness after entering the highway is going to delay convergence. However, in the original scenario, both mote carlo and TD (0) might be equally good.

Problem 4

Blackjack game was implemented and the following value function was obtained for no ace and ace states. It looks the same as that obtained in the book.

Fig 5.1

No usable ace plot for policy that hits when current value < 20



Usable ace plot for policy that hits when current value < 20

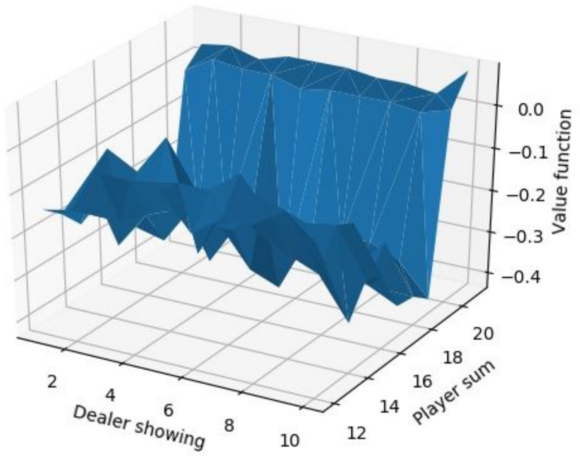
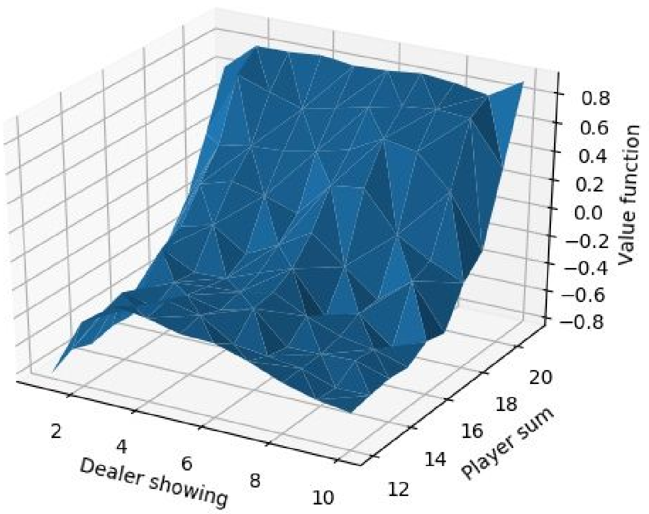
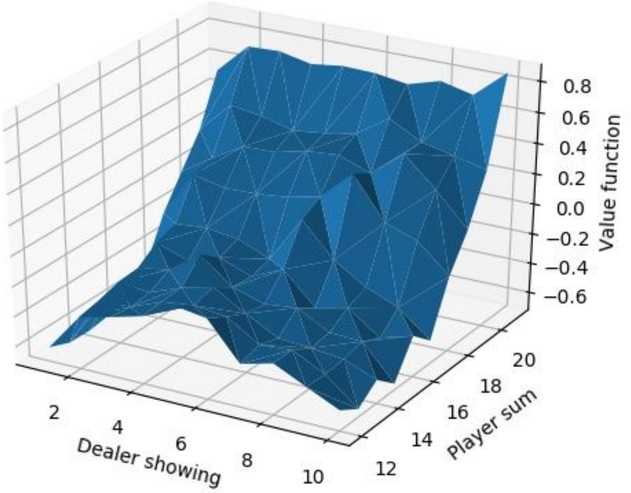


Fig 5.2 at 100000 epochs
No ace:

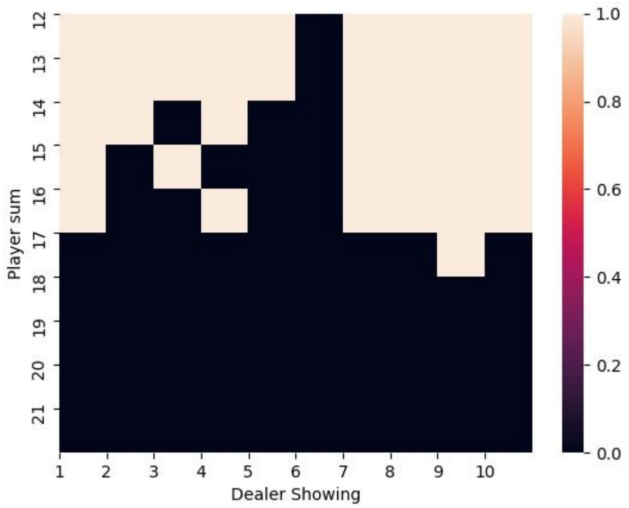


Usable ace:

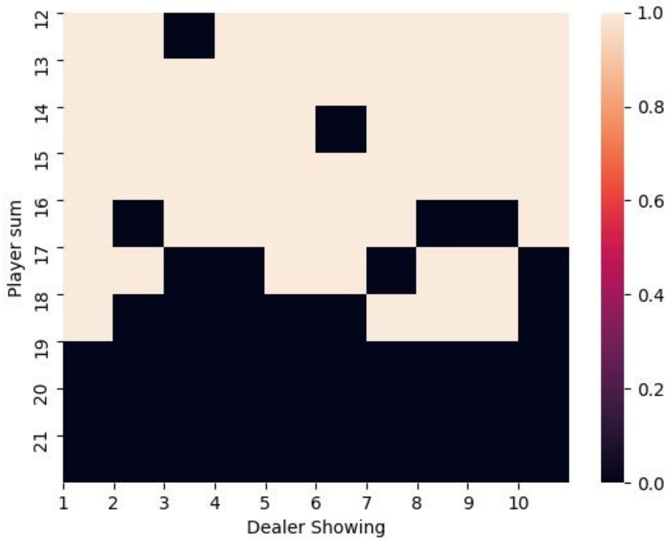


(Here, black denotes Stick and white denotes hit. Note that the y axis is increasing downwards)

No usable ace:

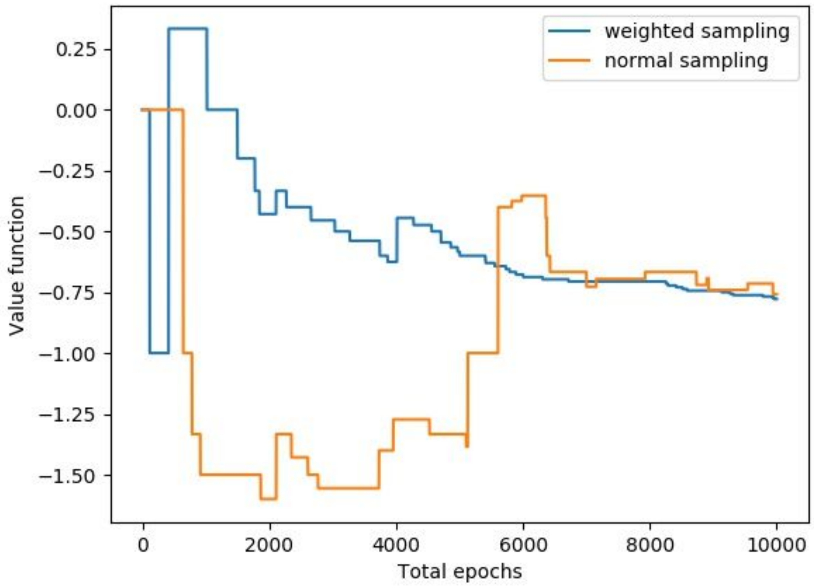


Usable ace:



There is mild noise due to low exploration in those areas.

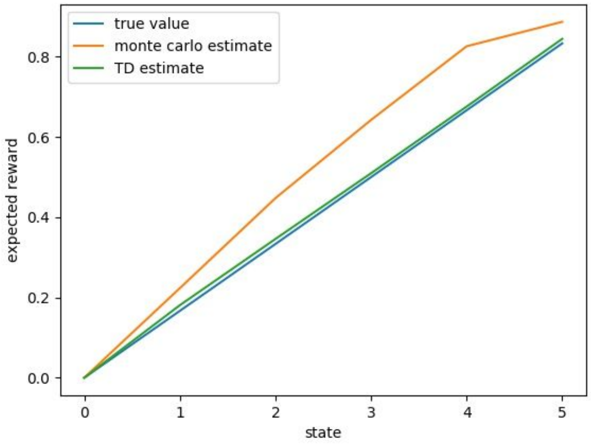
Fig 5.3



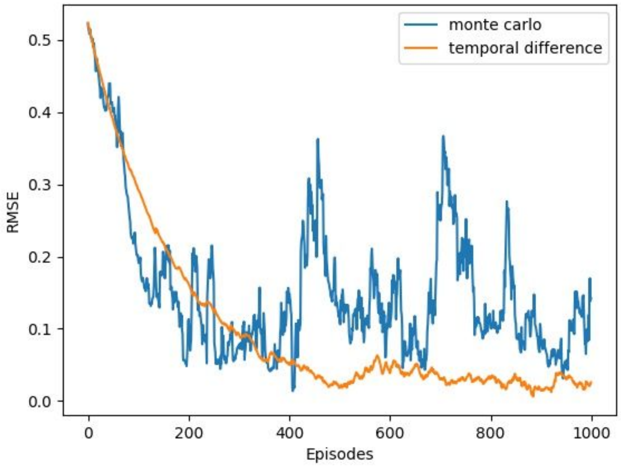
This is a plot of the value function against total epochs for a particular state for weighted sampling and normal sampling strategies. It can be clearly seen that both approaches converge to the same value, however, normal sampling is a lot more slower. This is because normal sampling is a high variance estimator as compared to weighted sampling.

Problem 6

In this problem, it can be observed that TD(0) reports values very close to the truth however, monte carlo is way off due to high values of alpha. The following is the obtained value for every state.



RMSE error vs training steps



Monte carlo oscillates a lot as expected due to high value of alpha and its high variance estimates. Otherwise, it looks pretty much similar to what was obtained in the book.

Ex 6.3

Initially, V(A) was set to 0.5. However, upon playing from A, the environment might have thrown us into a final state, giving a reward of 0. Therefore, TD algorithm would have updated V(a) as (1 - 0.1) * 0.5 + 0.1 * 0 = 0.45 (-0.05 change) which is the only change that would have happened as for all the other states s, V(S) = V(s') for all next states and immediate reward is 0.

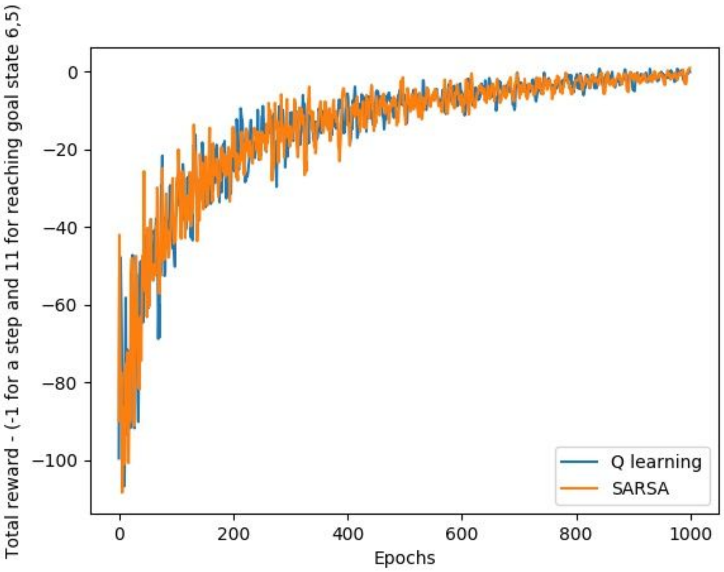
Ex 6.4

Yes, the conclusion will be affected by the choice of alpha that was used. In this graph, they used a reasonably high value of alpha. TD was able to learn well using this value, hence it converged faster. On the other hand, monte carlo has a high variance therefore such high values of alpha leads to oscillations and instability. For very small values of alpha, we can expect to get much better converge, however, at the cost of time.

Ex 6.5

The RMS error graph dips at a certain point and then starts to increase because of the way the value function was initialised. It won't occur unless the value function is initialised such that the value hits the optimal value, but diverges a bit because the other values didn't converge yet. Also, it may be possible that for higher values of alpha, the agent is getting biased by the recent episodes that it saw, therefore diverging slightly from the point of minima.

Problem 7



Both q learning and sarsa were implemented in this question. Epsilon greedy policy was used in both q learning and sarsa and it turned out that the reward behaviour for both the algorithms were the same, as expected. The reason for the same has been explained in the next question.

Problem 8

Yes, if the control policy for both Q learning and SARSA is the greedy policy, both the algorithms will be making the same action selections and weight updates, given that the initial value function is the same. This is because, suppose if both the algorithms start at a state S_0 and pick an Action A_0, and environment throws it into state S_1, Q learning will pick the maximum

Q value corresponding to state 1, which is equivalent to picking the greedy action in case of SARSA. Therefore, the updates in case of Q Learning and SARSA are the same. Q Learning will perform differently if its behaviour policy is not greedy.