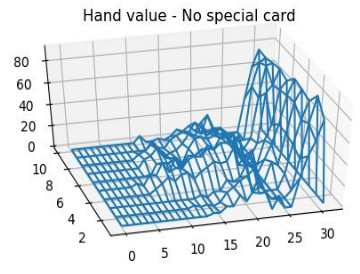
RL Assignment 1 Report

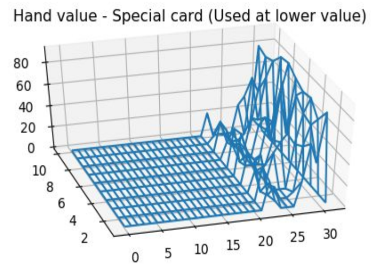
Friday, 09.04.20XX

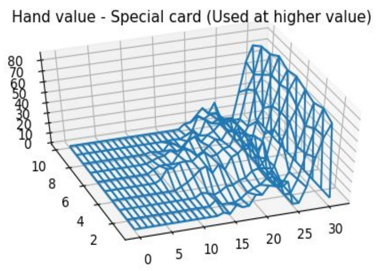
Policy Evaluation

Monte Carlo

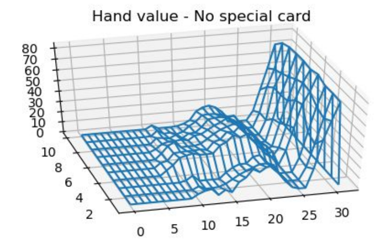
First Visit

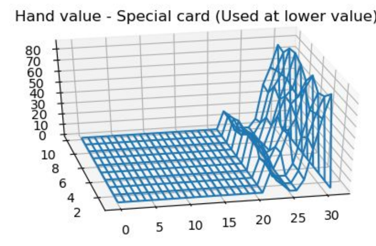


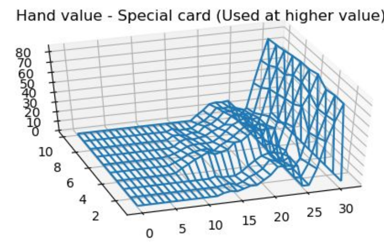




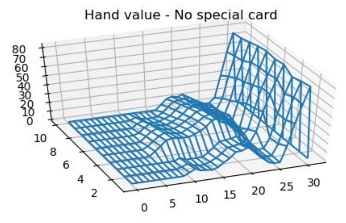
10K episodes

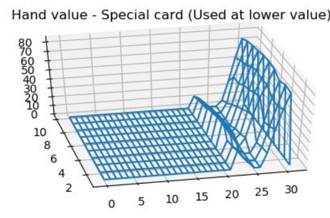


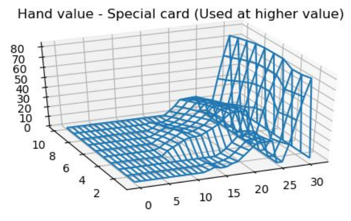




1 Million Episodes

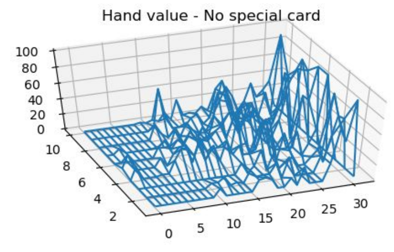


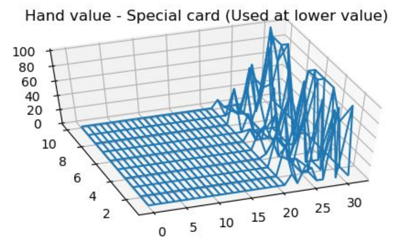


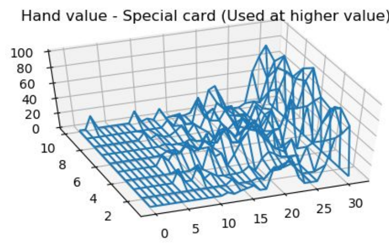


10 Million Episodes

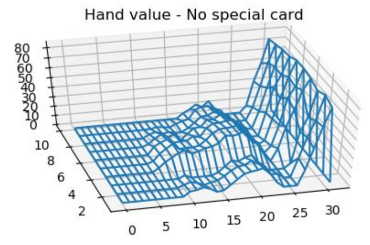
Every Visit

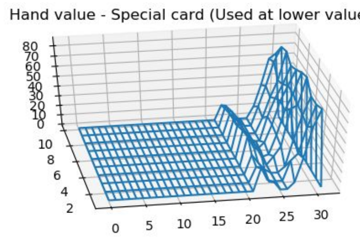


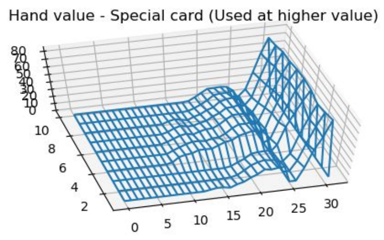




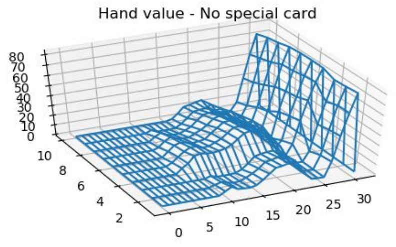
10K episodes

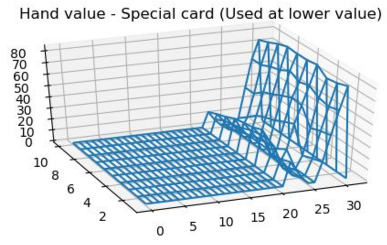


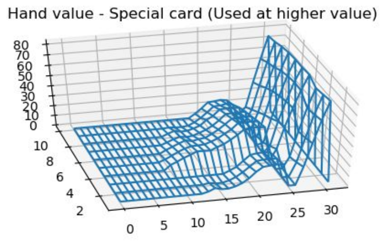




1 Million Episodes

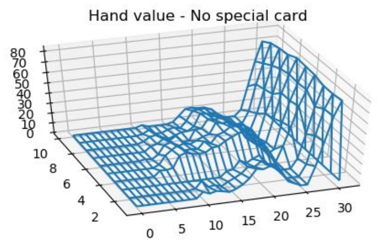


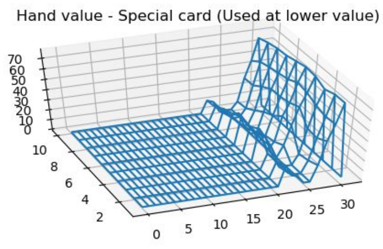


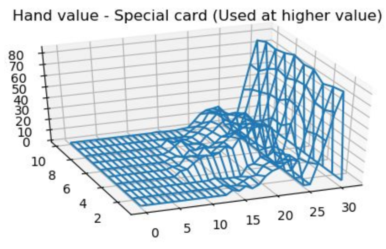


10 Million Episodes

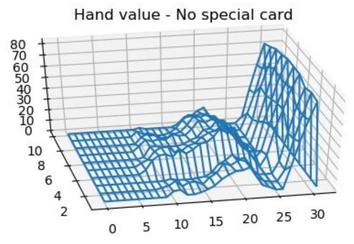
TD(K)

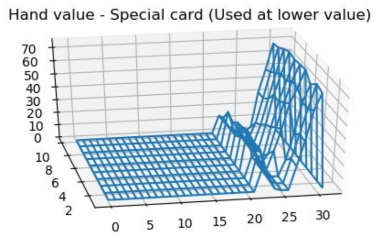


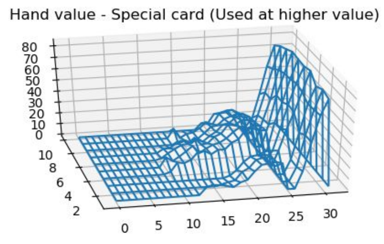




K = 20





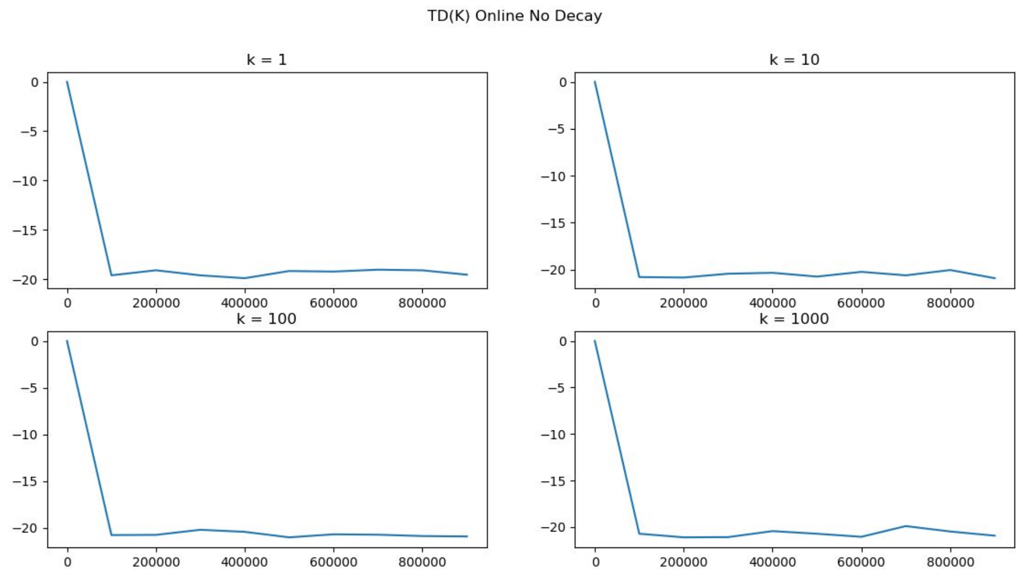


K = 1000

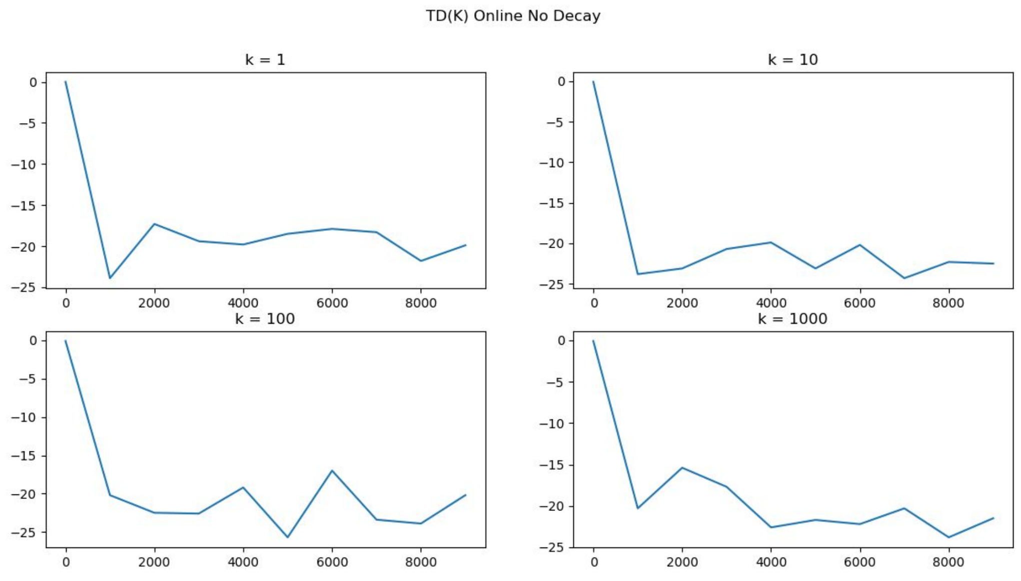
All the algorithms ultimately learn the same value function but every visit learns it the fastest due to more number of updates and TD(K) with k = 1000 can be seen as emulating MC while with k = 20 it is more like TD

Average Reward while Training

TD(K) No decay

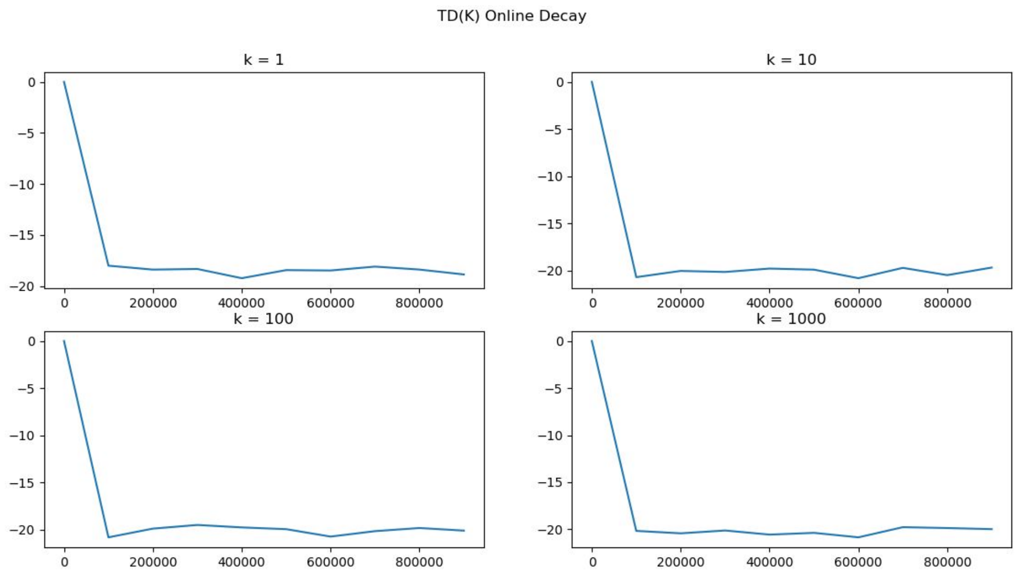


Trained upto million episodes with average rewards taken every 100k episodes

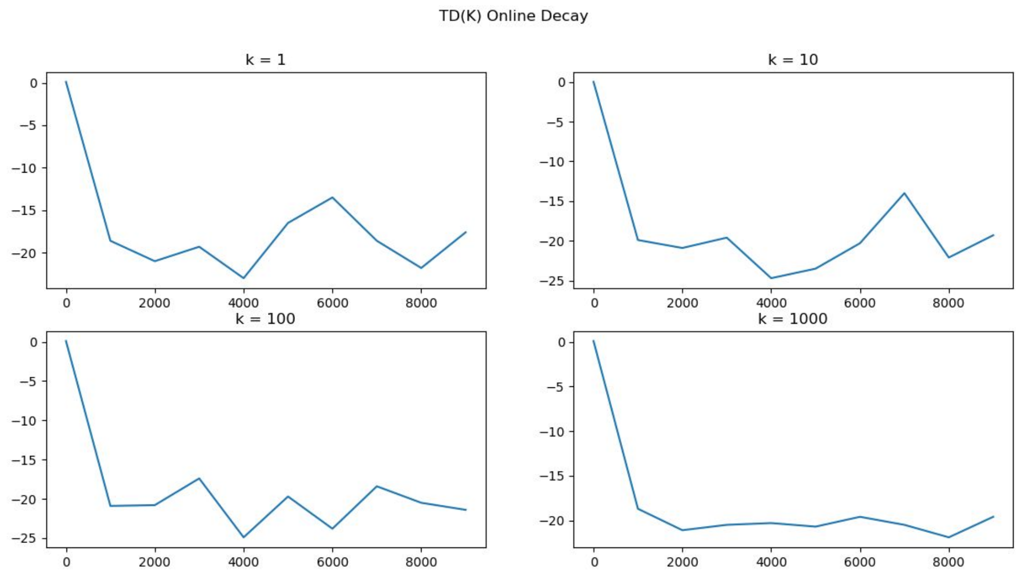


Trained upto 10k episodes with average rewards taken every 1k episodes

TD (K) with Decay

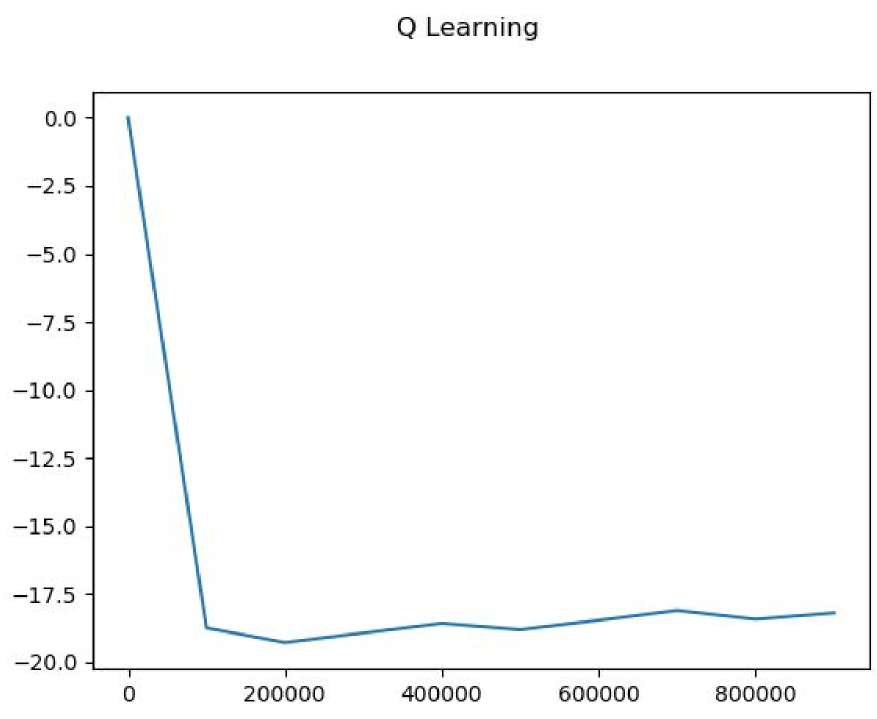


Trained upto million episodes with average rewards taken every 100k episodes

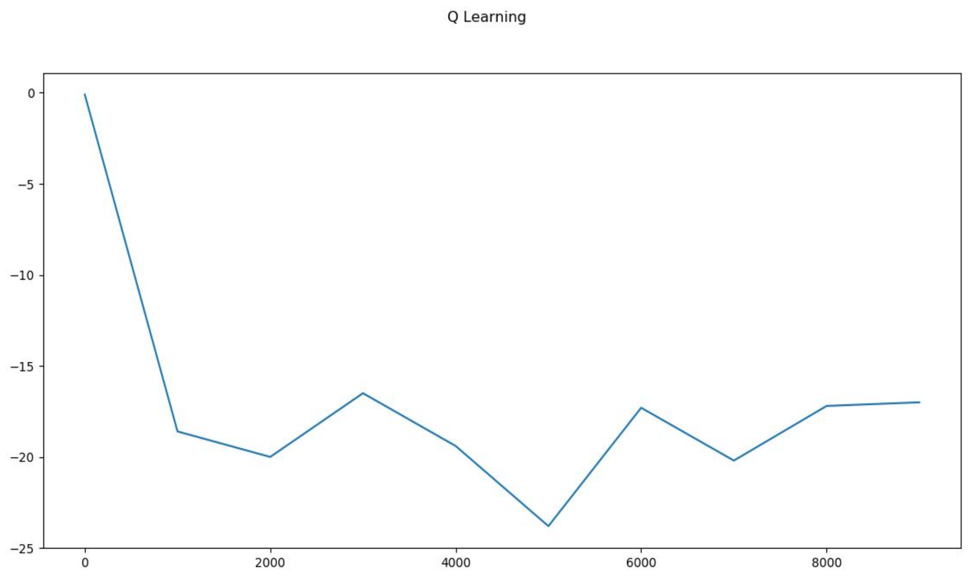


Trained upto 10k episodes with average rewards taken every 1k episodes

Q Learning



Trained upto a million episodes with average rewards taken every 100k episodes

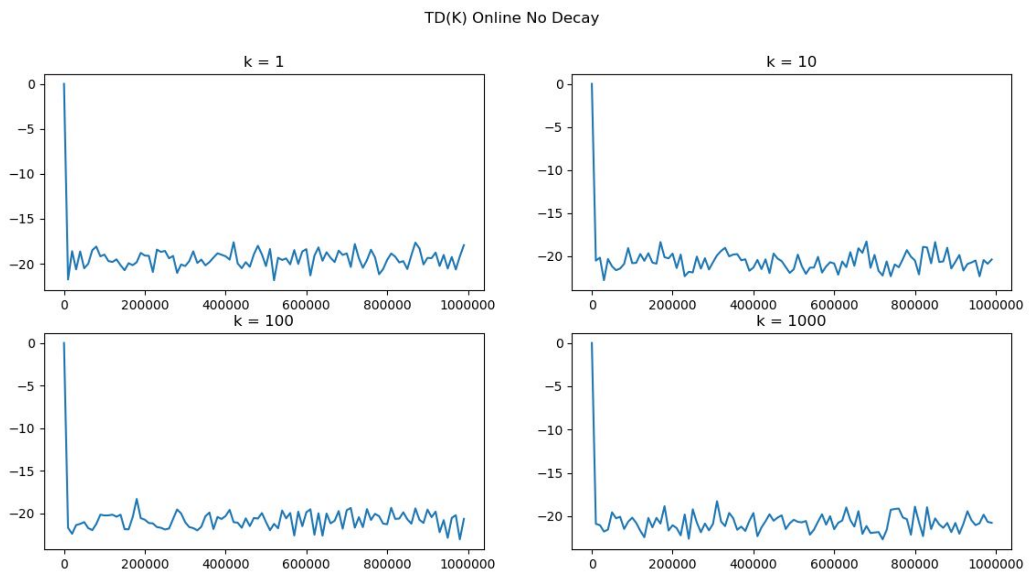


Trained upto 10k episodes with average rewards taken every 1k episodes

Here we can see that in the lesser training case the rewards are very varying and that is because of two things firstly the algorithm isn't really trained much throughout the 10k episodes and plus the initial distribution of cards has a major say when the rewards are averaged over only 1k episodes.

In one million case the algorithm is sufficiently trained even in the first 100k episodes and hence the reward is almost constant throughout in all the three algorithms. And also due to 100k being a large number we can assume that the initial distribution of cards will be almost the same in all the reward batches.

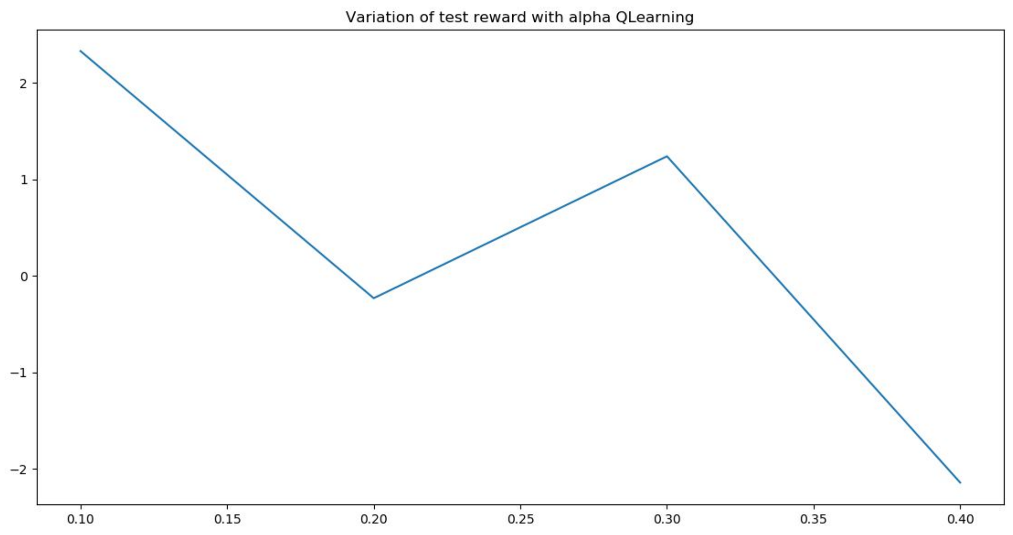
To find a properly increasing reward graph we need to pick a scale such that towards the end the algorithm is sufficiently trained and in the beginning it is not. For this I picked a scale of 1 million episodes and calculated reward over every 10k samples but it causes high variance due to card distribution as below



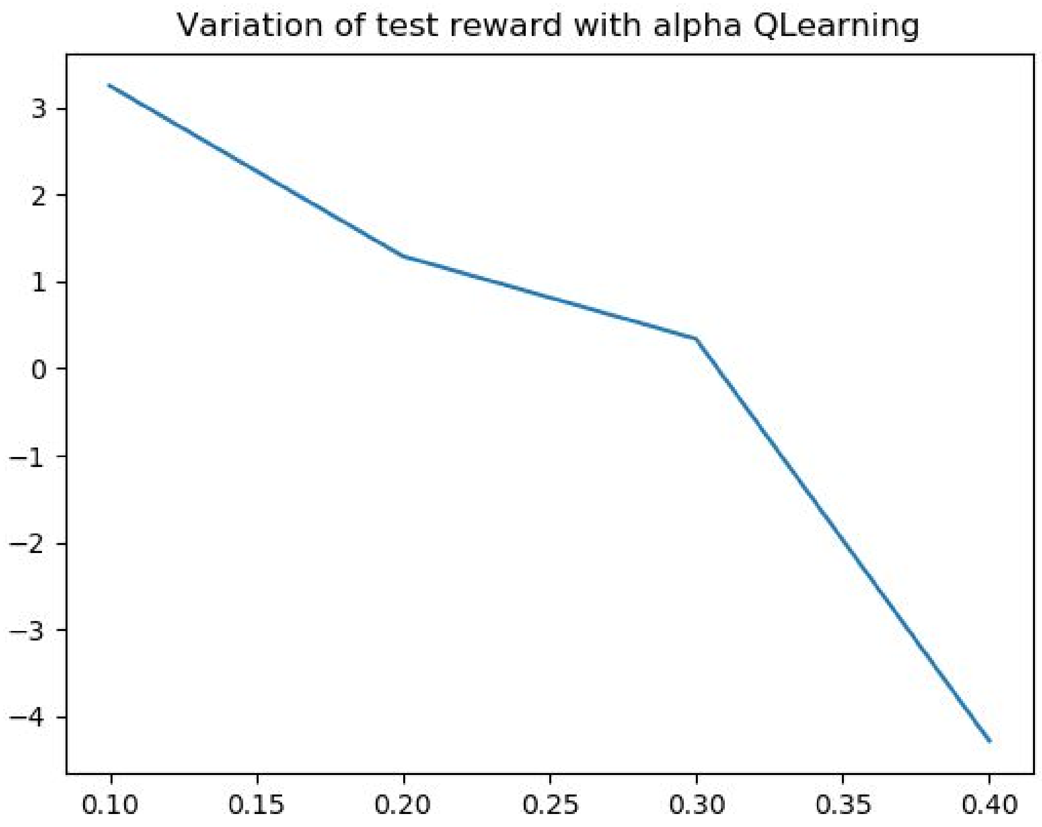
Testing the policy after training with different alphas

Q Learning

I ran the process multiple times to check if the results are coherent



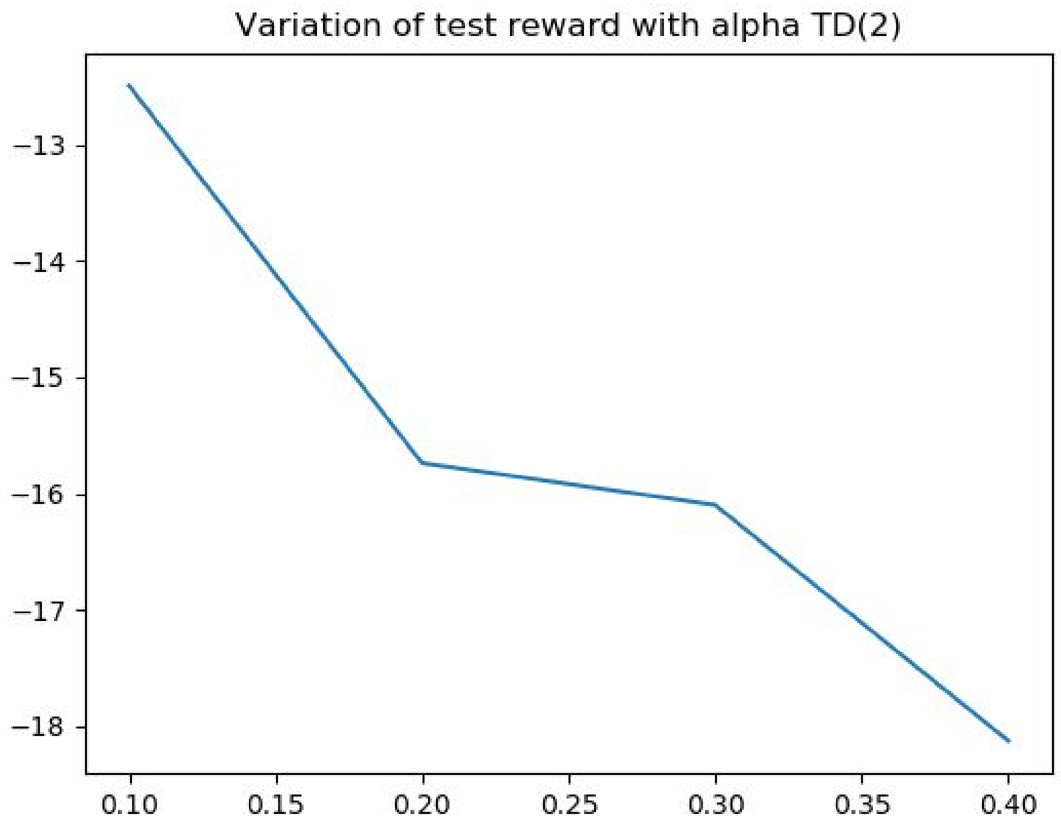
Simulation 1 - Trained over 100k episodes



Simulation 2 - Trained over 100k episodes

They were nearly the same and best performance was occuring for alpha = 0.1 and as we moved towards 0.4 the policy started to deteriorate that is probably because the value function couldn't properly converge ever with higher values of alpha. But it also didn't diverge because that would have given very bad average rewards, so the value function must have been oscillating constantly. To check if this is the case with other algorithms as well I ran it with TD(K) online.

TD(K) No decay

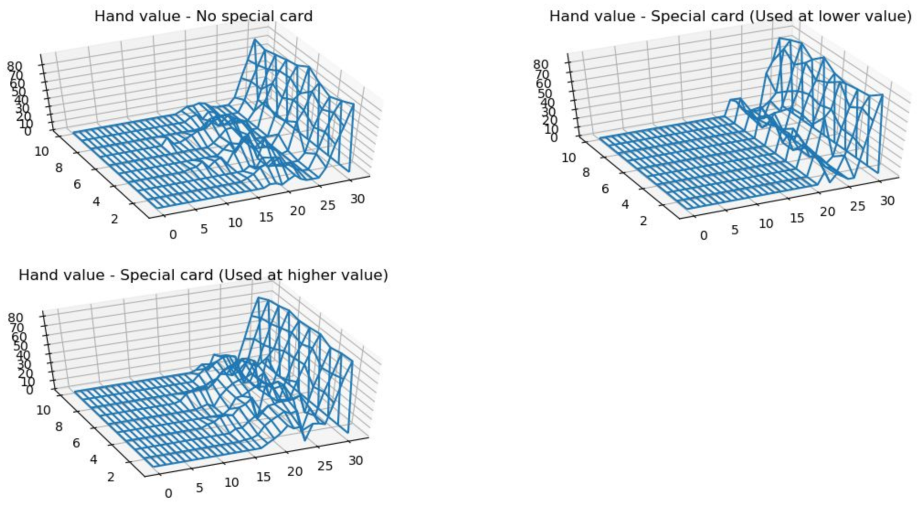


Simulation 3 - Trained over 100k episodes

Here also the results were nearly the same

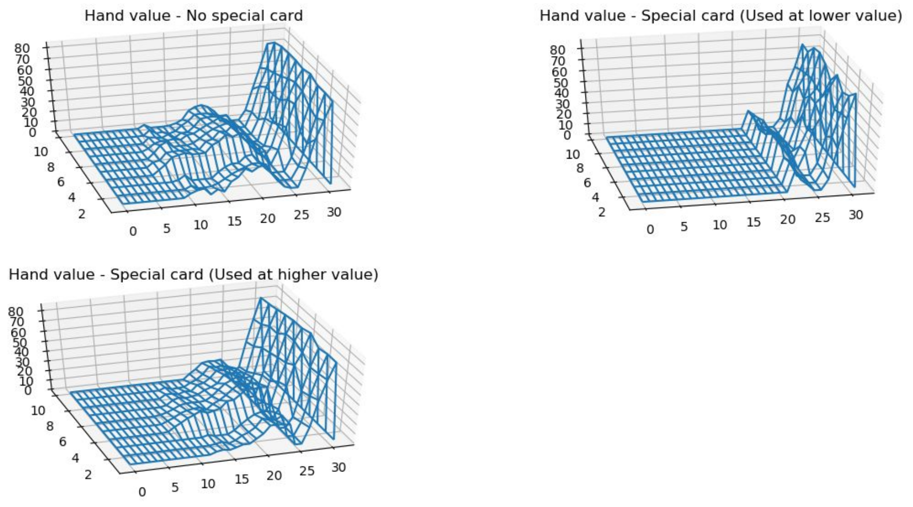
TD(λ)

Value function of policy obtained by TD(λ) is as follows :



Trained over 1 million episodes and evaluated using a million episodes (Monte Carlo)

While the original policy’s evaluation was:



‘Stick above 25’ policy’s evaluation using MC

As it can be seen TD learns a policy which also optimizes value functions of states where user hand value is around 15 and dealer hand value is 0-2, while the original policy is poor at that. The learnt policy is also better when the hand value is near about 25 while the actual policy has a sharp dip in value function due to sudden change of action.