CS698R: Deep Reinforcement Learning

Assignment #1

Name: Samarth Varma
Roll NO.: 180655

Solution to Problem 1: Multi-armed Bandits

1.
   - For $\alpha = 0$ and $\beta = 0$, the return for every episode was 0. The number of episodes in this case was 10.
   - For $\alpha = 1$ and $\beta = 1$, the return for every episode was 1. The number of episodes in this case was 10.
   - For $\alpha = 1$ and $\beta = 0$, the return for 10 episodes were 0,0,1,0,1,0,0,1,1 while for $\alpha = 0$ and $\beta = 1$, the return for 10 episodes were the complete opposite, 1,1,0,1,0,1,1,0,0
   - For $\alpha = 0.5$ and $\beta = 0.5$, for 10000 episodes, the return was 1 for 4945 episodes and 0 for the rest.

   Therefore, it could be concluded that the implementation for 2 Armed Bernoulli Bandit is correct as for both hyperparameters to be 0, the return should be 0 regardless of action and for the hyperparameters to be 1, the return should be 1 regardless of action. Therefore, when $\alpha = 1$ and $\beta = 0$, the return should be 1 when the action corresponding to α is taken, therefore when $\beta = 1$ and $\alpha = 0$, the actions corresponding to β should give a return of 1, which means the returns for episodes should be completely opposite to the prior case which it did have.

   For $\alpha = 0.5$ and $\beta = 0.5$, the probability of getting a reward of 1 is 0.5 and for over 10000 episodes, the number of episodes with a reward of 1 was 4945, which is roughly half.

   Therefore, the 2 Armed Bernoulli Bandit Environment is working perfectly.

2. For episode length of 10, the following results were inferred. The mean of the gaussian distributions are zero, the SD is varied.

   - for sd = 0.0, all the returns are 0
   - for sd = 0.2, the returns for each episodes are 0.047619212051000205, 0.2163970453493141, 0.44704934473669644, 0.7176077999120426, -0.2950287349666657, 0.6939616337788412,0.06174863310045048,0.27866484247161627, 0.16788008157154302,0.2794869284237169, 2.6153867864285556.
   - for sd higher than 0.2, the returns of the corresponding episodes are higher in magnitude in comparison for a sd of lower value.

     Therefore, the 10 Armed Gaussian Bandit Environment is working perfectly.

3. All the training algorithms are working and tested it manually. Refer to the Readme file of the repository for the function declarations.

4. Attached Plot of Average Reward Vs Time Step for 1000 Time Steps in Figure 1. The final result comes out to be a noisy distribution with UCB dominating on the upper side and pure exploration to be on the bottom side of the noise.

   If we follow the centre of the noise distribution of the different training algorithms, the average reward for the centre of Epsilon-decay and softmax is increasing while all the others are kind of stagnant.

   Pure Exploitation has the least average reward out of all the training algorithm that might be averaging around 0.5

   The hyperparameters taken are c = 0.2 for UCB, Epsilon = 0.5 for Epsilon Greedy, Epsilon = 1 to 0.01 for Epsilon-Decay (Linear) and exponential decay (from 100 to 0.01) for temperature in softmax equation.

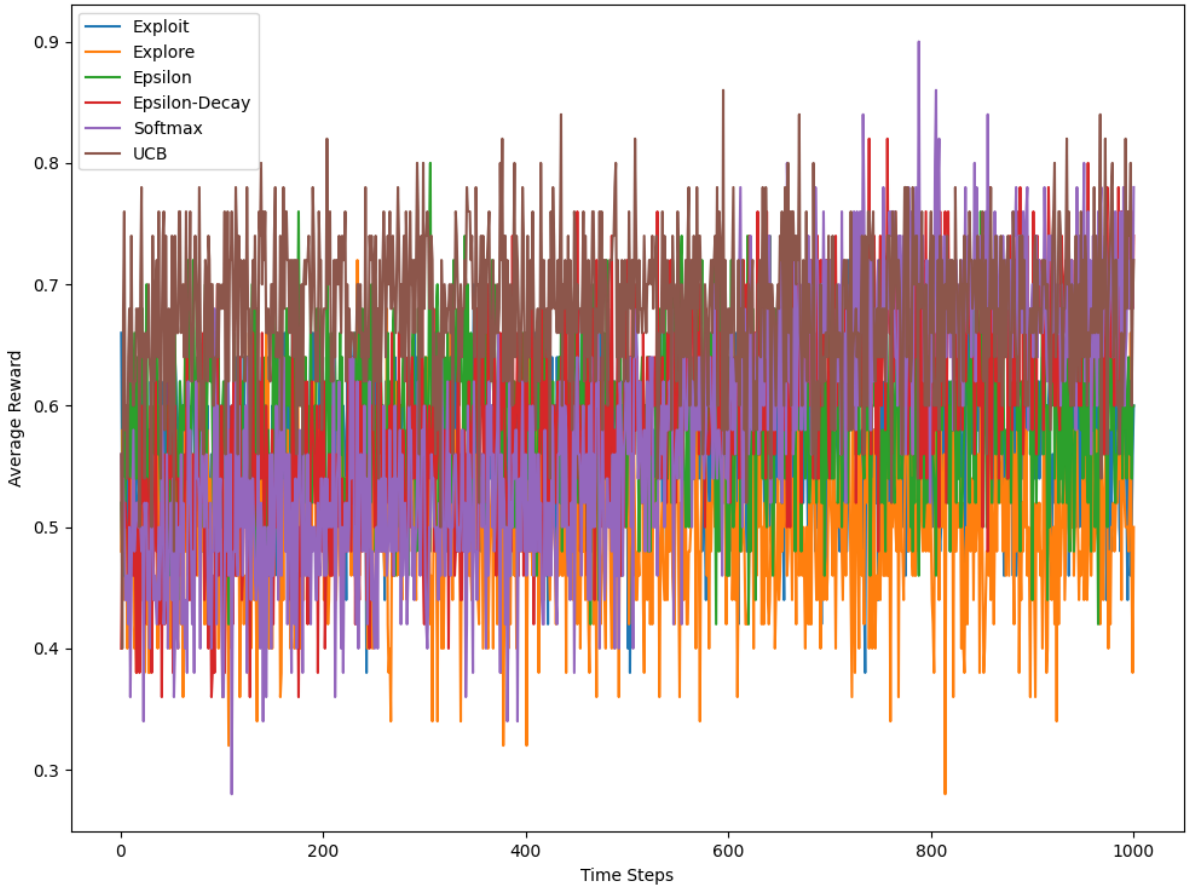

Figure 1: Q4: Average Reward vs Time Steps in 2 Arm Bernoulli Bandit

5. Attached Plot of Average Reward Vs Time Step for 1000 Time Steps in Figure 2 for 10 Arm Gaussian Bandit. The final result comes out to be a noisy distribution but is more spaced apart in comparison to 2 Arm Bernoulli Bandit. UCB is dominating in the beginning while Epsilon-Decay and Softmax catch up to it at the end.

   Pure Exploration has the least average reward out of all the algorithms which is more apparently here since the environment is more random in comparison to 2 arm bernoulli bandit.

   The hyperparameters taken are c = 0.2 for UCB, Epsilon = 0.5 for Epsilon Greedy, Epsilon = 1 to 0.01 for Epsilon-Decay (Linear) and exponential decay (from 100 to 0.01) for temperature in softmax equation.

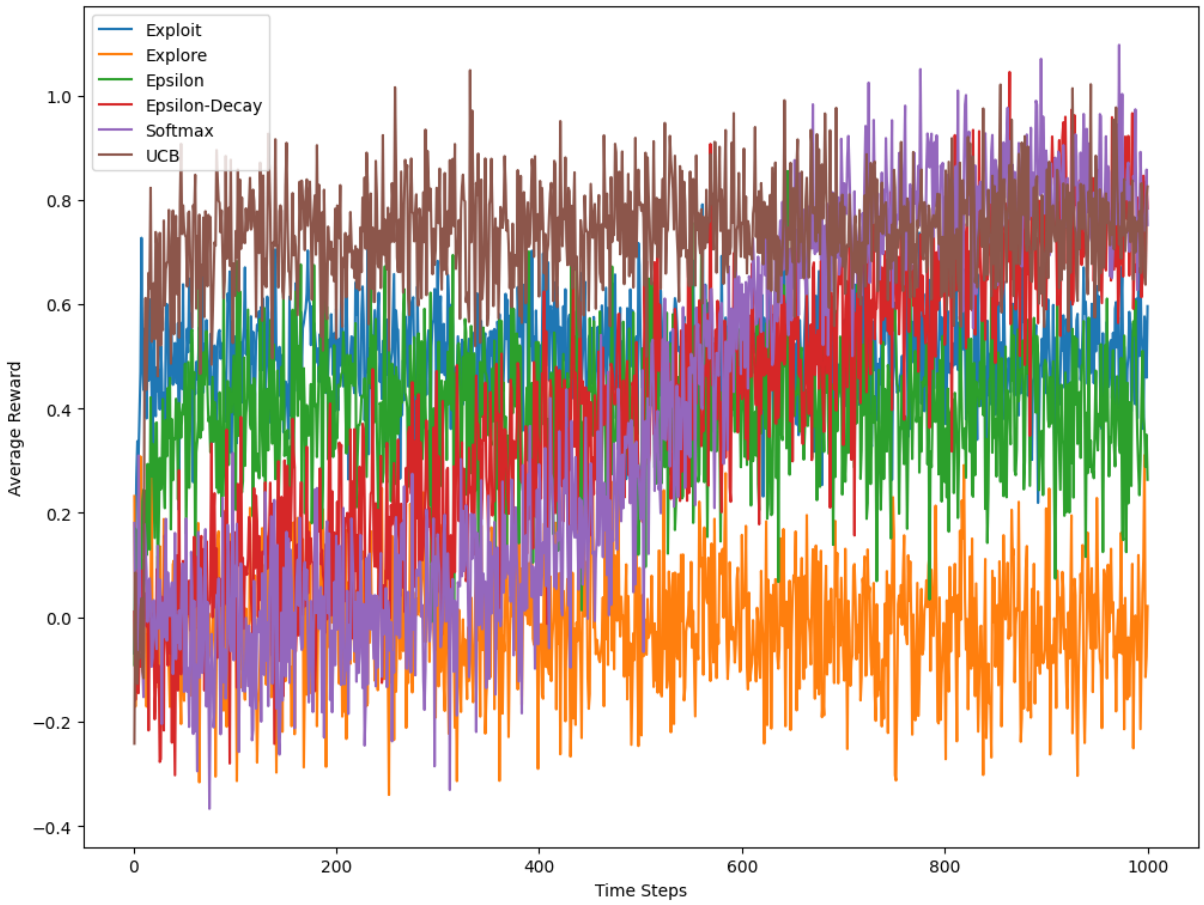

Figure 2: Q5: Average Reward vs Time Steps in 10 Arm Gaussian Bandit

6. Attached Plot of Return Vs Time Step for 1000 Time Steps in Figure 3 for 2 Arm Bernoulli Bandit. The final result comes out to be a straight line for pure exploration, epsilon-greedy, pure exploitation and UCB though the latter 2 have almost zero regret. For pure exploitation, it took the optimial action right off the bat and henceforth the regret is almost zero while for pure exploration, it's taking a random action and therefore the distribution of regret has to be consistent, which is a straight line of a positive slope.

   For softmax, the regret is increasing consistently but then nearing the end, it always takes the optimal action, henceforth the regret converges. This is the same for epsilon-decay except the graph is more parabolic-like and it also converges.

   Epsilon greedy is in the middle of pure exploitation and pure exploration.

   The hyperparameters taken are c = 0.2 for UCB, Epsilon = 0.5 for Epsilon Greedy, Epsilon = 1 to 0.01 for Epsilon-Decay (Linear) and exponential decay (from 100 to 0.01) for temperature in softmax equation.

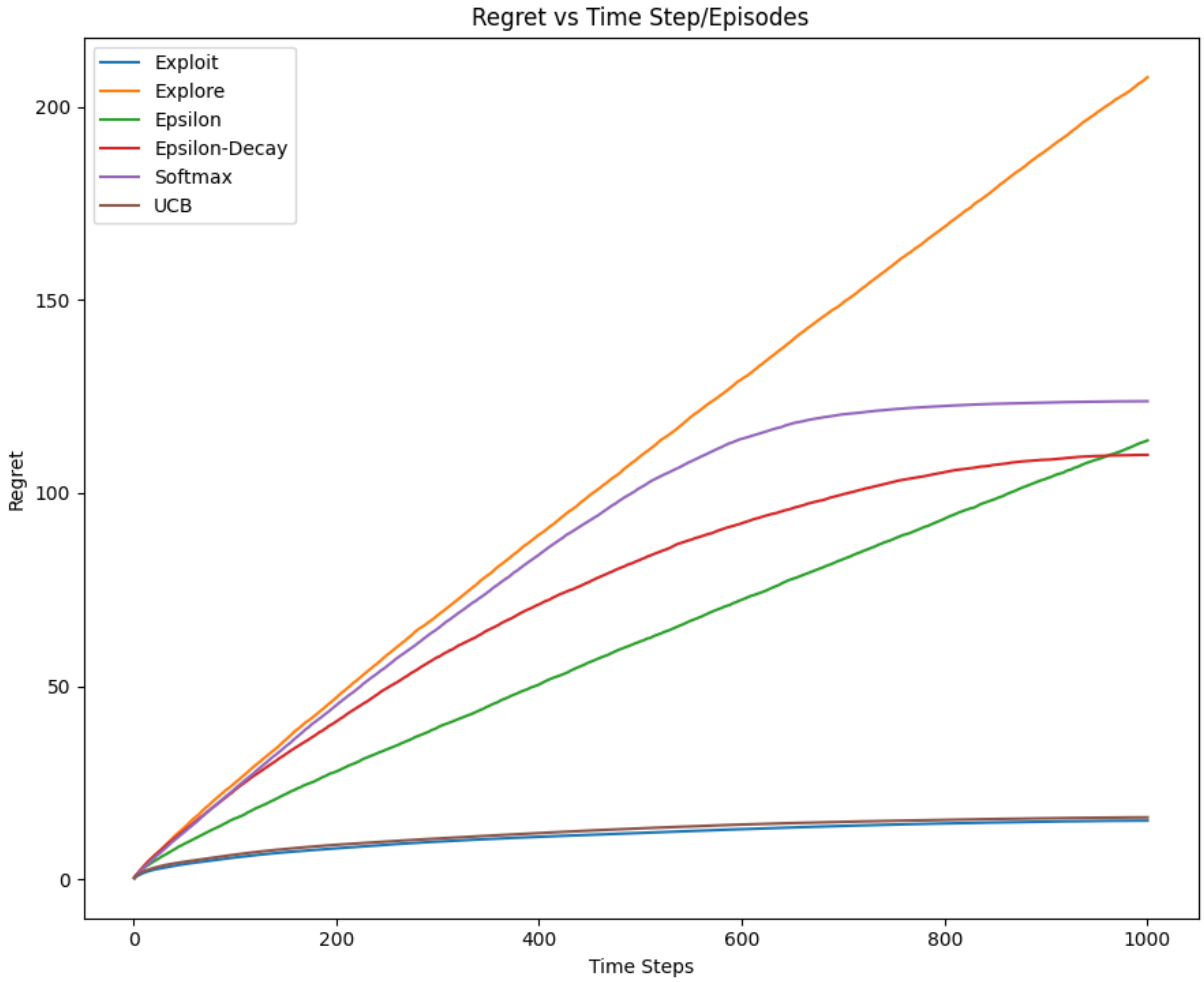

Figure 3: Q6: Regret vs Time Steps in 2 Arm Bernoulli Bandit

7. Attached Plot of Return Vs Time Step for 1000 Time Steps in Figure 4 for 10 Arm Gaussian Bandit. The final result comes out to be a straight line for every algorithm except Epsilon Decay, which comes out to be parabolic.

   The regret is linearly increasing for Exploration, which had the lowest rewards during the training, epsilon greedy is in the middle of pure exploitation and pure exploration. Pure Exploitation and UCB had the lowest Regret among all the algorithms.

   For softmax, the regret is increasing consistently but then nearing the end, it always takes the optimal action, henceforth the regret converges. This is the same for epsilon-decay except the graph is more parabolic-like and it also converges.

   Even though the graphs for 2 Arm Bernoulli Bandit and 10 Arm Gaussian Bandit look the same, the individual values of regret is more in 10 Arm Gaussian Bandit because of more choices and randomness in the environment.

   The hyperparameters taken are c = 0.2 for UCB, Epsilon = 0.5 for Epsilon Greedy, Epsilon = 1 to 0.01 for Epsilon-Decay (Linear) and exponential decay (from 100 to 0.01) for temperature in softmax equation.

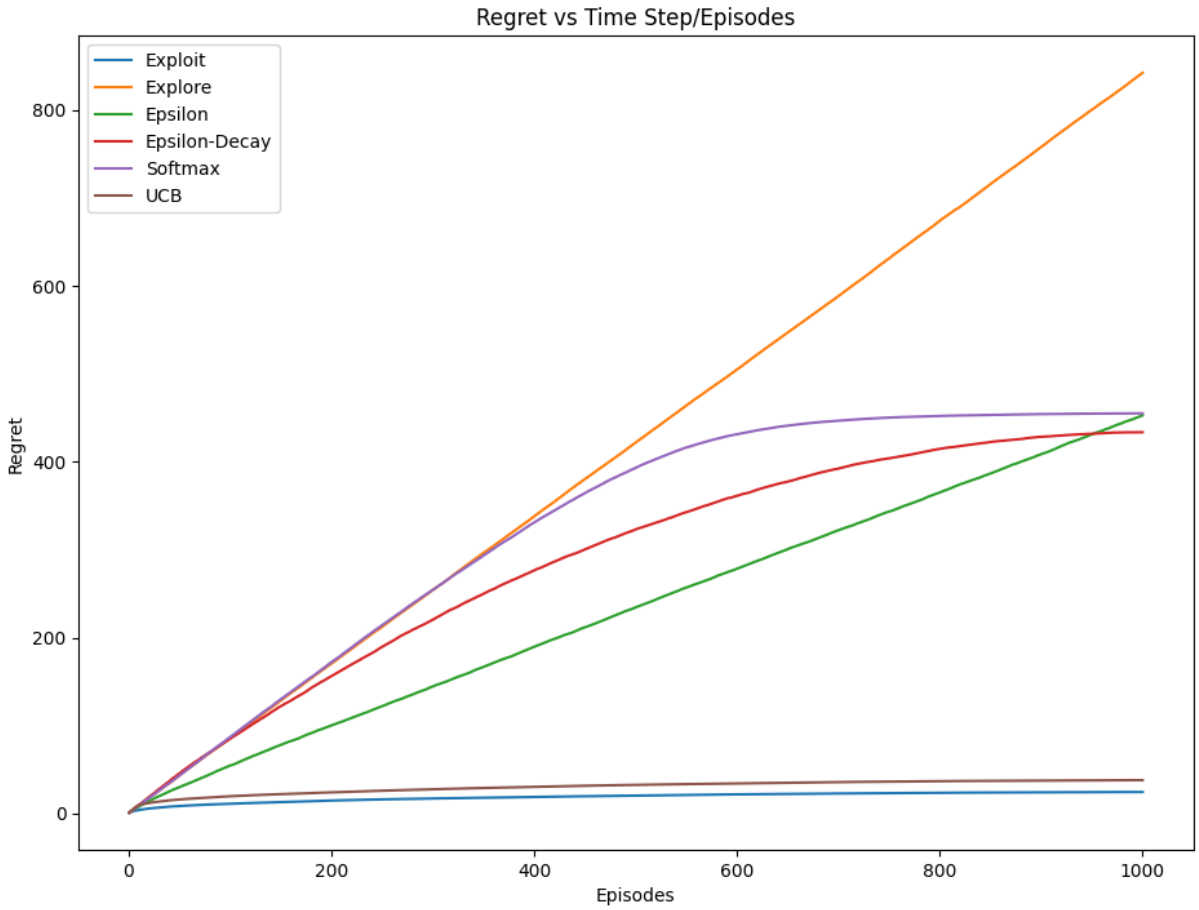

Figure 4: Q7: Regret vs Time Steps in 10 Arm Gaussian Bandit

8. for 2 Arm Bernoulli Bandit, the %Optimal Action vs Episodes is attached in Figure 5.

   Out of Random Luck, in the pure exploitation algorithm, the agent took the optimal decision in the first episode, therefore it took the same action (which is the optimal) in every step - therefore the plot is at 100 for every time step.

   Similarly, over the time - since one out of two action is optimal, pure exploration is getting 50 percent optimal over time while epsilon greedy converges around 75% (middle of pure exploitation and pure exploration).

   The graphs of UCB is that it increases quickly during the beginning, and then settles while Softmax is almost constant in the beginning and then picks up the pace.

   The hyperparameters taken are c = 0.2 for UCB, Epsilon = 0.5 for Epsilon Greedy, Epsilon = 1 to 0.01 for Epsilon-Decay (Linear) and exponential decay (from 100 to 0.01) for temperature in softmax equation.

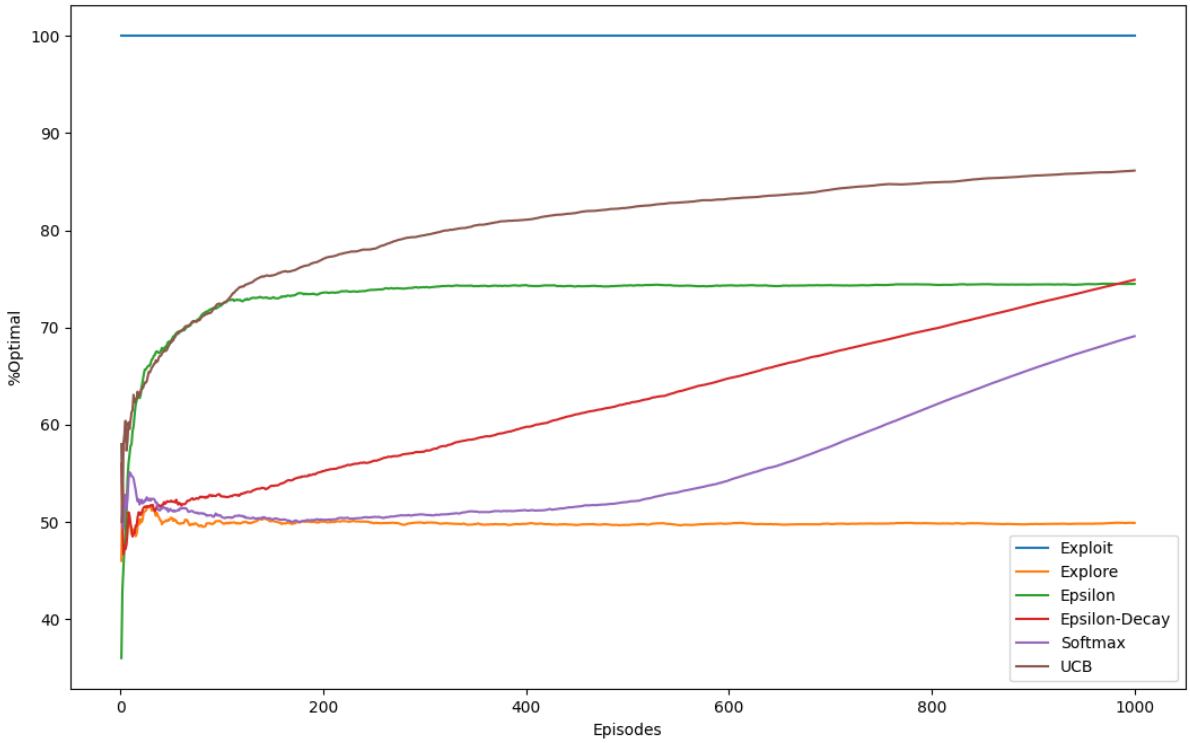

Figure 5: Q8: %Optimal Action vs Episodes in 2 Arm Bernoulli Bandit

9. Attached Plot of %Optimal Action Vs Episodes for 1000 Episodes in Figure 6 for 10 Arm Gaussian Bandit.

   Unlike the 2 Arm Bernoulli, in pure exploitation, the agent didn't take optimal decision, but after a few episodes it learns the optimal solution and then settles.

   The graph is almost the same as 2 Arm Bernoulli Bandit except that the individual values are less in comparison because of more choices and randomness in the environment.

   The hyperparameters taken are c = 0.2 for UCB, Epsilon = 0.5 for Epsilon Greedy, Epsilon = 1 to 0.01 for Epsilon-Decay (Linear) and exponential decay (from 100 to 0.01) for temperature in softmax equation.

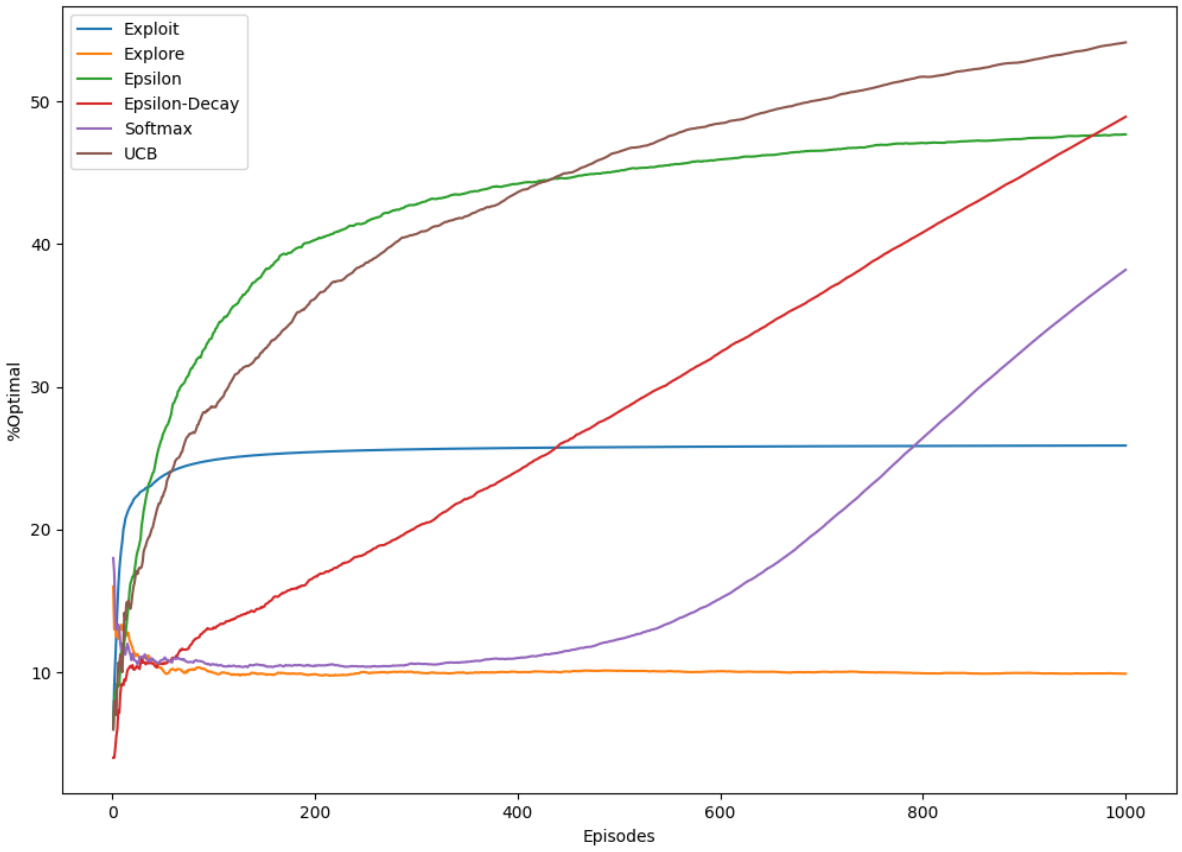

Figure 6: Q9: %Optimal Action vs Episodes in 10 Arm Gaussian Bandit

Solution to Problem 2: MC Estimates and TD Learning

1. The trajectory is working fine. The policy taken is action = 0 (left) for all states. This is implemented for all the questions.
2. Figure 7 depicts the decay.

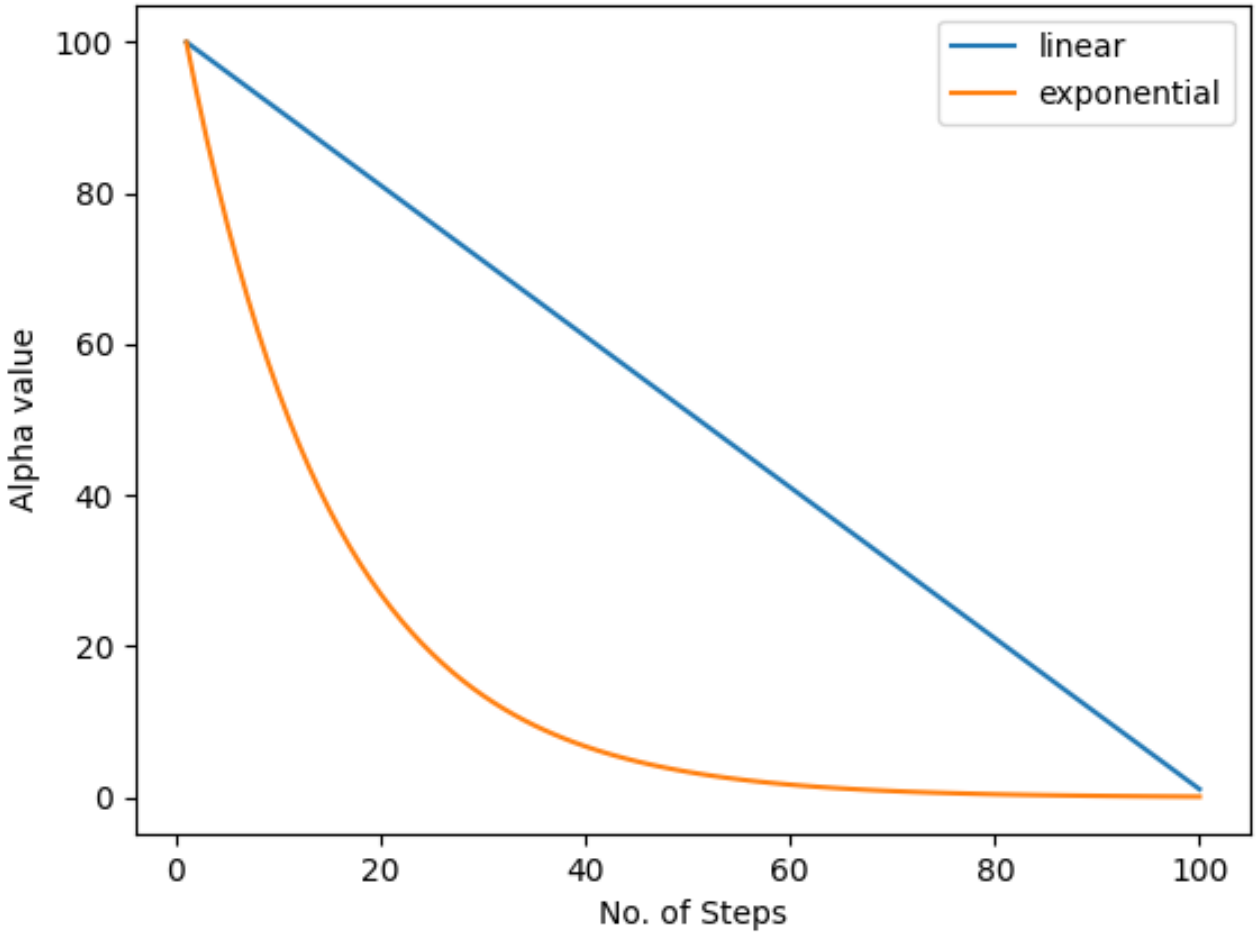

Figure 7: Q2: alpha vs time step

3. MonteCarlo implemented given in the readme. For high values of episode number and maximum number of steps, the algorithm converges at 0.5 for V(3) and the average return for discount = 1 also converges to 0.5.

4. TD learning implemented given in the readme. For high values of episode number and maximum number of steps, the algorithm converges at 0.5 for V(3) and the average return for discount = 1 also converges to 0.5.

5. Figure 8 depicts the MC-EVMC estimates through time. The estimation for the states closest to the terminal state that gives reward = 1 is the highest.

   The plots have high variance in the beginning. We know that $v_{e+1}(s) = v_e(s) + \alpha(e)[G_e - V_e(s)]$. This $\alpha(e)$ is decaying, and has the highest value in the beginning which causes the high variance. To add, G_t is an unbiased, but high variance estimate of the true value function $v_\pi(s)$. This G_t causes the main variance seen in the beginning. The graph jumps around the final true value as the MC Target is unbiased estimate of the true value.

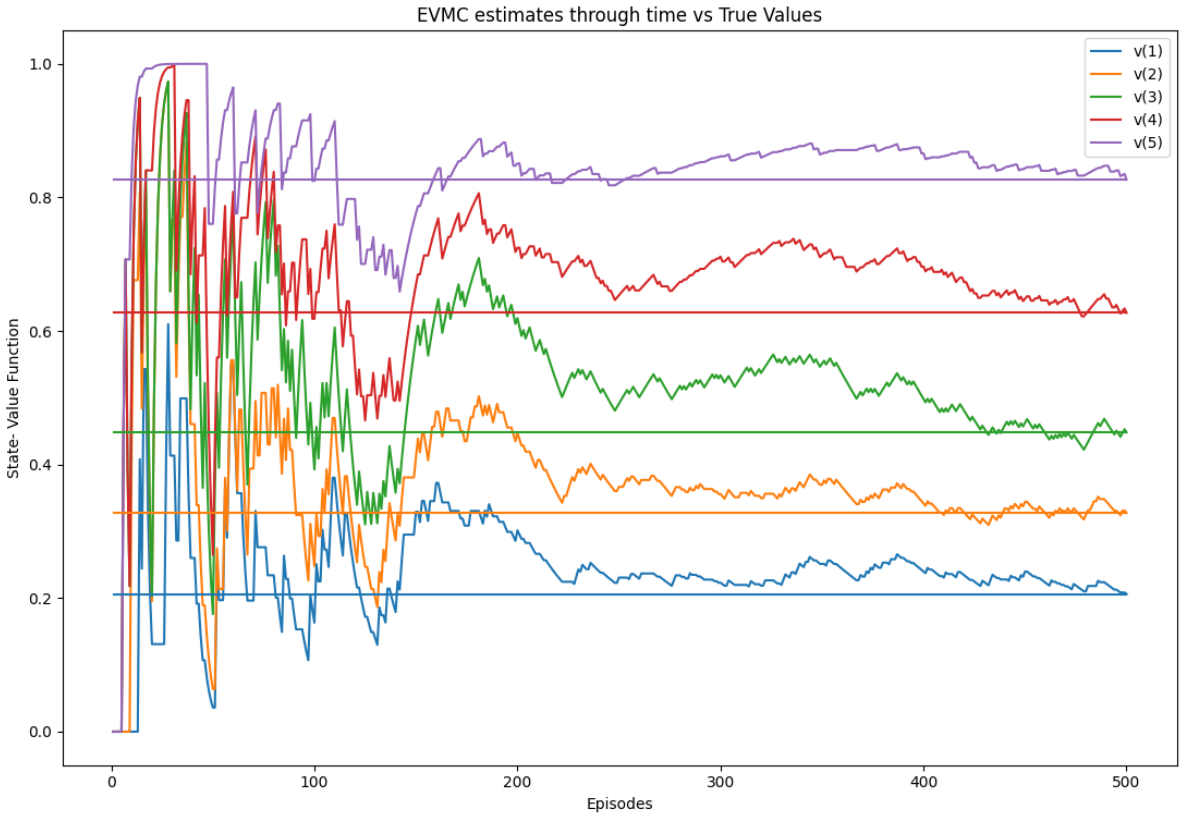

Figure 8: Q5: MC-EVMC estimate vs Episodes in RWE

6. Figure 9 depicts the MC-FVMC estimates through time. The estimation for the states closest to the terminal state that gives reward = 1 is the highest.

   The plots have high variance in the beginning. We know that $v_{e+1}(s) = v_e(s) + \alpha(e)[G_e - V_e(s)]$. This $\alpha(e)$ is decaying, and has the highest value in the beginning which causes the high variance. To add, G_t is an unbiased, but high variance estimate of the true value function $v_\pi(s)$. This G_t causes the main variance seen in the beginning. The graph jumps around the final true value as the MC Target is unbiased estimate of the true value.

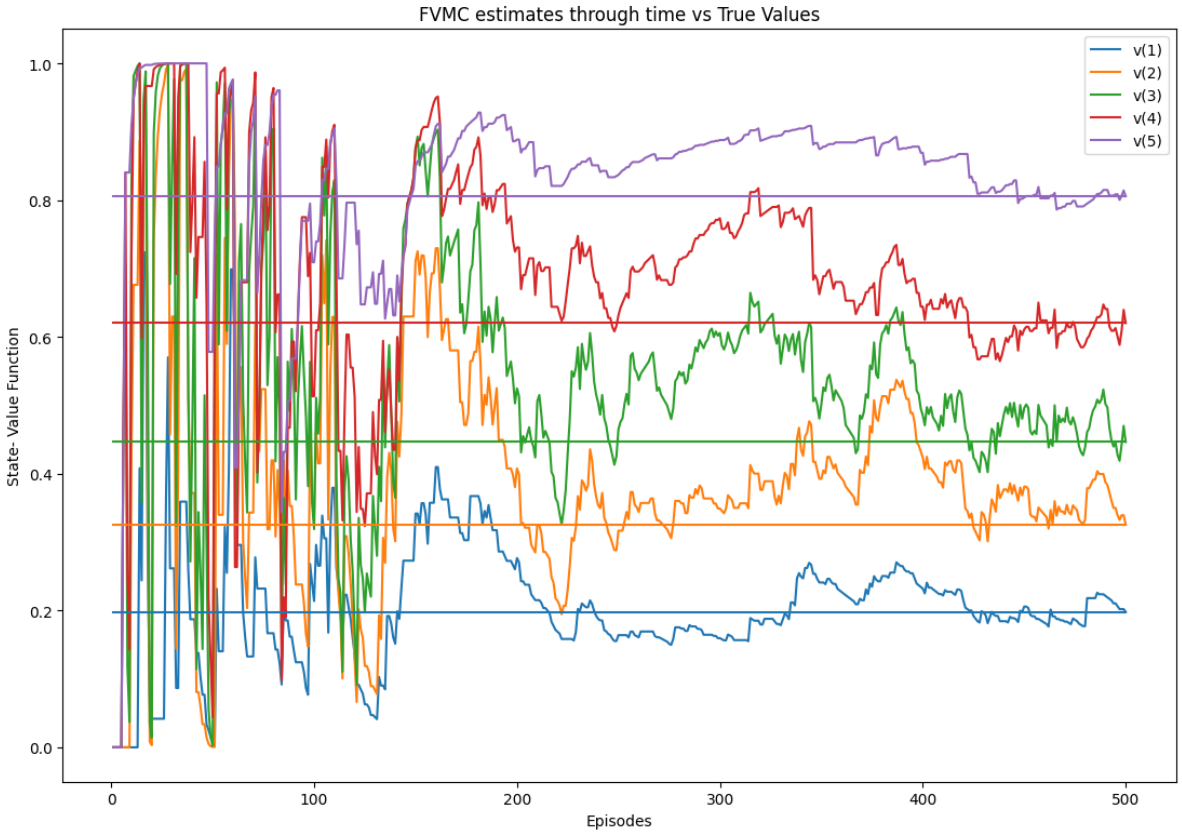

Figure 9: Q6: MC-FVMC estimate vs Episodes in RWE

7. Figure 10 depicts the TD estimates through time. The estimation for the states closest to the terminal state that gives reward = 1 is the highest.

   The plots have high variance in the beginning than at the end, where it settles down. Since the estimation of the true value is biased, the graph for individual states do not jump forth the final value.

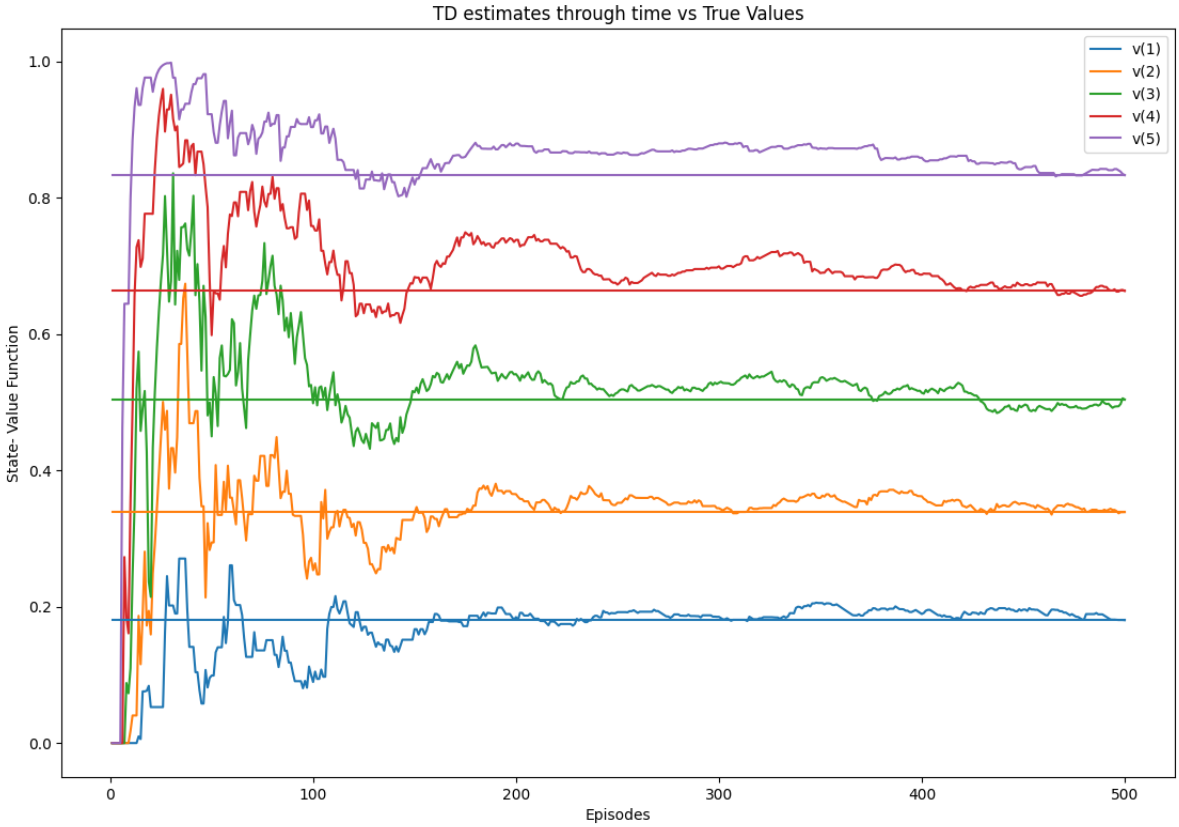

Figure 10: Q7: TD estimate vs Episodes in RWE

8. Figure 11 depicts the MC-EVMC estimates through log(time). The graph comes out to have the same characteristics as the normal MC-EVMC estimate through time. In the beginning, when the agent knows nothing - the variance is high as it settles down to it's true value at the end.

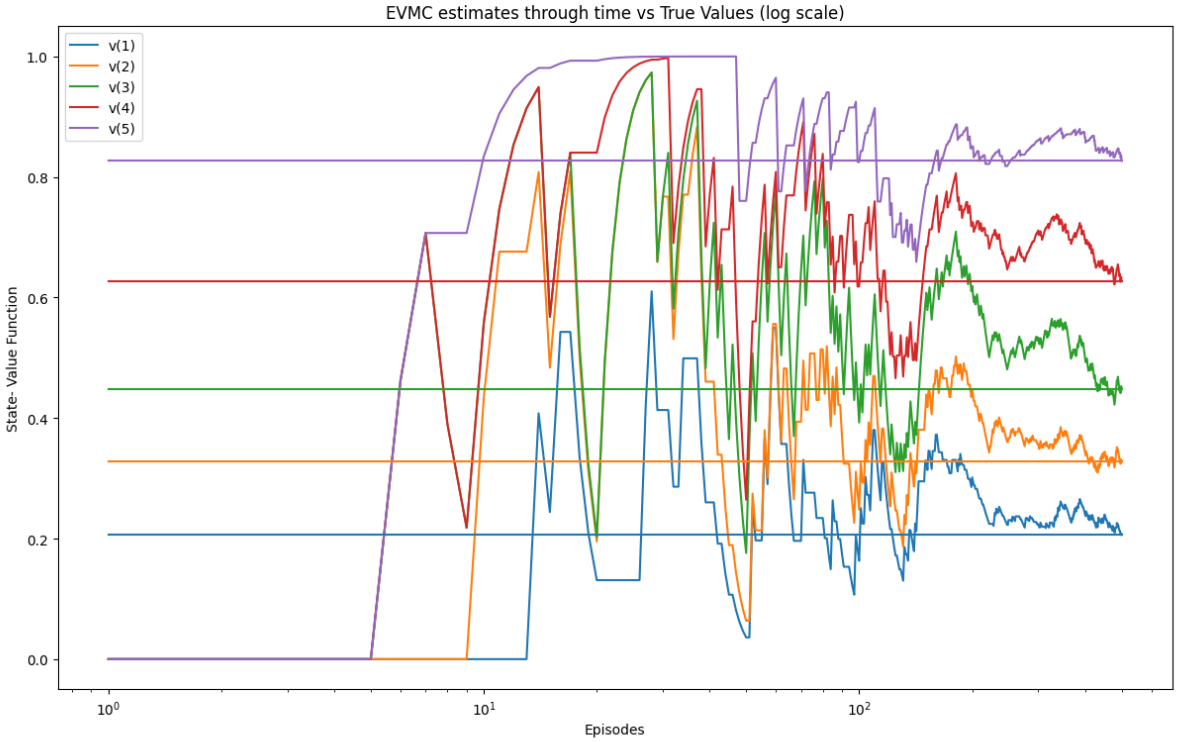

Figure 11: Q8: MC-EVMC estimate vs log(Episodes) in RWE

9. Figure 12 depicts the MC-FVMC estimates through log(time). The graph comes out to have the same characteristics as the normal MC-FVMC estimate through time. In the beginning, when the agent knows nothing - the variance is high as it settles down to it's true value at the end.

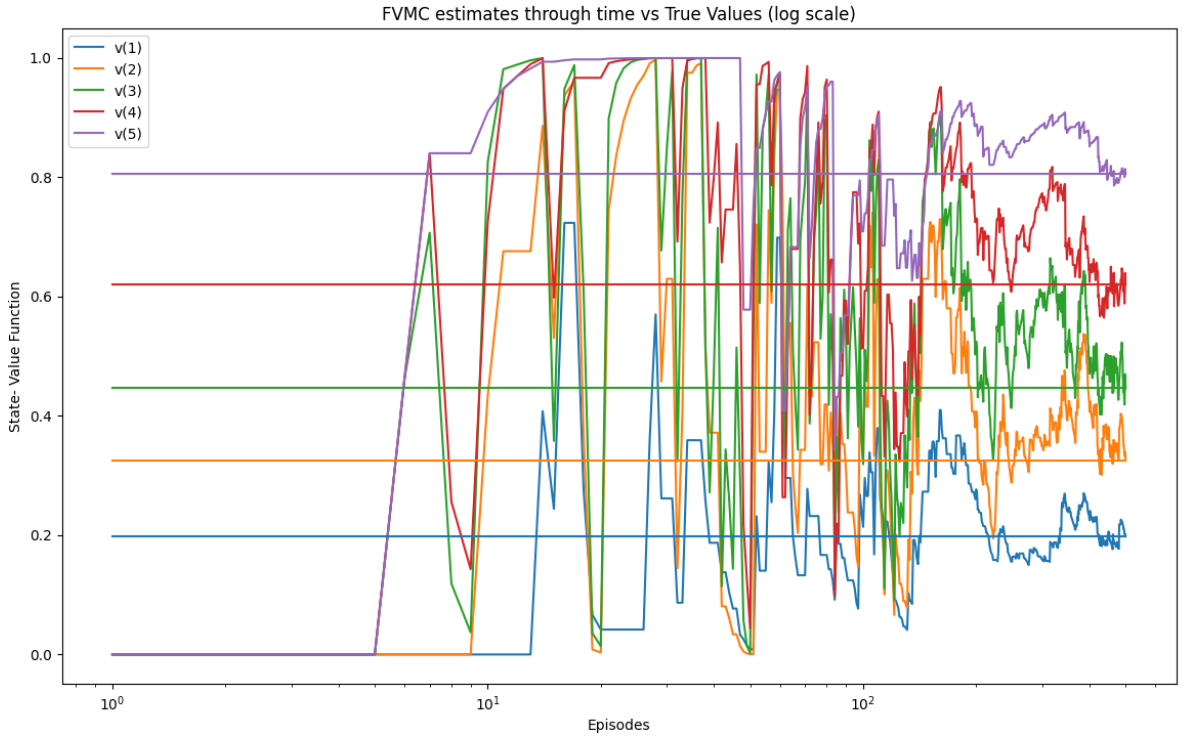

Figure 12: Q9: MC-FVMC estimate vs log(Episodes) in RWE

10. Figure 13 depicts the TD estimates through log(time). The graph comes out to have the same characteristics as the normal TD-learning estimate through time. The variance is high in the beginning as compared to end because the agent knows nothing. The plot for individual state either gradually goes up or down.

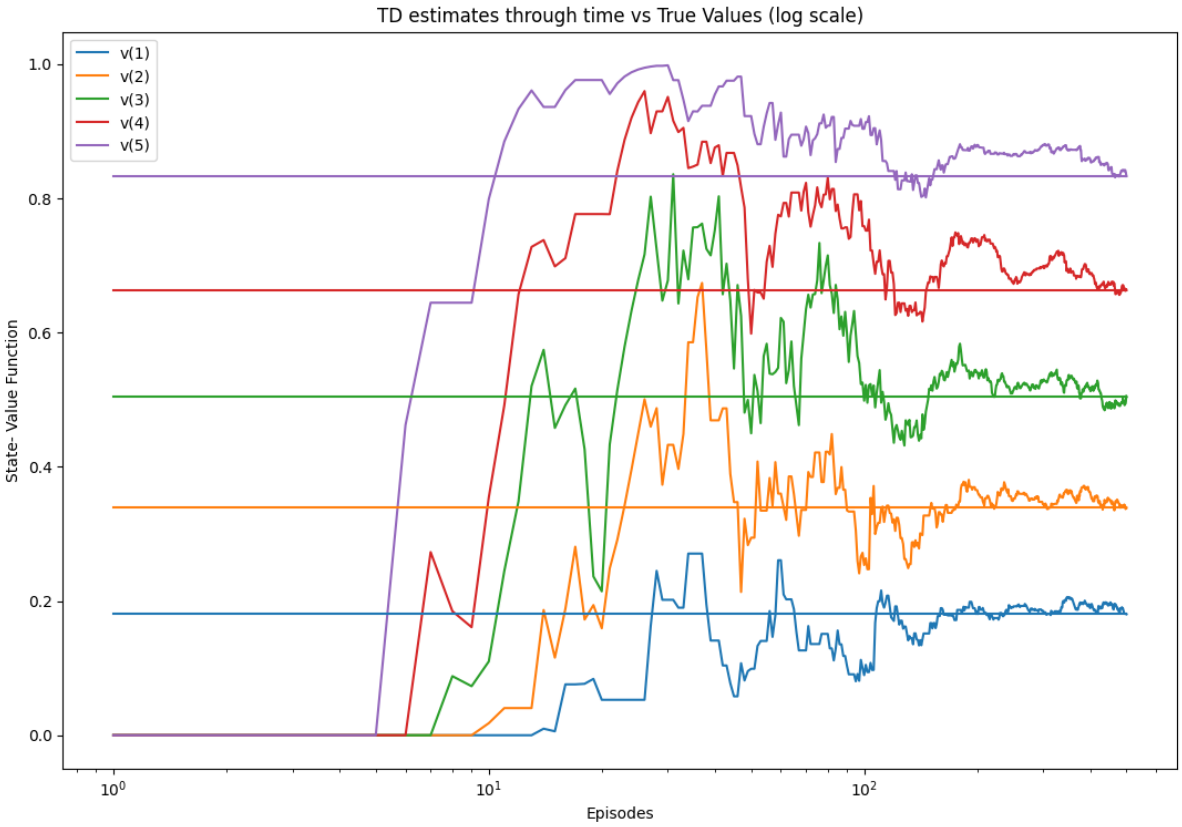

Figure 13: Q10: TD estimate vs log(Episodes) in RWE

11. In comparison of Monte-Carlo and TD learning, Monte-Carlo is sample inefficient - therefore to overcome the randomness, we take lots of samples from the trajectory. Therefore, the actual return has very high variance while in TD learning, we learn from an incomplete episode as we take one step and make an estimate (unlike in MC where we calculate from the entire trajectory/episode). Therefore, the variance in TD learning comes out to be only from $R_{(t+1)}$ and is henceforth very less than Monte-Carlo.

    The FMVC has higher variance than EVMC because FVMC also considers the already visited while EVMC completely ignores them. But in the end, both of them converge to the true value.

12. Figure 14 depicts the MC-EVMC target value (in blue) vs the final state value function (in green) for state = 3 (initial state). The target value is the true return, and for discount = 1; it's either 1 or 0. Therefore, the plots are either 1 or 0.

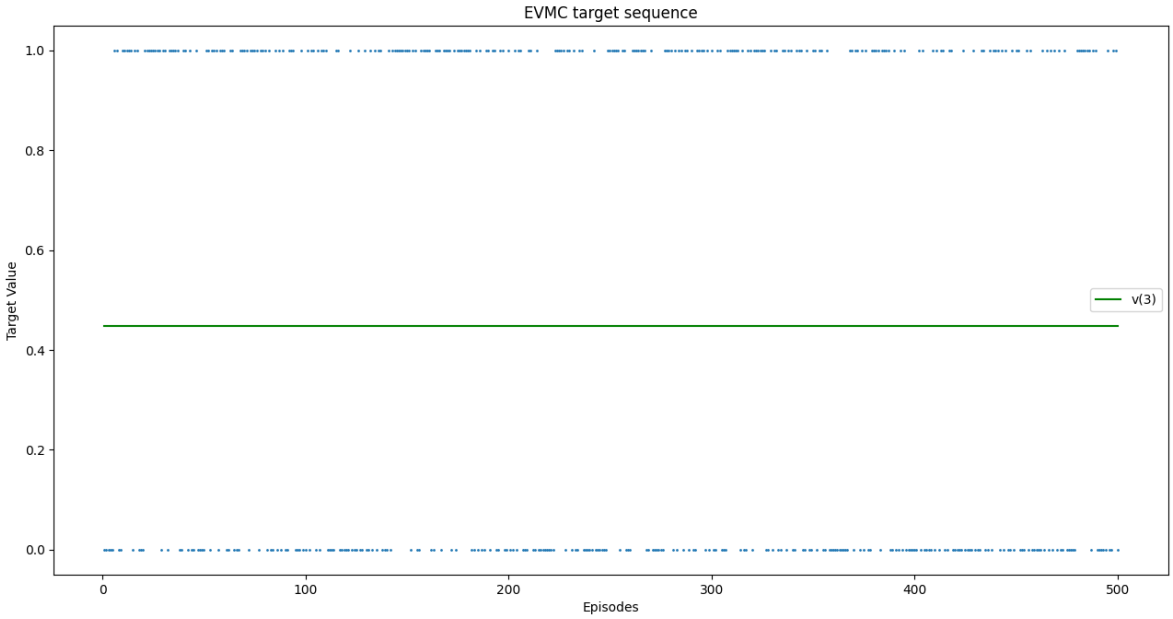

Figure 14: Q12: MonteCarlo EVMC vs Episodes in RWE

13. Figure 15 depicts the MC-FVMC target value (in blue) vs the final state value function (in green) for state = 3 (initial state). The target value is the true return, and for discount = 1; it's either 1 or 0. Therefore, the plots are either 1 or 0.

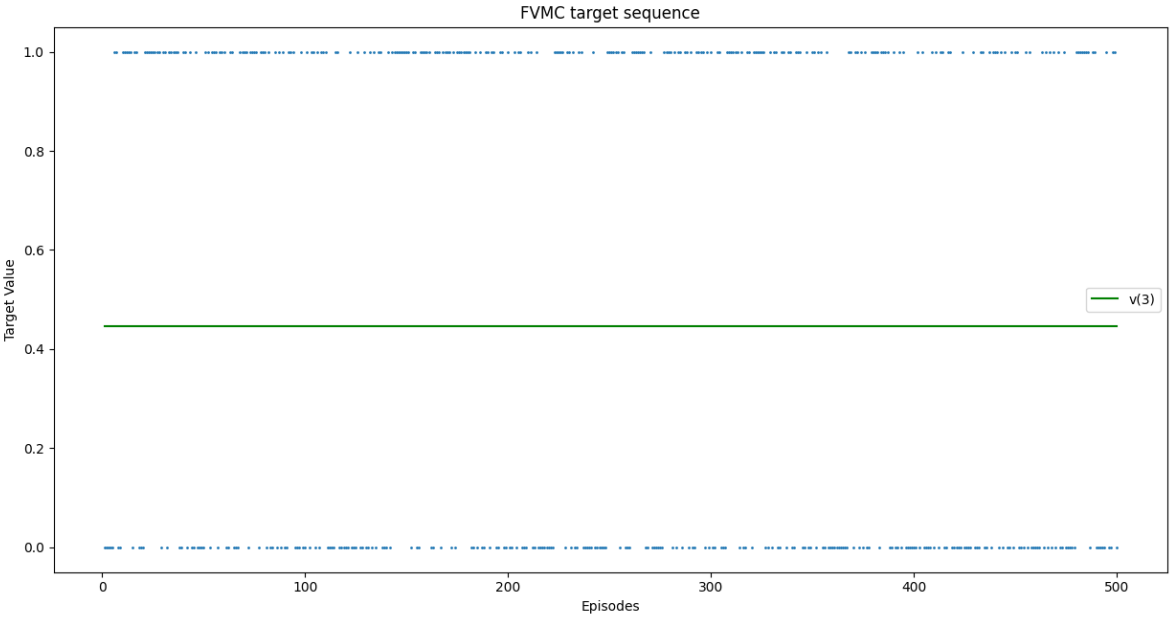

Figure 15: Q13: MonteCarlo FVMC vs Episodes in RWE

14. Figure 16 depicts the TD target value (in blue) vs the final state value function (in green) for state = 3 (initial state). The target value varies between 0 and 1 and slowly converges at the end.

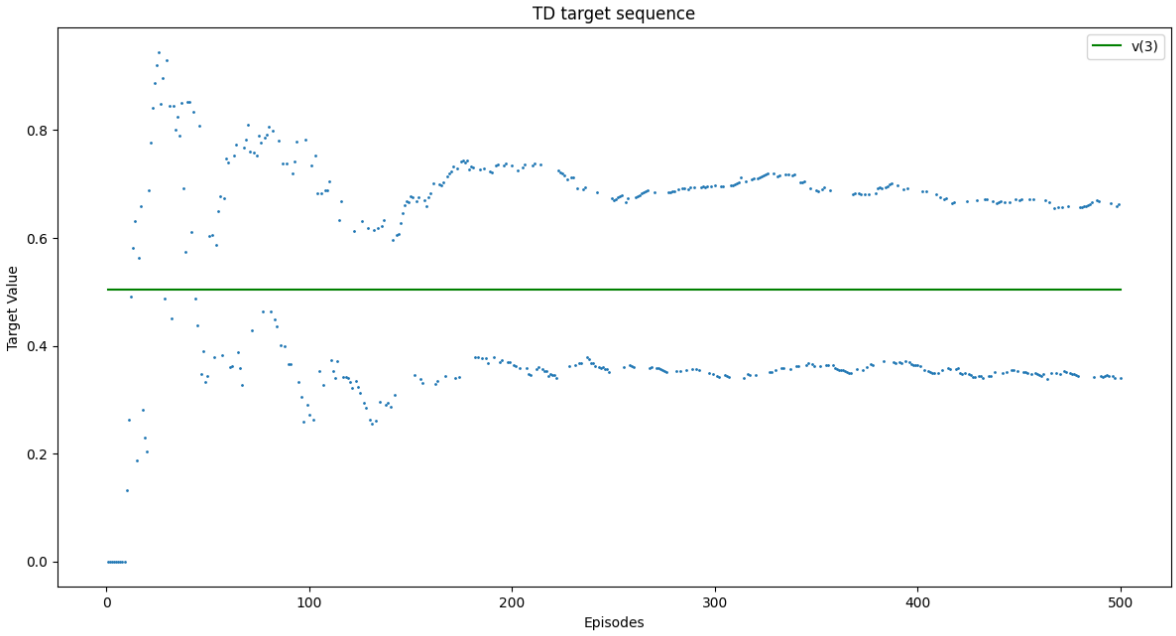

Figure 16: Q14: TD Target vs Episodes in RWE

15. In conclusion to Figure 14, Figure 15 and Figure 16 - the Target Value for EVMC and FVMC are the same because they are calculated the same way while in TD learning we have been estimating the target value which makes it varing between 0 and 1.

Citations/References

- CS698R - Introduction to Deep Reinforcement Learning Lectures by Prof. Ashutosh Modi, IIT Kanpur
- https://github.com/openai/gym/blob/master/docs/creating-environments.md
- https://github.com/openai/gym/blob/master/gym/core.py
- https://gym.openai.com
- https://towardsdatascience.com/beginners-guide-to-custom-environments-in-openai-s-gym-989371673952
- https://docs.scipy.org/doc/scipy/reference/generated/scipy.special.softmax.html
- https://matplotlib.org/stable/contents.html
- https://numpy.org/doc/