CS698R: Deep Reinforcement Learning

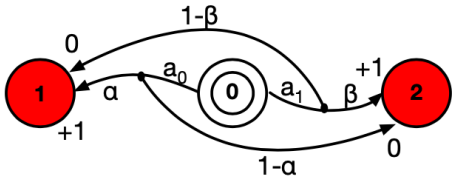
Assignment #1

Name: James Bond
Roll NO.: 0000007
Google Colab Link (as tiny URL link): www.tinyurl.com

Solution to Problem 1: Multi-armed Bandits

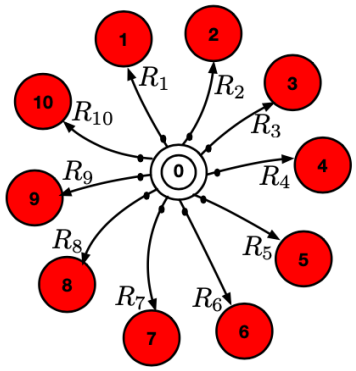
1. 2-armed bandits The environment described by below diagram runs for various values of (α, β) and tested manually for 10 episodes.

- $\alpha = 0, \beta = 0$: The reward is going to 0 at all times, and it has been verified in the implementation
- $\alpha = 1, \beta = 0$: here, the left action will always give 1 reward and right action will always give 0 reward, it has been verified for 100 episodes.
- $\alpha = 0, \beta = 1$:this case is similar to previous case
- $\alpha = 1, \beta = 1$: in this case, no matter what actions we take, we end with 1 reward and it has been verified for 100 episodes.



(a) Two-armed Bernoulli Bandit

2. 10-armed Gaussian bandits The environment described by below diagram runs for various values of (σ), the mean is set to 0 and tested manually for 10 episodes. In this case, multiple values of σ^2 has been tried, We observed that higher value of σ^2 is giving more reward



(b) Ten-armed Gaussian Bandit

3. Bandits Agents:

- **Pure Exploitation strategy**, **Pure Exploration strategy** and **Greedy Strategies** are implemented in a same function by varying ϵ between 0, 1.
- $\epsilon = 0$ will give pure exploitation behavior where as $\epsilon = 1$ will give pure exploration behavior.
- In Greedy epsilon function, $\epsilon = 0.0, 0.01, 0.05, 0.1, 0.2, 0.5, 1$ have been tried and observation have been reported in the below parts. In decaying greedy method, intial value of epsilon is set = 0.2 and results are reported corresponding to different decay rates = 0.01, 0.05, 0.1, 0.3, 0.5
- In Softmax strategy, average reward of linear decay of temperature have been reported in the below parts
- In UCB strategy, c = 0.01, 0.1, 0.5, 1, 2, 5 have been tried and their results have been documented in below parts.

4. 50 different 2-armed bandits

Key Observations: Below are the observation for all the 4 cases of plots.

- **Pure Exploitation**, alternatively $\epsilon = 0$ is giving the lowest average reward.Pure Exploration, alternatively $\epsilon = 1$ is also giving lowest reward, Figure 2
- One different between the two is the spikes, that is getting generated because of the exploratory nature of the algorithm. We can also observe that the spikes are becoming more evident as there is an increase in the ϵ value.
- , In the decaying epsilon greedy plot, Figure 1, we can observe that, epsilon = 0.2 with linear decay rate of 0.01 is giving the most reward.
- For softmax case, it can seen that as temperature is decreasing, the reward is increasing, the best average reward is achieved when temperature = 0.1
- For UCB case, we observed that average reward when c value is very low i.e. 0.01 or very high 5 is relatively lower compared to when c = 1, 2.

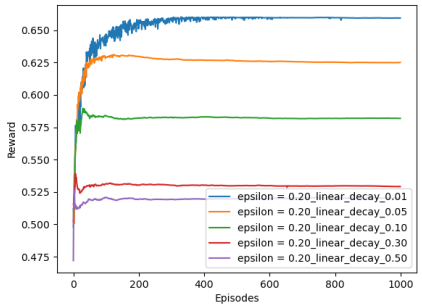


Figure 1: decaying epsilon greedy

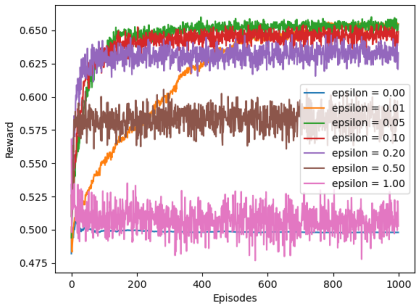


Figure 2: epsilon greedy

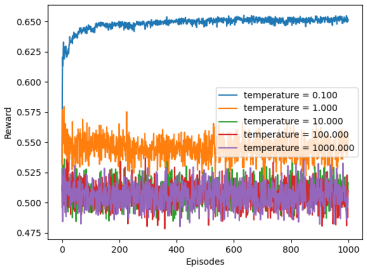


Figure 3: Softmax

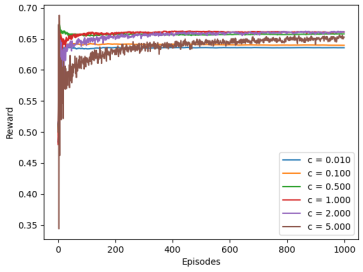


Figure 4: UCB Reward

5. 50 different 10-armed bandits
Key Observations

- **Pure Exploitation**, alternatively $\epsilon = 0$ is giving the lowest average reward.Pure Exploration, alternatively $\epsilon = 1$ is also giving lowest reward, Figure 6.
- One different between the two is the spikes, that is getting generated because of the exploratory nature of the algorithm. We can also observe that the spikes are becoming more evident as there is an increase in the ϵ value, Figure 6.
- For softmax case, it can seen that as temperature is decreasing, the reward is increasing, the best average reward is achieved when temperature = 0.1
- For UCB case, we observed that average reward when c value is very low i.e. 0.01 or very high 5 is relatively lower compared to when c = 1, 2.

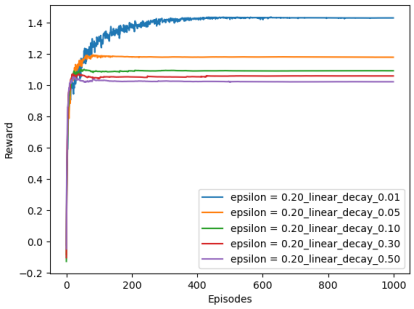


Figure 5: Decaying epsilon

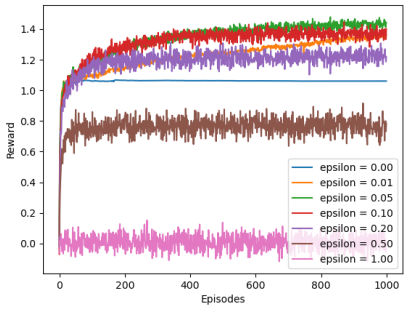


Figure 6: epsilon greedy

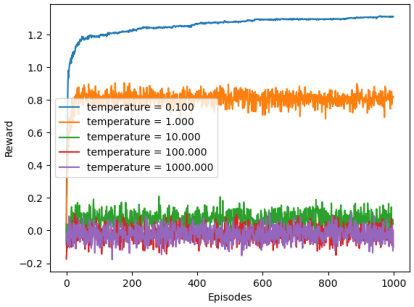


Figure 7: Softmax

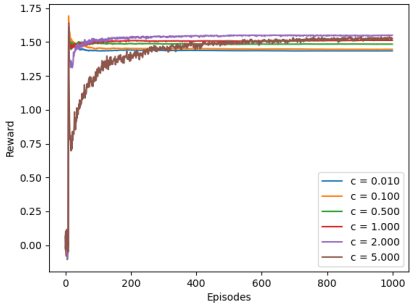


Figure 8: UCB Reward

6. 2-armed bandits: Regret vs episodes analysis Key Observation

- Regret is the expectation of difference between the actual value versus the optimal value. Hence, more closer our actions to the optimal actions, the lower the value of regret will be.
- **Pure Exploitation**, alternatively $\epsilon = 0$ is giving the highest average regret (as it has lowest reward). Most regret is present in the pure exploratory or pure exploitation scenarios as seen in the Figure 10.
- In decaying epsilon greedy method, Figure 9, higher the decay rate value, lower will be the resulting epsilon; higher decay corresponds to exploitation behavior and lower decay corresponds to exploration behavior. It can be seen that, higher exploration is giving more regret value.
- For softmax case, it can seen that as temperature is decreasing, the reward is increasing, hence, regret is decreasing, the lowest average regret is achieved when temperature = 0.1
- For UCB case, we observed that average regret when c value is very low i.e. 0.01 or very high 5 is relatively higher compared to when c = 1, 2.

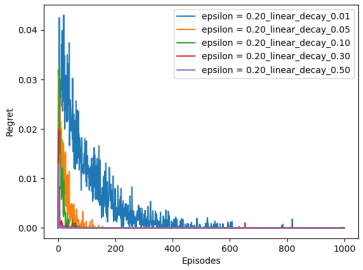


Figure 9: Decaying epsilon

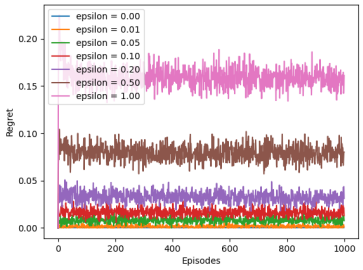


Figure 10: epsilon greedy

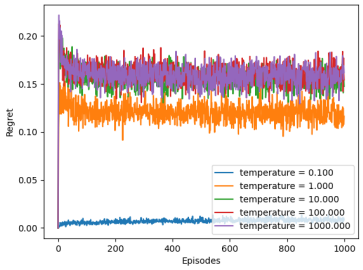


Figure 11: Softmax Regret

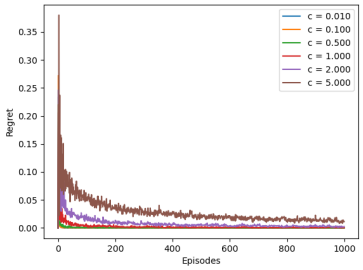


Figure 12: UCB Regret

7. 10-armed bandits: Regret vs episodes analysis Key Observation

- **Pure Exploitation**, alternatively $\epsilon = 0$ is giving the highest average regret (as it has lowest reward). Most regret is present in the pure exploratory or pure exploitation scenarios as seen in the Figure 14.
- In decaying epsilon greedy method, Figure 13, higher the decay rate value, lower will be the resulting epsilon; higher decay corresponds to exploitation behavior and lower decay corresponds to exploration behavior. It can be seen that, higher exploration is giving more regret value.
- For softmax case, it can seen that as temperature is decreasing, the reward is increasing, hence, regret is decreasing, the lowest average regret is achieved when temperature = 0.1
- For UCB case, we observed that average regret when c value is very low i.e. 0.01 or very high 5 is relatively higher compared to when c = 1, 2.

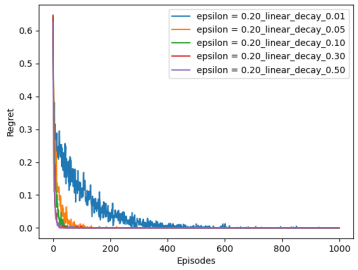


Figure 13: decaying epsilon

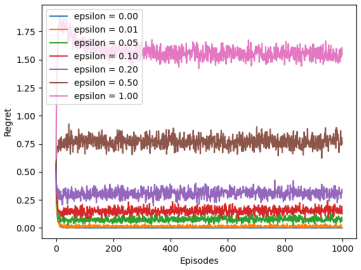


Figure 14: epsilon greedy

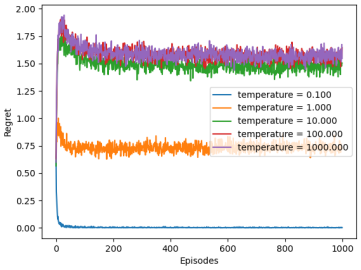


Figure 15: Softmax

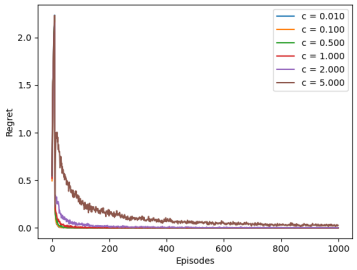


Figure 16: UCB

8 and 9. percentage of optimal actions vs episodes: 2-armed bandits and 10-armed bandits

- Below plots explain the percentage of optimal actions with episodes, hence, the best reward giving curve is going to have high optimal actions percentage value.
- In all the cases below, the plots are inline with the previous analysis done on the average reward and average regret.

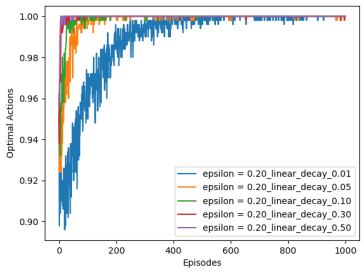


Figure 17: 2 arm - Decaying epsilon greedy

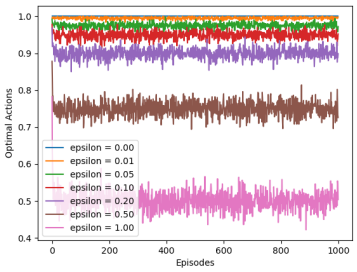


Figure 18: 2 arm - epsilon greedy

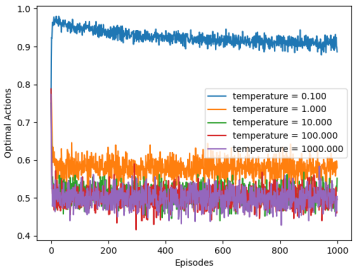


Figure 19: 2 arm - Softmax

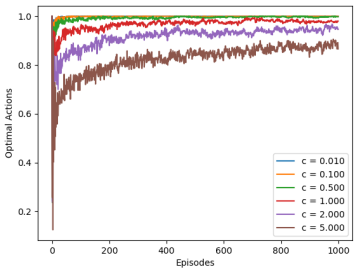


Figure 20: 2 arm - UCB

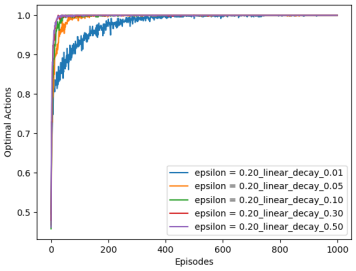


Figure 21: 10-arm: Decaying epsilon

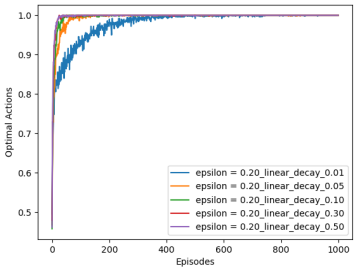


Figure 22: 10-arm: epsilon greedy

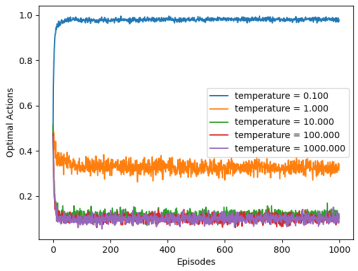


Figure 23: 10-arm: Softmax

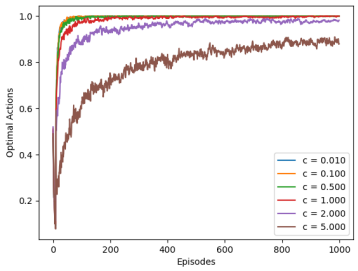


Figure 24: 10-arm: UCB

Solution to Problem 2: MC Estimates and TD Learning

1. We implement the function `generateTrajectory()` in code and to validate its correctness, we consider the following tests:
   - We use always left and always right policies and use `generateTrajectory()` with them. We observe that the actions taken are indeed always left and always right respectively.
   - To test the stochastic property of the environment we use `generateTrajectory()` with either policy and observe that the step taken is approximately left half of the time and right the other half.
2. We implement `decayAlpha()` and test it by plotting the values returned by the function. We observe the plot (Figure 25) is indeed in line with our expectations.

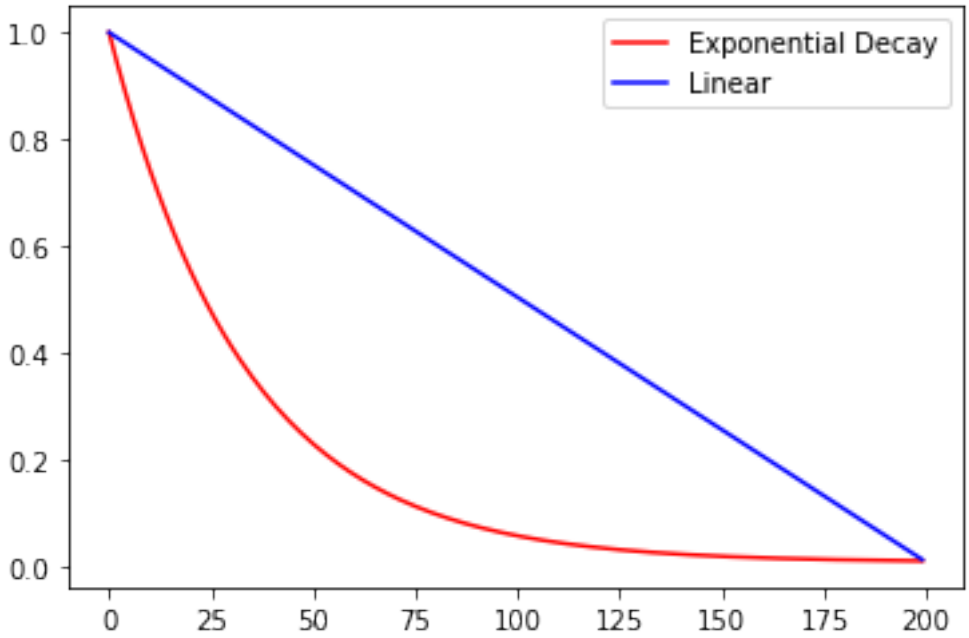


Figure 25: Plot for decayAlpha

3. To test our Monte-Carlo prediction algorithm, we compute the optimal value function using the bellman equations as follows:

We consider the action space to be $\{0, 1\}$ corresponding to $\{left, right\}$ and always left as a policy.

$$
\begin{aligned}
v(0) &= 0 \\
v(1) &= p(0,0|1,0)[0 + \gamma v(0)] + p(2,0|1,0)[0 + \gamma v(2)] \\
v(2) &= p(1,0|2,0)[0 + \gamma v(1)] + p(3,0|2,0)[0 + \gamma v(3)] \\
v(3) &= p(2,0|3,0)[0 + \gamma v(2)] + p(4,0|3,0)[0 + \gamma v(4)] \\
v(4) &= p(3,0|4,0)[0 + \gamma v(3)] + p(5,0|4,0)[0 + \gamma v(5)] \\
v(5) &= p(4,0|5,0)[0 + \gamma v(4)] + p(6,1|5,0)[1 + \gamma v(6)] \\
v(6) &= 0
\end{aligned}
$$

Thus we have,

$$
\begin{aligned}
v(0) &= 0 \\
v(1) &= \frac{1}{2}[\gamma 0] + \frac{1}{2}[\gamma v(2)] \\
v(2) &= \frac{1}{2}[\gamma v(1)] + \frac{1}{2}[\gamma v(3)] \\
v(3) &= \frac{1}{2}[\gamma v(2)] + \frac{1}{2}[\gamma v(4)] \\
v(4) &= \frac{1}{2}[\gamma v(3)] + \frac{1}{2}[\gamma v(5)] \\
v(5) &= \frac{1}{2}[\gamma v(4)] + \frac{1}{2}[1 + \gamma 0] \\
v(6) &= 0
\end{aligned}
$$

Simplifying,

$$
\begin{aligned}
v(0) &= 0 \\
v(1) - \frac{\gamma}{2} v(2) &= 0 \\
v(2) - \frac{\gamma}{2} v(1) - \frac{\gamma}{2} v(3) &= 0 \\
v(3) - \frac{\gamma}{2} v(2) - \frac{\gamma}{2} v(4) &= 0 \\
v(4) - \frac{\gamma}{2} v(3) - \frac{\gamma}{2} v(5) &= 0 \\
v(5) - \frac{\gamma}{2} v(4) &= \frac{1}{2} \\
v(6) &= 0
\end{aligned}
$$

Rewriting in matrix formulation,

$$
\begin{bmatrix}
1 & -\frac{\gamma}{2} & 0 & 0 & 0 \\
-\frac{\gamma}{2} & 1 & -\frac{\gamma}{2} & 0 & 0 \\
0 & -\frac{\gamma}{2} & 1 & -\frac{\gamma}{2} & 0 \\
0 & 0 & -\frac{\gamma}{2} & 1 & -\frac{\gamma}{2} \\
0 & 0 & 0 & -\frac{\gamma}{2} & 1
\end{bmatrix}
\begin{bmatrix} v(1) \\ v(2) \\ v(3) \\ v(4) \\ v(5) \end{bmatrix}
=
\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \frac{1}{2} \end{bmatrix}
$$

Solving with $\gamma = 0.99$ we get,

$$v(1) = 0.1501, v(2) = 0.3032, v(3) = 0.4624, v(4) = 0.6310, v(5) = 0.8124$$

We can now run our implementation of MC prediction and compare the values we obtain. We observe that the values generated by our prediction models are quite close to the actual values we computed analytically. This confirms that our implementation is correct.

4. For Temporal Difference as well we test our implementation in the same manner, namely comparing the generated values with the true ones. Here also we note that the values are quite close and hint to the correctness of our implementation.

   We do note that if we do not average over multiple runs and just consider one run of 500 episodes then the MC-FVMC gives us the closest estimates and mostly overestimates the true values, whereas MC-EVMC and TD both give underestimates with TD working better than MC-EVMC.

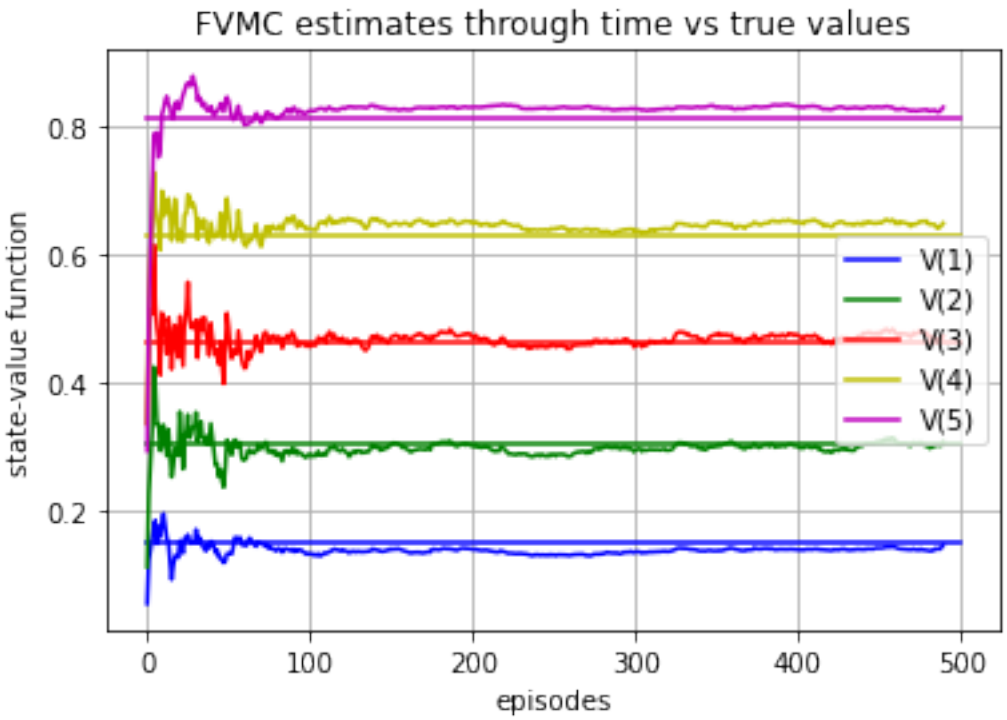


Figure 26: MC-FVMC vs True Values

5. We show the plot of MC-FVMC for non-terminal states in Figure 26
6. We show the plot of MC-EVMC for non-terminal states in Figure 27

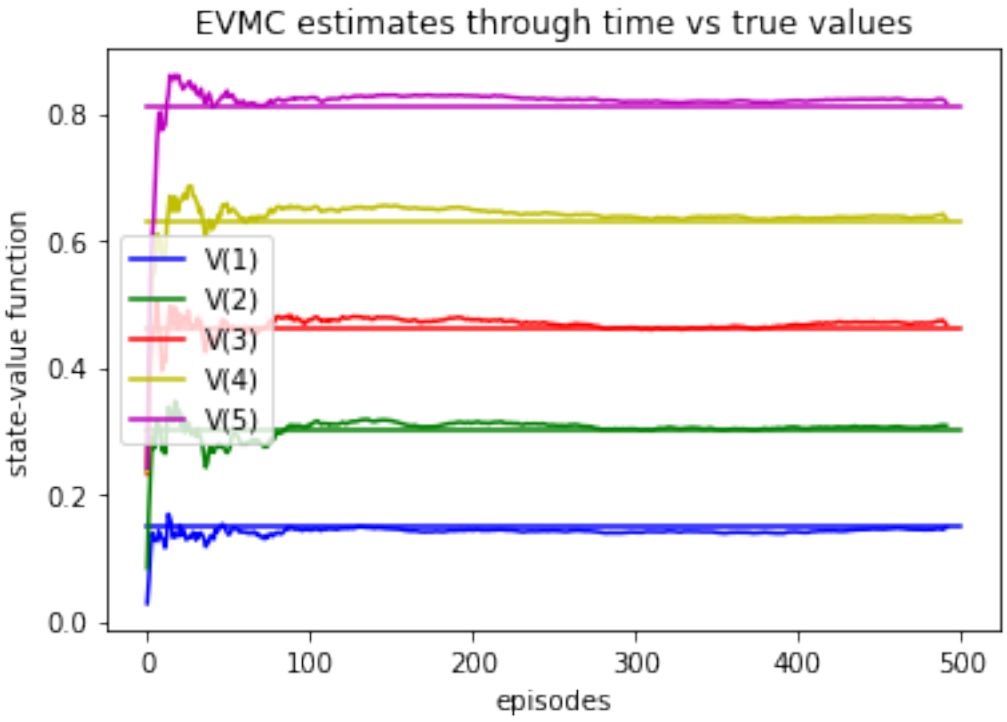


Figure 27: MC-EVMC vs True Values

Upon observation we note that in both FVMC and EVMC cases, the values of later states (closer to the final state) are generally over-estimated and earlier states (closer to the failure state) are underestimated.

This may be attributed to the fact that since MC is based on averaging, and it is far more likely to reach the final state from states closer to it, MC overestimates states that are close to the final state and vice-versa. We also note that EVMC tends to converge faster and is smoother than FVMC. This may suggest that considering all visits effectively allows for better averaging than just the first visit since it encodes more information.

7. We show the plot of TD for non-terminal states in Figure 28

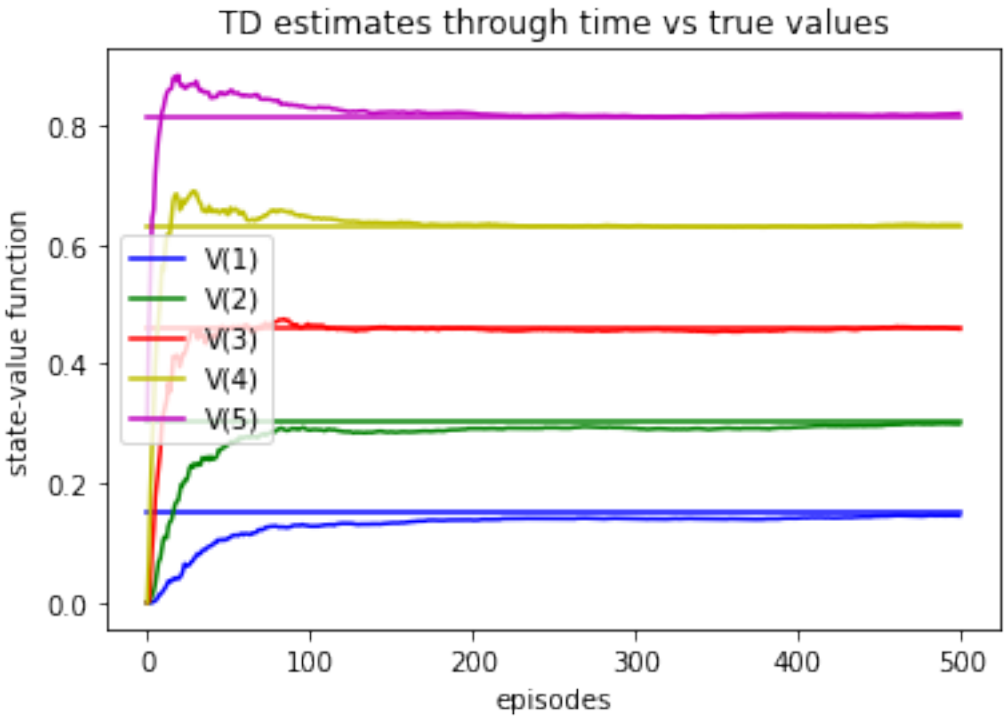


Figure 28: TD vs True Values

We observe that TD is the one which reaches closest to the true values among the techniques tested, it also has much less variance than MC methods. It also converges in a smoother manner than MC estimates. The over and under estimate issues are similar to MC methods but mostly affect the first few hundred episodes.

8. We show the plot of MC-FVMC (log-scale) for non-terminal states in Figure 29
9. We show the plot of MC-EVMC (log scale) for non-terminal states in Figure 30

   Plotting in log scale allows us to better analyze the behaviour for initial episodes. We note that both FVMC and EVMC start at underestimates for all states (EVMC more so) and slowly catch up to true values. FVMC shows more variance and spikes compared to EVMC. FVMC also has rather high peaks of overestimates which is much lower in EVMC plots.

10. We show the plot of TD (log scale) for non-terminal states in Figure 31

    In case of TD the plot start at gross underestimates with several starting at 0. They reach true values slowly with the lower states never going through an overestimate phase. The graphs are also very smooth compared to MC algorithms.

11. When we compare the three techniques we note that TD performs the best and reaches closest to the true values. Among MC, EVMC seems to perform better than FVMC. TD also shows the least amount of variance but takes the longest to converge (but as stated earlier they converge to better estimates). As such, it feels like if we have the luxury of running large number of episodes TD is a better choice. MC might be worth it if we cannot go for large number of episodes. Among MC, EVMC should be preferred over FVMC, since they converge at about the same time but EVMC has less variance.

12. We show the plot of target values for state 3 for MC-FVMC in Figure 32

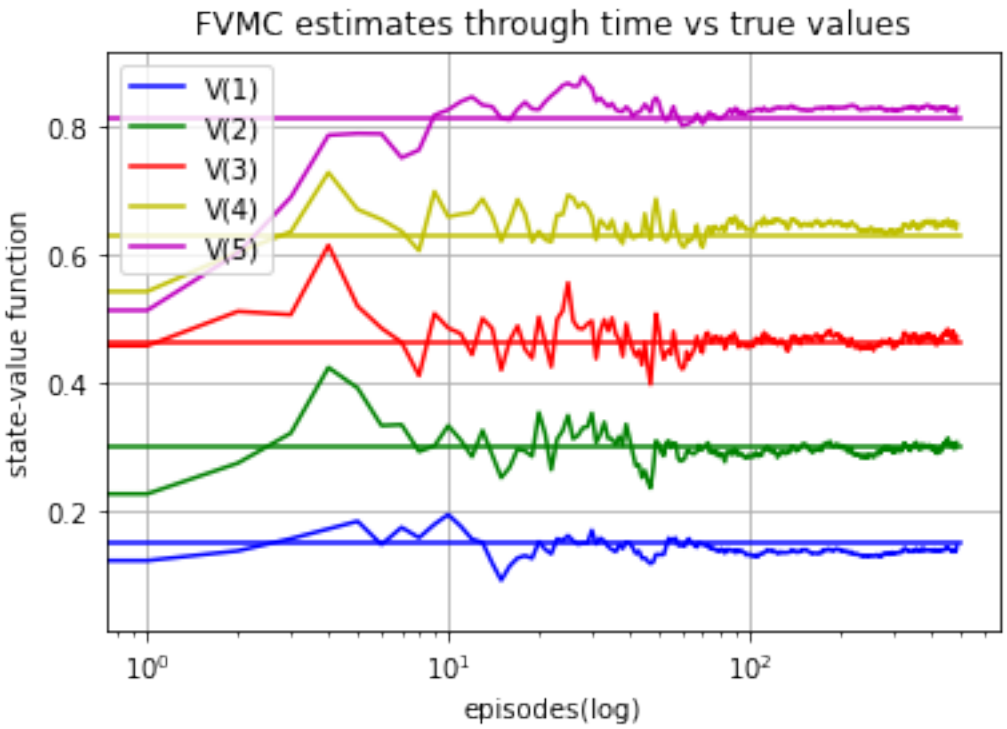


Figure 29: MC-FVMC vs True Values (Log scale)

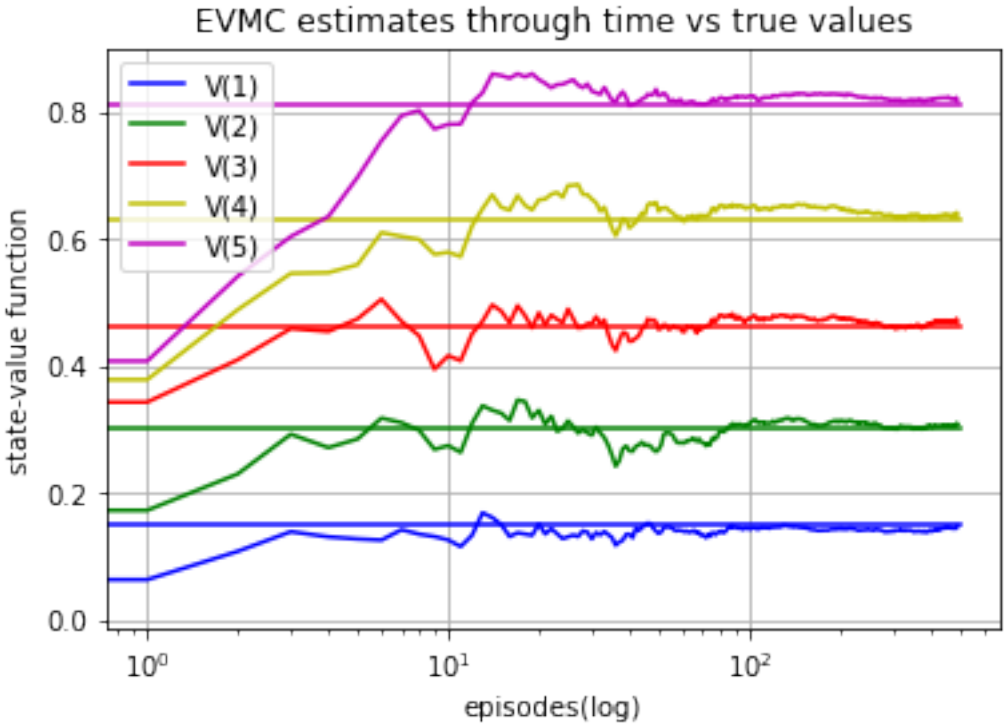


Figure 30: MC-EVMC vs True Values (Log scale)

13. We show the plot of target values for state 3 for MC-EVMC in Figure 33

    We note that the plot for EVMC is more polarized FVMC. This is because EVMC takes into account all visits and present a truer estimate of target value and is thus close to either 0 or 1.

14. We show the plot of target values for state 3 for TD in Figure 34

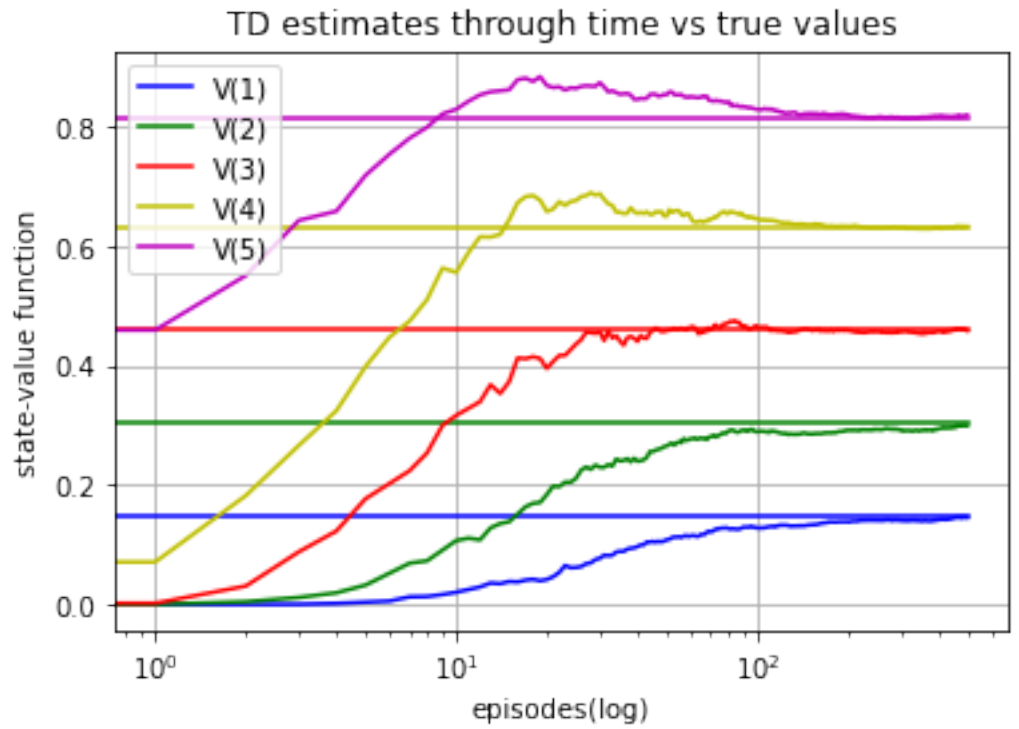


Figure 31: TD vs True Values (Log scale)

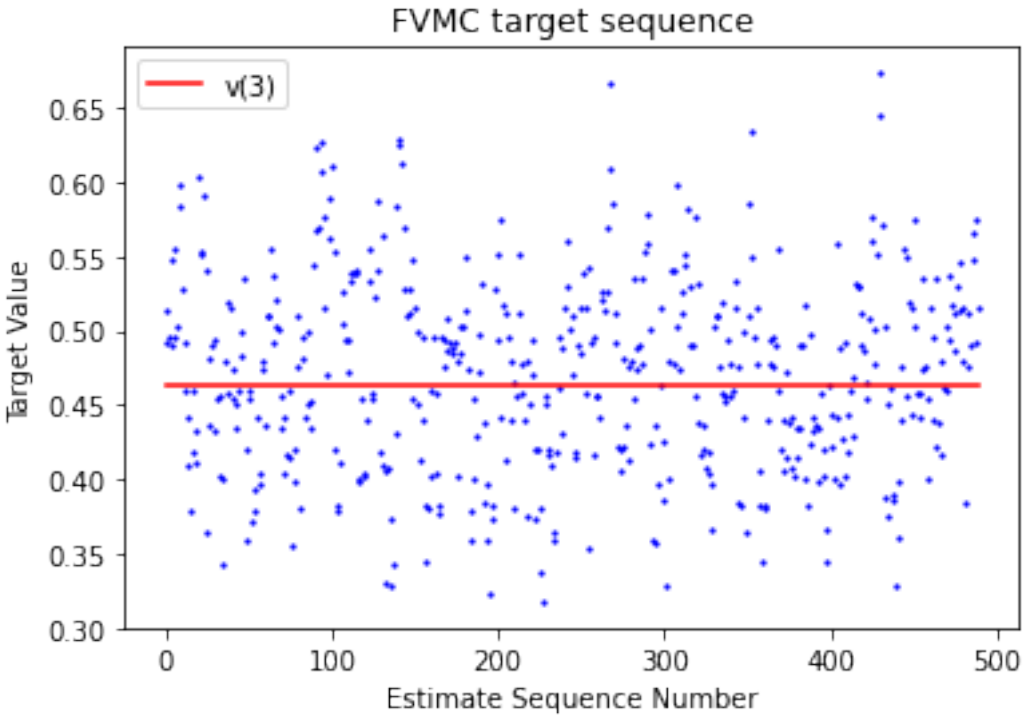


Figure 32: Target values for MC-FVMC

For TD the target value estimates are initially similar to FVMC and then slowly converge.

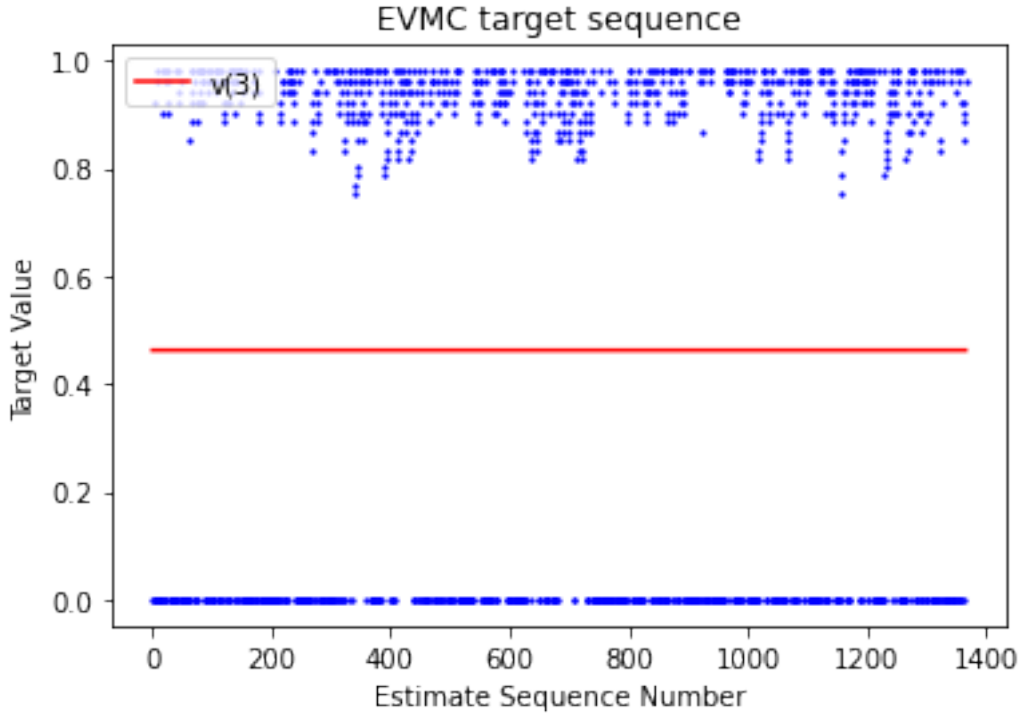


Figure 33: Target values for MC-EVMC

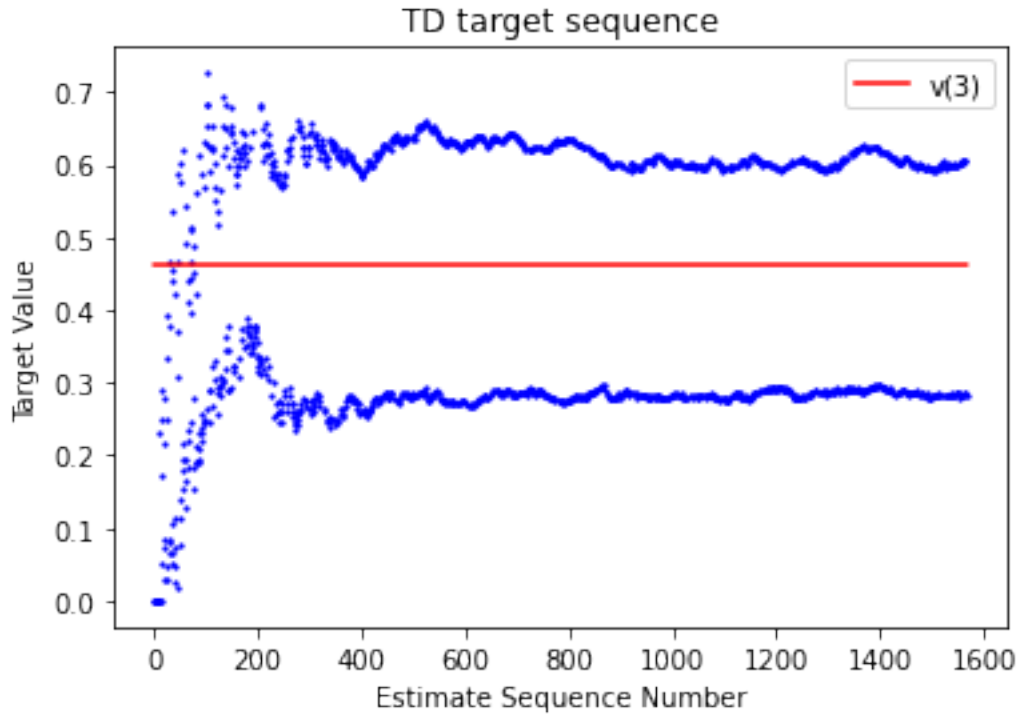


Figure 34: Target values for TD