CS698R: Deep Reinforcement Learning

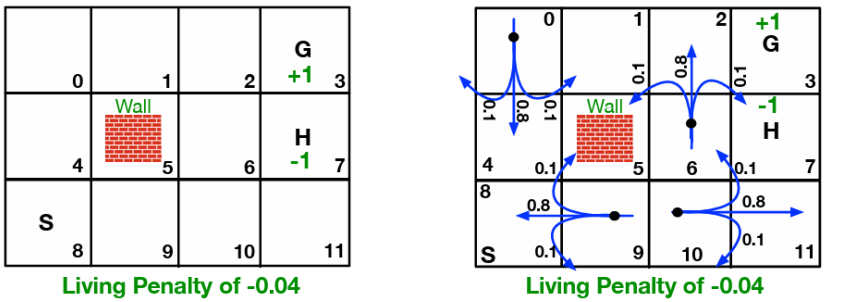
Mid-Semester Exam

Name: Rajarshi Dutta
Roll NO.: 200762

Solution to Problem 1: Random-Maze Environment Implementation

Problem 1: Notebook for problem 1
Problem 2: Notebook for Problem 2
Problem 3: Notebook for Problem 3

The given environment described below is a random maxe environment consisting of a grid of size (3 x 4), a highly stochastic environment with 11 states: **2** terminal states namely **goal** and **hole** states having rewards of +1 and -1 respectively. All the other non-terminal states has a reward of -0.04. For intended action, the agent has a 80% probability of taking that action and remaining 20% is distributed equally between every pair of orthogonal actions. The agent starts from state 8 with a a discoun factor of $\gamma = 0.99$. The figure(1) is shown below:



(a) Maze Environment (b) Maze Environment Transitions

Figure 1: Random Maze Environment

1. The evironment is created and the seed is set to **123**. The trajectory generated based on the given environement constraints on setting the required seed is shown in the table below:

Current State	Action	Reward	Next State	Done
8	2	-0.04	9	False
9	1	-0.04	10	False
10	1	-0.04	6	False
6	0	-0.04	6	False
6	0	-0.04	6	False
6	1	-0.04	2	False
2	3	1.0	3	True

Table 1: Transitions table

2. The second trajectory generated by setting the seed to 8888 is shown below:

State	Action	Reward	Next State	Done
8	1	-0.04	4	False
4	3	-0.04	8	False
8	0	-0.04	8	False
8	3	-0.04	9	False
9	3	-0.04	10	False
10	2	-0.04	11	False
11	2	1.0	7	True

Table 2: Transitions Table

Solution to Problem 2: RME Optimal Policy via Dynamic Programming

The solution provides the optimal values for all the states calculated using **Policy Iteration** and **Value Iteration** algorithm. The major difference is in **Policy iteration**, the agent calculates the values for all states initially using an adversarial policy and then it improves the policy from the q function by choosing the action that maximizes the expected return in each state. These two processes are repeated until the values converge. Whereas for the case of **Value Iteration**, the agent calculates the maximum possible return for each action in each state and then dynamically updates the value of each state with that maximum value.

1. For this question, **theta** is taken to be 10e-10. The **policy iteration** firstly calculates the value functions for the given environment using the adversarial policy and then performs **policy improvement** by updating the **Q values**. The initial adversarial policy was taken randomly from the set of all possible actions. The adversarial policy used for the given **Random Maze environment** is shown below:

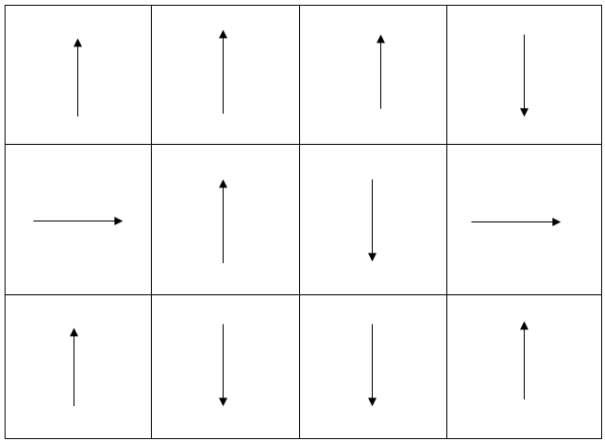


Figure 2: Initial Adversarial Policy

After performing **policy iteration** using the θ as 10e-10 and γ as 0.99, it was found that the optimal policy was reached within **3 iterations**. The optimal policy for the **Random Maze Experiment** is illustrated below:

The optimal state values for **Policy Iteration** are = [**0.824, 0.892, 0.954, 0, 0.764, 0, 0.688, 0, 0.697, 0.639, 0.606, 0.381**]

2. The second part is related to the implementation of **Value iteration** algorithm. This algorithm evaluates the value function until it converges to the optimal value function. Once the optimal values are obtained, the optimal policy for the environment can be calculated from them.

After performing **value iteration** using the θ as 10e-10 and γ as 0.99, it was found that the optimal policy was reached within **25 iterations**. The optimal policy for the **Random Maze Experiment** is illustrated below:

The optimal state values for **Value Iteration** algorithm are = [**0.824, 0.892, 0.954, 0, 0.764, 0, 0.688, 0, 0.697, 0.638, 0.606, 0.381**]

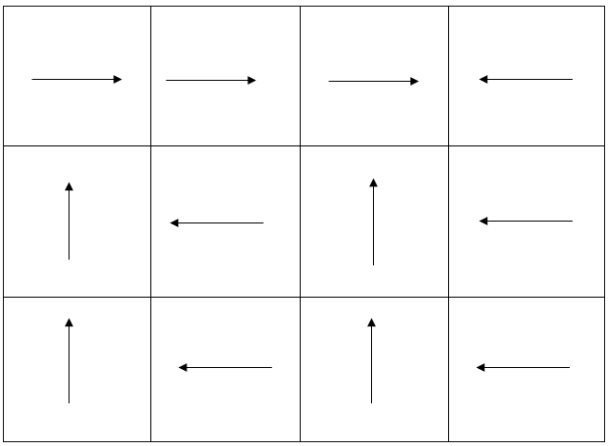


Figure 3: Final optimal Policy (Policy Iteration)

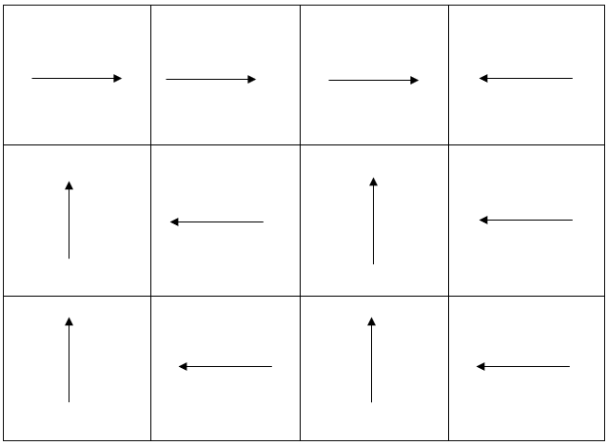
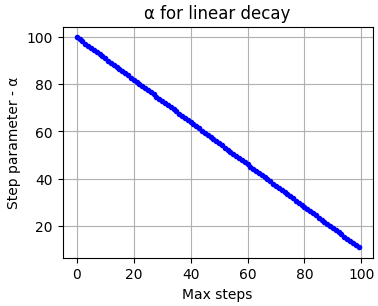


Figure 4: Final optimal Policy (Value iteration)

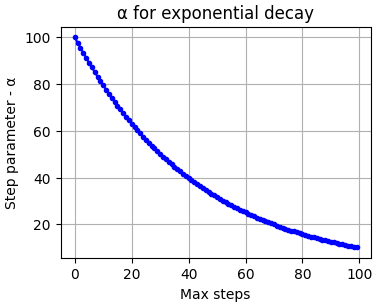
Solution to Problem 3: RME Prediction with MDP Unknown

1. The environment resets the agent to the state 8 of the all the nodes every time a new episode is initiated. The optimal state-value functions obtained though is independent of the starting point of the agent. The generateTrajectory() function generates a new episode containing the experience tuple (state, actions and indicators whether epsiode has terminated or not) and MC and TD plots are obtained y varying the step parameter implemented in the function decayAlpha().

2. The plots for linear and exponential decay of the step parameter is shown below. The **initial value** is set to and **final value** is set to , **max steps** is set to **100**



(a) Linear Decay



(b) Exponential Decay

3. The **FVMC** plots for all the non terminal states are shown below. From these plots, we can infer that the values for state 10 and 11 doesn't change at all and the variations of all the state values are quite noisy especially because of the episodic nature of Monte Carlo algorithm. The peaks are comparatively lower in the case of **EVMC** as compared to **FVMC** algorithm and there is no significant difference between the values of the non terminal states in both the methods

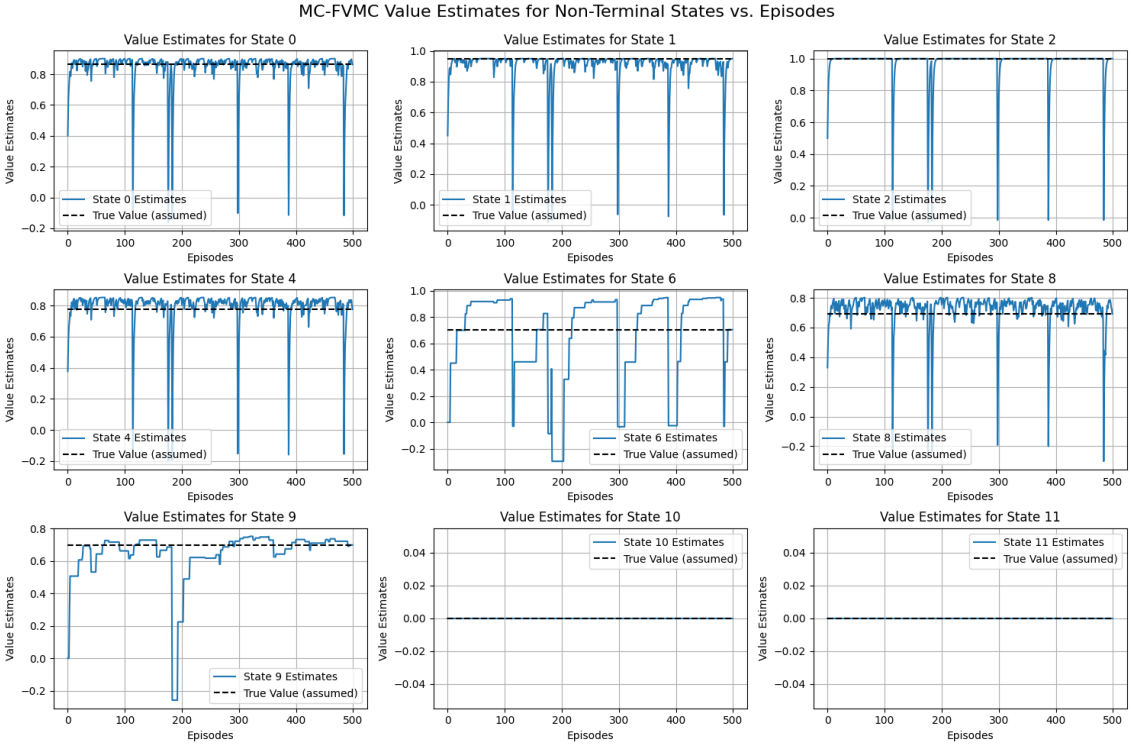


Figure 6: FVMC plots for constant $\alpha = 0.5$

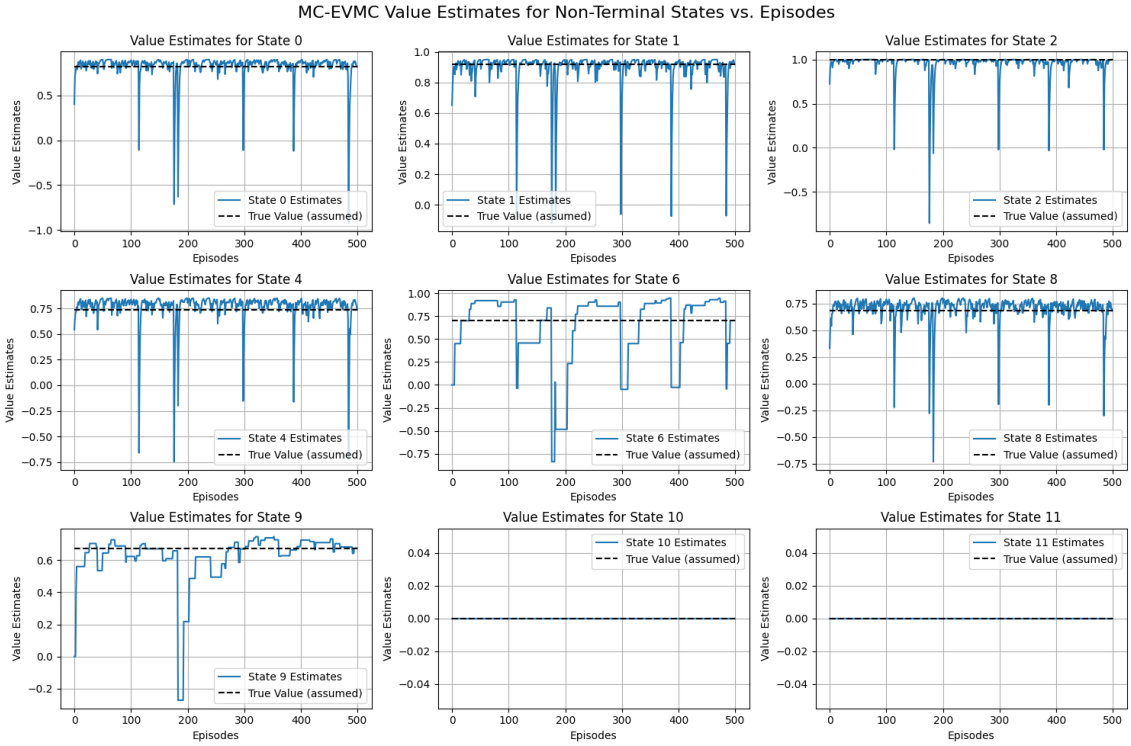


Figure 7: EVMC plots for constant $\alpha = 0.5$

4. The next set of plots depicts the **TD** estimates of all non terminal states. These plots are relatively smoother as compared to the **MC** plots due to the step nature of the algorithm. There are much less overshoots and there is a significant difference in the vairation of **state 9** as compared to that in the case of **MC** plots.

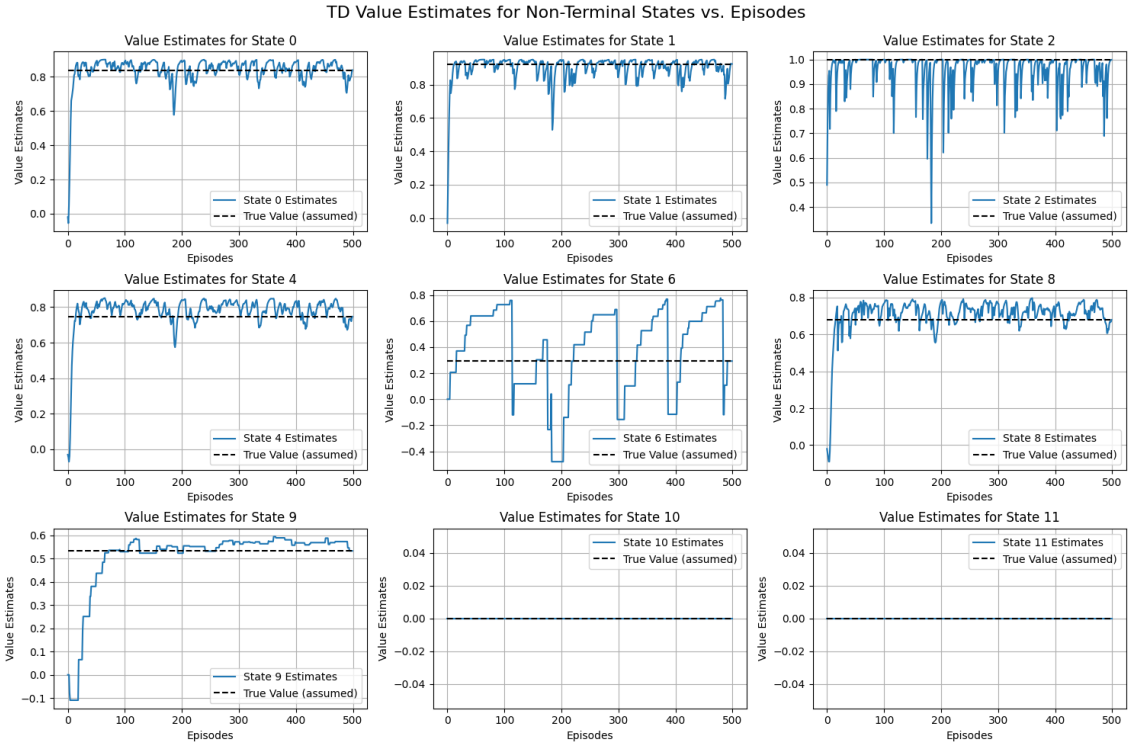


Figure 8: TD plots for constant $\alpha = 0.5$

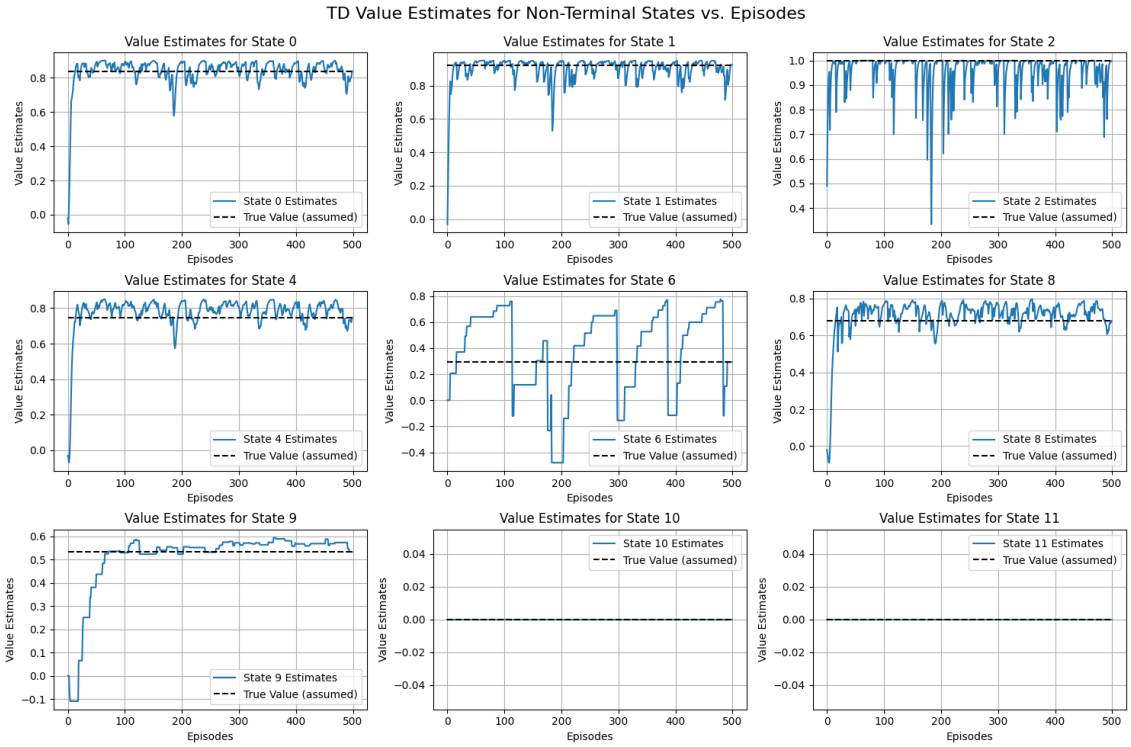


Figure 9: TD plots for constant $\alpha = 0.5$

5. The next plots depicts the **n Step TD algorithm** where in which is the parameter that determines the number of steps that we want to look ahead before updating the value function. For the case of low n, the updates are based on the immediate reward and the estimated value of the next state, leading to high bias because it doesn't consider the actual outcomes of future actions beyond the immediate next step also this leads to quicker convergence since for high values of n, the agent accumulates rewards over broader states and slowly high n step TD approaches the traditional MC algorithm.

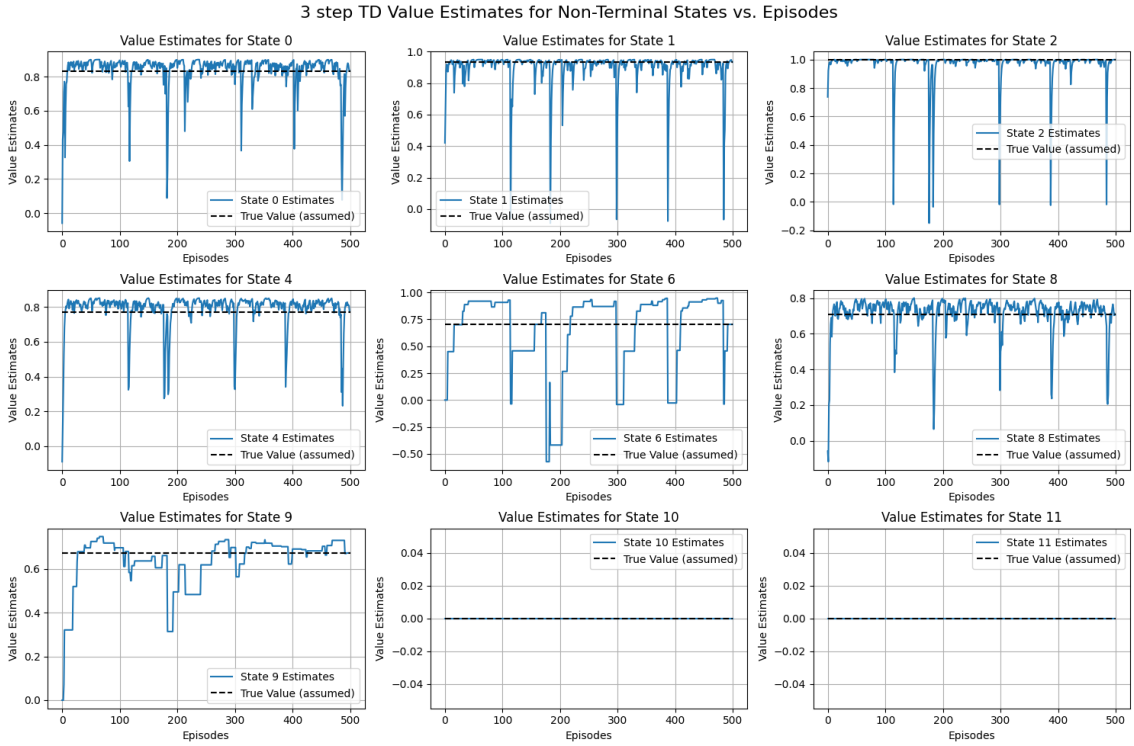


Figure 10: n step TD plots for constant $\alpha = 0.5$ and n = 3

6. The next set of plots depicts the **TD Λ algorithm**. The plot depicts that the algorithm instantly converges to the optimal value which is indicated by no variations in the values of the states in the environment.

7. The next figures depicts the averaged plots for **EVMC**, **FVMC**, **TD**, **3 step TD** and **6 step TD**, **TD (0.3)** algorithms averaged over varying seeds ranging from 100 to 300.The averaged estimates are over**500** episodes with α decaying from **0.5** to **0.01**. These plot eliminates the noise and its clear that TD plots converge way more quickly with less variance than Monte Carlo estimates.

8. The next figures depicts the averaged plots for **EVMC**, **FVMC**, **TD**, **3 step TD** and **6 step TD**, **TD (0.3)** algorithms averaged over varying seeds ranging from 100 to 300.The averaged estimates are over**500** episodes with α decaying from **0.5** to **0.01**. These plot eliminates the noise and its clear that TD plots converge way more quickly with less variance than Monte Carlo estimates. The episodes are taken in the log scale which helps us to zoom into the initial stages of the learning process.

9. The **FVMC-target** and **EVMC-target** plots for state 4 is shown below. The plot depicts a higher variance in the beginning of episodes and the values finally becomes stationary which indicates that the learning is almost complete. The values don't show an upward or downward trend which denotes that the algorithm is stable and not diverging and there are no sudden explorations in later episodes. Since, **Monte Carlo estimation** targets involves calculations of returns after an entire episode, the plot also denotes sufficient exploration of the environment by the agent.

10. The **TD-target** plot for state 4 is shown below. The target plot depicts that the the **TD-targets** initially have a high variance which denotes that the agent is beginning to explore and dynamics. The agent is updating its estimates over a wide variety of possible experiences. The faster convergence also denotes that the step parameter is comparatively good otherwise the values might have converged slower. Since, the target values come closer to each other, this indicates that the learning is almost complete and the targets have converged to a final value

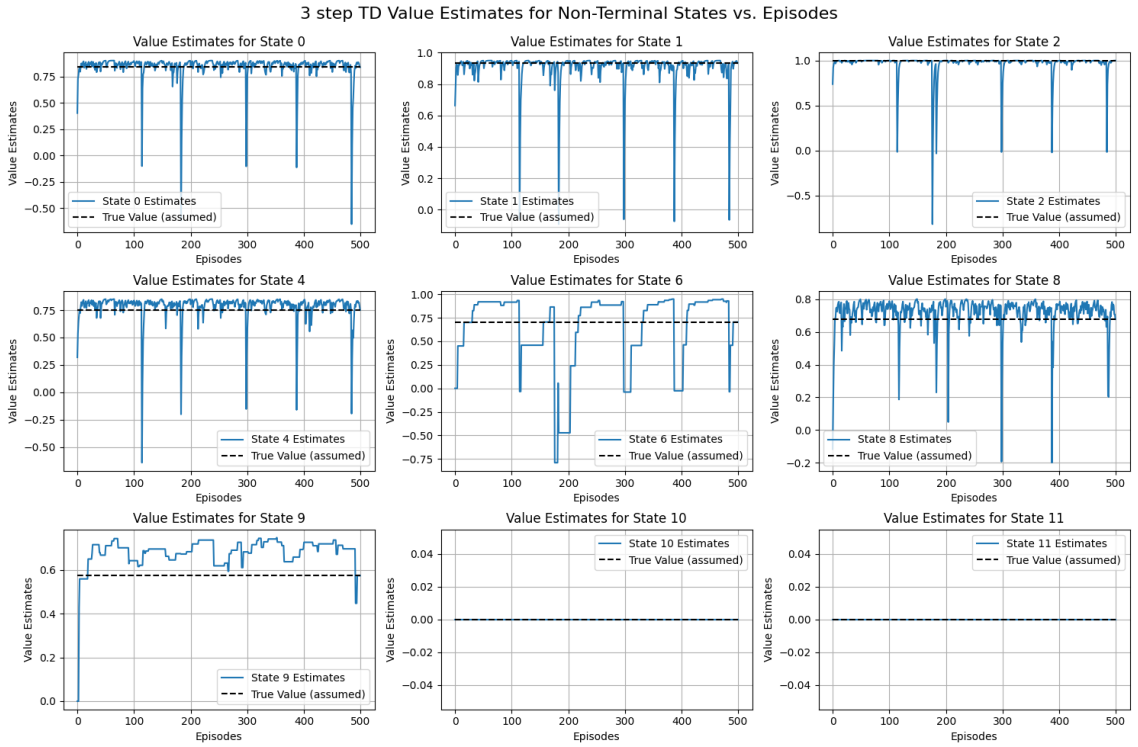
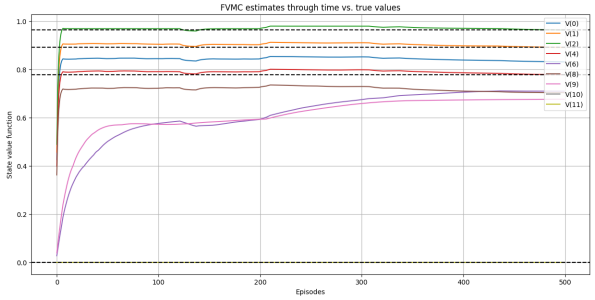
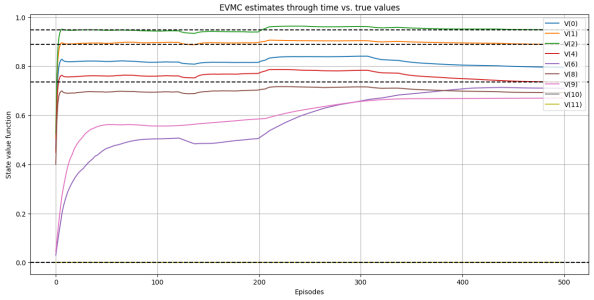


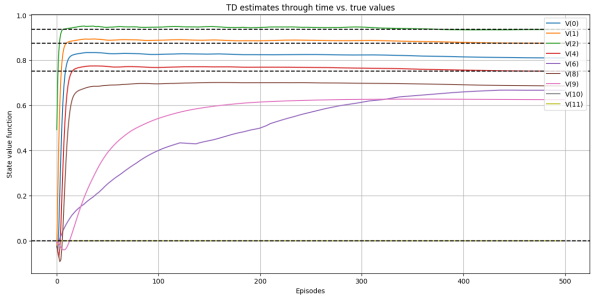
Figure 11: n step TD plots for constant $\alpha = 0.5$ and n = 6



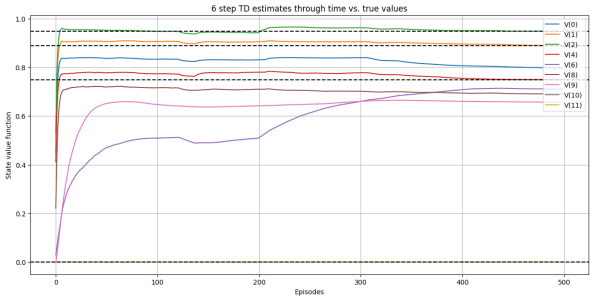
(a) FVMC Averaged out value estimates over 500 episodes



(b) EVMC Averaged out value estimates over 500 episodes

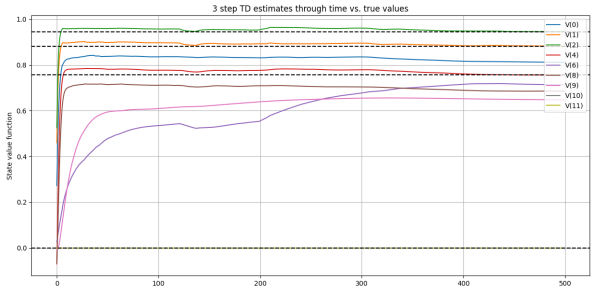


(a) TD Averaged out value estimates over 500 episodes

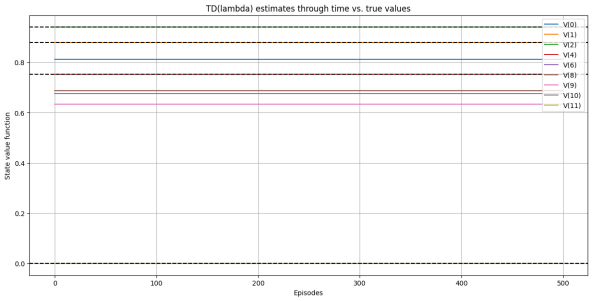


(b) 6 Step TD Averaged out value estimates over 500 episodes

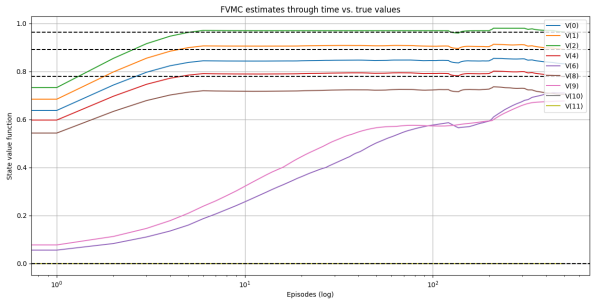
11. A comparative analysis of the algorithms has been incorporated into a bar plot depicting the optimal values for all of the non terminal states in the **Random Mazke Environment**. The figure is illustrated below:



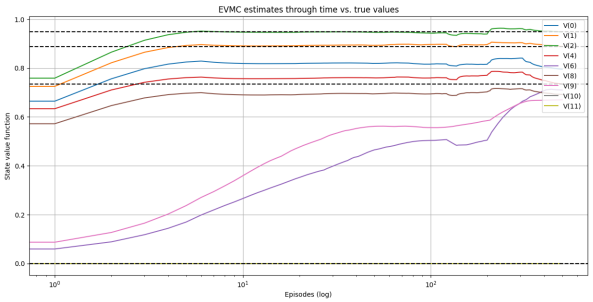
(a) 3 Step TD Averaged out value estimates over 500 episodes



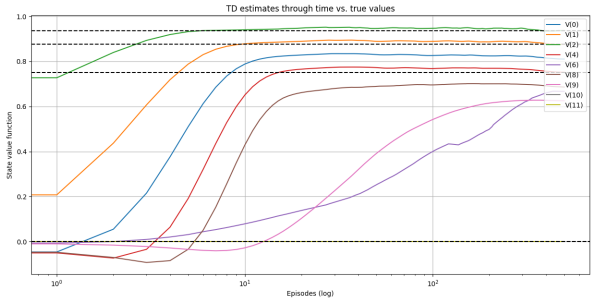
(b) TD(0.3) Averaged out value estimates over 500 episodes



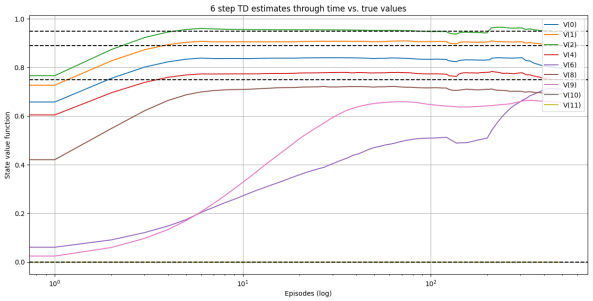
(a) FVMC Averaged out value estimates over 500 episodes (log scale)



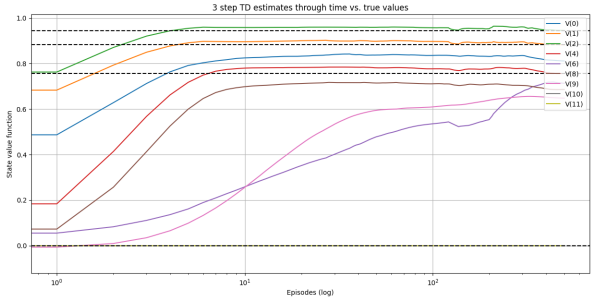
(b) EVMC Averaged out value estimates over 500 episodes (log scale)



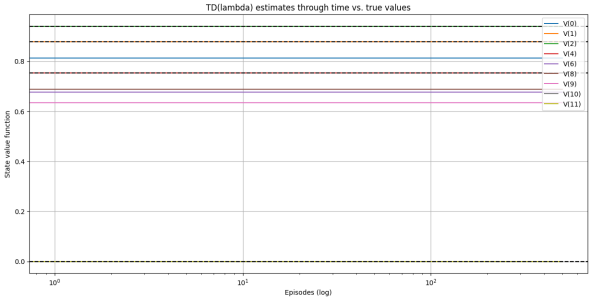
(a) TD Averaged out value estimates over 500 episodes (log scale)



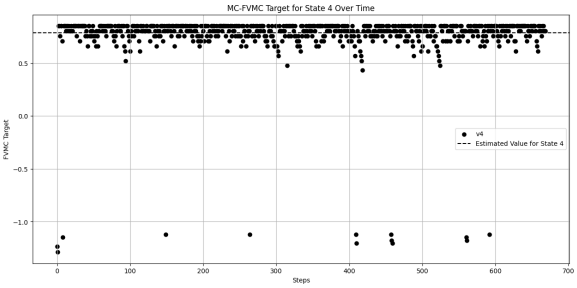
(b) 6 Step TD Averaged out value estimates over 500 episodes (log scale)



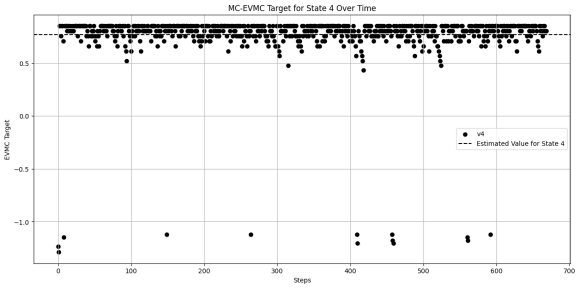
(a) 3 Step TD Averaged out value estimates over 500 episodes



(b) TD(0.3) Averaged out value estimates over 500 episodes



(a) FVMC-Targets for state 4



(b) EVMC Targets for state 4

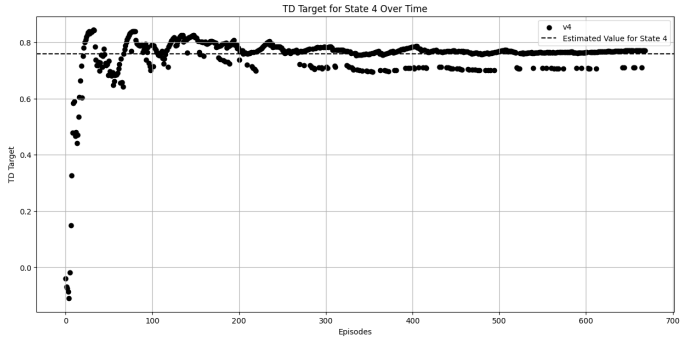


Figure 19: TD Targets for state 4

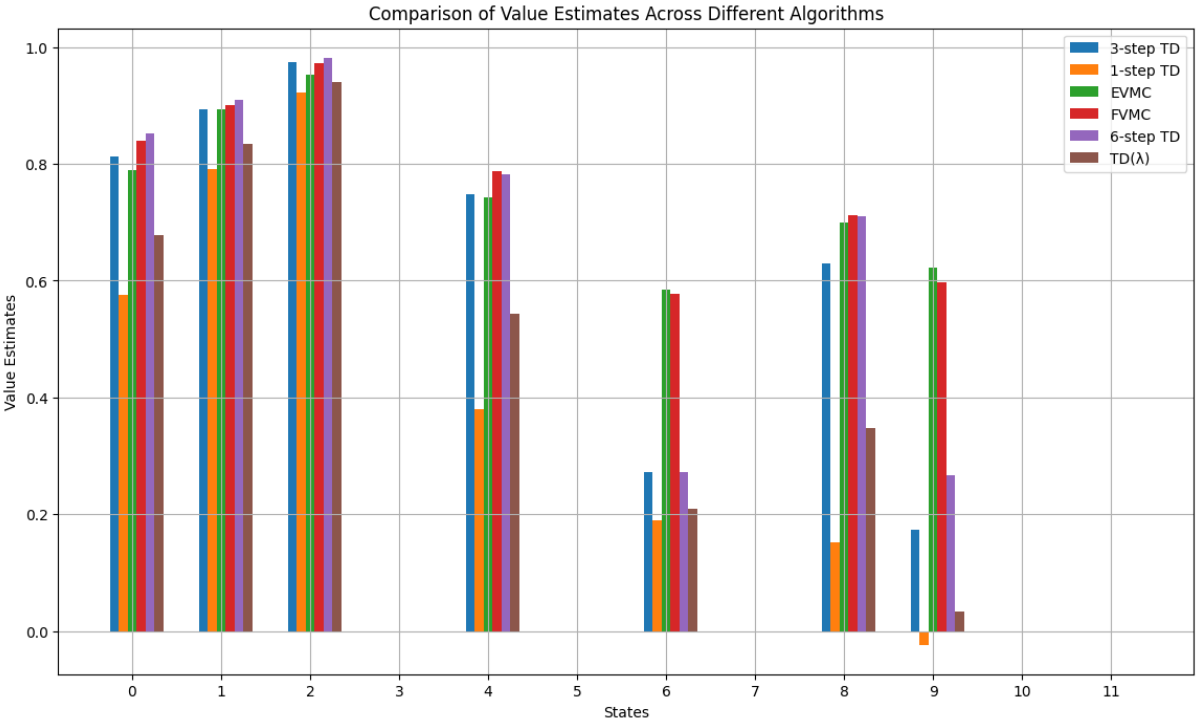


Figure 20: Optimal Values of states for different algorithms