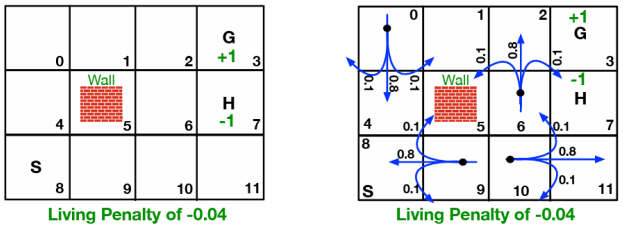
CS780: Deep Reinforcement Learning

Assignment #2

Name: Rajarshi Dutta
Roll NO.: 200762

The given environment(1) described below is a random maze 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 discount factor of $\gamma = 0.99$. The figure(1) is shown below:



(a) Maze Environment (b) Maze Environment Transitions

Figure 1: Random Maze Environment

Solution to Problem 1: Monte Carlo Control

Monte Carlo Control is an off-policy algorithm for estimation of the optimal policy of the environment without requiring the environment's model. This is essentially an episodic learning algorithm which accumulates the reward fro value calculation over an entire episode and uses the estimated state values for calculation of the optimal policy. The policy is generally calculated via the *epsilon*-greedy algorithm.

1. The given plot(2) depicts the state values for the non terminal states in the given random walk environment averaged over **50** different instances of the Random Maze Environment. The number of episodes chosen are **2000** and the seeds are chosen ranging from **100** to **150**. The plot shows that all of the states values initially starts below 0 and gradually converge towards their true estimates. The curves are relatively smooth since the estimates are averaged over various instances.

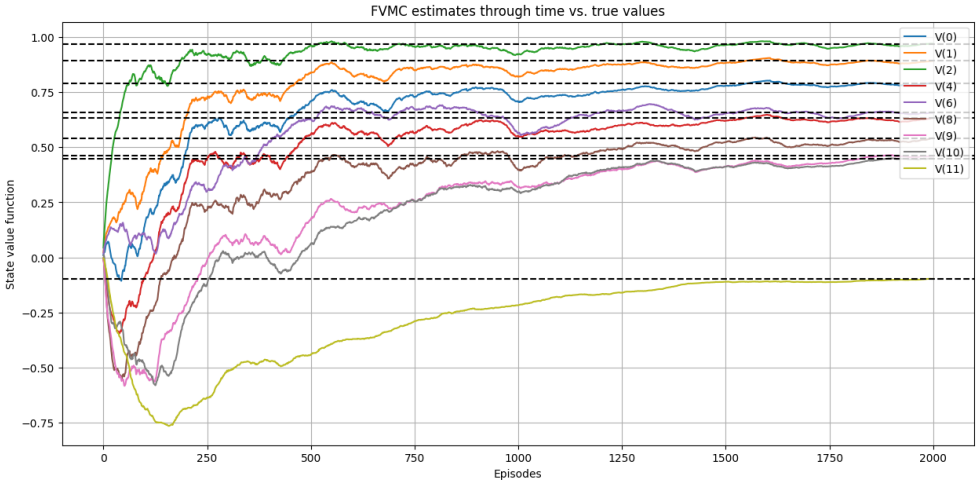


Figure 2: Monte Carlo Control value estimates vs Episodes

2. The next plot(3) depicts the Q function evolution over different episodes of the environment averaged over 50 instances. The 4 subplots depicts the averaged q values over episode evolution for different actions permissible in the given random maze environment.

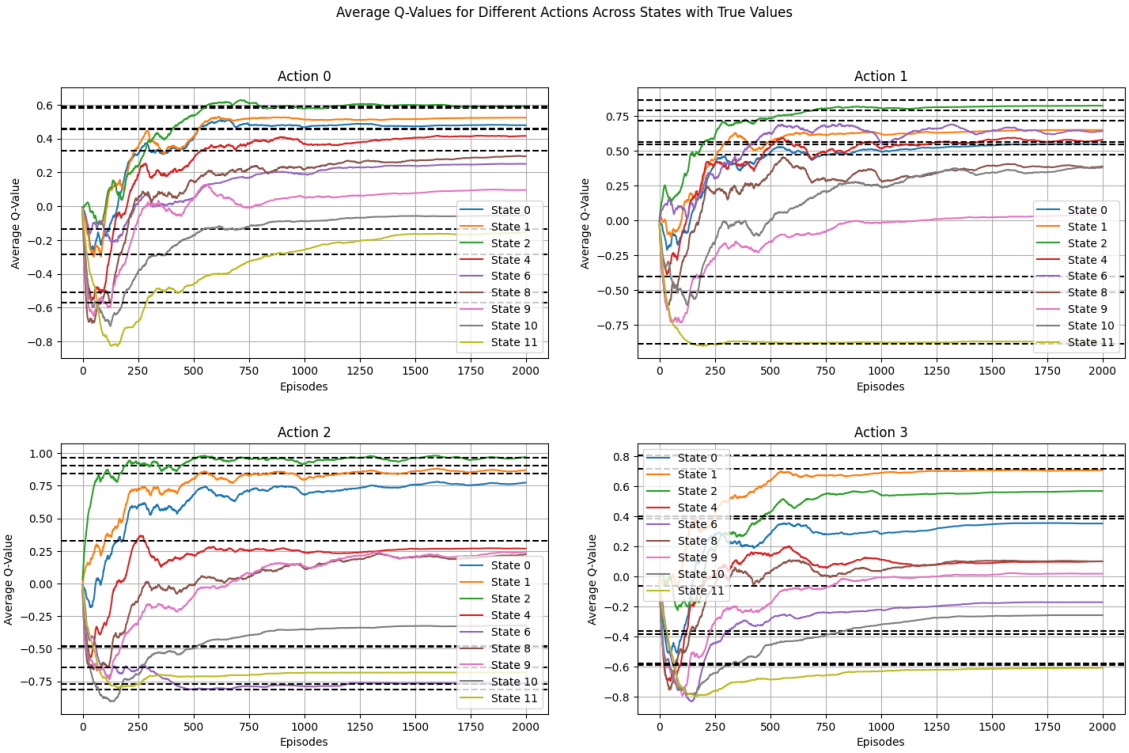


Figure 3: Monte Carlo Control Q value estimates vs Episodes

3. The plots are averaged over seeds ranging from **100** to **150** where on every start of episode, the environment is reset using an episode seed calculated from global seed.

4. The next plot(4) illustrates the optimal policy calculated via the Monte Carlo Control algorithm for the given environment.

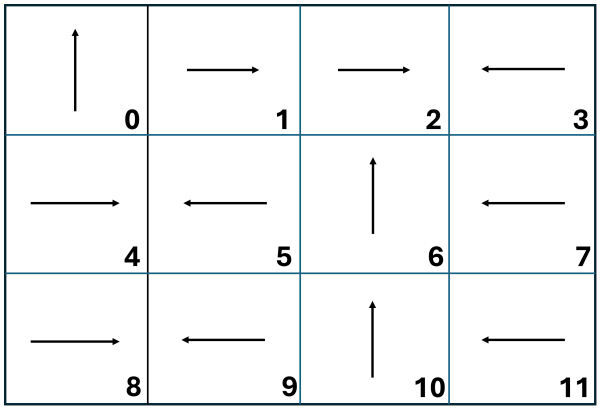


Figure 4: Optimal policy using Monte Carlo Control

5. The hyperparameters used for the algorithm mainly includes the `numEpisodes` which is set to 2000, the `decaytype` is set to **exponential**, `mctype` is set to **EVMC**, the discount factor (γ) is set to 0.99, the `maxSteps` is set to 100 and the `global_seed` is set to 123.

6. As observed from the plots, the convergence towards the true state values takes a lot of episodes in the case of Monte Carlo Control algorithm especially for the states 8, 9, 10, 11 etc. The optimal value for the state 11 seems to converge to a negative value at the end of 2000 episodes.

Solution to Problem 2: SARSA (TD Control)

SARSA (TD Control) is an example of online learning where the agent learns from every experience it gathers in each time step while interacting with the given stochastic environment. Also a single policy is used to make decisions by the agent which is also used for data generation. The agent gradually assess its actions and learns to improve the policy which leads to convergence. Hence, this is an example of on-policy algorithm.

1. The given plot(5) depicts the state values for the non terminal states in the given random walk environment averaged over **50** different instances of the Random Maze Environment. The number of episodes chosen are **2000** and the seeds are chosen ranging from **100** to **150**. The plot shows that all of the states values initially starts below 0 and gradually converge towards their true estimates. The curves are relatively smooth since the estimates are averaged over various instances.

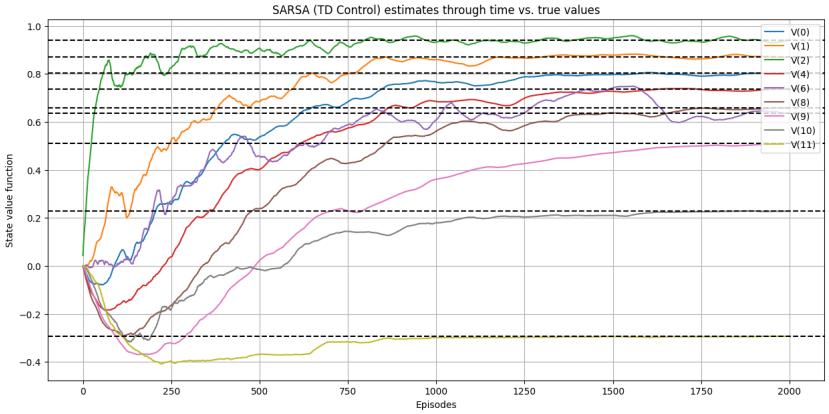


Figure 5: SARSA (TD Control) value estimates vs Episodes

2. The next plot(6) depicts the Q function evolution over different episodes of the environment averaged over 50 instances.

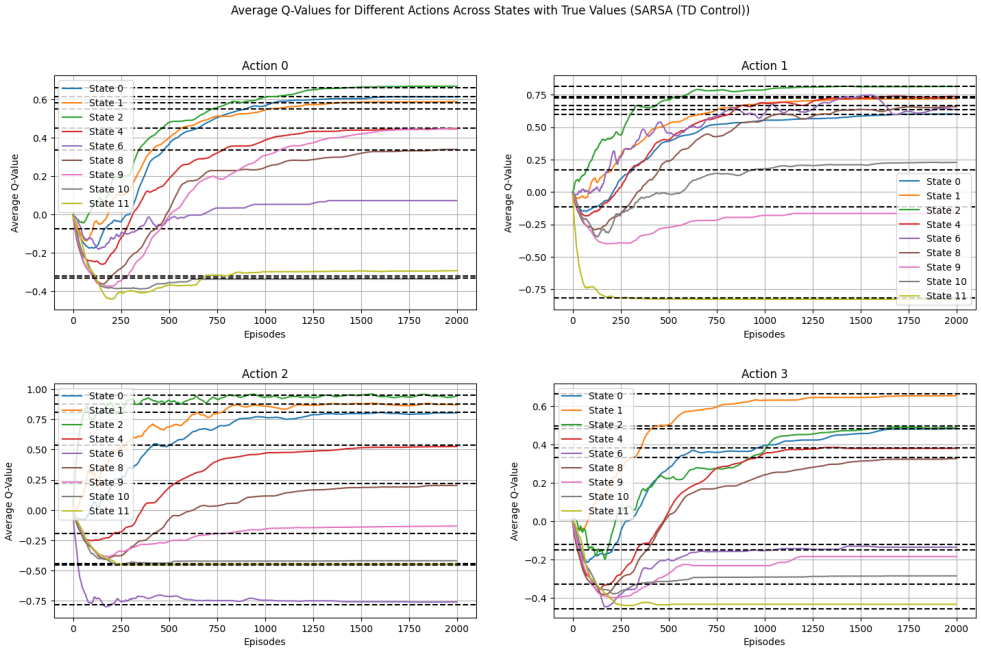


Figure 6: SARSA (TD Control) Q value estimates vs Episodes

3. The plots are averaged over seeds ranging from **100** to **150** where on every start of episode, the environment is reset using an episode seed calculated from global seed.

4. The next plot(7) illustrates the optimal policy calculated via the SARSA (TD Control) algorithm for the given environment.

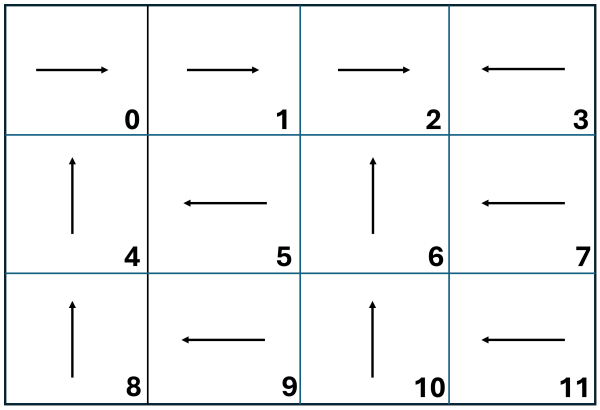


Figure 7: Optimal policy using SARSA (TD Control)

5. The hyperparameters used for the algorithm mainly includes the `numEpisodes` which is set to **2000**, the `decaytype` is set to **exponential**, the discount factor (γ) is set to **0.99**, the `maxSteps` is set to **100** and the `global_seed` is set to **123**.

6. We find that the state values evolution over 2000 episodes is way smoother as compared to that of the Monte Carlo Control estimates. The rate of convergence is also larger for the case of TD Control.

Solution to Problem 3: Q Learning

Q Learning is a model free off-policy algorithm that performs bootstrapping by directly evaluating the best policy while generating experiences using that policy. Here in every target updation, we greedily choose the action which maximizes the q function. Here, two different policies are used and the agent tries to find the optimal policy different fro the policy used for data generation from interactions with the given stochastic environment. These policies are named as target and behavioural policy respectively.

1. The given plot(8) depicts the state values for the non terminal states in the given random walk environment averaged over **50** different instances of the Random Maze Environment. The number of episodes chosen are **2000** and the seeds are chosen ranging from **100** to **150**. The plot shows that all of the states values initially starts below 0 and gradually converge towards their true estimates. The curves are relatively smooth since the estimates are averaged over various instances

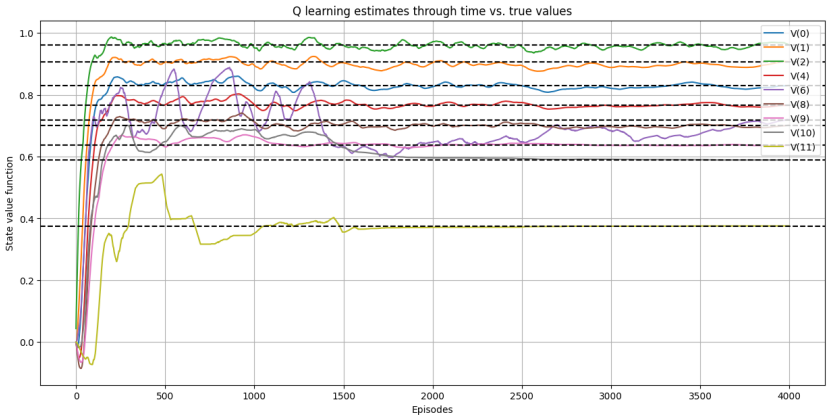


Figure 8: Q Learning value estimates vs Episodes

2. The next plot(9) depicts the Q function evolution over different episodes of the environment averaged over 50 instances.

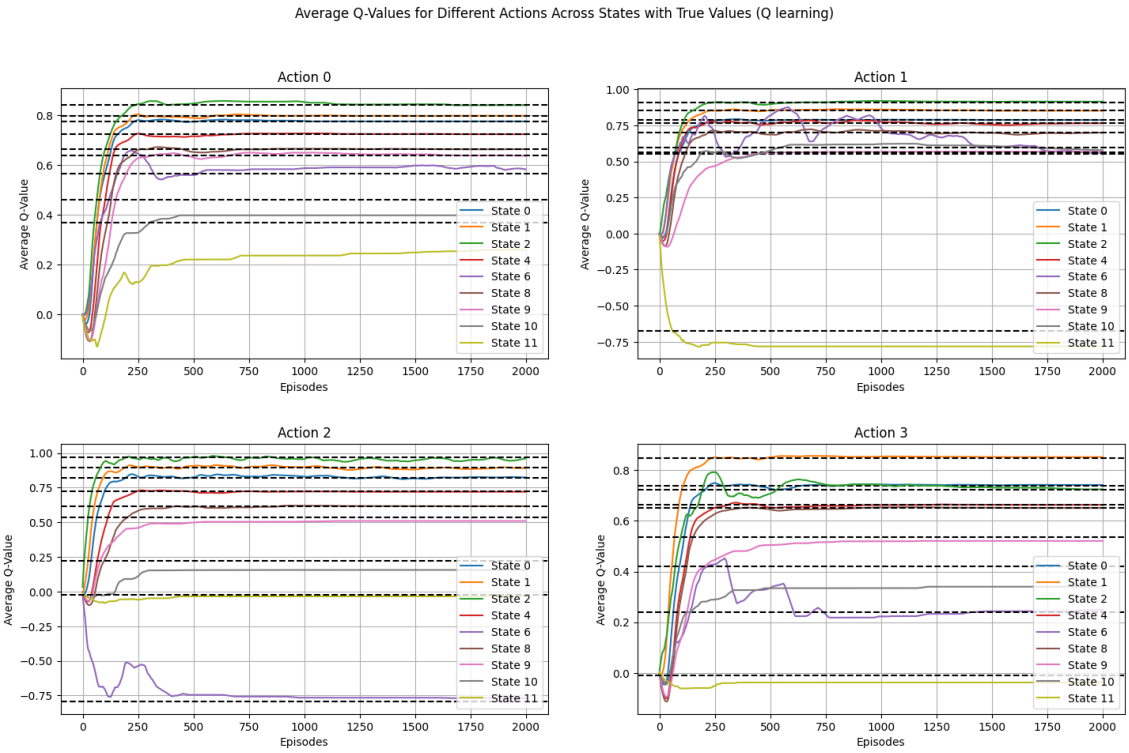


Figure 9: Q Learning Q value estimates vs Episodes

3. The plots are averaged over seeds ranging from **100** to **150** where on every start of episode, the environment is reset using an episode seed calculated from global seed.

4. The next plot(10) illustrates the optimal policy calculated via the Q Learning algorithm for the given environment.

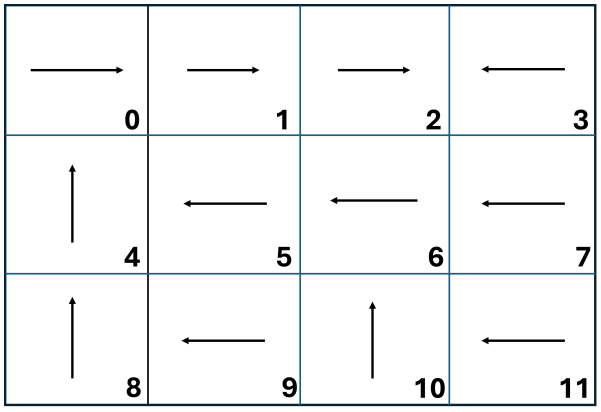


Figure 10: Optimal policy using Q Learning algorithm

5. The hyperparameters used for the algorithm mainly includes the `numEpisodes` which is set to **2000**, the `decaytype` is set to **exponential**, the discount factor (γ) is set to **0.99**, the `maxSteps` is set to **100** and the `global_seed` is set to **123**.

6. As observed from the plot, the state value estimates gradually approaches their optimal values way more rapidly as compared to both Monte Carlo Control algorithm and SARSA Control. But it is observed that the q value estimates seems to converge to positive values for all the states.

Solution to Problem 4: Double Q Learning

Double Q Learning essentially solves the overestimation problem of Q learning algorithm. This problem is referred to as the **Maximization Bias** problem which states that the same samples are used to estimate the best action as well as the action value function. Thus if an action's value is overestimated, then its overestimated value will be used as the target. Instead, we can have two different action-value estimates and the **best** action can be chosen based on the values of the first action-value estimate and the target can be decided by the second action-value estimate. Finally the average of the two q values are taken as the final q function.

1. The given plot(11) depicts the state values for the non terminal states in the given random walk environment averaged over **50** different instances of the Random Maze Environment. The number of episodes chosen are **2000** and the seeds are chosen ranging from **100** to **150**. The plot shows that all of the states values initially starts below 0 and gradually converge towards their true estimates. The curves are relatively smooth since the estimates are averaged over various instances

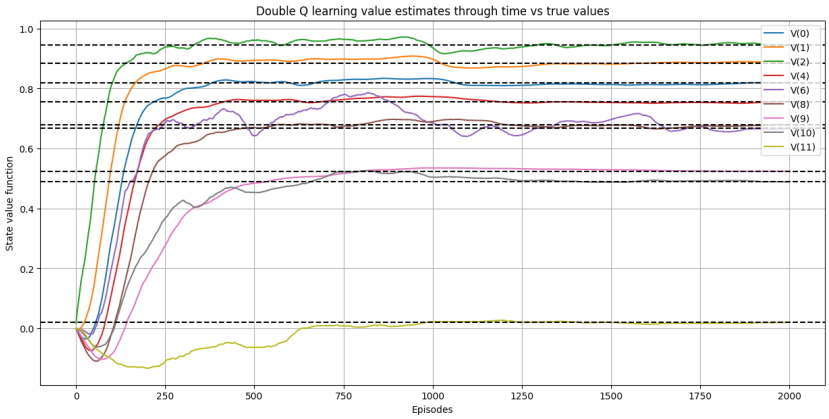


Figure 11: Double Q Learning value estimates vs Episodes

2. The next plot(12) depicts the Q function evolution over different episodes of the environment averaged over 50 instances.

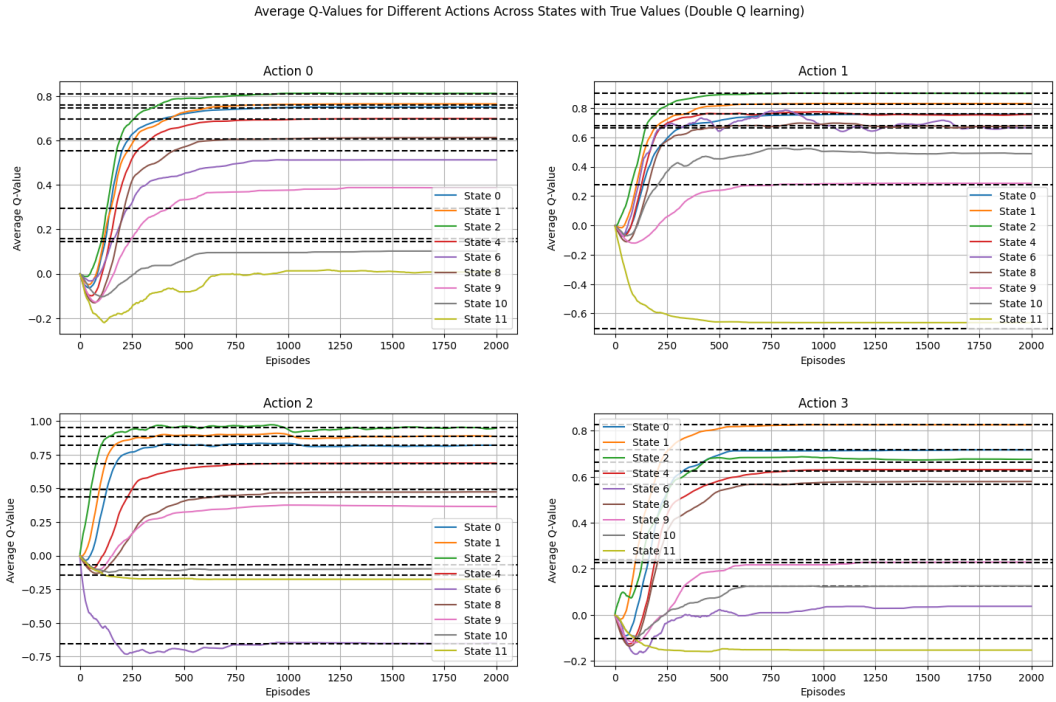


Figure 12: Double Q Learning Q value estimates vs Episodes

3. The plots are averaged over seeds ranging from **100** to **150** where on every start of episode, the environment is reset using an episode seed calculated from global seed.

4. The next plot(13) illustrates the optimal policy calculated via the Double Q Learning algorithm for the given environment.

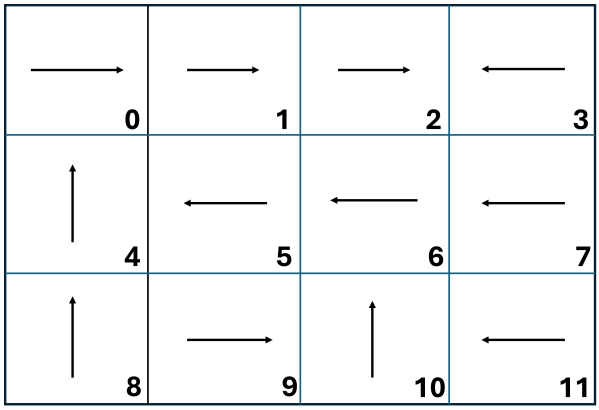


Figure 13: Optimal policy using Double Q Learning algorithm

5. The hyperparameters used for the algorithm mainly includes the `numEpisodes` which is set to **2000**, the `decaytype` is set to **exponential**, the discount factor (γ) is set to **0.99**, the `maxSteps` is set to **100** and the `global_seed` is set to **123**.

6. The value estimates for Double Q learning seems to converge a bit slower than that of the previous Q learning algorithm. The evolution of the value estimates is quite smooth for the case of double Q learning algorithm. The value estimates for state 11 seems to hover longer in the negative side in the initial episodes and gradually converges to positive value for the case of double Q learning algorithm.

Solution to Problem 5: Comparing Control Algorithms

1. The given plot(14) depicts the % Policy Success Rate calculated for each of the algorithms. It denotes the number of times the agent was able to reach the goal state for every policy calculated after each episode for each of the algorithms

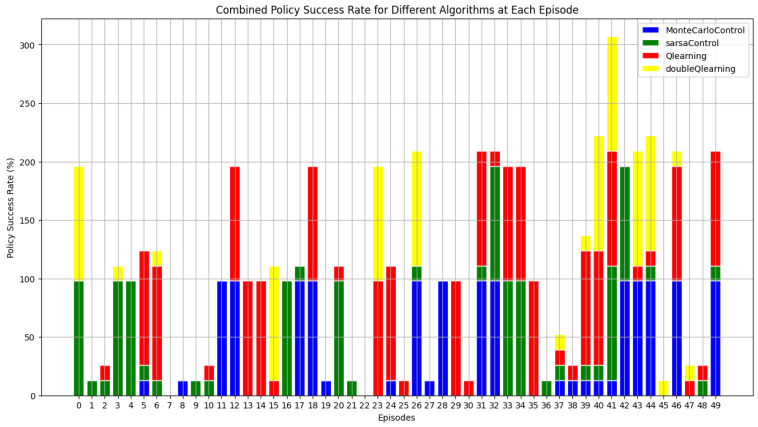


Figure 14: % Policy Success Rate vs Episode evolution

2. The plot(14) shows that for most of the episodes, **Double Q learning** is able to achieve close to 100% policy success Rate as compared to other algorithms. **Monte Carlo Control** initially fails to achieve good policy success rate but after subsequent training, it also achieves close to 100% policy success rate similar to **Q learning**. As observed from the plot, **Q learning** has the largest frequency in achieving good policy success rate after 50 episodes of training the agent.

3. The given plot(15) depicts the Expected Return starting from starting state (State **8**) vs Episodes for all the algorithms.

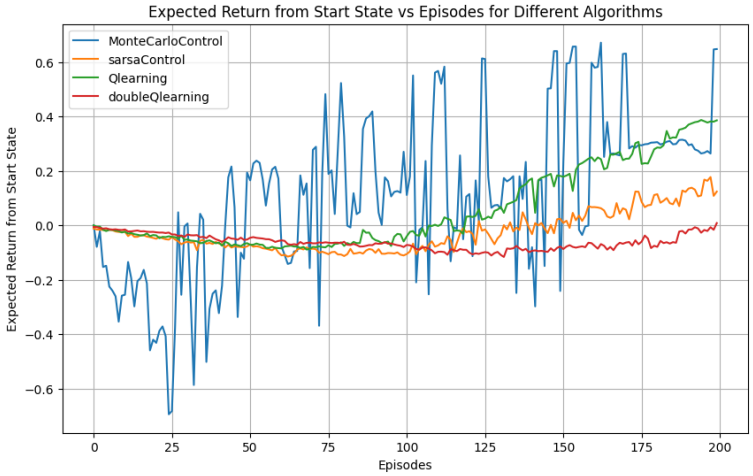


Figure 15: Expected Reward vs Episodes evolution

4. The plot(15) illustrates that the expected return for **Monte Carlo Control** has the largest oscillations compared to the other algorithms. The return evolution for **Double Q learning** is the smoothest and hovers around 0 whereas there is an increasing trend for the case of **Monte Carlo Control** and **Q Learning** algorithms with increasing episodes.

5. The given plot(16) depicts the State Estimation Error vs Episodes for all the algorithms which are calculated by taking the mean absolute error for all the state values from their optimal values for each algorithm.

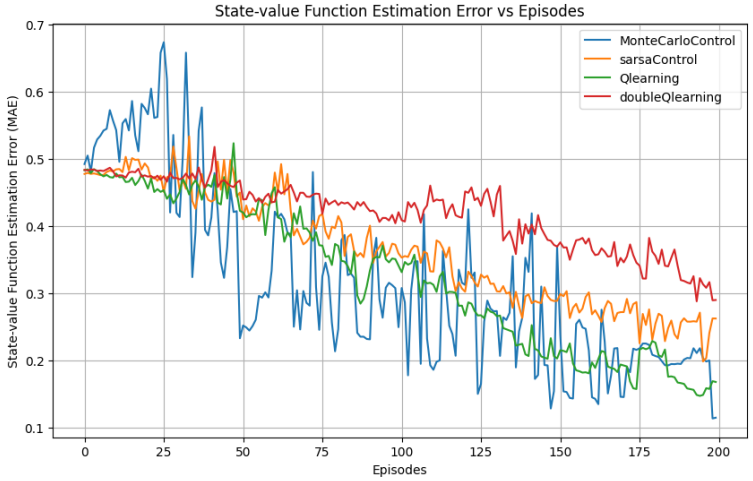


Figure 16: State Estimation Error vs Episodes evolution

6. The plot (16) shows that for **Q learning** algorithm, the state estimation error decreases most rapidly as compared to other algorithms, suggesting better convergence to the optimal values. As observed, the **Monte Carlo Control** shows the most variance in its decreasing trend and the plot for **Double Q Learning** has the least variance.

Solution to Problem 6: SARSA(λ) with Replacing traces

SARSA(*lambda*) is essentially a class of the λ-control algorithms. This algorithm uses eligibility traces to keep a track of the recent visited states and action pairs. When a given state action pair is encountered, the trace value is updated to 1 for that pair and the remaining sets are decayed by a factor of λ. The replacing traces ensure that the algorithm focuses on the recently visited states and allows for faster learning and convergence.

1. The given plot(17) depicts the state values for the non terminal states in the given random walk environment averaged over **50** different instances of the Random Maze Environment. The number of episodes chosen are **2000** and the seeds are chosen ranging from **100** to **150**. The plot shows that all of the states values initially starts below 0 and gradually converge towards their true estimates. The curves are relatively smooth since the estimates are averaged over various instances.

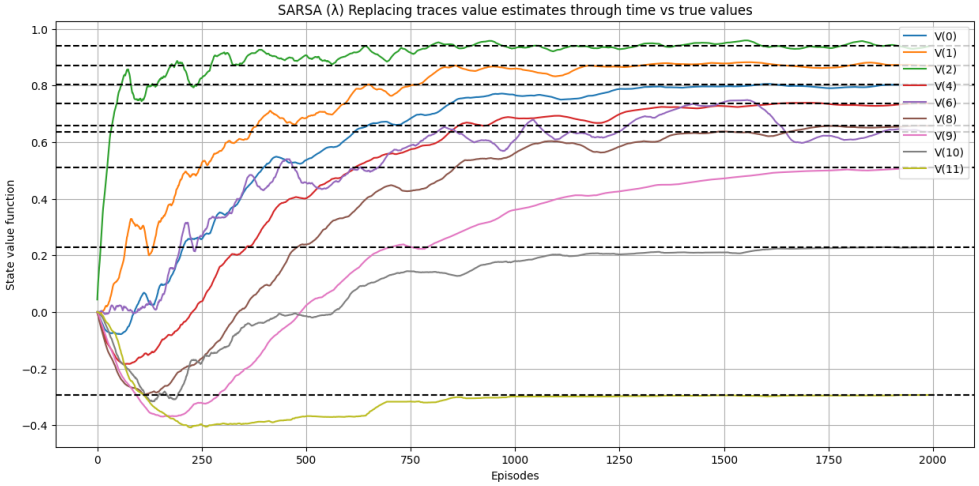


Figure 17: SARSA(λ) with replacing traces value estimates vs Episodes

2. The next plot(18) depicts the Q function evolution over different episodes of the environment averaged over 50 instances.

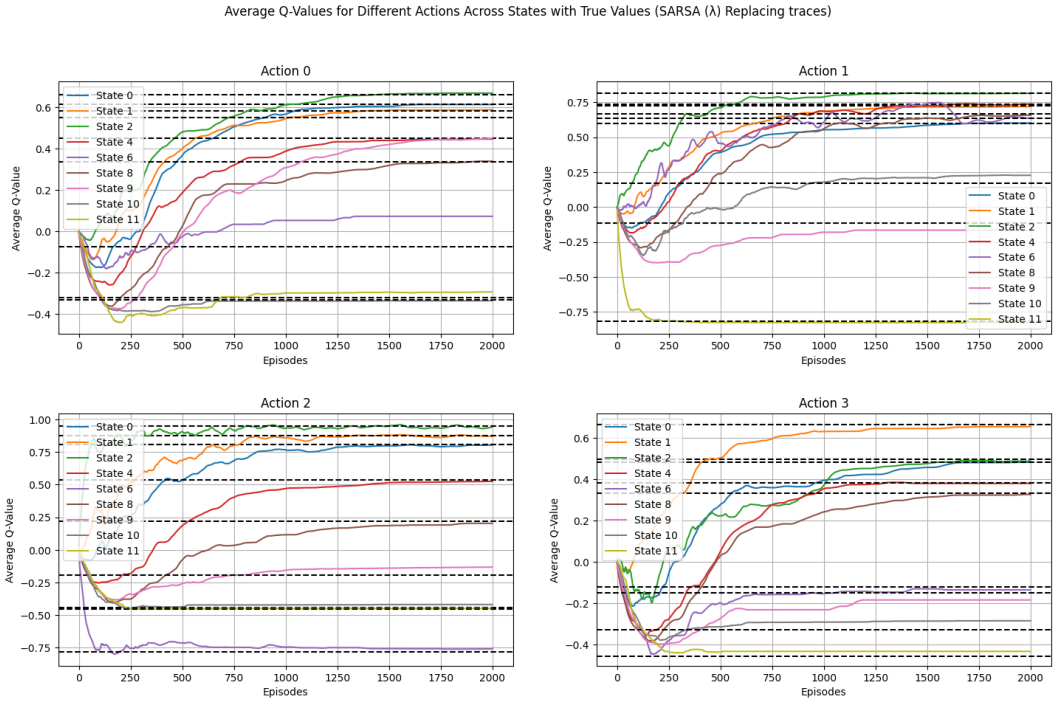


Figure 18: SARSA(λ) with replacing traces Q value estimates vs Episodes

3. The plots are averaged over seeds ranging from **100** to **150** where on every start of episode, the environment is reset using an episode seed calculated from global seed.

4. The next plot(19) illustrates the optimal policy calculated via the SARSA(λ) with replacing traces algorithm for the given environment.

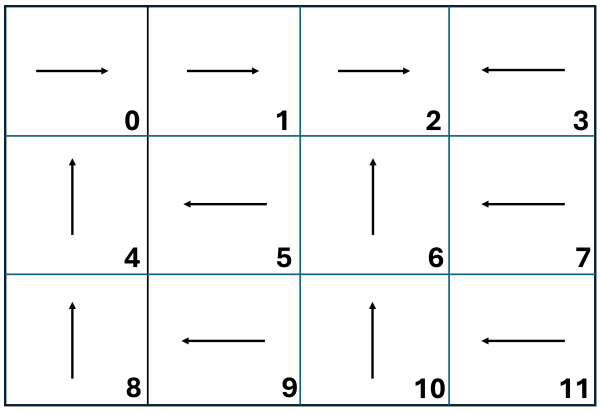


Figure 19: Optimal policy using SARSA(λ) with replacing traces

5. The hyperparameters used for the algorithm mainly includes the `numEpisodes` which is set to **2000**, the `decaytype` is set to **exponential**, the discount factor (γ) is set to **0.99**, the `tracedecay` factor λ is set to 0.3, the `maxSteps` is set to **100** and the `global_seed` is set to **123** and `replaceTrace` is set to **True**.

6. As observed from the value estimates, the estimates are initially noisy but they gradually converge and approaches a steady value. The plots for Double Q learning are way smoother than this value estimates evolution and the plot almost looks similar to that of SARSA(TD Control) algorithm. Also, the algorithm converges to their true values later as compared to that of Double Q learning algorithm.

Solution to Problem 7: SARSA(λ) with Accumulating traces

SARSA(λ) is essentially a class of the λ-control algorithms. This algorithm uses eligibility traces to keep a track of the recent visited states and action pairs. Accumulating traces provide a mechanism to assign credit for a reward to not only the most recent state-action pair but also to those preceding it. This is especially useful in situations where the effects of actions are not immediately apparent, and rewards are delayed. They can handle long term dependencies in decision making.

1. The given plot(20) depicts the state values for the non terminal states in the given random walk environment averaged over **50** different instances of the Random Maze Environment. The number of episodes chosen are **2000** and the seeds are chosen ranging from **100** to **150**. The plot shows that all of the states values initially starts below 0 and gradually converge towards their true estimates. The curves are relatively smooth since the estimates are averaged over various instances.

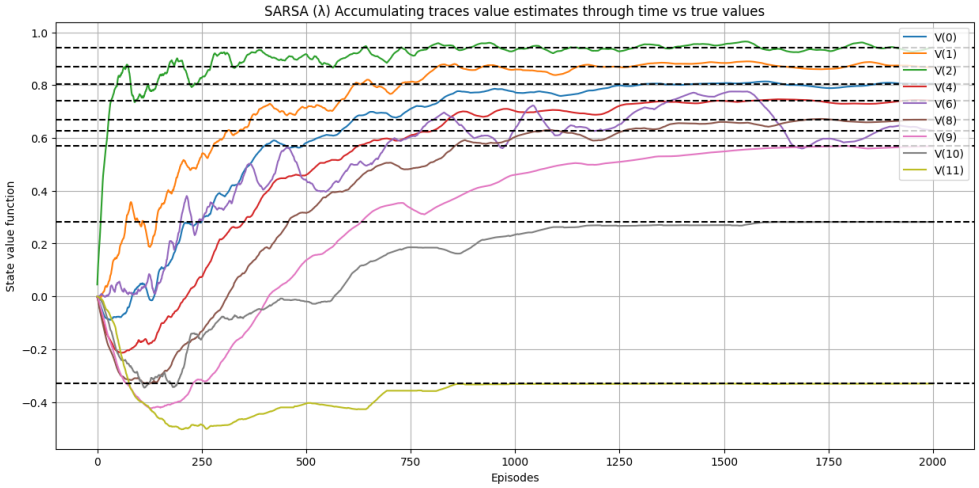


Figure 20: SARSA(λ) with accumulating traces value estimates vs Episodes

2. The next plot(21) depicts the Q function evolution over different episodes of the environment averaged over 50 instances.

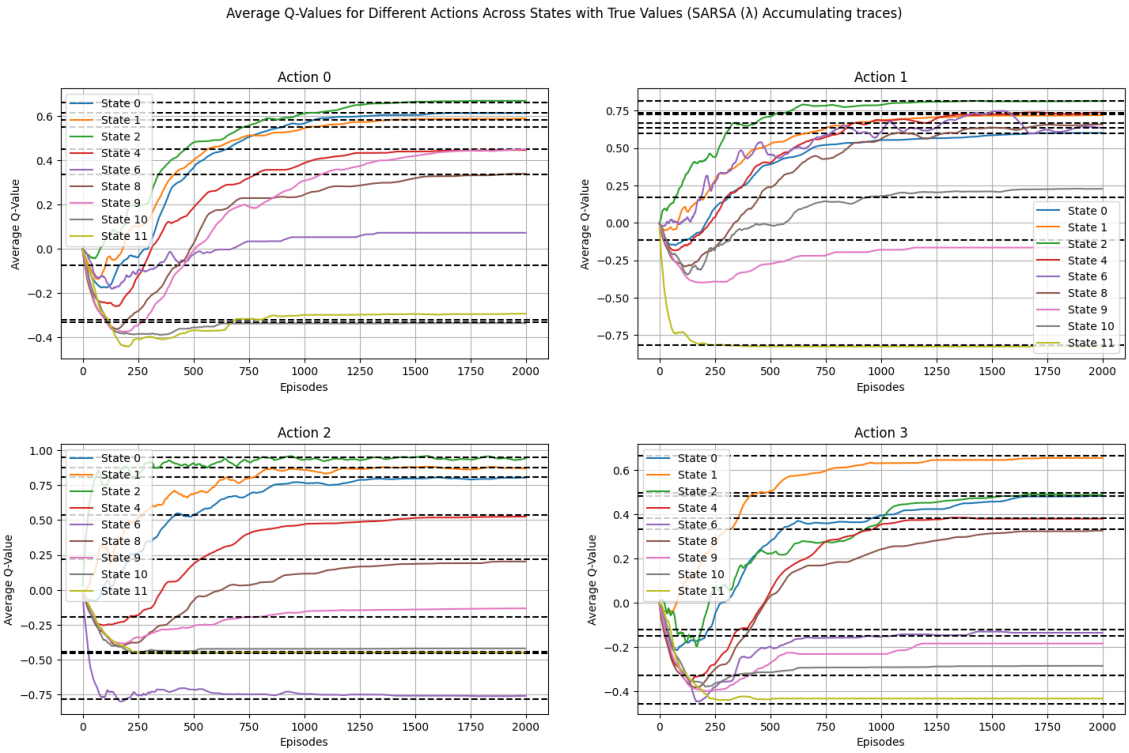


Figure 21: SARSA(λ) with accumulating traces Q value estimates vs Episodes

3. The plots are averaged over seeds ranging from **100** to **150** where on every start of episode, the environment is reset using an episode seed calculated from global seed.

4. The next plot(22) illustrates the optimal policy calculated via the SARSA(λ) with accumulating traces algorithm for the given environment.

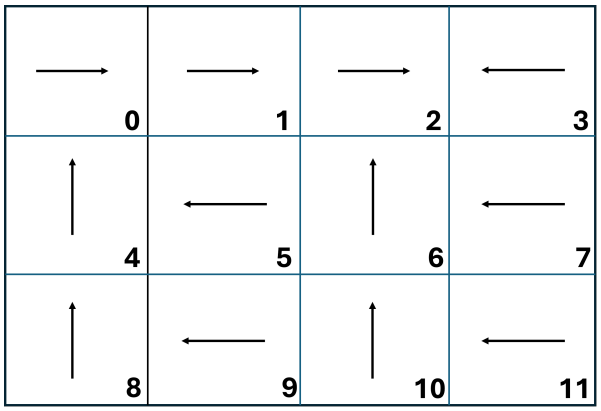


Figure 22: Optimal policy using SARSA(λ) with accumulating traces

5. The hyperparameters used for the algorithm mainly includes the `numEpisodes` which is set to **2000**, the `decaytype` is set to **exponential**, the discount factor (γ) is set to **0.99**, the `tracedecay` factor λ is set to 0.3, the `maxSteps` is set to **100** and the `global_seed` is set to **123** and `replaceTrace` is set to **False**.

6. As observed from the value estimates, the estimates are initially noisy but they gradually converge and approaches a steady value within. The value estimates converges a bit faster than those for the SARSA(λ) with replacing traces. Otherwise, the remaining pattern for the value estimates plots looks the same for both the SARSA(λ) control algorithms. Similarly, convergence requires a larger number of episodes.

Solution to Problem 8: Q(λ) with Replacing Traces

Q(λ) with replacing traces is an example of off policy λ-control algorithm. Since replacing traces are used, the trace is reset to 1 every time the similar state-action pair is encountered whereas other pairs are decayed by a factor γ.

1. The given plot(23) depicts the state values for the non terminal states in the given random walk environment averaged over **50** different instances of the Random Maze Environment. The number of episodes chosen are **2000** and the seeds are chosen ranging from **100** to **150**. The plot shows that all of the states values initially starts below 0 and gradually converge towards their true estimates. The curves are relatively smooth since the estimates are averaged over various instances.

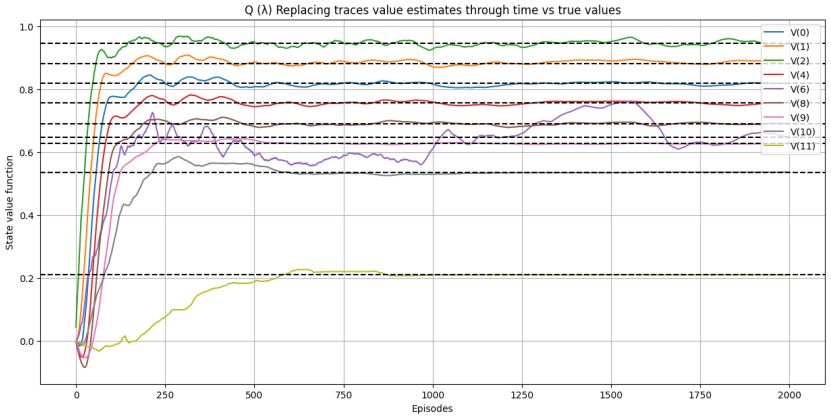


Figure 23: Q(λ) with replacing traces value estimates vs Episodes

2. The next plot(24) depicts the Q function evolution over different episodes of the environment averaged over 50 instances.

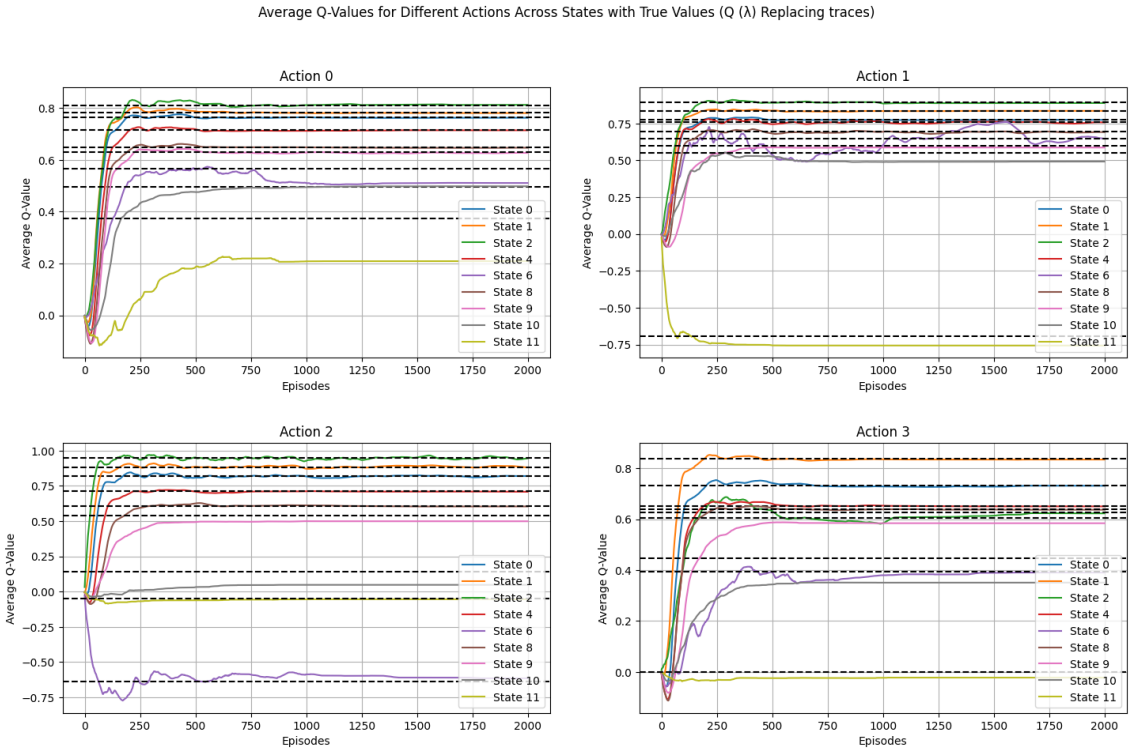


Figure 24: Q(λ) with replacing traces Q value estimates vs Episodes

3. The plots are averaged over seeds ranging from **100** to **150** where on every start of episode, the environment is reset using an episode seed calculated from global seed.

4. The next plot(25) illustrates the optimal policy calculated via the Q(λ) with replacing traces algorithm for the given environment.

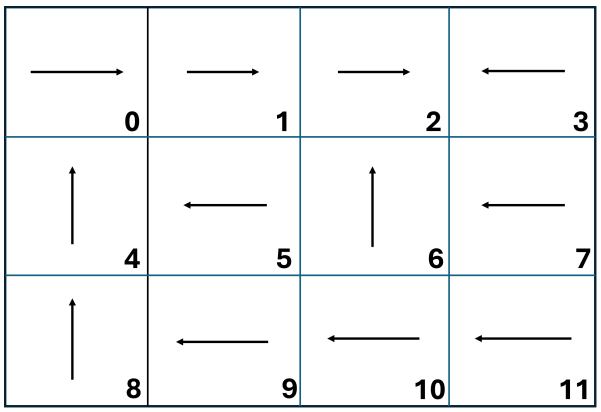


Figure 25: Optimal policy using Q(λ) with replacing traces

5. The hyperparameters used for the algorithm mainly includes the `numEpisodes` which is set to **2000**, the `decaytype` is set to **exponential**, the discount factor (γ) is set to **0.99**, the `tracedecay` factor λ is set to 0.3, the `maxSteps` is set to **100** and the `global_seed` is set to **123** and `replaceTrace` is set to **True**.

6. The state value evolution over through episodes is much smoother than that of ordinary Q learning approach, primarily attributed to the eligibility traces. The state values automatically converges near the true values in the vicinity of initial episodes. The q value estimates are also smoother than that of Q learning values.

Solution to Problem 9: Q(λ) with Accumulating Traces

Q(λ) with Accumulating traces is an example of off policy λ-control algorithm. Since Accumulating traces are used, the trace is updated by 1 every time the similar state-action pair is encountered whereas other pairs are decayed by a factor γ. They help to keep track of past experiences of the agent.

1. The given plot(26) depicts the state values for the non terminal states in the given random walk environment averaged over **50** different instances of the Random Maze Environment. The number of episodes chosen are **2000** and the seeds are chosen ranging from **100** to **150**. The plot shows that all of the states values initially starts below 0 and gradually converge towards their true estimates. The curves are relatively smooth since the estimates are averaged over various instances.

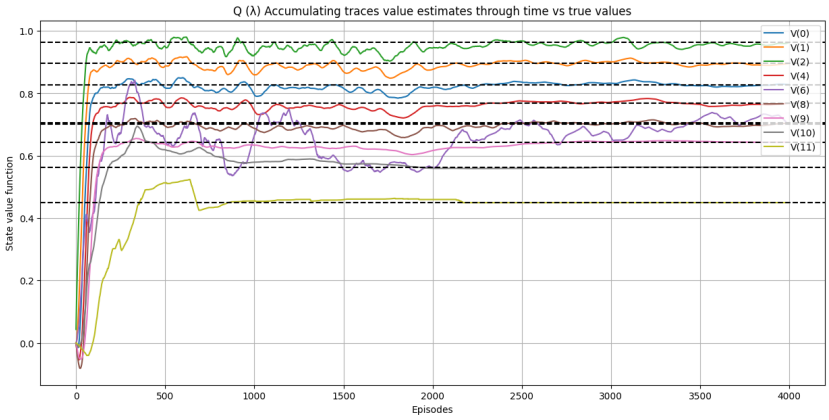


Figure 26: Q(λ) with accumulating traces value estimates vs Episodes

2. The next plot(27) depicts the Q function evolution over different episodes of the environment averaged over 50 instances.

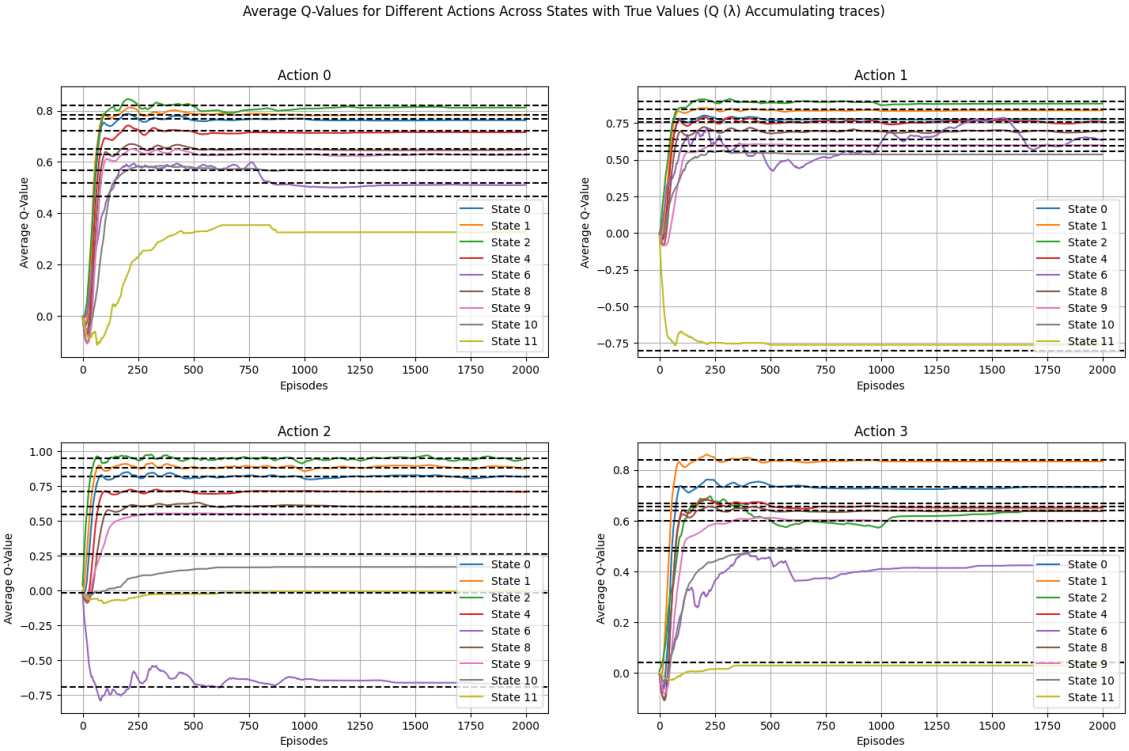


Figure 27: Q(λ) with accumulating traces Q value estimates vs Episodes

3. The plots are averaged over seeds ranging from **100** to **150** where on every start of episode, the environment is reset using an episode seed calculated from global seed.

4. The next plot(28) illustrates the optimal policy calculated via the Double Q LearningQ(λ) with accumulating traces algorithm for the given environment.

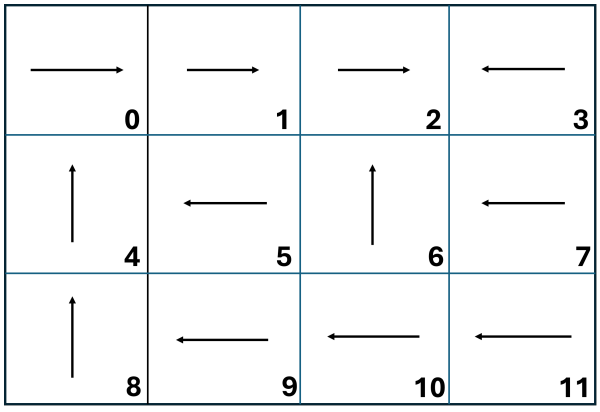


Figure 28: Optimal policy using Q(λ) with accumulating traces

5. The hyperparameters used for the algorithm mainly includes the `numEpisodes` which is set to **2000**, the `decaytype` is set to **exponential**, the discount factor (γ) is set to **0.99**, the `tracedecay` factor λ is set to 0.3, the `maxSteps` is set to **100** and the `global_seed` is set to **123** and `replaceTrace` is set to **False**.

6. Both the state value estimates for the case of replacing and accumulating traces look similar except for the fact that the value estimates of state 11 converges to a greater value for the case of accumulating traces. The value estimates for state 8 seems to be resistant towards normal convergence as other states which leads to its values shooting up in the later episodes as well.

Solution to Problem 10: Dyna-Q Algorithm

Dyna-Q is an example of Model based Reinforcement learning approach that essentially combines the direct RL with the planning approach which automatically builds a model from which additional experiences are generated for improved learning. While planning, it uses the learned model to simulate experiences where the model randomly selects previous state-action pair, using it to predict the next state and reward and automatically updates the Q values as if the transitions were actually experienced.

1. The given plot(29) depicts the state values for the non terminal states in the given random walk environment averaged over **50** different instances of the Random Maze Environment. The number of episodes chosen are **2000** and the seeds are chosen ranging from **100** to **150**. The plot shows that all of the states values initially starts below 0 and gradually converge towards their true estimates. The curves are relatively smooth since the estimates are averaged over various instances.

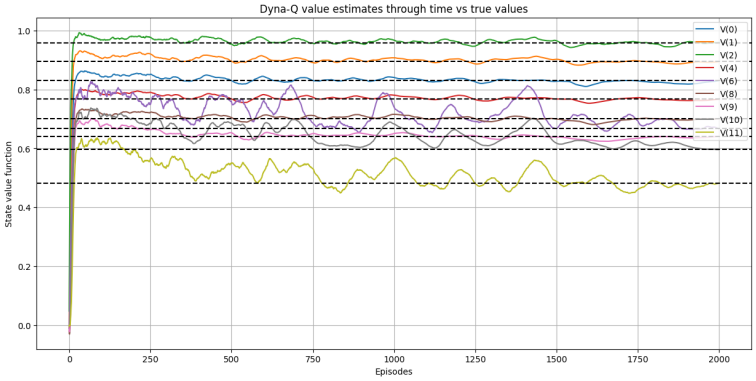


Figure 29: Dyna-Q value estimates vs Episodes

2. The next plot(30) depicts the Q function evolution over different episodes of the environment averaged over 50 instances.

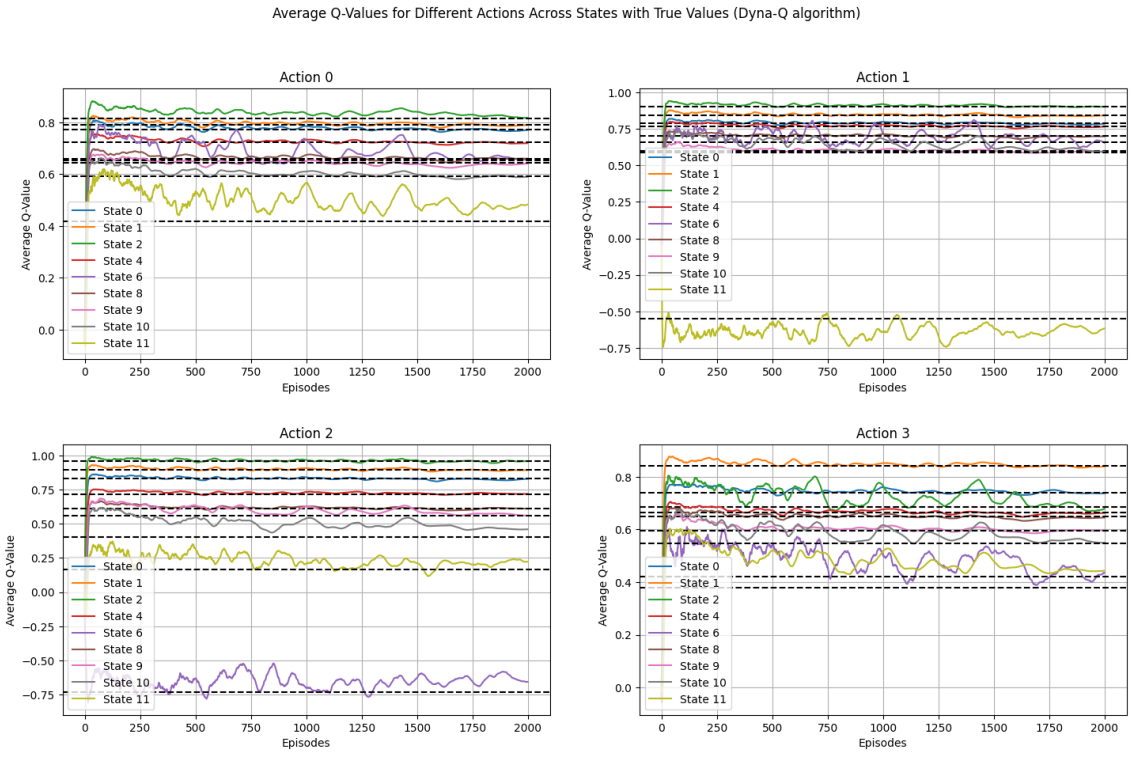


Figure 30: Dyna-Q Q value estimates vs Episodes

3. The plots are averaged over seeds ranging from **100** to **150** where on every start of episode, the environment is reset using an episode seed calculated from global seed.

4. The next plot(31) illustrates the optimal policy calculated via the Dyna-Q algorithm for the given environment.

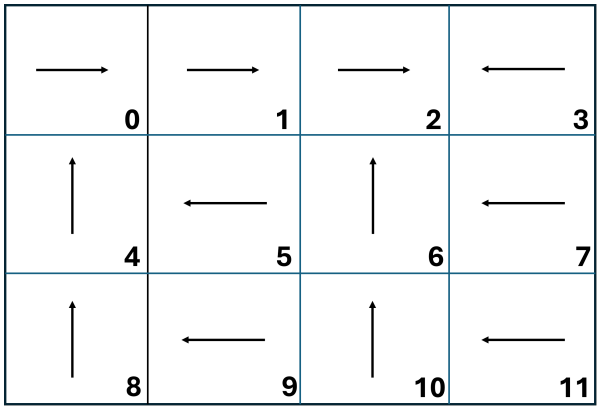


Figure 31: Optimal policy using Dyna-Q algorithm

5. The hyperparameters used for the algorithm mainly includes the `numEpisodes` which is set to **2000**, the `decaytype` is set to **exponential**, the discount factor (γ) is set to **0.99**, the `tracedecay` factor λ is set to 0.3, the `maxSteps` is set to **100** and the `global_seed` is set to **123** and `numPlanning` is set to 5.

6. The state value estimates seems to converge rapidly to the true values over small episodes but it is observed that for the later states (namely states 8, 9, 10, 11) the values starts deviating and oscillating around the true values in the initial episodes and the variation becomes a bit noisier. The plot also shows a sharp rise of almost all the state value estimates from their initial values.

Solution to Problem 11: Trajectory Sampling Algorithm

Trajectory Sampling is also another example of model-based planning algorithm that improves upon the **Dyna-Q** algorithm which during the planning step uniformly selects a state by choosing state which is likely to appear in the current episode. Trajectory Sampling essentially gathers trajectories (states and rewards) that are more likely to occur in the immediate future.

1. The given plot(32) depicts the state values for the non terminal states in the given random walk environment averaged over **50** different instances of the Random Maze Environment. The number of episodes chosen are **2000** and the seeds are chosen ranging from **100** to **150**. The plot shows that all of the states values initially starts below 0 and gradually converge towards their true estimates. The curves are relatively smooth since the estimates are averaged over various instances.

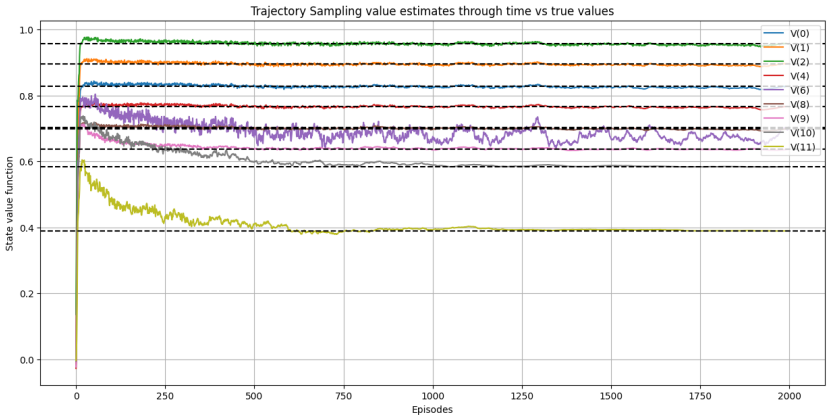


Figure 32: Trajectory Sampling value estimates vs Episodes

2. The next plot(33) depicts the Q function evolution over different episodes of the environment averaged over 50 instances.

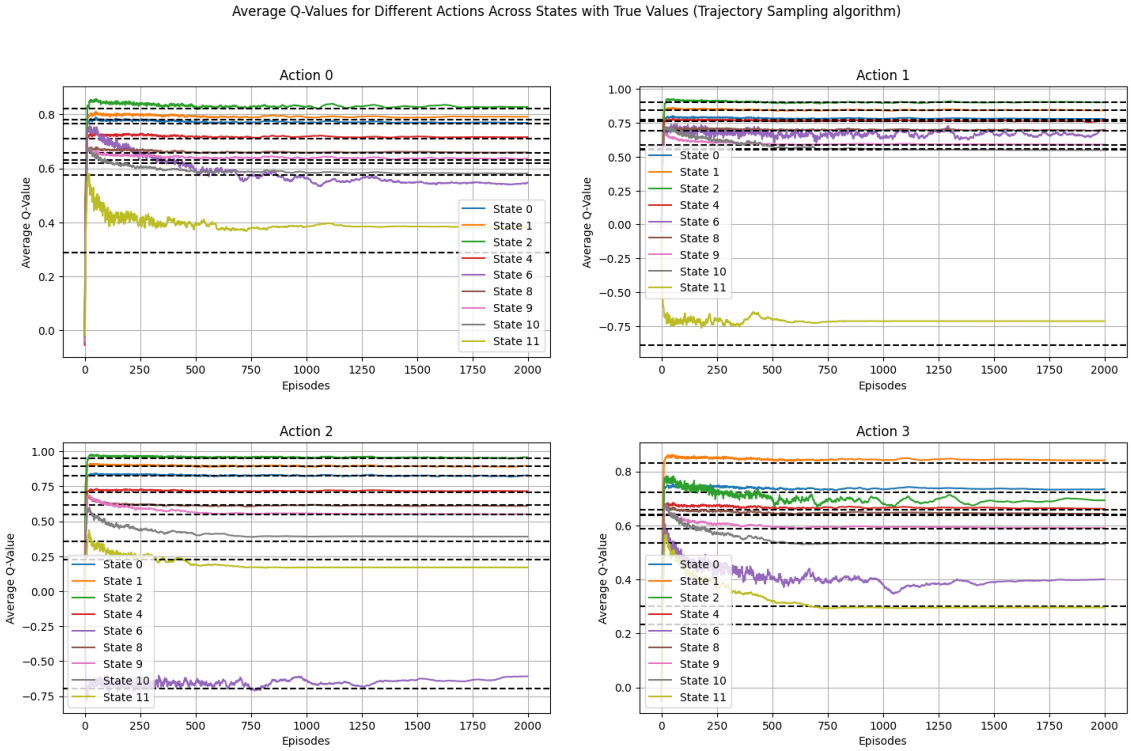


Figure 33: Trajectory Sampling Q value estimates vs Episodes

3. The plots are averaged over seeds ranging from **100** to **150** where on every start of episode, the environment is reset using an episode seed calculated from global seed.

4. The next plot(34) illustrates the optimal policy calculated via the Trajectory Sampling algorithm for the given environment.

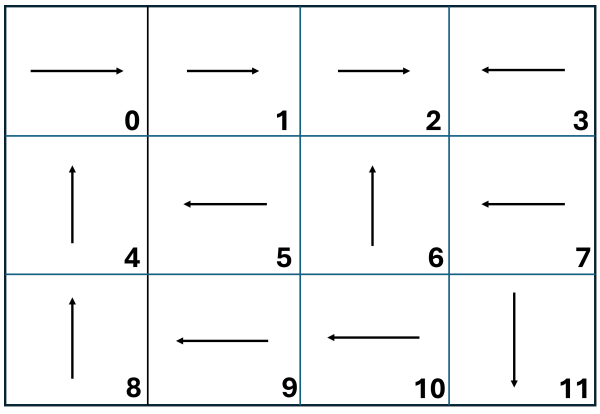


Figure 34: Optimal policy using Trajectory Sampling algorithm

5. The hyperparameters used for the algorithm mainly includes the `numEpisodes` which is set to **2000**, the `decaytype` is set to **exponential**, the discount factor (γ) is set to **0.99**, the `tracedecay` factor λ is set to 0.3, the `maxSteps` is set to **100** and the `global_seed` is set to **123** and `maxTrajectory` is set to 10.

6. The trajectory sampling value estimates essentially follows the rapid convergence over the episodes. The values shows a very steep rise from their initial random values and rapidly converges with the quite lost of minute variations over the episodes. The states (namely 1, 2, 4, 6) seems to reach the true values within

300 episodes whereas it takes a bit of time for the later states to reach their optimal values. Overall, this algorithm shows most rapid convergence but at the cost of increased variance.

Solution to Problem 12: Comparing Control Algorithms

1. The given plot(35) depicts the % Policy Success Rate calculated for each of the algorithms. It denotes the number of times the agent was able to reach the goal state for every policy calculated after each episode for each of the algorithms

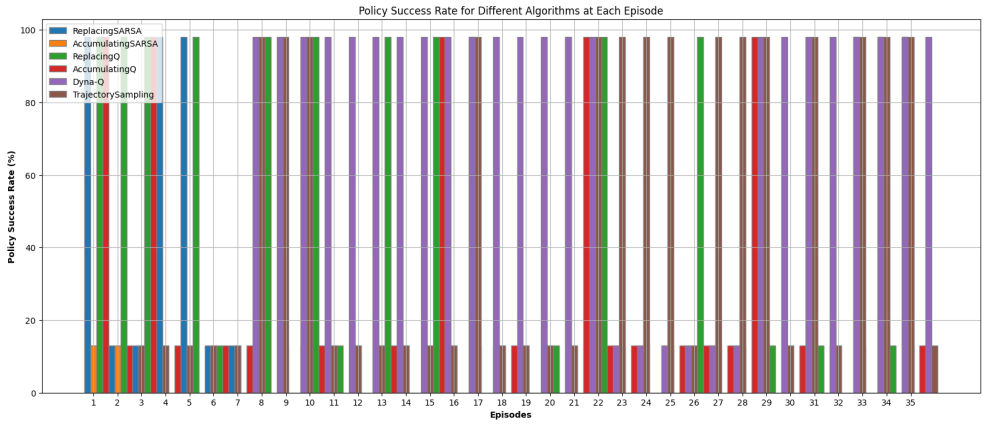


Figure 35: % Policy Success Rate vs Episode evolution

2. The plot(35) shows that **Dyna-Q** algorithm has the most number of successful policy rates in various episodes followed by **Trajectory Sampling**. In the initial episodes,**SARSA(λ)** has some policy hits. Mostly towards the end of episodes, **Replacing Q**, **Accumulating Q** algorithms has a smaller percentage of policy success rate other than Dyna-Q and trajectory sampling.

3. The given plot(36) depicts the Expected Reward starting from the start state (State **8**) vs episodes for all the algorithms.

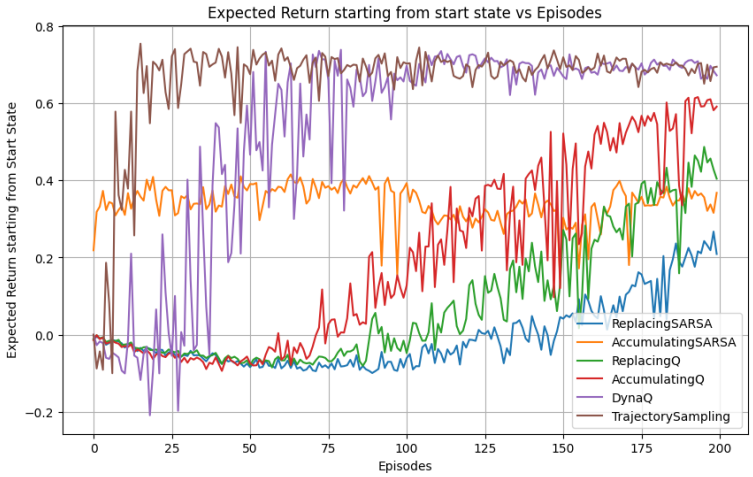


Figure 36: Expected Reward vs Episodes evolution

4. The plot(36) illustrates that the expected return for **Trajectory Sampling**, **Dyna-Q** increases in the initial episodes and almost attains a steady value with minimal oscillations. The other algorithms like **Replacing Q**, **Accumulating Q** and **Replacing SARSA** shows an increasing trend after 100 episodes. The highest reward is achieved by **Trajectory Sampling** followed by Dyna-Q algorithm.

5. The next plot(37) shows the Estimated value error calculated by the mean absolute error of all the states over episodes for all the algorithms.

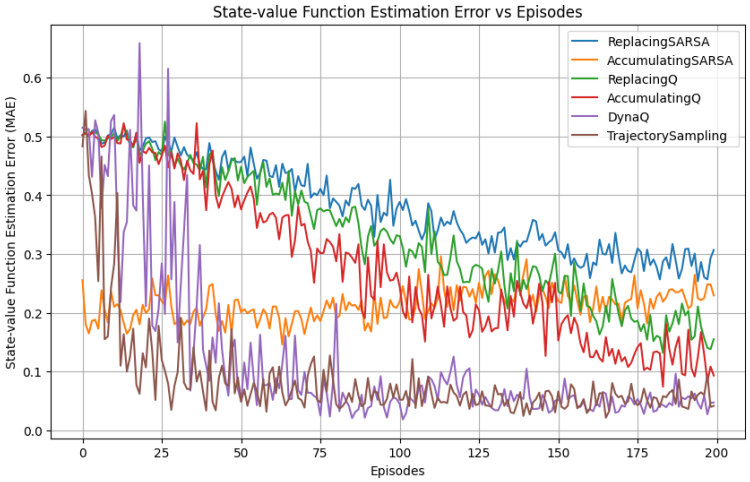


Figure 37: Estimated Value error vs Episodes evolution

6. The plot(37) illustrates that the error decreases very rapidly especially for **Trajectory Sampling** and **Dyna-Q** followed by **Accumulating Q** algorithm. The decrease essentially starts steadily in the initial episodes. The least decrease happens in the case of **Replacing SARSA** over the episodes but the rate of decrement in steady.