CSE 4/574:Introduction to Machine Learning, Sections A,C&D
Spring 2023

Assignment 3

Defining and Solving Reinforcement Learning Task

Report

- Narasimha Gaonkar – ngaonkar

- Rohan Venkatesh Sirigeri – rohanven

Part 1

1. In our assignment, we used a Grid-World environment with the theme of the popular game called Super Mario. The grid built is a 4X4 grid where the RL agent i.e., Mario moves through the grid in four different directions to reach its end goal i.e., Princess.

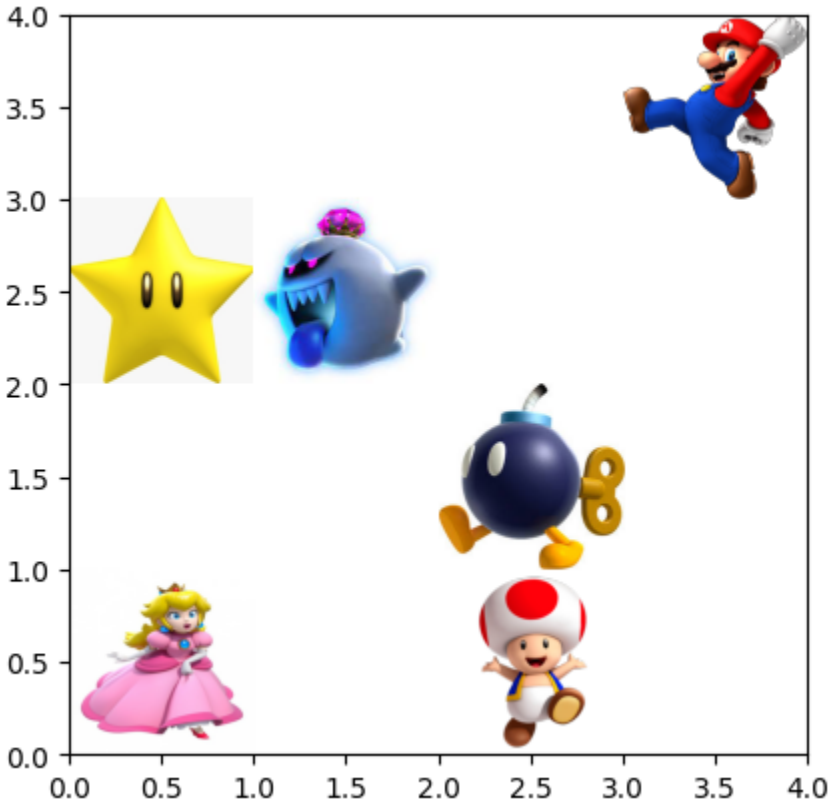
   The states formed in the grid is a list i.e., {S1 = (0,0), S2 = (0,1), S3 = (0,2), S4 = (0,3), S5 = (1,0), S6 = (1,1), S7 = (1,2), S8 = (1,3), S9 = (2,0), S10 = (2,1), S11 = (2,2), S12 = (2,3), S13 = (3,0), S14 = (3,1), S15 = (3,2), S16 = (3,3)}. Mario's starting state is S4 and the princess he should win is in state S13.

   Mario can take the set of actions in this grid: {Up, Down, Right, Left}.

   The rewards placed in the grid which Mario can obtain in the process are as follows:

   - Star: State is S8 and Reward is 6.
   - Mushroom: State is S15 and Reward is 5.
   - Boo: State is S6 and Reward is -5.
   - Bomb: State is S11 and Reward is -6.

2. Our SuperMario grid environment looks like this:



3. Safety in AI:

   Several steps must be taken to guarantee an AI agent's safety when functioning in a grid setting. The agent must first choose just those acts that are permitted by the environment. This may be accomplished by properly organizing the action area and making sure the agent abides by the environment's norms. In order to prevent the agent from entering invalid states, the agent should also be restricted to moving only inside the specified state-space.

   Additionally, the surroundings must be planned to prevent dangerous scenarios, such as obstructions or dangers that might injure the agent or other people. To prevent over-adapting to a particular subset of the environment, which might result in surprising behavior under novel circumstances, the agent should also be trained on a safe and diversified range of events. The agent's conduct must also be continually observed and assessed in order to identify and resolve any potential safety concerns.

Part 2 and Part 3

1. The SARSA (State-Action-Reward-State-Action) tabular reinforcement learning method is used to train a Q-function that calculates the predicted total of upcoming rewards for a certain state-action combination. Since SARSA is an on-policy approach, it acquires knowledge of the Q-function for the same policy that it employs to produce behavior.

The update function for SARSA is:

Q(s, a) = Q(s, a) + alpha * (reward + gamma * Q(s', a') - Q(s, a))

where Q(s, a) is the current estimate of the expected sum of future rewards for taking action a in state s, alpha is the learning rate, gamma is the discount factor, reward is the immediate reward received for taking action a in state s, and s' and a' are the next state and action, respectively, that the agent selects using its current policy.

Key features of SARSA include:

- It is an on-policy method, meaning that it learns the Q-function for the same policy it is using to generate behavior.
- It can handle stochastic policies and environments.
- It is guaranteed to converge to the optimal Q-function for a given policy if the learning rate is sufficiently small and the policy converges to the optimal policy.

SARSA offers several benefits, including the capacity to manage stochastic settings and policies, as well as its under certain circumstances guaranteed convergence. Among its drawbacks are its comparatively sluggish convergence rate and sensitivity to hyperparameters like the learning rate and discount factor. Additionally, SARSA may experience the "bootstrapping" phenomenon, in which the agent updates its estimations of the Q-function based on its own current estimates. This may cause the genuine Q-values to be overestimated or underestimated.

The Q-learning tabular reinforcement learning technique is used to train a Q-function, which calculates the predicted total of future rewards for a given state-action combination. Being an off-policy approach, Q-Learning studies the Q-function for a different policy from the one it employs to produce behavior.

The update function for Q-Learning is:

$$Q(s, a) = Q(s, a) + \alpha * (\text{reward} + \gamma * \max_{a'}(Q(s', a')) - Q(s, a))$$

where Q(s, a) is the current estimate of the expected sum of future rewards for taking action a in state s, alpha is the learning rate, gamma is the discount factor, reward is the immediate reward received for taking action a in state s, s' is the next state that the agent transitions to, and max_a'(Q(s', a')) is the maximum expected sum of future rewards over all actions a' in the next state s'.

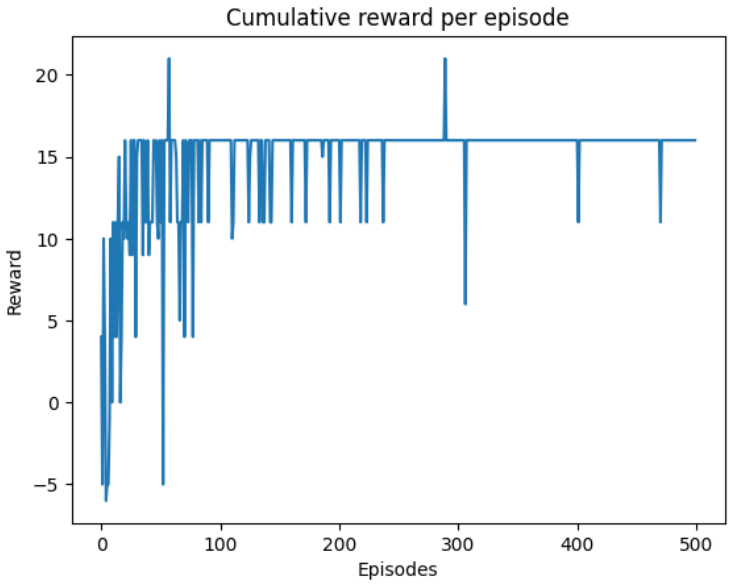
Key features of Q-Learning include:

- It is an off-policy method, meaning that it learns the Q-function for a different policy than the one it is using to generate behavior.
- It can handle stochastic policies and environments.
- It is guaranteed to converge to the optimal Q-function regardless of the behavior policy, as long as all state-action pairs are visited an infinite number of times and the learning rate satisfies certain conditions.

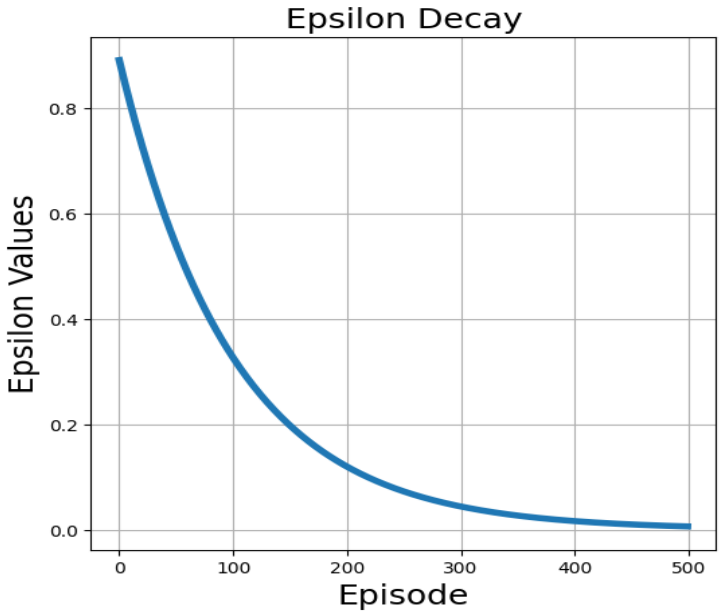
Q-Learning provides a number of benefits, including as the capacity to deal with stochastic environments and policies, convergence guarantees under specific circumstances, and simplicity and convenience of use. Its propensity for over- or underestimating Q-values, sensitivity to hyperparameters like the learning rate and discount factor, and relatively sluggish convergence rate in comparison to other approaches are drawbacks. In addition, the "exploration-exploitation" tradeoff, where the agent must choose between doing actions that it is positive will result in large rewards (exploitation) and taking activities that it is unsure of but that can result in even greater rewards (exploration), might negatively affect Q-Learning.

2. SARSA Results:

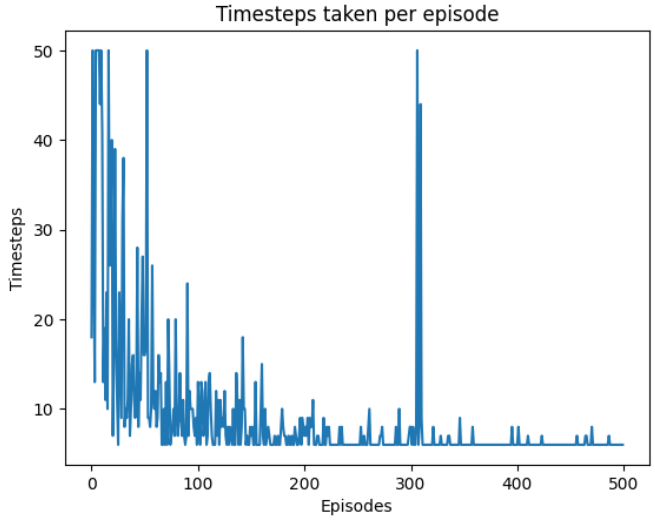
- Total Reward per Episode:



- Epsilon Decay:

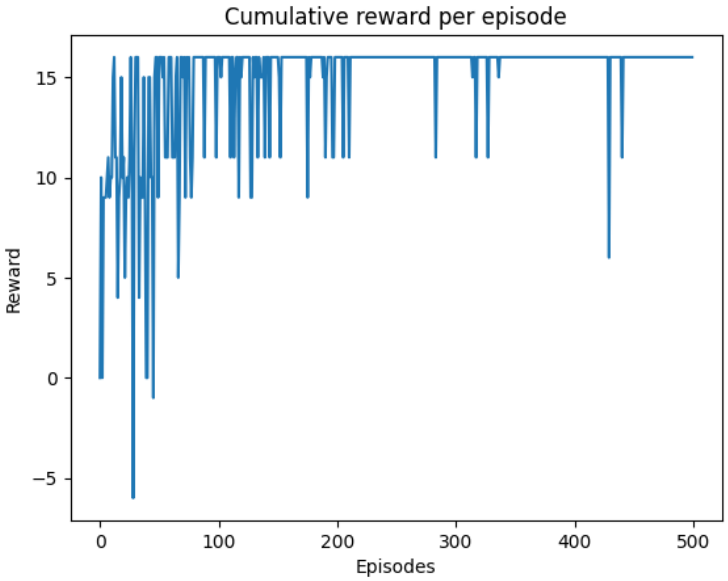


- Timesteps per Episode:

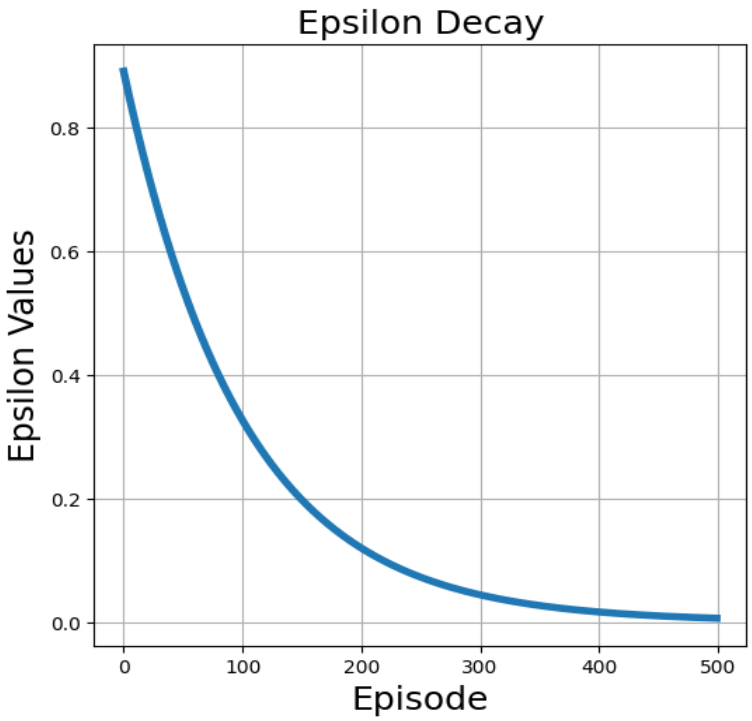


Q-Learning Results:

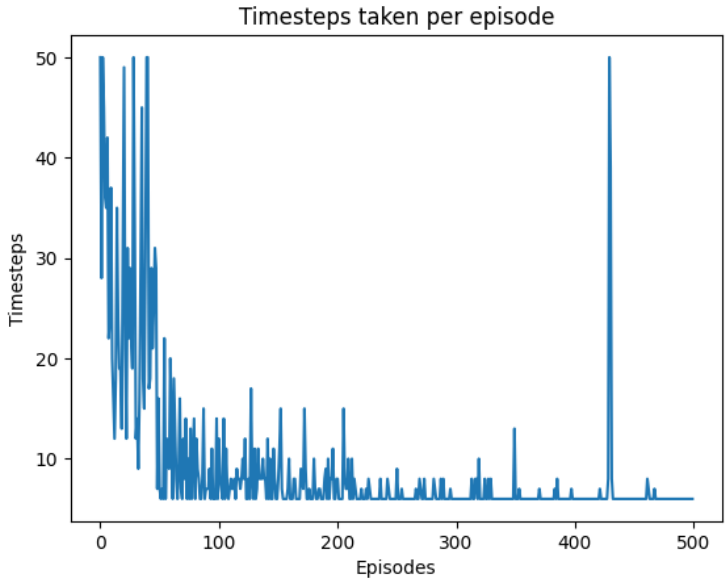
- Total Reward per Episode:



- Epsilon Decay:



- Timesteps per Episode:



For SARSA and Q-Learning, both reach the maximum reward of around 100 episodes.

3. After training both SARSA and Q-Learning in the same environment, we can plot the reward dynamics of both algorithms on the same graph. Interpreting the results would involve examining the graph and comparing the performance of the two algorithms in terms of their convergence rate and variance. Specifically, we would analyze the shape and trend of the learning curves, as well as the magnitude and frequency of reward fluctuations. This analysis would allow us to make informed conclusions about the strengths and weaknesses of each algorithm and which one may be better suited for the specific task at hand. Some of the interpretations made were:

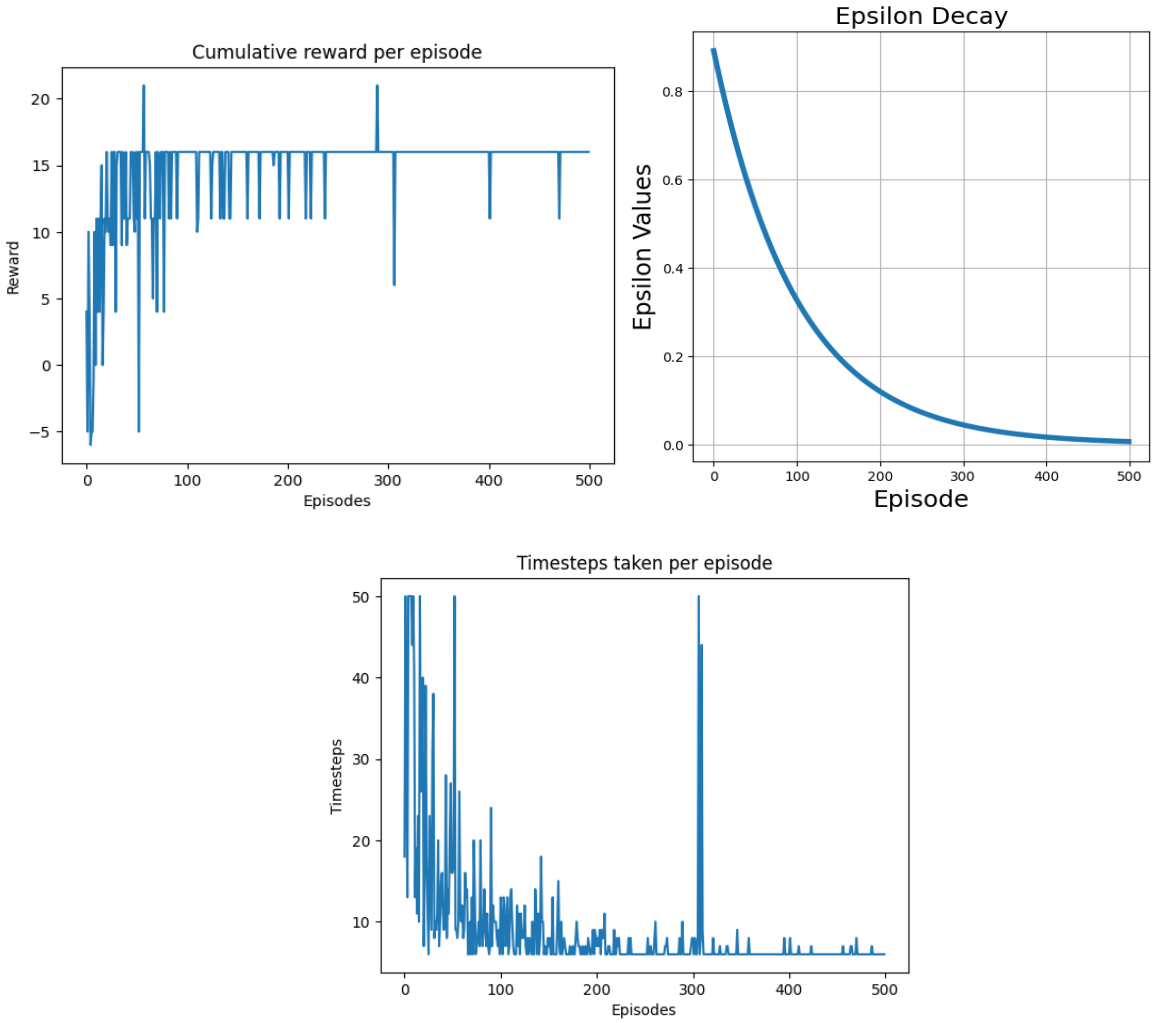
- In the Episodes v/s Reward plot, we can see that both of the algorithms reached the maximum reward near the 100th epoch. But in Q-Learning, it reached faster than SARSA.
- In the Episodes v/s Timesteps plot, we can see that Timestep decreased in Q-Learning near the 45th episode. But in SARSA, timesteps decreased near the 65th episode.

4. Hyperparameter Configurations

Hyperparameter 1:

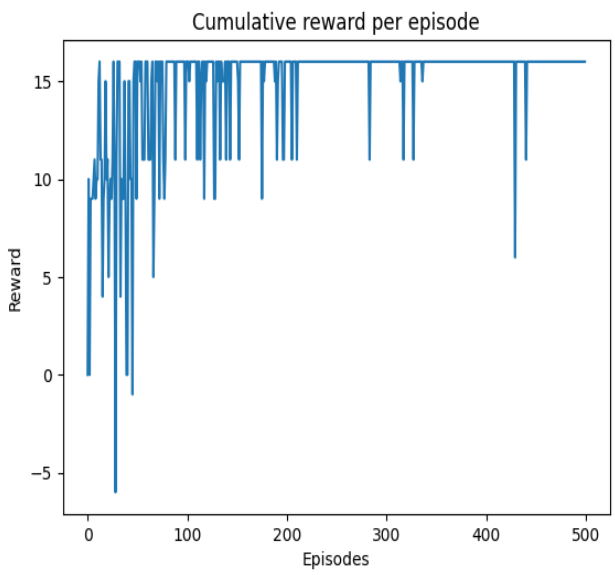
(episodes: 500, discount_rate: 0.9, learning_rate: 0.1, epsilon:0.9)

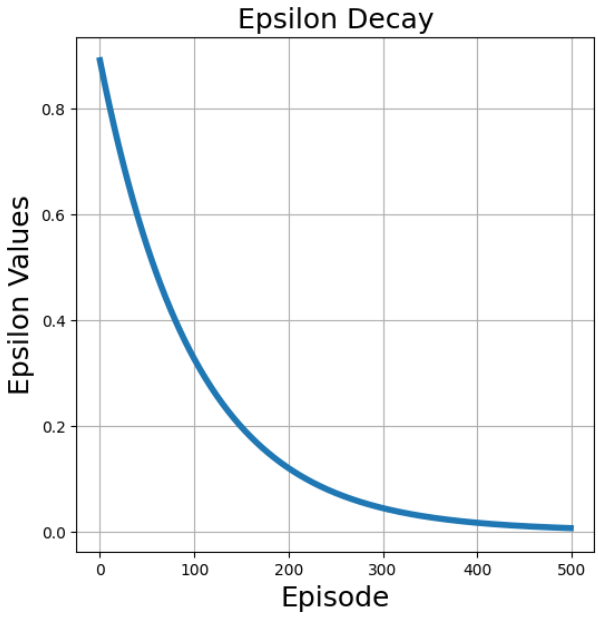
SARSA:

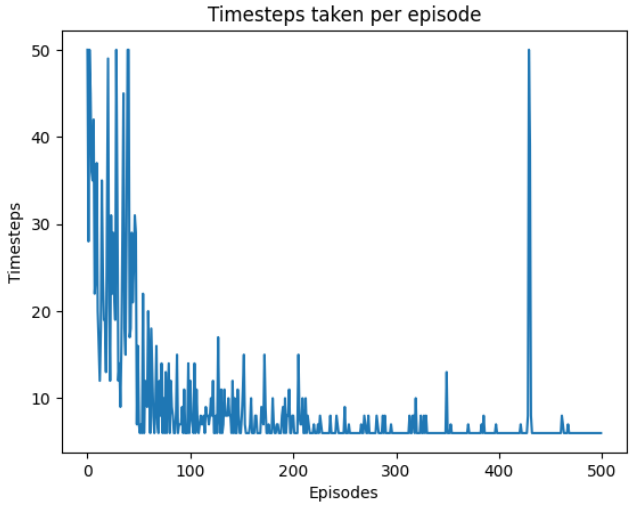


Above are the Episodes v/s Reward, Epsilon Decay, and Episodes v/s Timesteps to achieve reward plots. As the episodes progress, the agent learns from the algorithm and finds the shortest path with the maximum reward. From this set of hyperparameters, our agent reaches the maximum reward near the 100th episode with the minimum number of timesteps i.e., 6 timesteps. Once the agent learned the optimal path for reaching the goal with maximum rewards, we can see that agent takes the same path for the rest of the episodes.

Q-Learning:





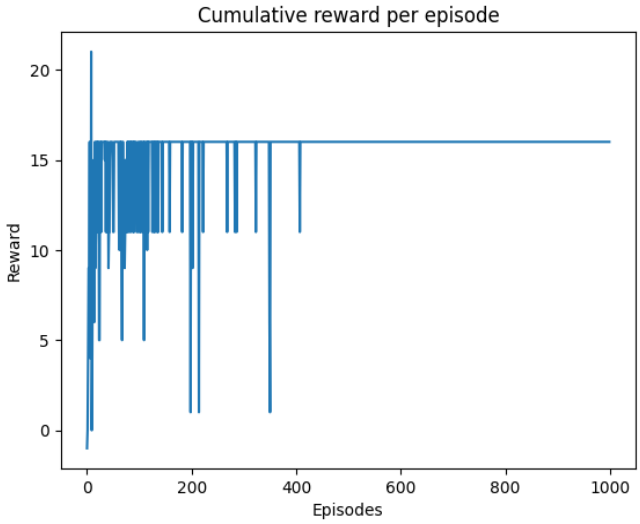


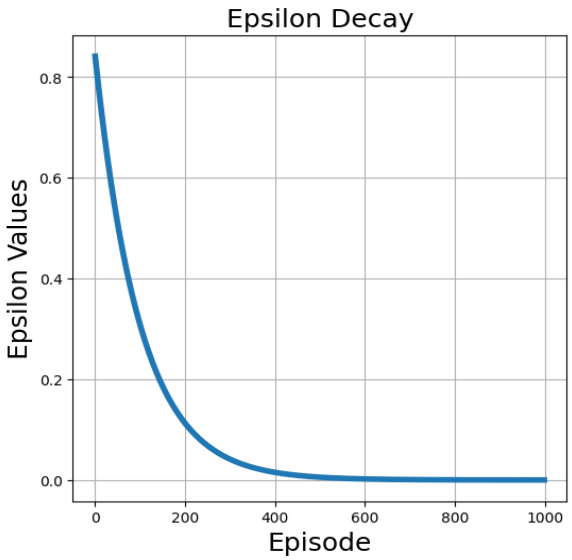
Above are the Episodes v/s Reward, Epsilon Decay, and Episodes v/s Timesteps to achieve reward plots. As the episodes progress, the agent learns from the algorithm and finds the shortest path with the maximum reward. From this set of hyperparameters, our agent reaches the maximum reward near the 100th episode with the minimum number of timesteps i.e., 6 timesteps. Once the agent learned the optimal path for reaching the goal with maximum rewards, we can see that agent takes the same path for the rest of the episodes.

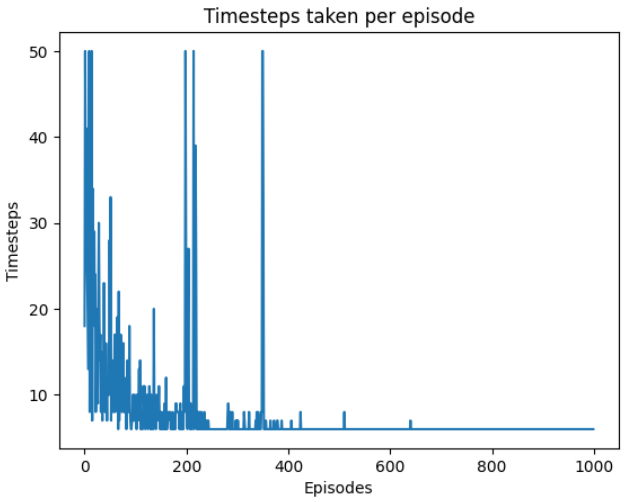
Hyperparameter 2:

(episodes: 1000, discount_rate: 0.95, learning_rate: 0.1, epsilon: 0.85)

SARSA:

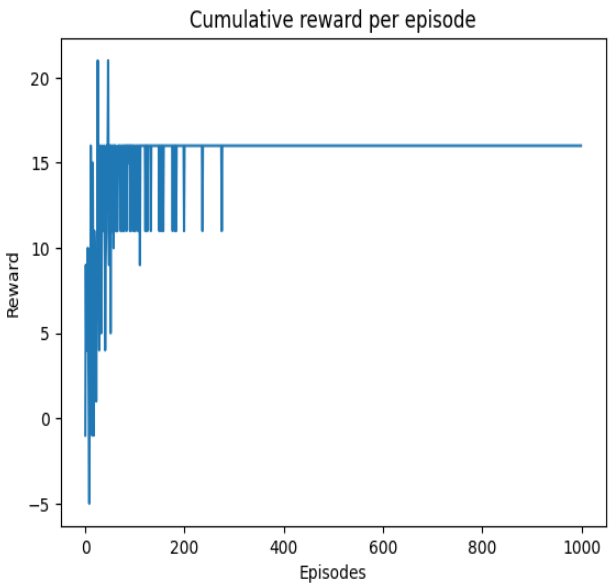


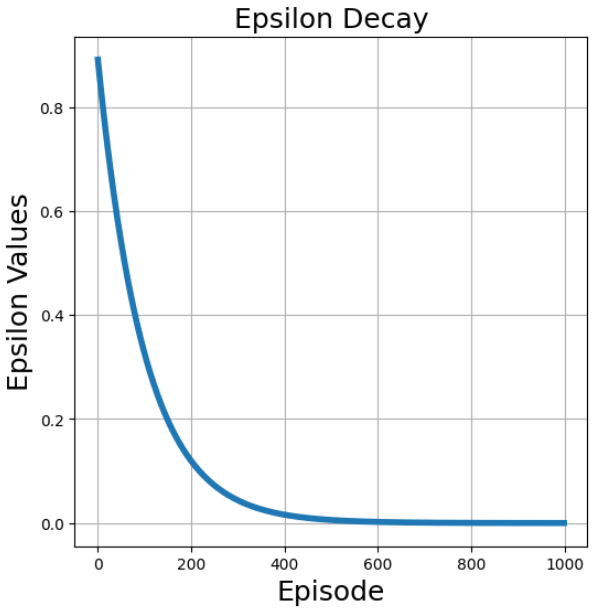


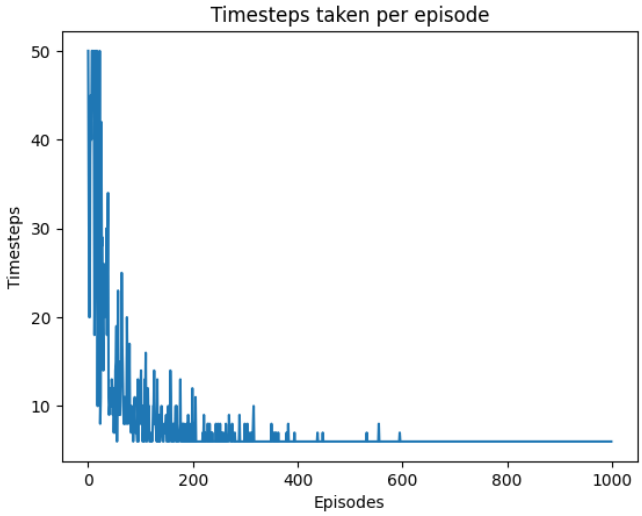


Above are the Episodes v/s Reward, Epsilon Decay, and Episodes v/s Timesteps to achieve reward plots. As the episodes progress, the agent learns from the algorithm and finds the shortest path with the maximum reward. From this set of hyperparameters, our agent reaches the maximum reward near the 30th episode with the minimum number of timesteps i.e., 6 timesteps. Once the agent learned the optimal path for reaching the goal with maximum rewards, we can see that agent takes the same path for the rest of the episodes.

Q-Learning:





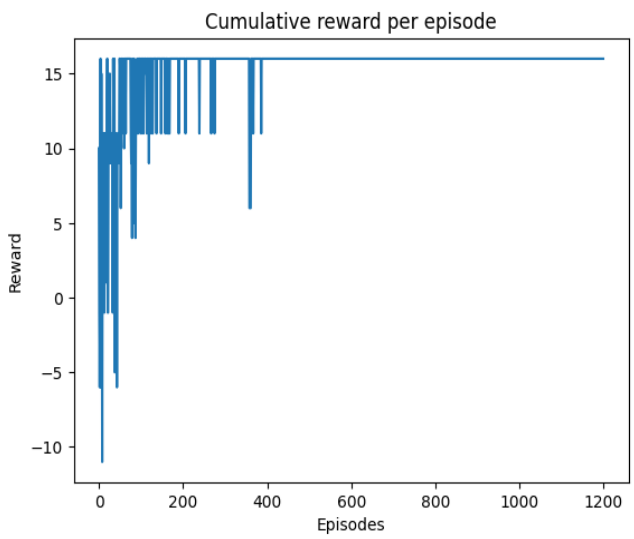


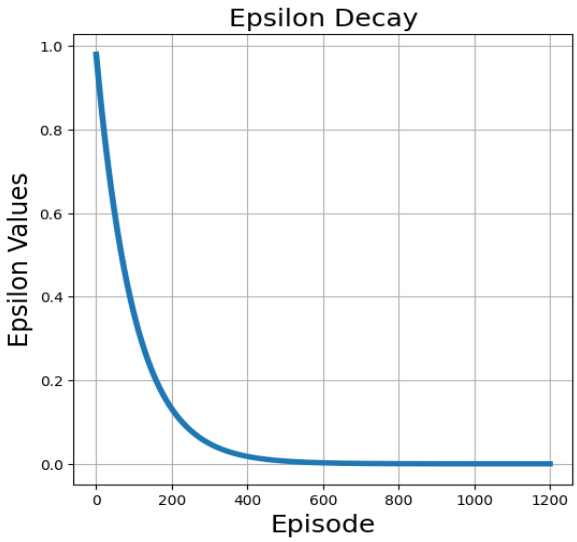
Above are the Episodes v/s Reward, Epsilon Decay, and Episodes v/s Timesteps to achieve reward plots. As the episodes progress, the agent learns from the algorithm and finds the shortest path with the maximum reward. From this set of hyperparameters, our agent reaches the maximum reward near the 70th episode with the minimum number of timesteps i.e., 6 timesteps. Once the agent learned the optimal path for reaching the goal with maximum rewards, we can see that agent takes the same path for the rest of the episodes.

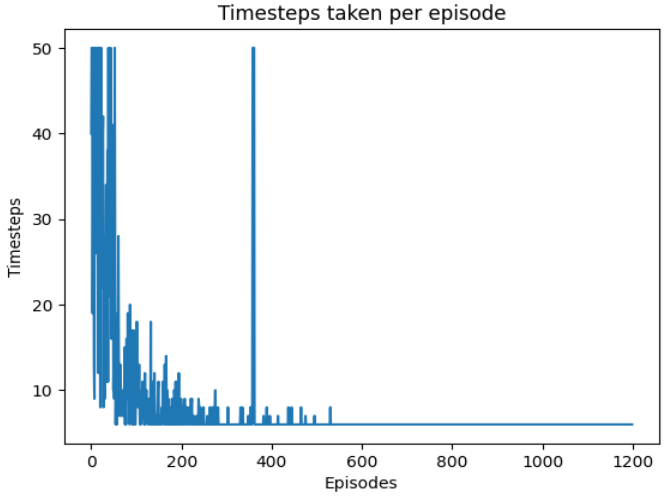
Hyperparameter 3:

(episodes: 1200, discount_rate: 0.99, learning_rate: 0.1, epsilon: 0.99)

SARSA:

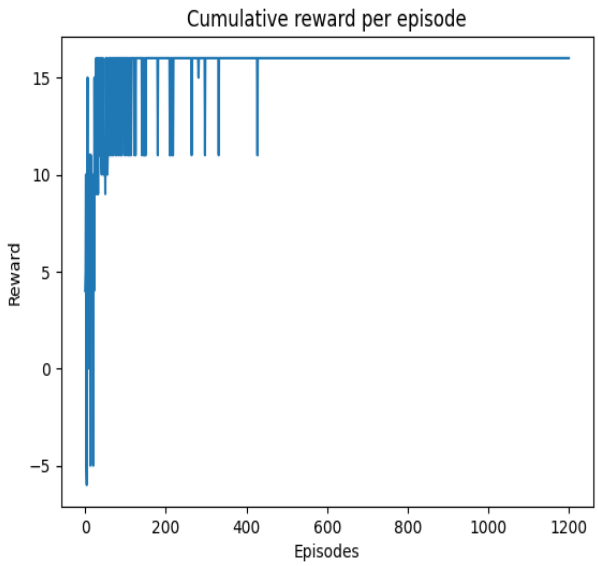


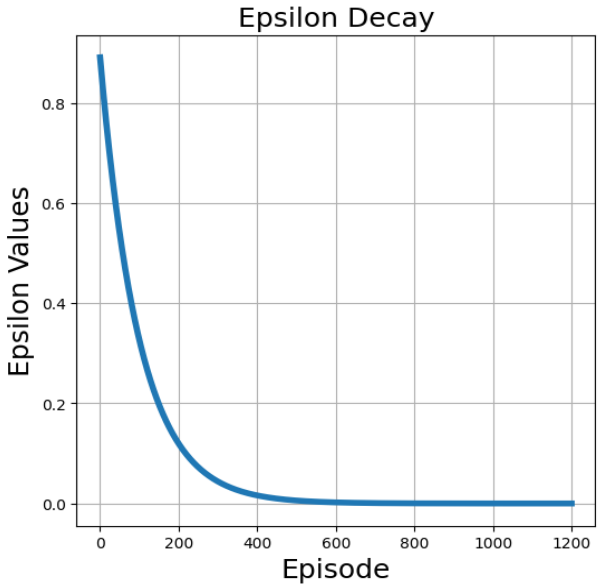


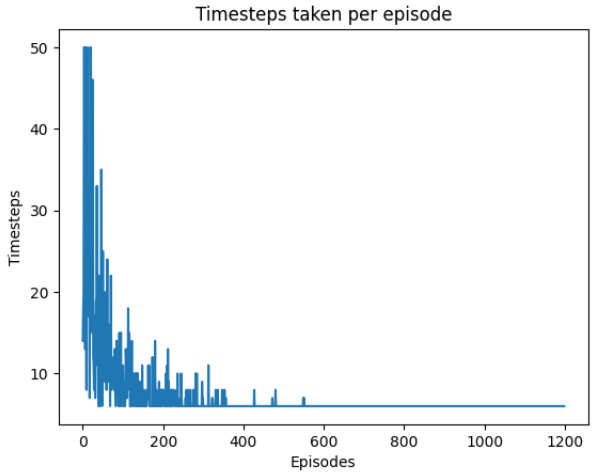


Above are the Episodes v/s Reward, Epsilon Decay, and Episodes v/s Timesteps to achieve reward plots. As the episodes progress, the agent learns from the algorithm and finds the shortest path with the maximum reward. From this set of hyperparameters, our agent reaches the maximum reward near the 60th episode with the minimum number of timesteps i.e., 6 timesteps. Once the agent learned the optimal path for reaching the goal with maximum rewards, we can see that agent takes the same path for the rest of the episodes.

Q-Learning:







Above are the Episodes v/s Reward, Epsilon Decay, and Episodes v/s Timesteps to achieve reward plots. As the episodes progress, the agent learns from the algorithm and finds the shortest path with the maximum reward. From this set of hyperparameters, our agent reaches the maximum reward near the 45th episode with the minimum number of timesteps i.e., 6 timesteps. Once the agent learned the optimal path for reaching the goal with maximum rewards, we can see that agent takes the same path for the rest of the episodes.

Best Hyperparameters that we would recommend for this grid environment:

- SARSA:

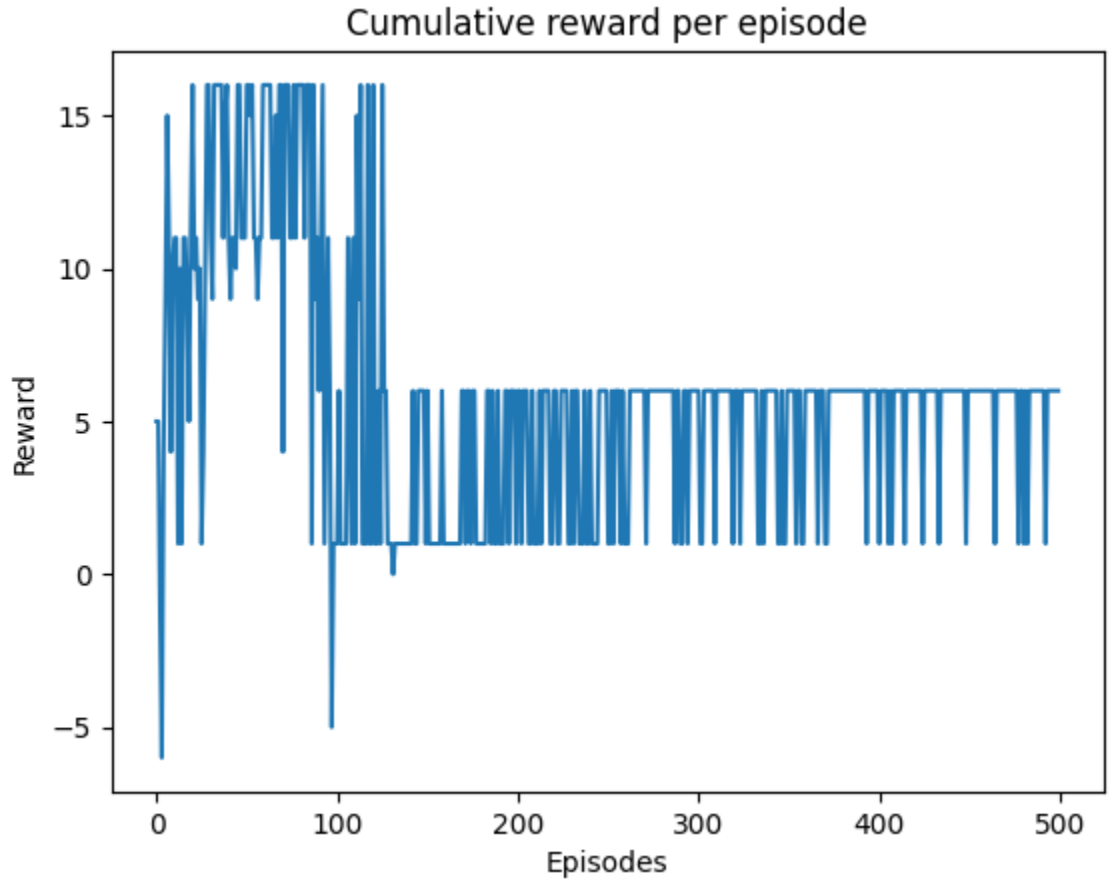
  (episodes: 500, discount_rate: 0.9, learning_rate: 0.1, epsilon:0.9)

- Q-Learning:

  (episodes: 1000, discount_rate: 0.95, learning_rate: 0.1, epsilon: 0.85)

Bonus

We have implemented SARSA 2-step Bootstraping for the existing SARSA model we had. We observed that SARSA is working more efficiently than the Bootstrapping algorithm. This is interpreted from the below Episodes v/s Reward plot:



References:

1. Assignment_0 .ipynb file.
2. Assignment_1 .ipynb file.
3. Assignment_2 .ipynb file.
4. Pandas: "https://pandas.pydata.org/"
5. Numpy: "https://numpy.org/doc/stable/reference/random/generated/numpy.random.rand.html"
6. Gymnasium: https://gymnasium.farama.org/
7. Optuna: https://optuna.org/

Contribution:

Name	Contribution
Narasimha Gaonkar	50%
Rohan Venkatesh Sirigeri	50%