Assignment 1: Maze Solver using RL

Anonymous Author(s)
Affiliation
Address
`email`

1 Environment

The problem defines two types of environments: deterministic and stochastic. In a deterministic environment, the agent will transition to the next state based on the action taken with no uncertainty, but in a stochastic environment, the agent will move to the next state at random with a probability defined by GridEnv. Stochastic environment follows the below action dynamics, keys are actions and values are its corresponding probabilities

```
{'w': 0.95, 's': 0.98, 'd': 0.96, 'a': 0.99}
```

1.1 Objective

Solve maze by reaching goal state optimally avoiding forbidden states. They include out-of-bounds positions, walls and already visited states.

1.2 Actions

- Left (a) : Agent moves left when action 'a' is executed
- Right (d) : Agent moves right when action 'd' is executed
- Up (w) : Agent moves upwards when action 'w' is executed
- Down (s) : Agent moves downwards when action 's' is executed

1.3 States

Agent position tuple (x, y) determines the current state. Environment with 16 states has been chosen for convenience but can be easily modified by passing a larger maze to GridEnv class. States with value of 1.0 are legal positions in which our agent can move, whilst others with a value of 0.0 are walls that our agent should avoid

1.4 Rewards

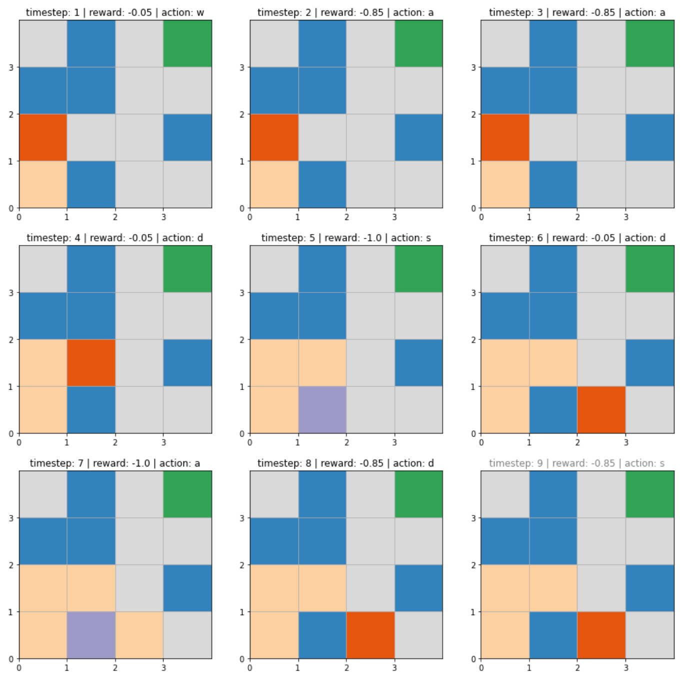
Initially rewards were given according to the euclidean distance between agent and the goal, but policy was not able to converge in a limited time. Later, new dynamics were chosen and they seem to work well for the current task. The given pairs represent the appropriate reward dynamics, with key representing the current state type and value being the reward

- Goal : 20.0
- Wall : -5.0
- Out of bounds : -0.75
- Visited: -0.85
- Legal : -0.05

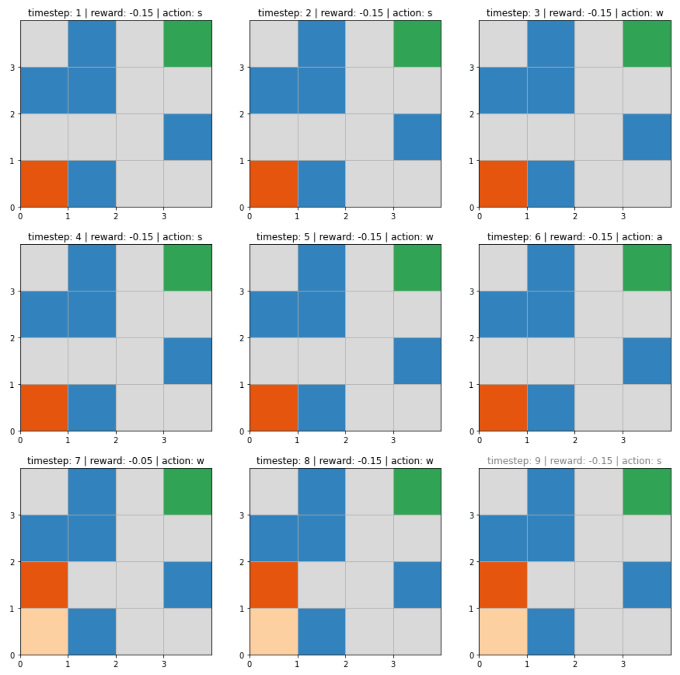
2 Visualizations

The agent is represented by an orange square, the target by a green square, the maze walls are blue, and the visited positions are light orange. Wall color will be changed to purple if the agent is current at its position

2.1 Deterministic Environment


timestep: 1 | reward: -0.05 | action: w
timestep: 2 | reward: -0.85 | action: a
timestep: 3 | reward: -0.85 | action: a
timestep: 4 | reward: -0.05 | action: d
timestep: 5 | reward: -1.0 | action: s
timestep: 6 | reward: -0.05 | action: d
timestep: 7 | reward: -1.0 | action: a
timestep: 8 | reward: -0.85 | action: d
timestep: 9 | reward: -0.85 | action: s

2.2 Stochastic Environment


timestep: 1 | reward: -0.15 | action: s
timestep: 2 | reward: -0.15 | action: s
timestep: 3 | reward: -0.15 | action: w
timestep: 4 | reward: -0.15 | action: s
timestep: 5 | reward: -0.15 | action: w
timestep: 6 | reward: -0.15 | action: a
timestep: 7 | reward: -0.05 | action: w
timestep: 8 | reward: -0.15 | action: w
timestep: 9 | reward: -0.15 | action: s

3 Safety in AI

The task of any intelligent agent in Reinforcement Learning is to optimize the problem by taking the best possible steps towards the goal, but this can be risky. For example, in the case of self-driving cars, the agent's goal is to reach a destination and the cost metric can be time; in this case, we must ensure that the agent follows some rules and regulations while traversing.

In most real-world problems, the environment dynamics are unknown to the agent, so it must explore to learn. However, while exploring, the agent may end up in certain unlawful and harmful states; to solve this issue, we must supply the agent with some previous information about the environment.

Likewise, in my environment I've ensured to clip the out of bound positions to the edges and negative rewards are associated with the state action pairs leading to illegal states

4 Tabular Methods

4.1 Q-Learning

QL is a model-free algorithm, it learns state values by experience, it works by taking an action and updating the Q table using the immediate reward and Q value estimate of the next state

4.1.1 Key features

- Q table is updated after every action
- Behavioural policy is different from update policy hence its an off policy model
- It high bias and low variance as it uses next best Q value and it is an estimate and doesn't depend on all the future actions
- It can learn before final outcome
- Works with both episodic and continuing tasks

4.1.2 Update function

```
q[state][action] = q[state][action] + learning_rate * (reward + (
    ↪ gamma * max(q[next_state]) - q[state][action]))
```

4.2 Monte Carlo

MC is also a model-free algorithm and it learns by sampling, it estimates the state values by averaging returns from all sampled episodes

4.2.1 Key features

- Q value for a state-action pair is updated at the end of every episode
- Behavioural policy is different from update policy hence its an off policy model
- It high variance as the Q value depends on all the future actions which can be random
- It can learn only after final outcome
- Works only with episodic tasks

4.2.2 Update function

```
returns = reward(t + 1) + discount_factor ^ 1 * reward(t + 2) +
    ↪ discount_factor ^ 2 * reward(t + 3) + .... discount_factor ^
    ↪  (t - 1) * reward(T)
q[state][action] = old_q + alpha * (returns - old_q)
```

5 Q-Learning - Training

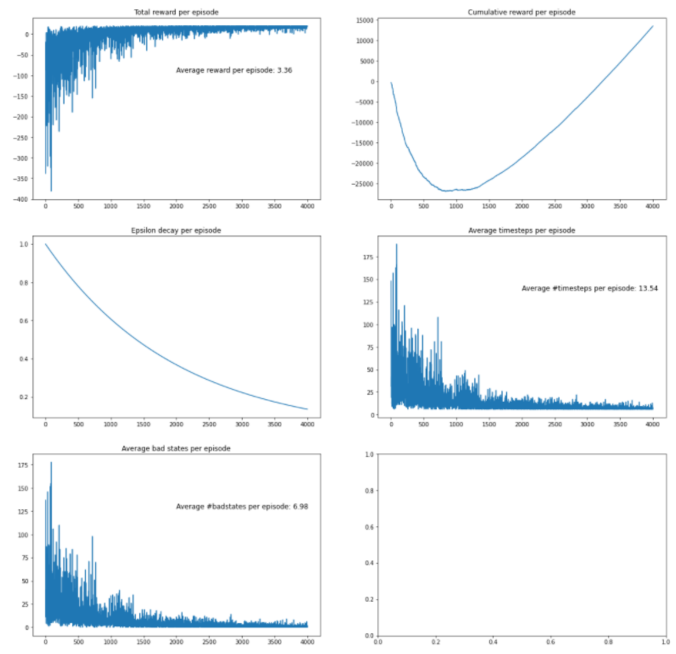
The QL algorithm was able to converge to an optimal policy based on the epsilon decay rate; the higher the decay rate, faster it could converge. Furthermore, when the discount factor is closer to one, it performed better

5.1 Tuning Discount Factor

Algorithm functioned best when gamma=1.0; I believe this is because a higher discount factor gives more weight to the best Q value of the next state-action pair while updating the value of the current state

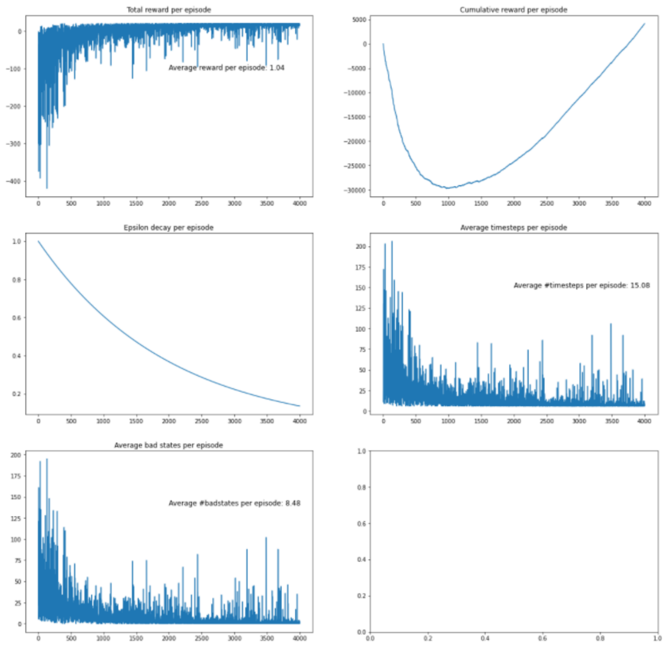
5.1.1 Deterministic Environment

- Epsilon start = 1.0, Epsilon decay factor = 0.9995, Discount factor = 1.0, Episodes = 4000, Step size = 0.01



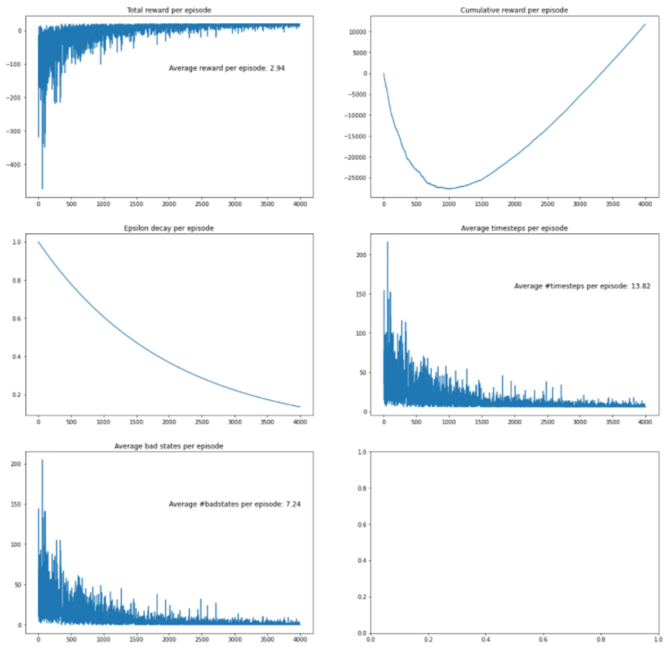
With the above hyper-parameters policy converged at around 1000 episodes with average total reward around 3

- Epsilon start = 1.0, Epsilon decay factor = 0.9995, Discount factor = 0.0, Episodes = 4000, Step size = 0.01



With the above hyper-parameters policy converged at around 1000 episodes with an average total reward around 1

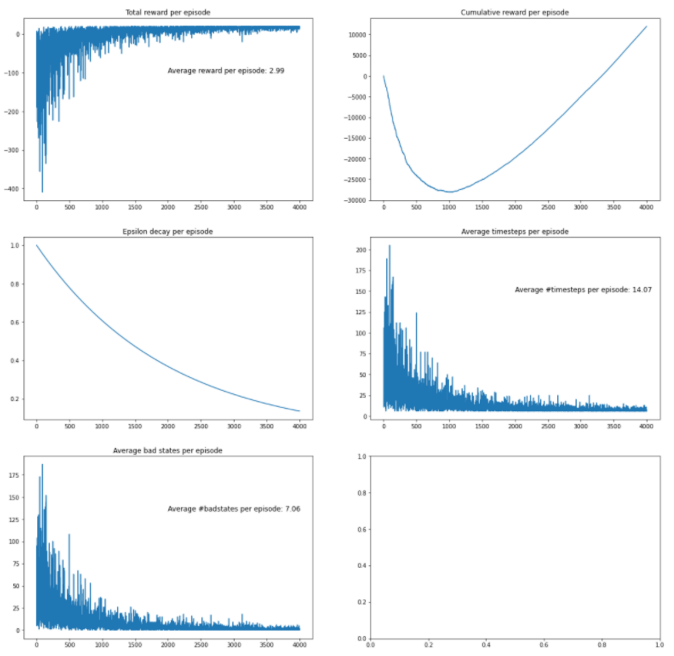
- Epsilon start = 1.0, Epsilon decay factor = 0.9995, Discount factor = 0.5, Episodes = 4000, Step size = 0.01



With the above hyper-parameters policy converged at around 1000 episodes with average total reward around 3

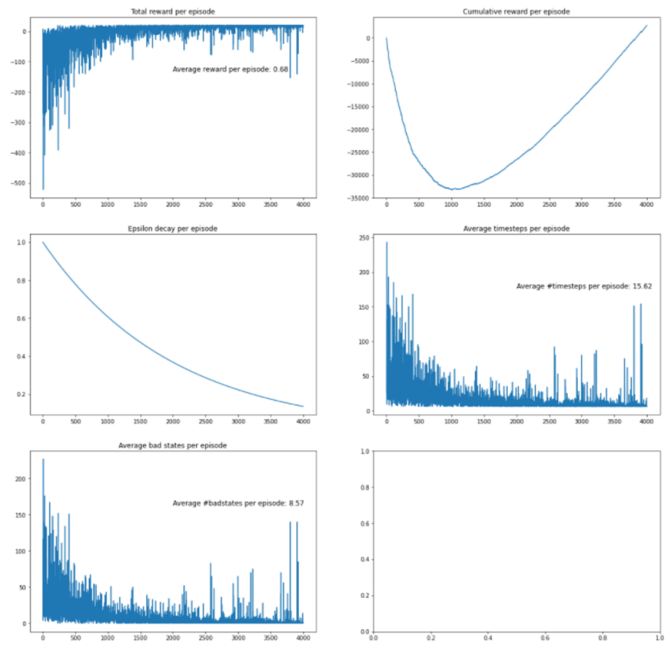
5.1.2 Stochastic Environment

- Epsilon start = 1.0, Epsilon decay factor = 0.9995, Discount factor = 1.0, Episodes = 4000, Step size = 0.01



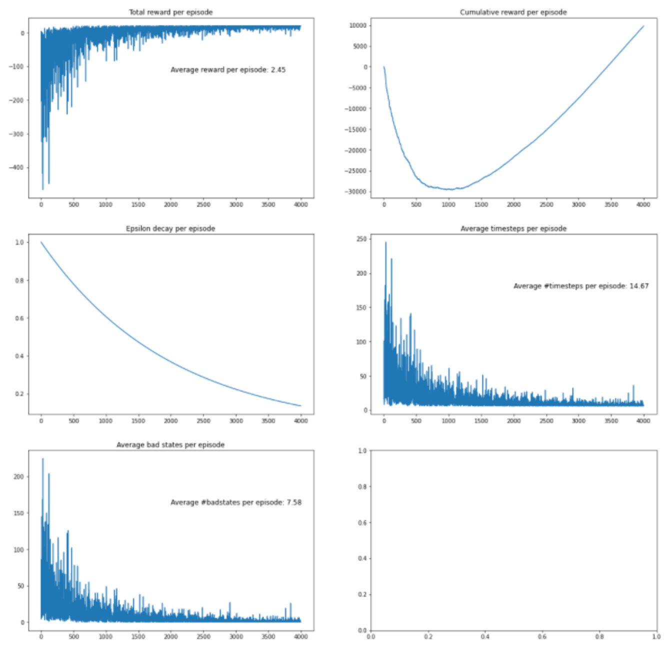
With the above hyper-parameters policy converged at around 1000 episodes with average total reward around 3

- Epsilon start = 1.0, Epsilon decay factor = 0.9995, Discount factor = 0.0, Episodes = 4000, Step size = 0.01



With the above hyper-parameters policy converged at around 1000 episodes with average total reward around 0.5

- Epsilon start = 1.0, Epsilon decay factor = 0.9995, Discount factor = 0.5, Episodes = 4000, Step size = 0.01



With the above hyper-parameters policy converged at around 1000 episodes with average total reward around 2.5

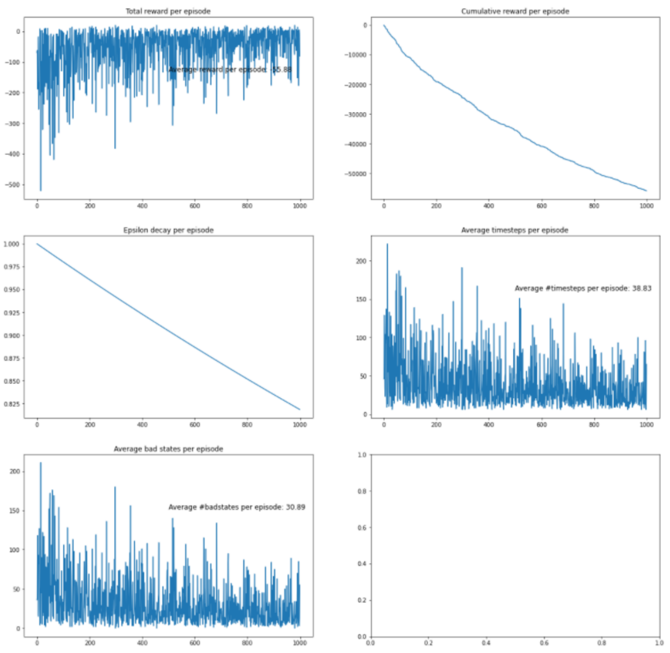
5.2 Tuning Epsilon Decay

The algorithm performed best when the decay rate was set to 0.995; I believe this is because a higher epsilon value lets the agent to explore more; nevertheless, I believe there should be a balance between exploration and exploitation

5.2.1 Deterministic Environment

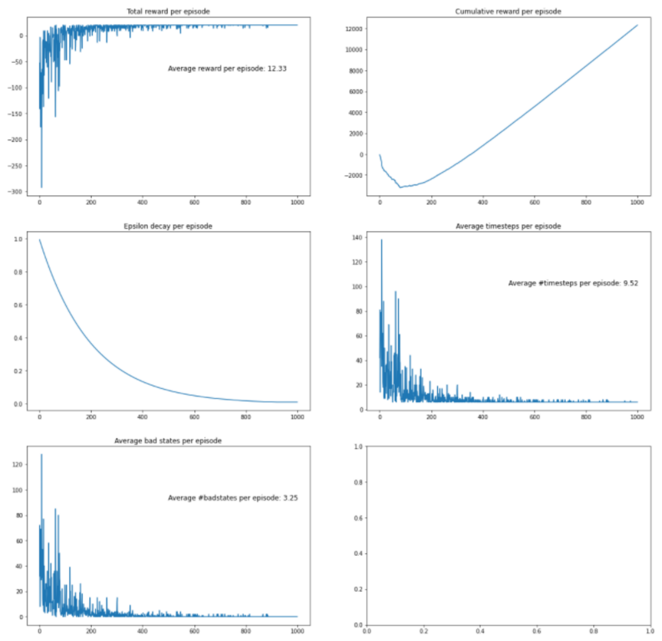
For deterministic environment, among the hyper-parameters tested the one which is most optimal is when (epsilon start = 1.0, decay factor = 0.995, gamma = 0.9, step size = 0.01)

- Epsilon start = 1.0, Epsilon decay factor = 0.9998, Discount factor = 0.9, Episodes = 1000, Step size = 0.01



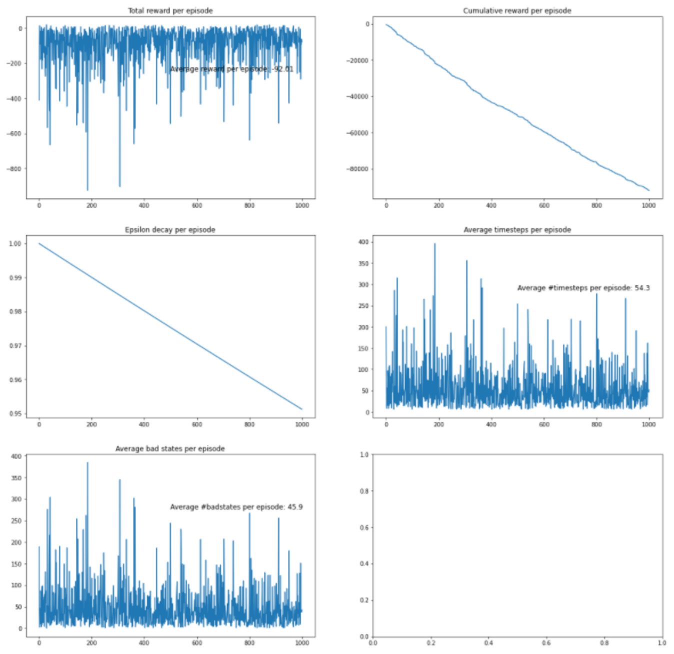
With the above hyper-parameters policy couldn't converge even after 1000 episodes and the average total reward around -55

- Epsilon start = 1.0, Epsilon decay factor = 0.9950, Discount factor = 0.9, Episodes = 1000, Step size = 0.01



With the above hyper-parameters policy converged at around 150 episodes with average total reward around 12

- Epsilon start = 1.0, Epsilon decay factor = 0.99995, Discount factor = 0.9, Episodes = 1000, Step size = 0.01

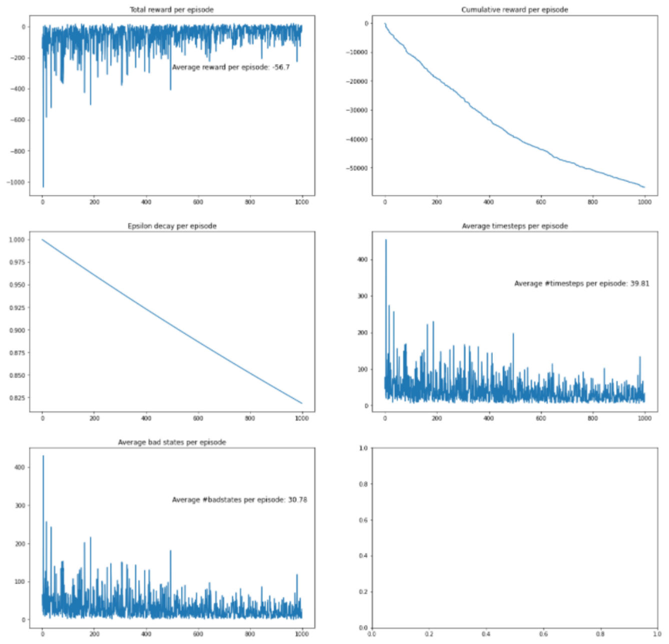


With the above hyper-parameters policy couldn't converge even after 1000 episodes and the average total reward around -90

5.2.2 Stochastic Environment

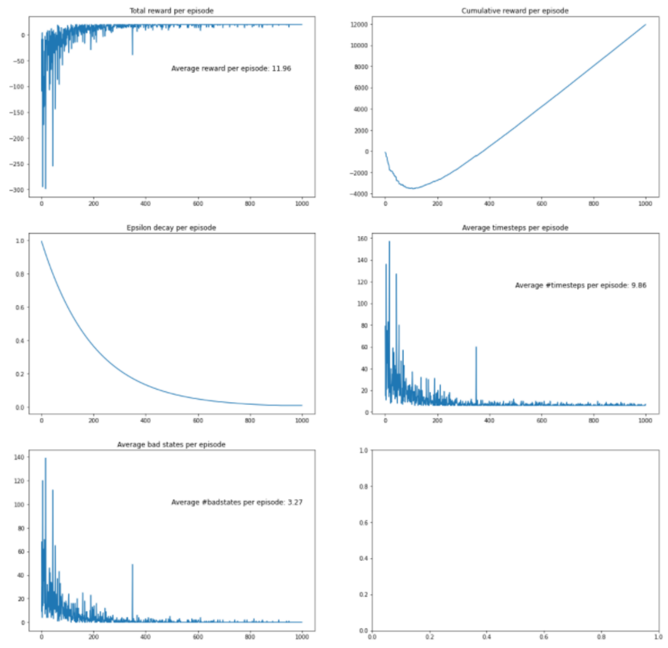
For stochastic environment, among the hyper-parameters tested the one which is most optimal is when (epsilon start = 1.0, decay factor = 0.995, gamma = 0.9, step size = 0.01)

- Epsilon start = 1.0, Epsilon decay factor = 0.9998, Discount factor = 0.9, Episodes = 1000, Step size = 0.01



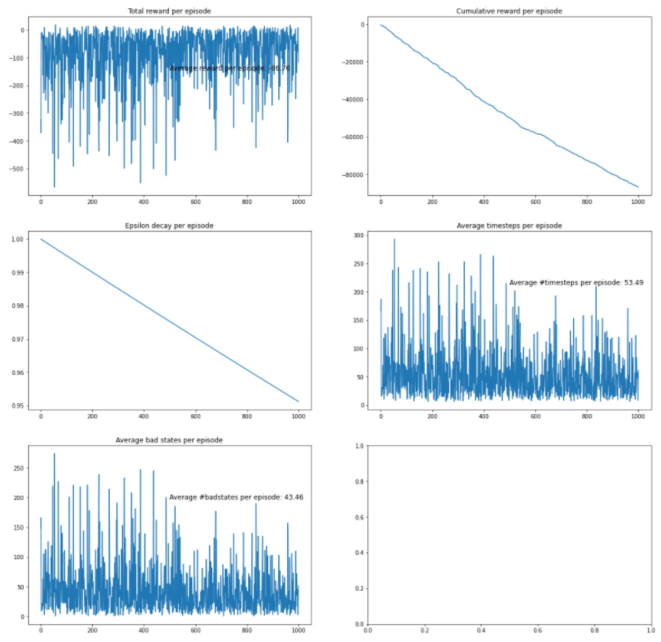
With the above hyper-parameters policy couldn't converge even after 1000 episodes and the average total reward around -56

- Epsilon start = 1.0, Epsilon decay factor = 0.9950, Discount factor = 0.9, Episodes = 1000, Step size = 0.01



With the above hyper-parameters policy converged at around 150 episodes with average total reward around 12

- Epsilon start = 1.0, Epsilon decay factor = 0.99995, Discount factor = 0.9, Episodes = 1000, Step size = 0.01



With the above hyper-parameters policy couldn't converge even after 1000 episodes and the average total reward around -85

6 Monte Carlo - Training

Like Q-Learning, Monte Carlo was also able to converge to an optimal policy based on the epsilon decay rate; the higher the decay rate, the faster it could converge. Furthermore, when the discount factor is near to one, the algorithm performed better

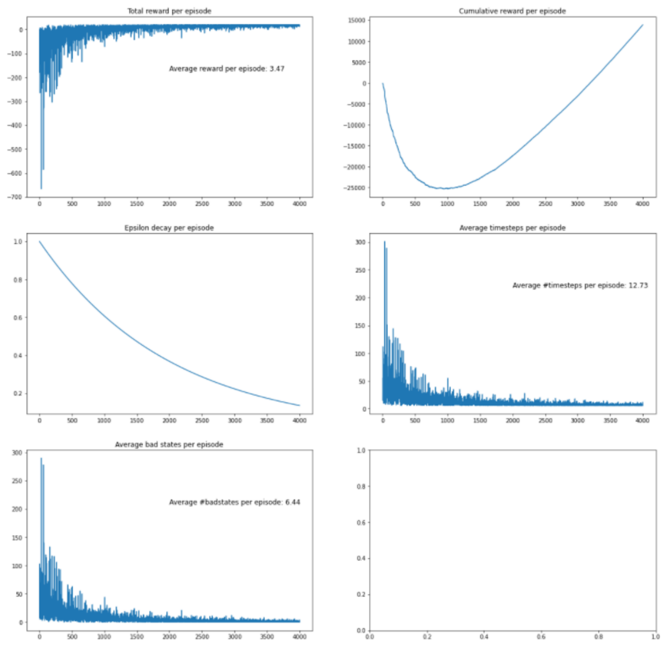
6.1 Tuning Discount Factor

The algorithm functioned best when gamma=1.0; As the discount factor is 1, the returns which was used to update Q value considered all the future state rewards

6.1.1 Deterministic Environment

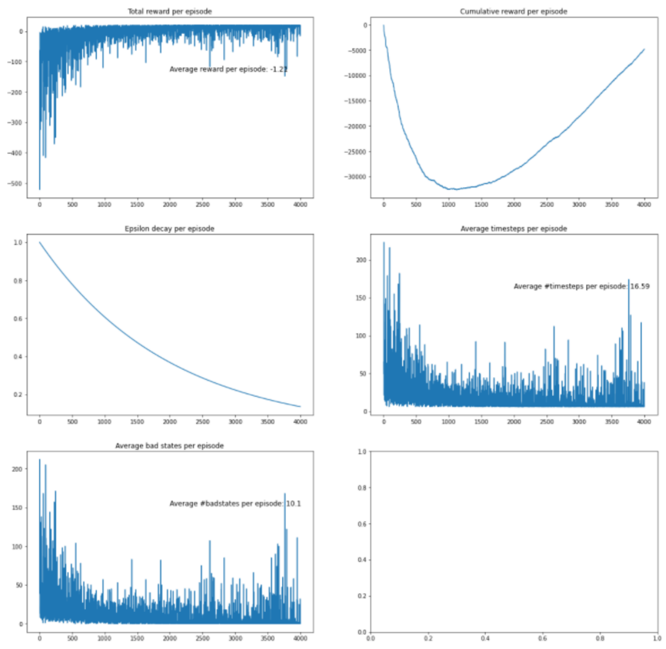
For deterministic environment, among the hyper-parameters tested the one which is most optimal is when (epsilon start = 1.0, decay factor = 0.995, gamma = 0.9)

- Epsilon start = 1.0, Epsilon decay factor = 0.9995, Discount factor = 1.0, Episodes = 4000



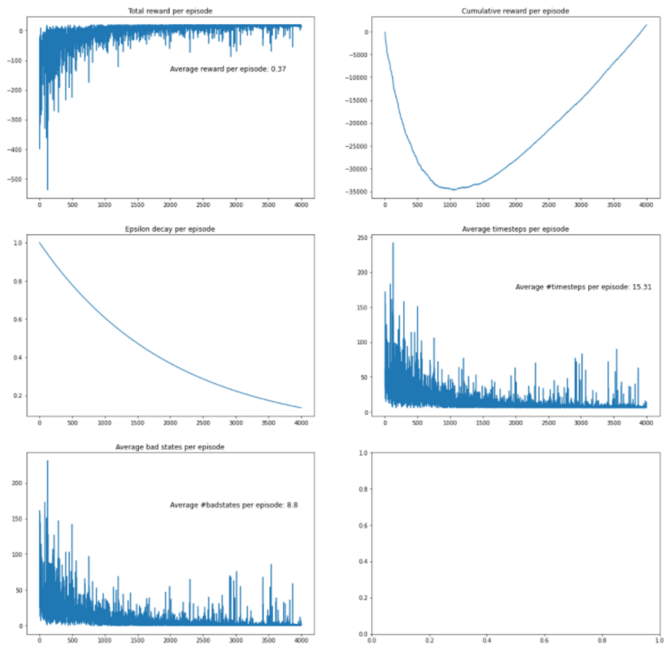
With the above hyper-parameters policy converged at around 1000 episodes with average total reward around 3

- Epsilon start = 1.0, Epsilon decay factor = 0.9995, Discount factor = 0.0, Episodes = 4000



With the above hyper-parameters policy converged at around 1200 episodes with average total reward around -1

- Epsilon start = 1.0, Epsilon decay factor = 0.9995, Discount factor = 0.5, Episodes = 4000

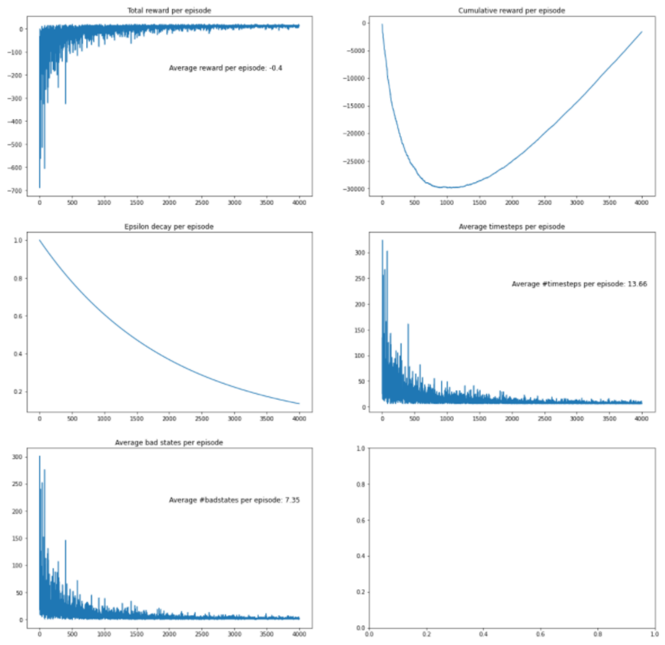


With the above hyper-parameters policy converged at around 1000 episodes with average total reward around 0

6.1.2 Stochastic Environment

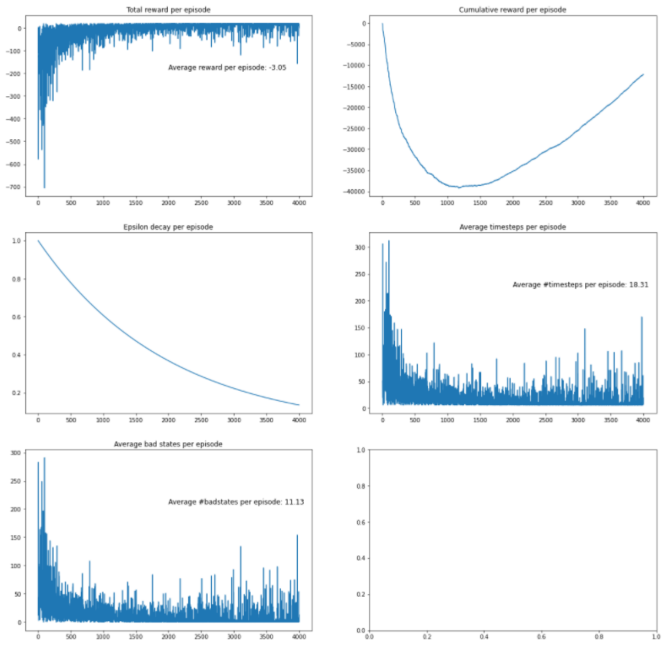
For stochastic environment, among the hyper-parameters tested the one which is most optimal is when (epsilon start = 1.0, decay factor = 0.995, gamma = 1.0)

- Epsilon start = 1.0, Epsilon decay factor = 0.9995, Discount factor = 1.0, Episodes = 4000



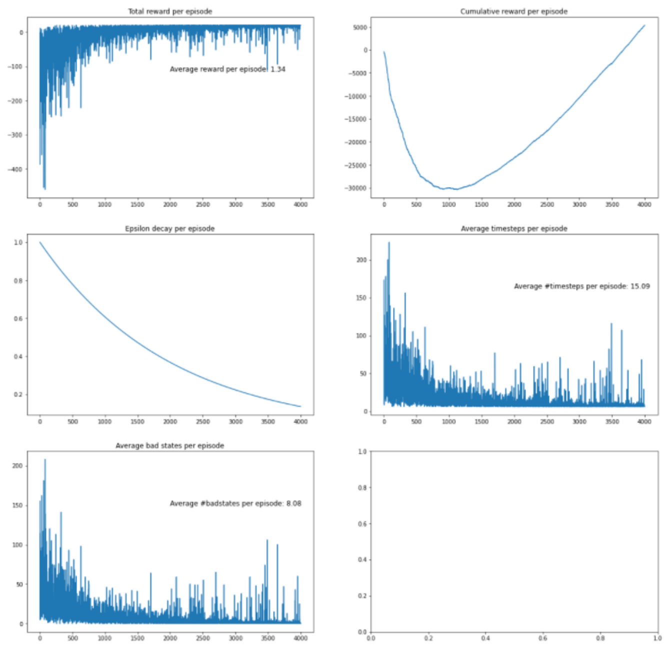
With the above hyper-parameters policy converged at around 1000 episodes with average total reward around 0

- Epsilon start = 1.0, Epsilon decay factor = 0.9995, Discount factor = 0.0, Episodes = 4000



With the above hyper-parameters policy converged at around 1200 episodes with average total reward around -3

- Epsilon start = 1.0, Epsilon decay factor = 0.9995, Discount factor = 0.5, Episodes = 4000



With the above hyper-parameters policy converged at around 1000 episodes with average total reward around 1

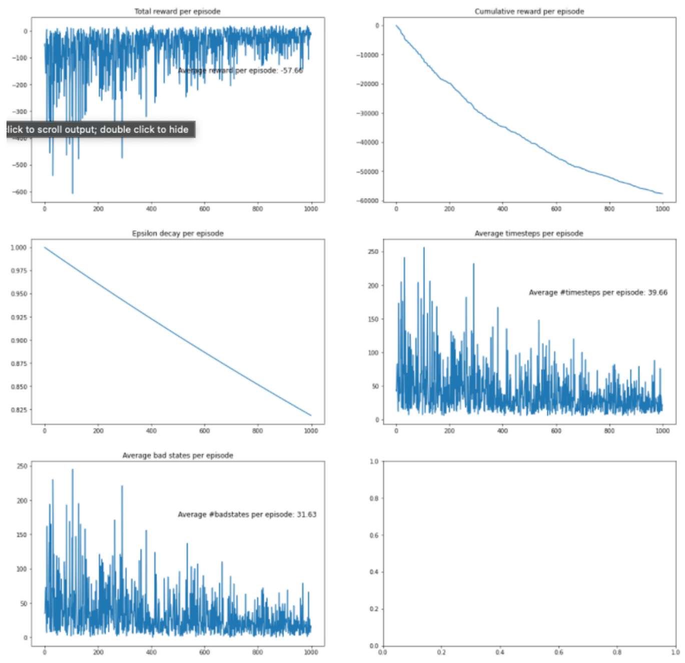
6.2 Tuning Epsilon Decay

The algorithm performed best when the decay rate was set to 0.995; I believe this is because a higher epsilon value causes the agent to explore more hence not taking the greedy steps; nevertheless, I believe there should be a balance between exploration and exploitation.

6.2.1 Deterministic Environment

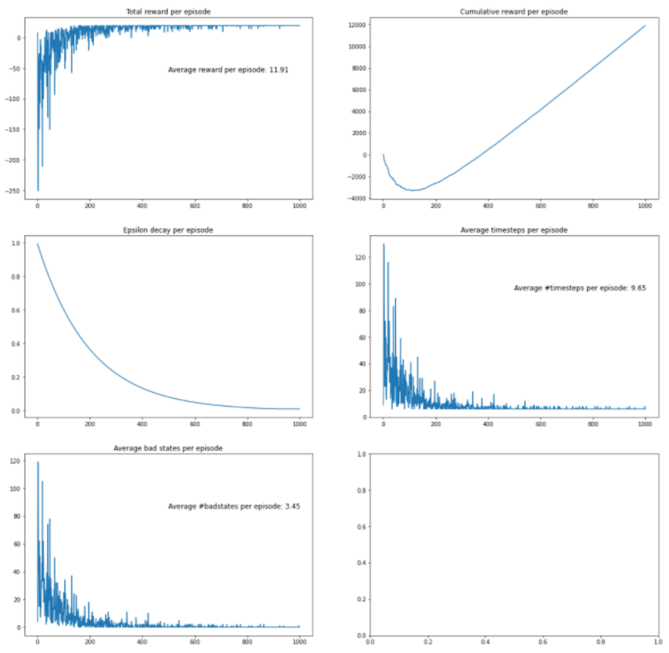
For deterministic environment, among the hyper-parameters tested the one which is most optimal is when (epsilon start = 1.0, decay factor = 0.995, gamma = 0.9)

- Epsilon start = 1.0, Epsilon decay factor = 0.9998, Discount factor = 0.9, Episodes = 1000



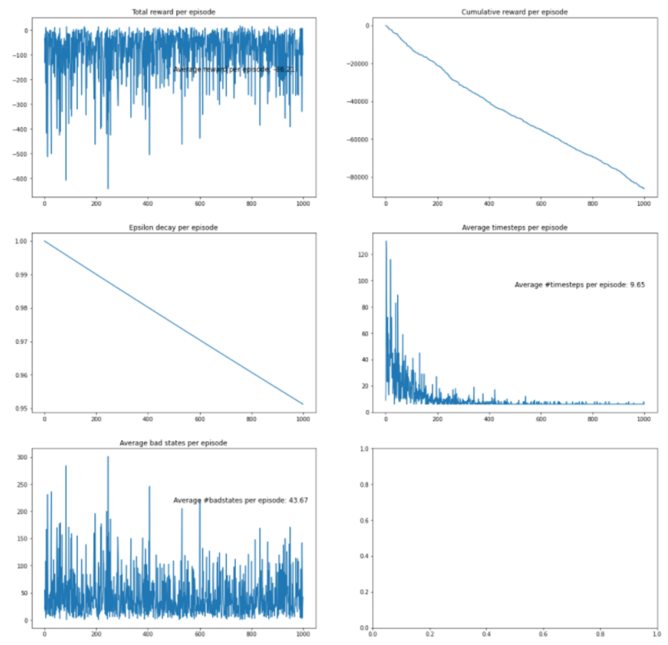
With the above hyper-parameters policy couldn't converge even after 1000 episodes and the average total reward around -58

- Epsilon start = 1.0, Epsilon decay factor = 0.9950, Discount factor = 0.9, Episodes = 1000



With the above hyper-parameters policy converged at around 150 episodes with average total reward around 12

- Epsilon start = 1.0, Epsilon decay factor = 0.99995, Discount factor = 0.9, Episodes = 1000

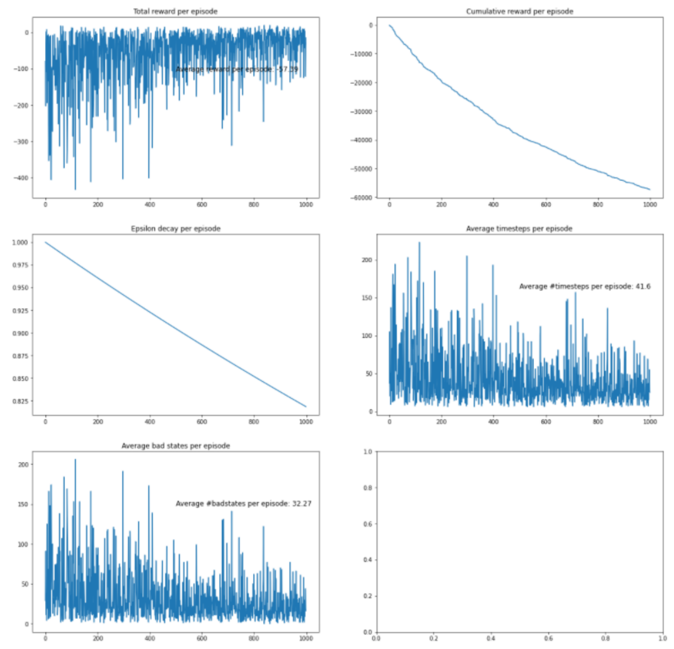


With the above hyper-parameters policy couldn't converge even after 1000 episodes and the average total reward around -86

6.2.2 Stochastic Environment

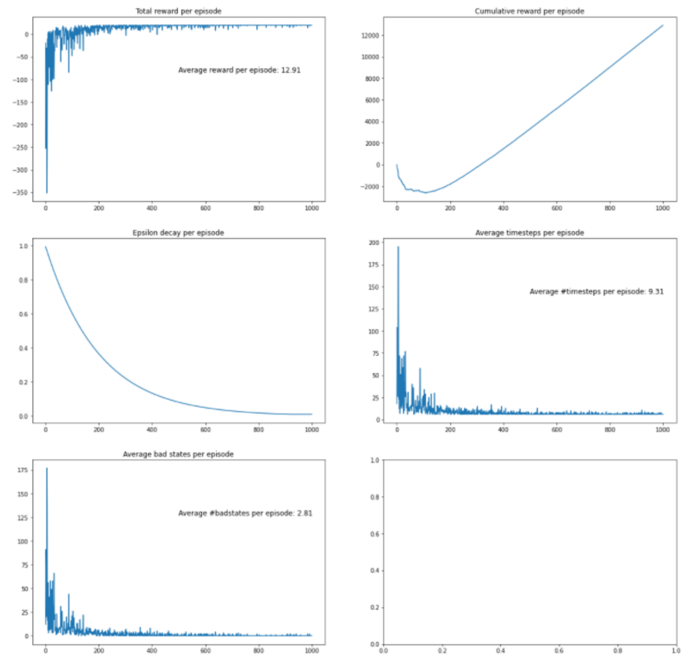
For stochastic environment, among the hyper-parameters tested the one which is most optimal is when (epsilon start = 1.0, decay factor = 0.995, gamma = 1.0)

- Epsilon start = 1.0, Epsilon decay factor = 0.9998, Discount factor = 0.9, Episodes = 1000



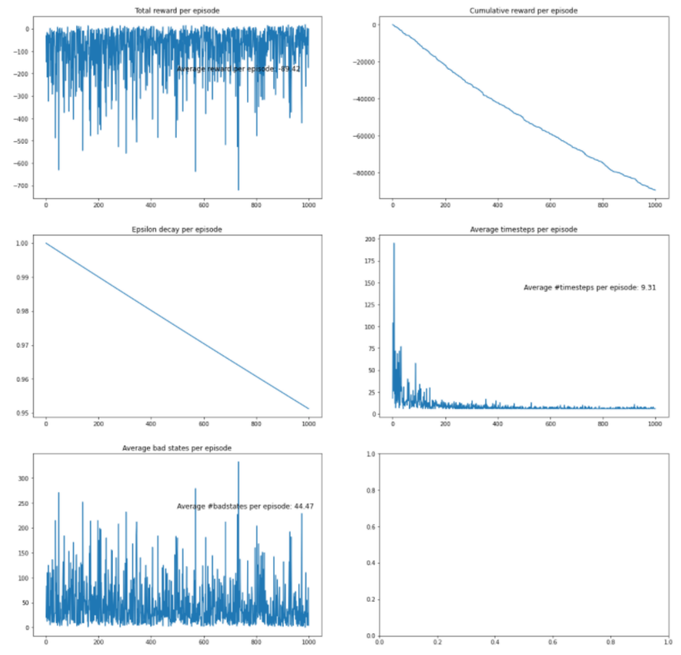
With the above hyper-parameters policy couldn't converge even after 1000 episodes and the average total reward around -57

- Epsilon start = 1.0, Epsilon decay factor = 0.9950, Discount factor = 0.9, Episodes = 1000



With the above hyper-parameters policy converged at around 150 episodes with average total reward around 13

- Epsilon start = 1.0, Epsilon decay factor = 0.99995, Discount factor = 0.9, Episodes = 1000



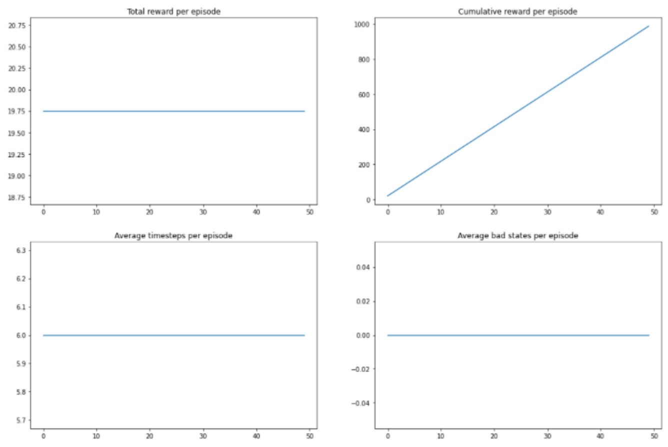
With the above hyper-parameters policy couldn’t converge even after 1000 episodes and the average total reward around -90

7 Q-Learning - Evaluation

A total of 50 episodes were evaluated where the agent takes the best actions as per optimal policy. As the policy is optimal agent traverses through the best path through the maze in deterministic environment but there are slight deviations in the stochastic environment as expected

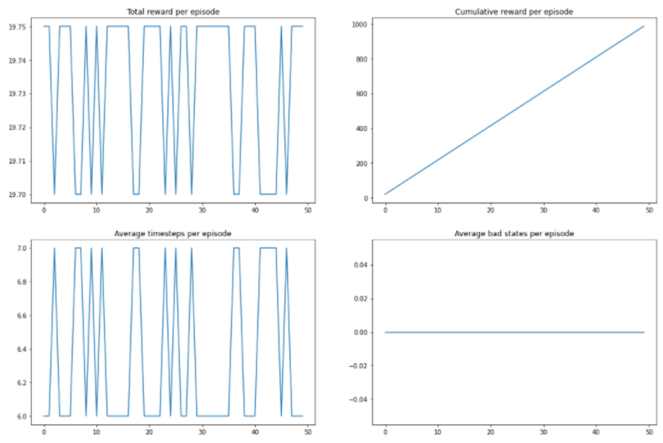
7.1 Deterministic Environment

- Epsilon start = 1.0, Epsilon decay factor = 0.9995, Discount factor = 1.0, Episodes = 4000, Step size = 0.01



7.2 Stochastic Environment

- Epsilon start = 1.0, Epsilon decay factor = 0.9995, Discount factor = 1.0, Episodes = 4000, Step size = 0.01

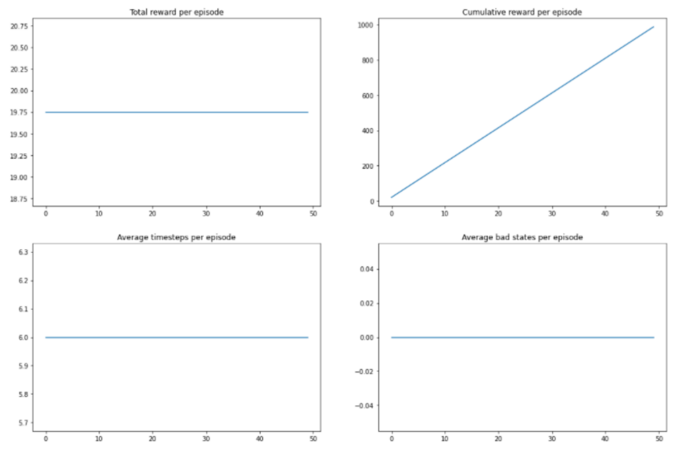


8 Monte Carlo - Evaluation

A total of 50 episodes were evaluated where the agent takes the best actions as per optimal policy. As the policy is optimal agent traverses through the best path through the maze in deterministic environment but there are slight deviations in the stochastic environment as expected

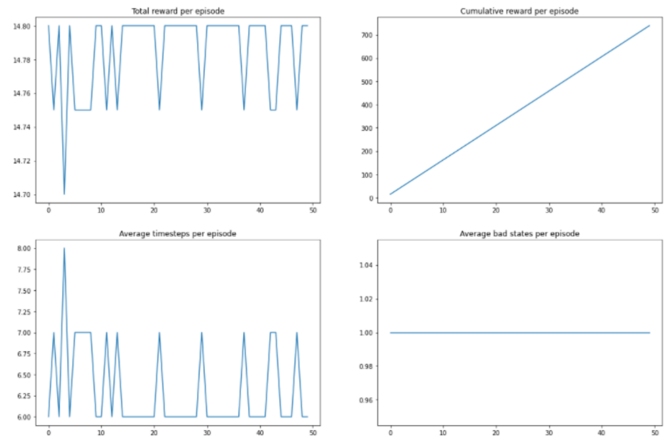
8.1 Deterministic Environment

- Epsilon start = 1.0, Epsilon decay factor = 0.9995, Discount factor = 1.0, Episodes = 4000



8.2 Stochastic Environment

- Epsilon start = 1.0, Epsilon decay factor = 0.9995, Discount factor = 1.0, Episodes = 4000



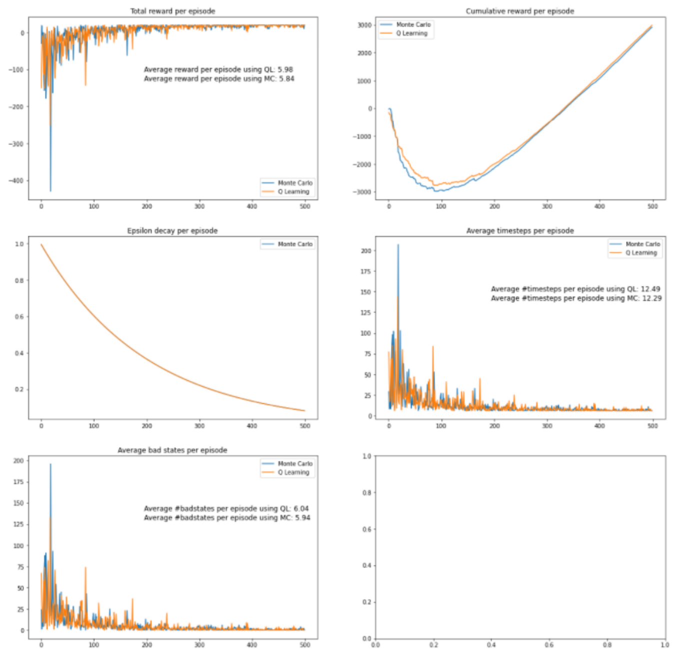
9 Monte Carlo Vs Q-Learning

In the current environment both Monte Carlo and Q-Learning algorithms were able to converge similarly in deterministic environment but Q-Learning is slightly better in stochastic environment. It consistently gave better average reward per episode, lower average no of bad states/time-steps per episode

9.1 Deterministic Environment

With current hyper-parameter configuration both the algorithms were able to converge in around 100 episodes in deterministic environment

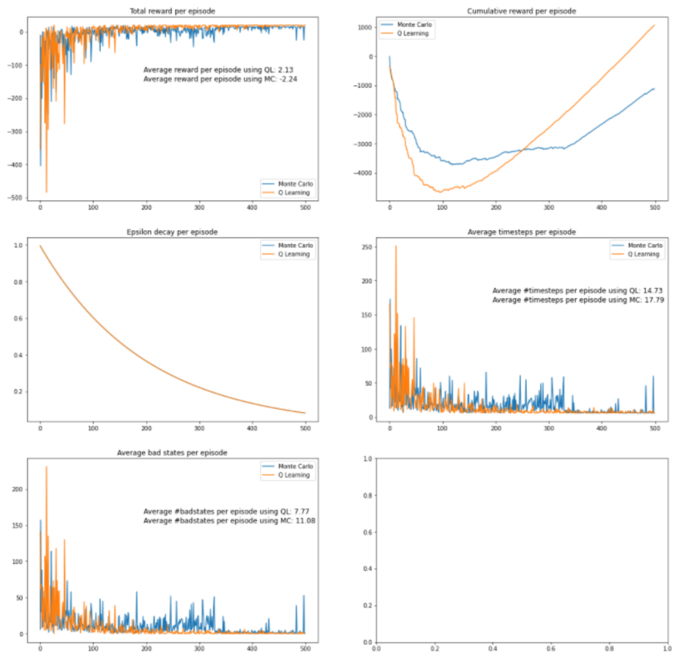
- Epsilon start = 1.0, Epsilon decay factor = 0.995, Discount factor = 1.0, Episodes = 500, Step size = 0.01



9.2 Stochastic Environment

With current hyper-parameter configuration QL converge in around 100 episodes and MC in around 200 episodes in stochastic environment

- Epsilon start = 1.0, Epsilon decay factor = 0.995, Discount factor = 1.0, Episodes = 500, Step size = 0.01


Total reward per episode
Average reward per episode using QL: 2.13
Average reward per episode using MC: -2.24
Monte Carlo
Q Learning
Cumulative reward per episode
Epsilon decay per episode
Average timesteps per episode
Average #timesteps per episode using QL: 14.73
Average #timesteps per episode using MC: 17.79
Average bad states per episode
Average #badstates per episode using QL: 7.77
Average #badstates per episode using MC: 11.08