Defining Reinforcement Learning Environment

Ankit Shaw
Compunter Science and Engineering
University at Buffalo
`ashaw7@buffalo.edu`

Abstract

In Reinforcement Learning (RL), an Environment is the playground where an Agent interacts. Environment have multiple states and the Agent interacts by taking an action to move to the next state. With each action-state pair there is a reward associated which the Agent gets. This reward gives the Agent some hint on whether the action was good or bad. In this paper, we are going to define deterministic and stochastic environment, provide visualization for our custom environments and explain how deterministic environments are different from stochastic environments. Towards the end we will also see how safety checks where put in place for the environments so that the agent only takes actions that are allowed and navigates within the defined state-space. The code for the environment can be found on github.[1]

1 Deterministic Environment

An Environment is said to be Deterministic when next state outcome is determined by the current state and action taken. For instance, in a grid environment if we taken an action left then the agent will always go towards left.

1.1 Definition

1.1.1 States

In our Grid Environment defined the size of the environment is (5 x 5). There are 25 states in total.

Set of States: S = {S0, S1, S2, ... S25}

Each individual cell in the grid is an unique states. These states can have different reward associated to them as well.

1.1.2 Actions

An agent can take four actions in the environment. The actions could be Left, Right, Up, Down. These are denoted by 'L', 'R', 'U', 'D' respectively.

Set of Actions: S = {'L', 'R', 'U', 'D'}

1.1.3 Reward

We have five rewards in our grid environment. Two penalty rewards are -3,2 and there are three positive rewards 2, 4, 10.

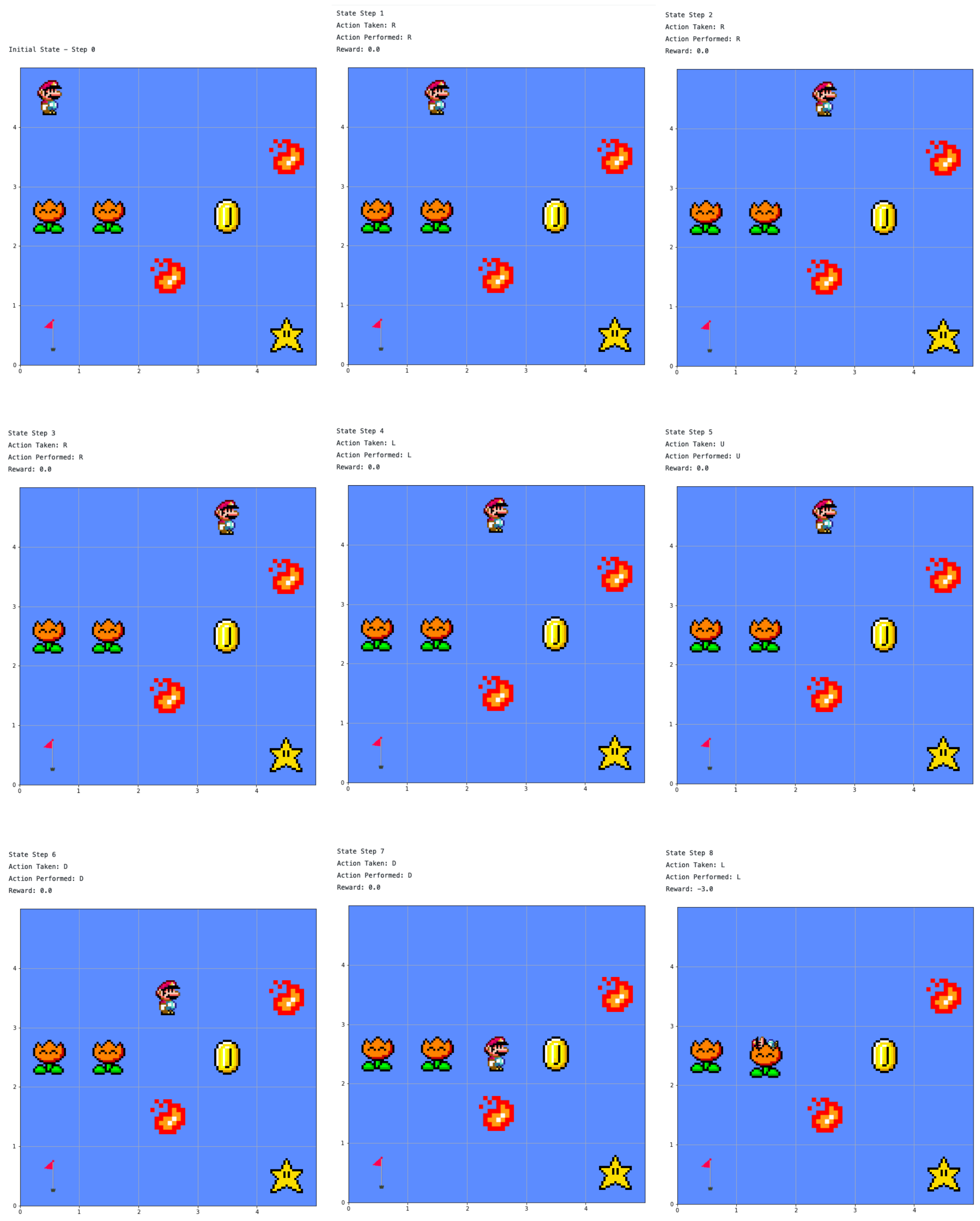
[1]https://github.com/ankitshaw/ub-cse546-reinforcement-learning-a1.git

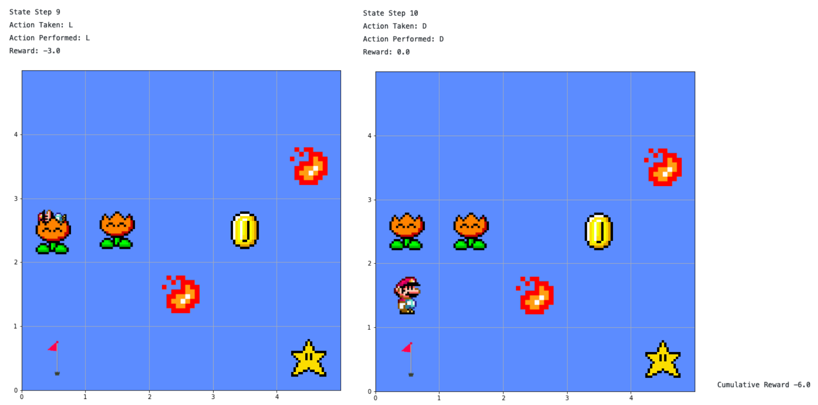
Set of Rewards: R = {-3, -2, 2, 4, 10}

1.2 Objective

The main objective of an agent in the environment is to reach the terminal or goal state and attain the maximum discounted cumulative reward.

1.3 Visualization





2 Stochastic Environment

An Environment is said to be Stochastic when next state outcome cannot not be determined by the current state and action taken. So there is always uncertainty with actions taken. For instance, in a grid environment if we taken an action left then the agent may or may not always go towards left. There is always a probability that the agent may go right or up but not left.

2.1 Definition

2.1.1 States

In our Grid Environment defined the size of the environment is (5 x 5). There are 25 states in total.

$$\text{Set of States: } S = \{S0, S1, S2, ... S25\}$$

Each individual cell in the grid is an unique states. These states can have different reward associated to them as well.

2.1.2 Actions

An agent can take four actions in the environment. The actions could be Left, Right, Up, Down. These are denoted by 'L', 'R', 'U', 'D' respectively.

$$\text{Set of Actions: } S = \{\text{'L', 'R', 'U', 'D'}\}$$

2.1.3 Reward

We have five rewards in our grid environment. Two penalty rewards are -3,2 and there are three positive rewards 2, 4, 10.

$$\text{Set of Rewards: } R = \{-3, -2, 2, 4, 10\}$$

2.2 Objective

The main objective of an agent in the stochastic environment is to reach the terminal or goal state and attain the maximum discounted cumulative reward.

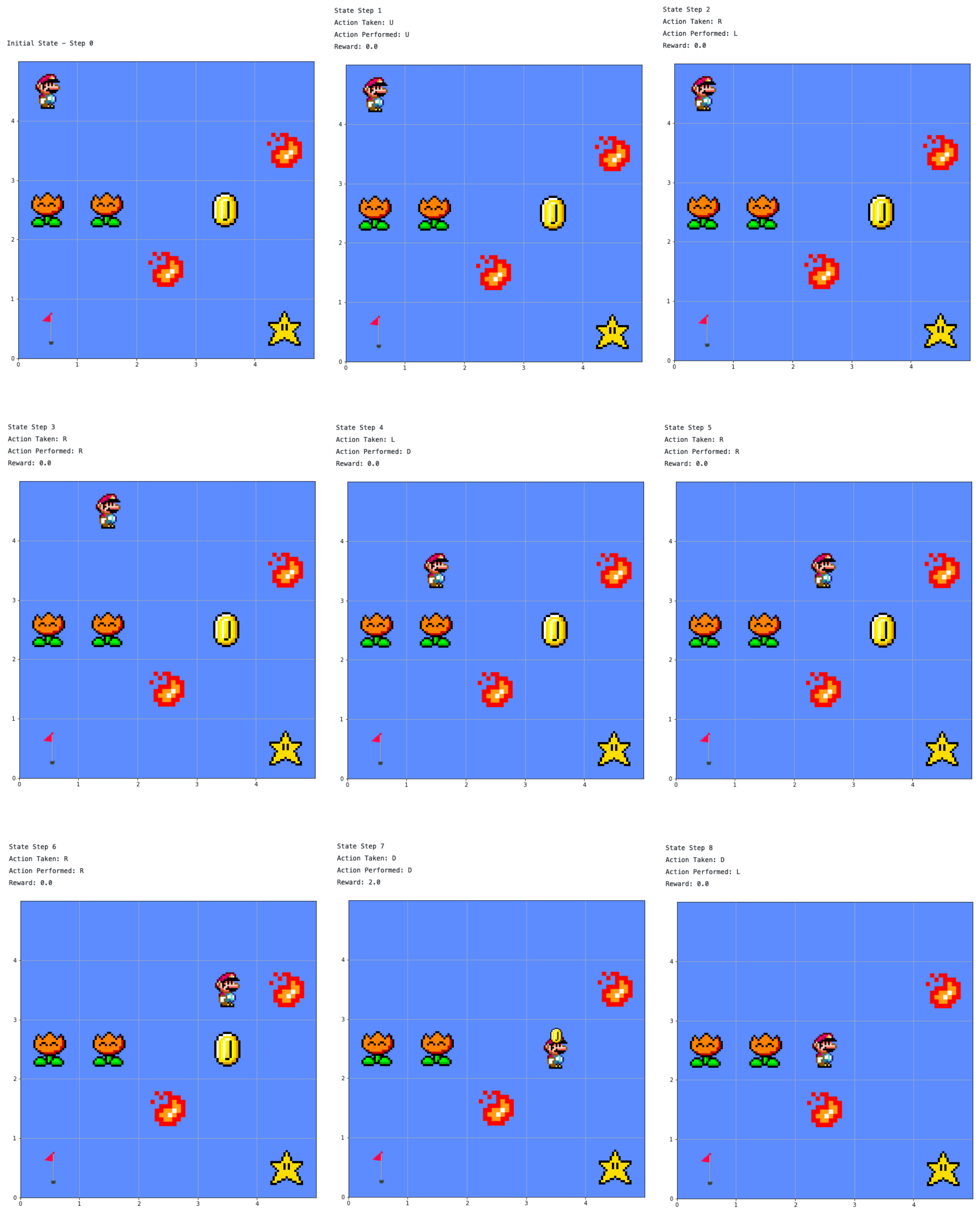
2.3 Stochastic Condition

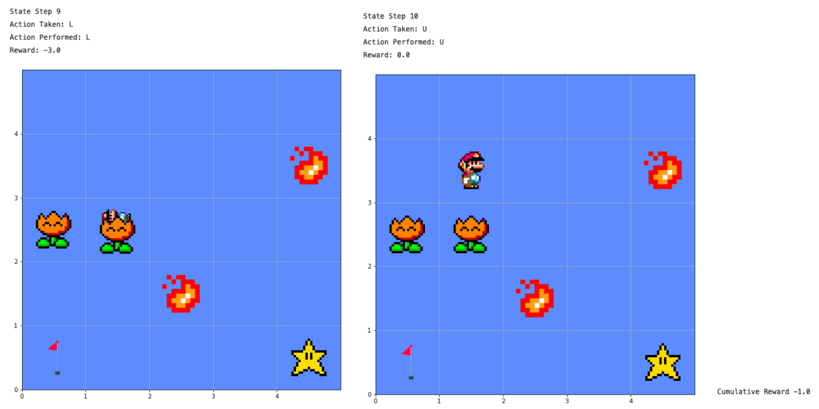
To define the stochastic environment we introduce a parameter: p_transition. p_transition defines the transition probability of taking an action a1 at a given state S1 to move to state s2. To keep things simple we use the same p_transition for all the state-action pair. In simple terms, p_transition tells the probability of executing an action if the action is taken by the agent.

For example: Given a p_transition p, When an agent performs an action a1, then probability that the same action is executed is p. And, the probability executing an action other than a1 is (1 - p_transition)/(action_count - 1).

This is how we introduce the stochasticity or uncertainty in our environment.

2.4 Visualization





3 Difference between Deterministic and Stochastic Environment

The main difference between Deterministic and Stochastic environment is the element of uncertainty associated with an action and state transition. In a Deterministic environment next state outcome is determined by the current state and action taken. For instance, in a grid environment if we taken an action left then the agent will always go towards left. There is no uncertainty in that. While on-the-other in a Stochastic environment next state outcome cannot not be determined by the current state and action taken. There is always an uncertainty associated to the transition. For instance, in a grid environment if we taken an action left then the agent may or may not always go towards left. There is always a probability that the agent may go right or up but not left.

4 Safety in AI

To ensure safety in our environment we design it in such a way that agent interact within a limit of our definition. To ensure that the agent navigates within our defined state-space we do a check on every new state transition to ensure that it lies within the state-space. For our grid environment we check whether the new state coordinates are within the grid shape we defined. Similarly, we filter any actions that are not part of our action set before we execute that action. So if an agent taken an action outside of our action set then the agent remains within that set.

5 Reference

1. Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction.
2. Lecture Slides
3. Icons for Rendering credit - Sandro Pereira/ph03nyX, ph03nyx.deviantart.com

Solving Reinforcement Learning Environment

Ankit Shaw
Compunter Science and Engineering
University at Buffalo
ashaw7@buffalo.edu

Abstract

In Reinforcement Learning (RL), there are different algorithms and methods to solve an environment such that an Agent learns the optimal policy to navigate in the environment to reach the end goal with maximum reward. In this report we will briefly go through Tabular Solution methods for training our Agent to find optimal policy. We then explain two tabular methods and use it to train our Mario Agent on the environment that we had defined in Part 1. Finally we will evaluate the methods and compare their performance on both Deterministic and Stochastic Environment. The code for the environment can be found on github.[1]

1 Tabular Solution Method

Tabular Solution methods are used to solve Reinforcement Learning problems in which the state and action space are small enough to represent the approximate value function as tables or arrays.[1] These problems are simplest form of RL problems and we often find the exact optimal policy. Following are different methods:

1. Dynamic Programming.
   (a) Policy Evaluation
   (b) Policy Iteration
2. Monte Carlo
   (a) First-Visit Monte Carlo Estimation
   (b) Every-Visit Monte Carlo Estimation
3. Temporal Difference
   (a) TD(0)
   (b) SARSA
   (c) Q-Learning
   (d) Double Q-Learning

From the above methods we will be using **Q-Learning and SARSA** methods to train our agent to solve the RL environment.

2 SARSA

SARSA is an on-policy model-free reinforcement learning algorithm that learns the optimal policy by using the current estimate of the optimal policy to update the Q-table and generate new behavior. The Q-values of each state is updated based Q-value of next state and next action pair.

[1]https://github.com/ankitshaw/ub-cse546-reinforcement-learning-a1.git

Some of the features of SARSA are:

1. It is model free i.e, it doesn't require model of the environment.
2. It is an on-policy algorithm.

2.1 Update Function

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

$S_t - A_t \leftarrow$ Current State-Action pair
$S_{t+1} \leftarrow$ Next state
$A_{t+1} \leftarrow$ Next Action $R_t \leftarrow$ Immediate reward for the State-Action pair
$Q(S_t, A_t) \leftarrow$ Q-Value for Current State-Action pair / Prediction Value
$R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) \leftarrow$ Target Value
$R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \leftarrow$ Loss
$Q(S_{t+1}, A_{t+1}) \leftarrow$ Q-Value for the next state next action pair. $\alpha \leftarrow$ Learning Rate
$\gamma \leftarrow$ Discount Factor

2.2 Algorithm

```
Initialise Q-Table
Loop over episodes:
    Reset environment
    From current state select action A using e-greedy policy
    Loop over steps:
        Take Action A and observe next state S'
        From next state S' select action A'.
        Using update function update Q-Table value for current S-A pair
        Set current state S to next state S'
        Set current action A to next action A'
        Break if terminal state reached
```

2.3 Hyperparameters

Hyperparameter are values that are used to control the learning in Machine Learning and Reinforcement Learning algorithms. Having correct set of parameters are important for successfully train the algorithms. The important hyperparameters we need to decide on before we start training are:

1. $\alpha \leftarrow$ Learning Rate
2. $\gamma \leftarrow$ Discount Factor
3. $\lambda \leftarrow$ Decay Rate
4. $\epsilon \leftarrow$ init. e-greedy policy
5. $episodes \leftarrow$no of episodes
6. $steps \leftarrow$ max step size of each episode

2.4 Training - Deterministic Environment

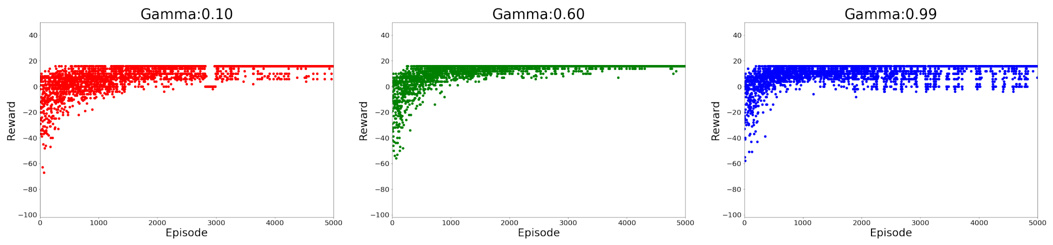
The agent for our environment is trained using the algorithm SARSA in the previous section written in Python programming language.

2.4.1 Hyperparameter Tuning

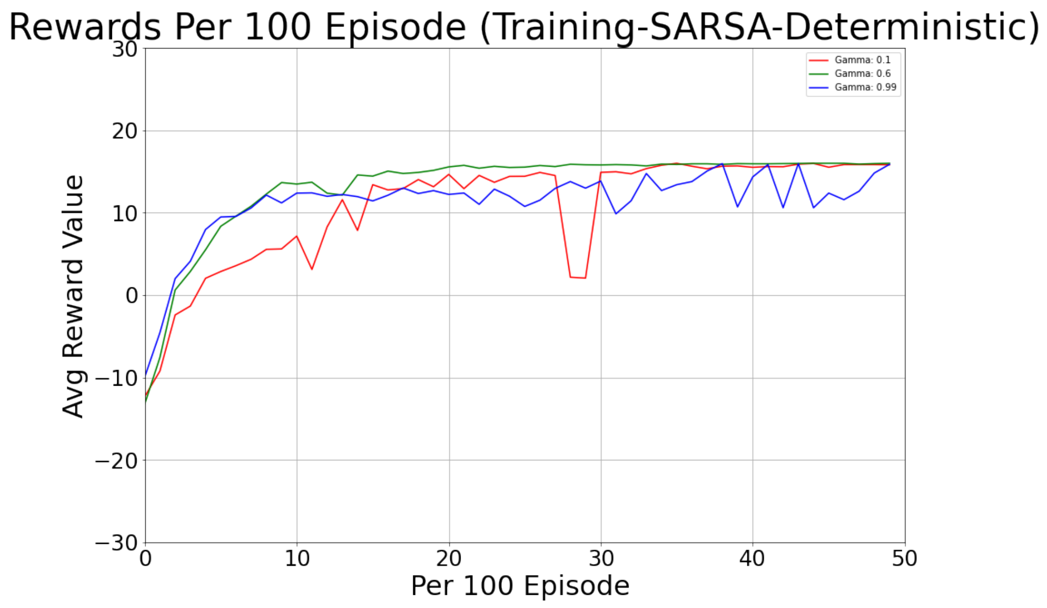
For better training result we fine-tuned the parameters discount factor γ and step size per episode $steps$ to get the optimal values for our training.

2.3.1.1. Discount Factor - γ

For fine-tuning γ we try with three different values **0.10, 0.60, 0.99**. Following are the results that we observed. In the visualization in the next page we can see the rewards per episode for different values of gamma. We can observe that the convergence is slowest for $\gamma = 0.10$ followed



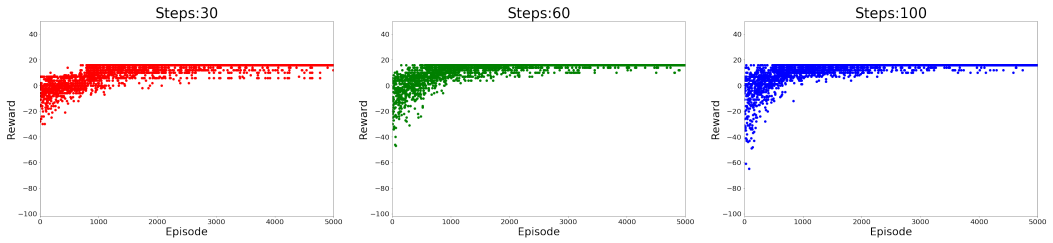
by $\gamma = 0.6$ and then $\gamma = 0.99$ Another thing we observe that for $\gamma = 0.99$ the convergence is slightly faster than $\gamma = 0.60$ initially. But at the same time the reward values are sparse for former value than the latter. The rewards are much more stable for $\gamma = 0.99$ as we progress with more episodes. Let us now also see the average rewards for every 100 episode to understand this better. From the average rewards per 100 episode graph we can clearly see that for $\gamma = 0.60$ the



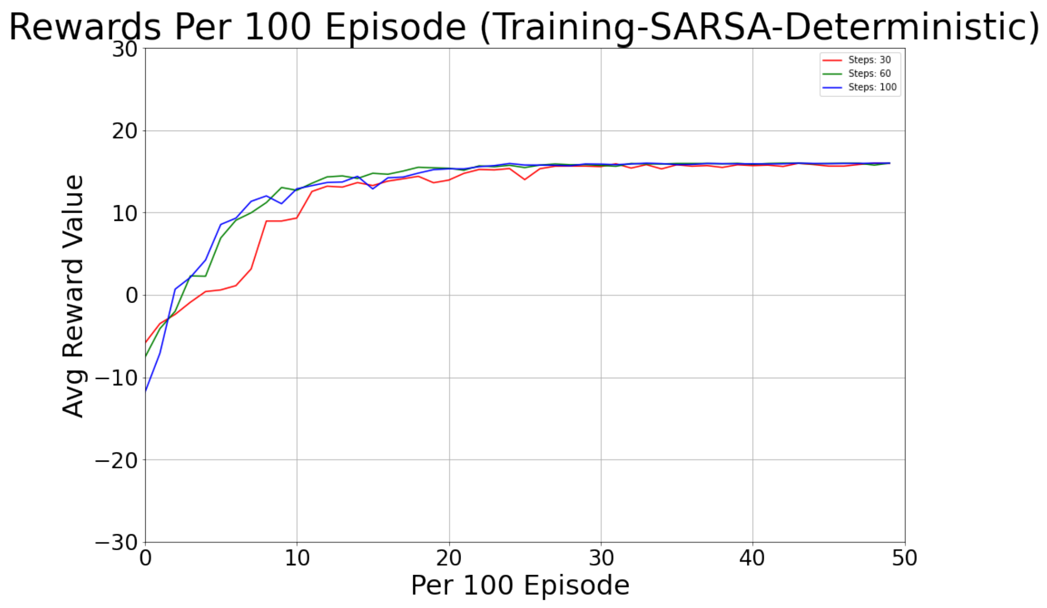
convergence is slightly slower than $\gamma = 0.90$. But the average rewards are better for $\gamma = 0.60$ as the episodes progress. We can understand from this that for gamma value close to 0 the agent cares more about the immediate reward. While for gamma value closer to 1 the agent gives more importance to the future long-term reward. Hence, this could be the reason for sparser rewards value per episode as the episodes progress for $\gamma = 0.90$. The agent is looking to further improve rewards in the long-term. For our problem we go with the gamma value 0.60 for training.

2.3.1.2. Step Size - $steps$

For fine-tuning $steps$ i.e the maximum step size per episode, we try with three different values bf 30, 60, 100. Following are the results that we observed.



In the above visualization can see the rewards per episode for different values of steps. We can observe that the convergence is slowest for $steps = 30$ followed by $steps = 60$ and then $steps = 100$ The convergence for $steps = 100$ is slightly faster than $steps = 60$, even though the rewards for initial few episodes for the former were worst than that of latter steps size.
Let us now also see the average rewards for every 100 episode to understand this better. From the



average rewards per 100 episode graph we can clearly see that the average rewards for $steps = 100$ was worse than the other two values. But the policy improve very quickly, it beats the performance with $steps = 30$ and $steps = 60$and the average reward is better throughout the later episodes. What we can understand from this is that larger steps size is better for our agent to learn for our environment. The larger step size gives the agent more time to explore the environment and collect more information about the environment every episode than the smaller steps size. Also given, we are using epsilon-greedy policy of exploration and exploitation, we want to explore more in the initial episodes for better performance. For our problem we select the steps size 100 for training.

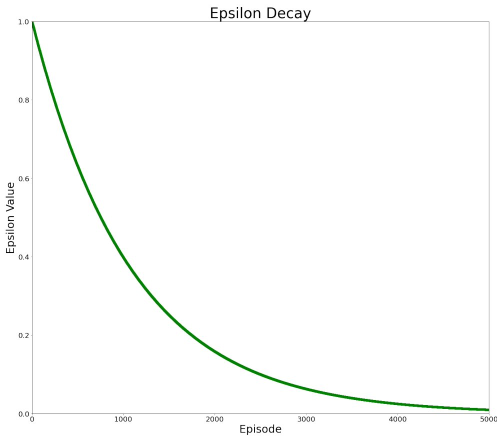
2.4.2 Training

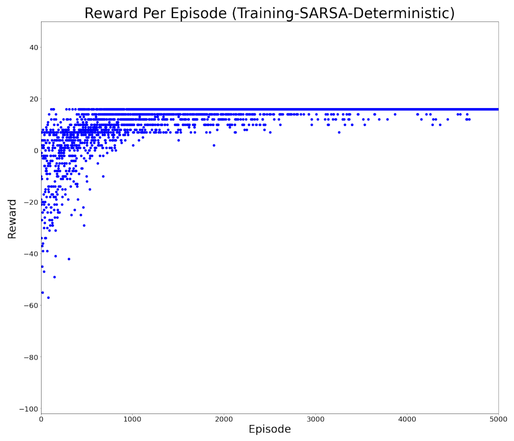
After we tune the parameters as seen in the previous section we move to training the algorithm with the other set of hyperparameters. The final hyperparameters values we use for the training are as follows:

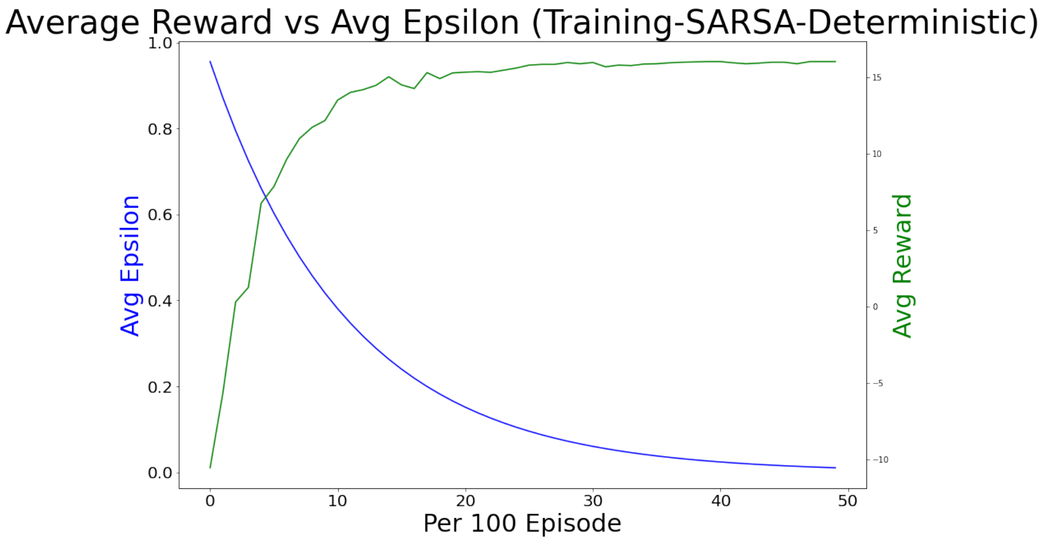
1. $\alpha \leftarrow 0.10$
2. $\gamma \leftarrow 0.6$
3. $\lambda \leftarrow 0.999$
4. $\epsilon \leftarrow 1.0$
5. $episodes \leftarrow 5000$

6. $steps \leftarrow 100$

Following are the results after training SARSA for Deterministic Environment. The first graph is of the epsilon decay over time and the second graphs shows the reward received per episode. In the third graph, we can see the average rewards trend with respect to avg epsilon over every 100 episodes. We can clearly see that over time the agent learns the optimal policy and is able to receive the max reward possible i.e 16.

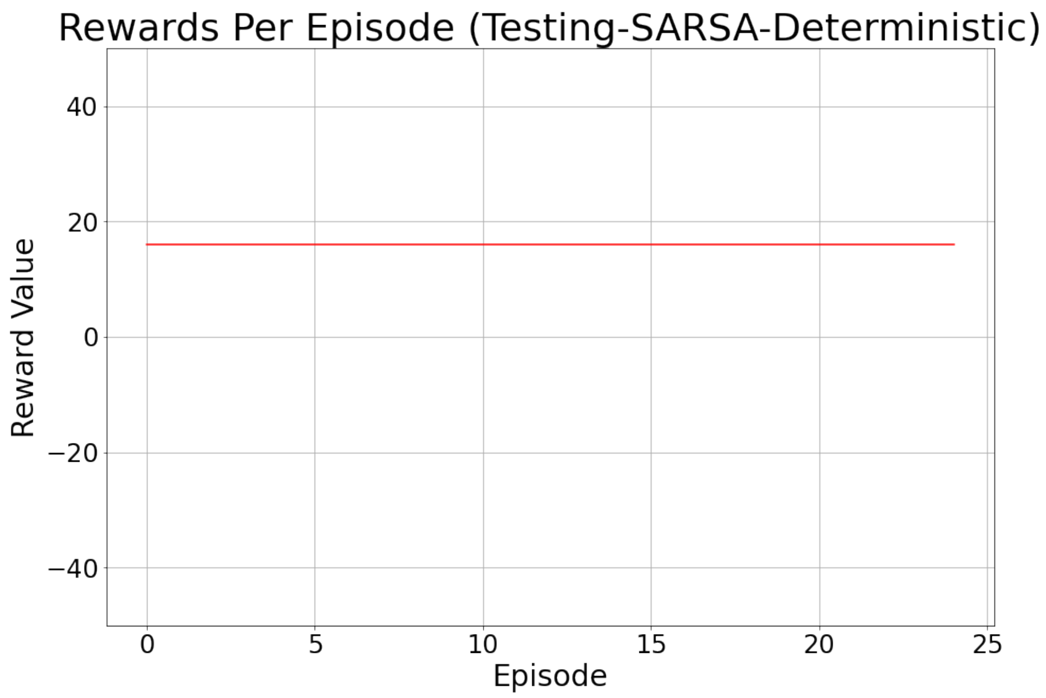






2.4.3 Evaluation

Let us evaluate the policy learned by agent. For evaluation the agent select the greedy action from the learnt policy. Following is the result of running the agent for 25 episodes.



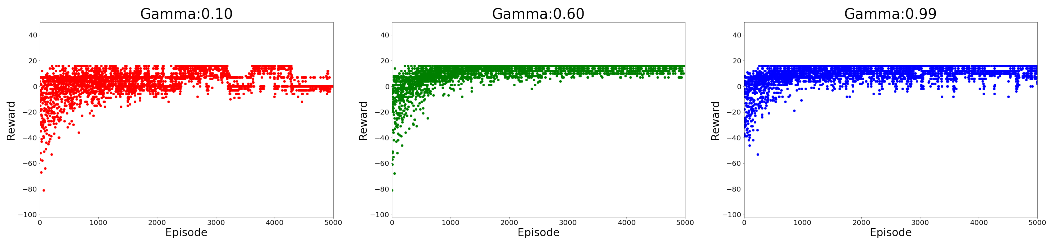
2.5 Training - Stochastic Environment

2.5.1 Hyperparameter Tuning

We do the same parameters tuning for stochastic environment.

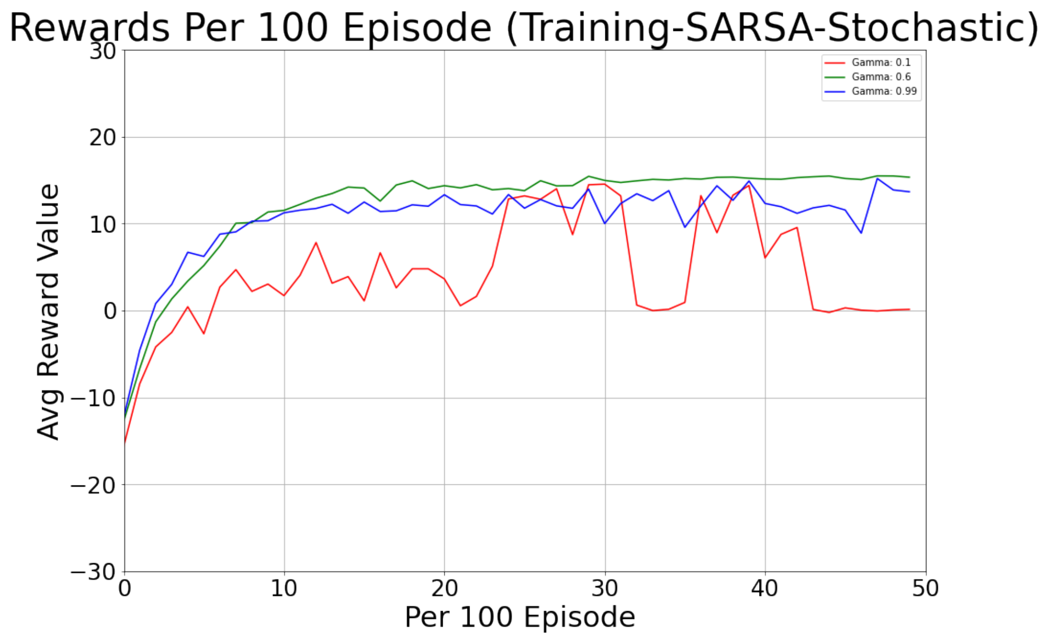
2.5.1.1. Discount Factor - γ

For fine-tuning γ we try with three different values bf 0.10, 0.60, 0.99. Following are the results that we observed, which are very similar to what we got for Deterministic environment. In the visualization we can see the rewards per episode for different values of gamma. We can observe



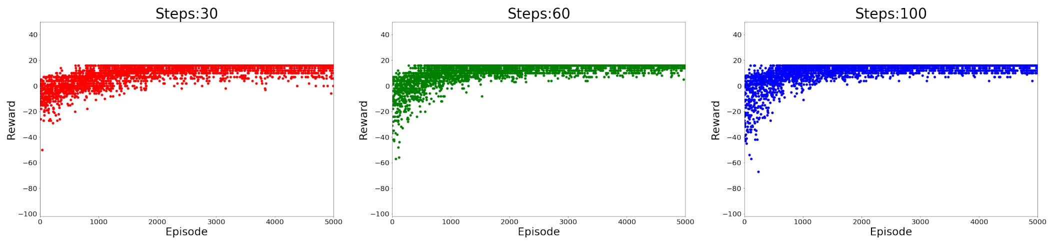
that the convergence is slowest for $\gamma = 0.10$ followed by $\gamma = 0.6$ and then $\gamma = 0.99$ during the initial few episodes. The rewards are much more stable for $\gamma = 0.60$ than $\gamma = 0.99$ as we progress with more episodes. Let us now also see the average rewards for every 100 episode to understand this better.

From the average rewards per 100 episode graph we can clearly see that for $\gamma = 0.60$ the convergence is slightly slower than $\gamma = 0.90$. But the average rewards are better for $\gamma = 0.60$ as the episodes progress as the episodes progress. We observe that $\gamma = 0.60$ though had lower average rewards in the initial episodes beats the other two values in the later episodes. Hence, we choose the $\gamma = 0.60$ for our problem.



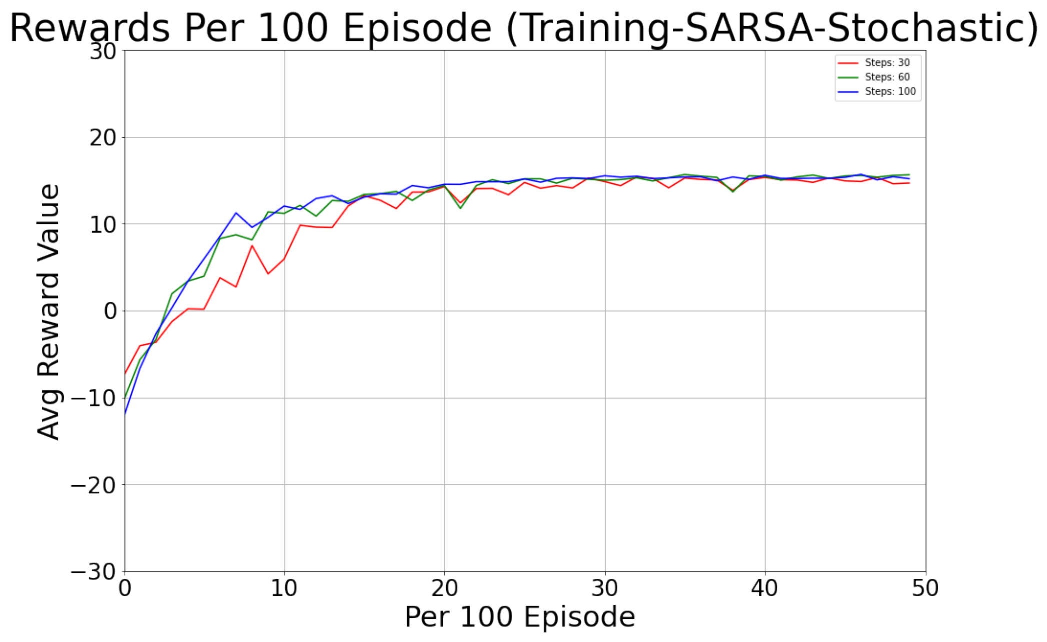
2.5.1.2. Step Size - *steps*

For fine-tuning $steps$ i.e the maximum step size per episode, we try with three different values bf 30, 60, 100. Following are the results that we observed. In the above visualization



can see the rewards per episode for different values of steps. We can observe that the convergence is slowest for $steps = 30$ while the performance with $steps = 60$ and $steps = 100$ are very similar. Let us now also see the average rewards for every 100 episode to understand this better.

From the average rewards per 100 episode graph we can clearly see that the average rewards for $steps = 100$ and $steps = 60$ have very similar performance throughout all the episodes. This again confirms our conclusion that large step size helps agent perform better, as it is able to explore more in the environment. Looking at the results we select the steps size 100 for training.

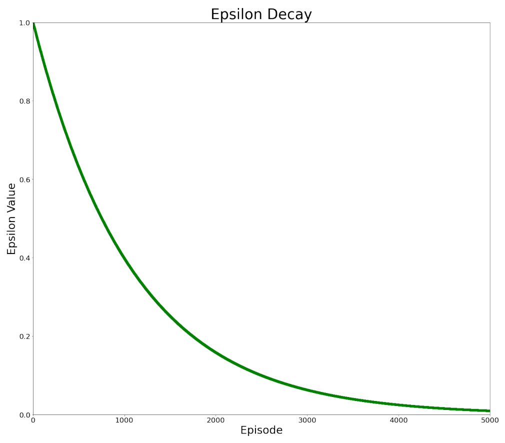


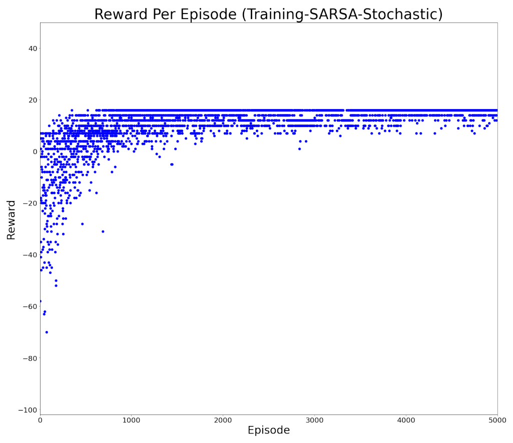
2.5.2 Training

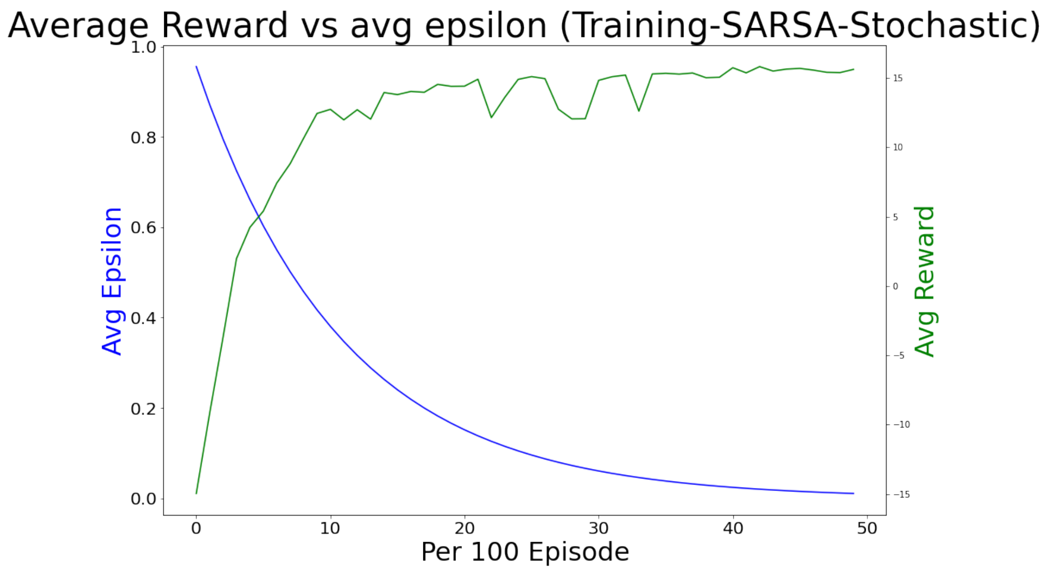
The final hyperparameters values we use for the training are as follows:

1. $\alpha \leftarrow 0.10$
2. $\gamma \leftarrow 0.6$
3. $\lambda \leftarrow 0.999$
4. $\epsilon \leftarrow 1.0$
5. $episodes \leftarrow 5000$
6. $steps \leftarrow 100$

Following are the results after training SARSA for Stochastic Environment. The first graph is of the epsilon decay over time and the second graphs shows the reward received per episode. In the third graph, we can see the average rewards trend with respect to avg epsilon over every 100 episodes. Because of stochasticity in the environment it some times end up in the penalty cells which reduces it's maximum reward.

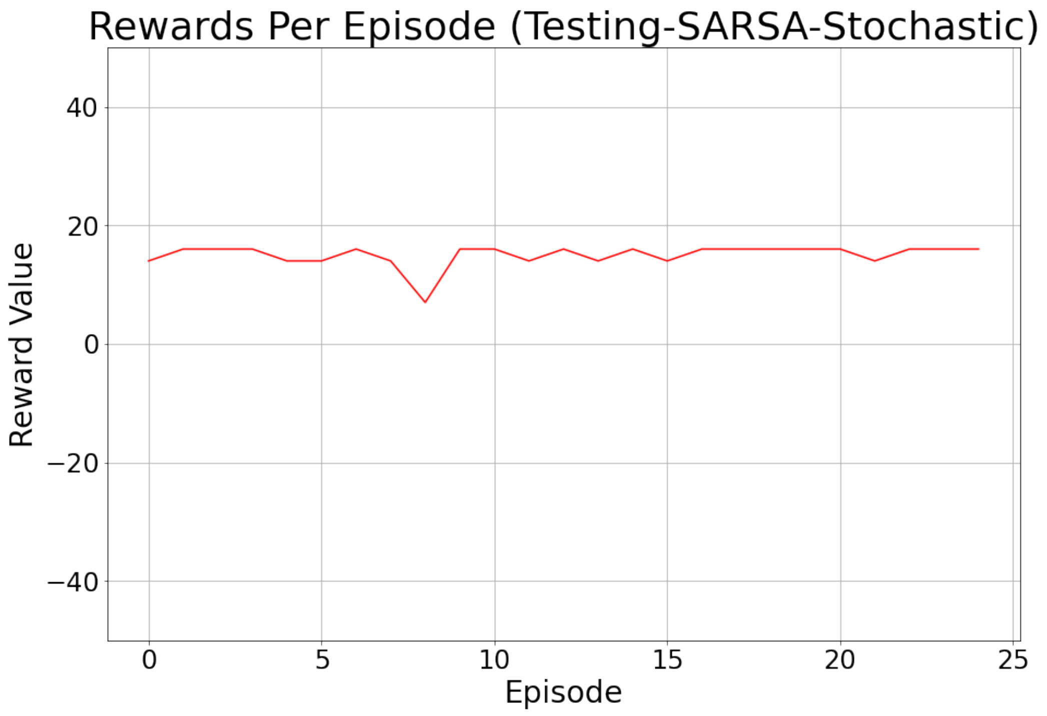






2.5.3 Evaluation

Let us evaluate the policy learned by agent. For evaluation the agent select the greedy action from the learnt policy. Following is the result of running the agent for 25 episodes.



3 Q-Learning

Q-Learning is an off-policy model-free reinforcement learning algorithm that directly approximates the optimal action-value function (q*) from the learned action-value function. It uses only the next states best action to estimate the current state Q-value, hence off-policy.

Some of the features of Q-Learning are:

1. It is model free i.e, it doesn't require model of the environment.
2. It is independent of the agent policy hence an off-policy algorithm.

3.1 Update Function

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\,[R_{t+1} + \gamma\, max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

$S_t - A_t \leftarrow$ Current State-Action pair
$S_{t+1} \leftarrow$ Next state
$R_t \leftarrow$ Immediate reward for the State-Action pair
$Q(S_t, A_t) \leftarrow$ Q-Value for Current State-Action pair / Prediction Value
$R_{t+1} + \gamma\, max_a Q(S_{t+1}, a) \leftarrow$ Target Value
$R_{t+1} + \gamma\, max_a Q(S_{t+1}, a) - Q(S_t, A_t) \leftarrow$ Loss
$\alpha \leftarrow$ Learning Rate
$\gamma \leftarrow$ Discount Factor

3.2 Algorithm

```
Initialise Q-Table
Loop over episodes:
    Reset environment
    Loop over steps:
        From current State S select Action A using e-greedy policy
        Take Action A and observe next state S'
        Using update function update Q-Table Value for current S-A pair
        Set current state S to next state S'
        Break if terminal state reached
```

3.3 Hyperparameters

Hyperparameter are values that are used to control the learning in Machine Learning and Reinforcement Learning algorithms. Having correct set of parameters are important for successfully train the algorithms. The important hyperparameters we need to decide on before we start training are:

1. $\alpha \leftarrow$ Learning Rate
2. $\gamma \leftarrow$ Discount Factor
3. $\lambda \leftarrow$ Decay Rate
4. $\epsilon \leftarrow$ init. e-greedy policy
5. $episodes \leftarrow$no of episodes
6. $steps \leftarrow$ max step size of each episode

3.4 Training - Deterministic Environment

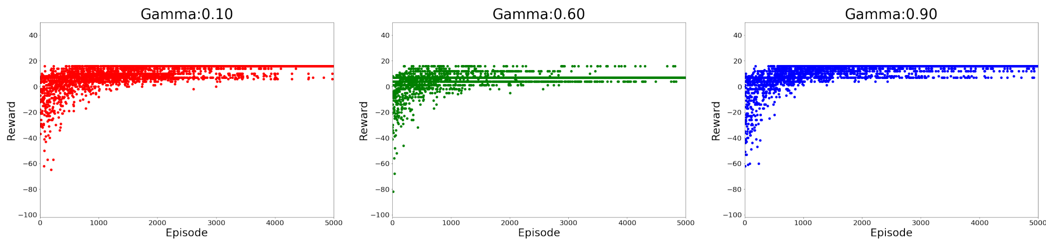
The agent for our environment is trained using the algorithm Q-Learning in the previous section written in Python programming language.

3.4.1 Hyperparameter Tuning

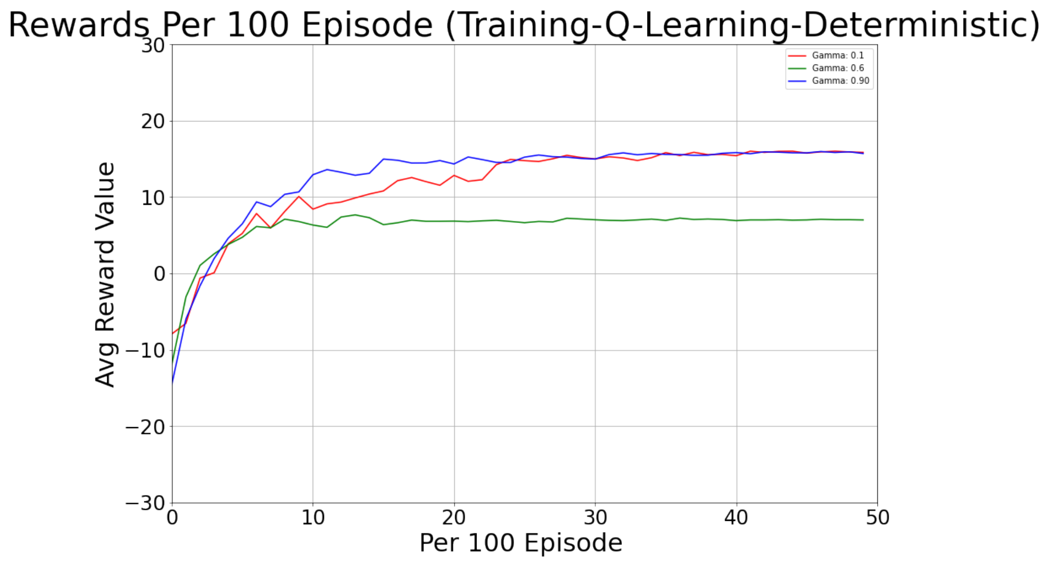
For better training result we fine-tuned the parameters discount factor γ and step size per episode $steps$ to get the optimal values for our training.

2.3.1.1. Discount Factor - γ

For fine-tuning γ we try with three different values bf 0.10, 0.60, 0.90. Following are the results that we observed. In the visualization in the next page we can see the rewards per episode for different values of gamma. We can observe that the convergence is slowest for $\gamma = 0.10$ followed



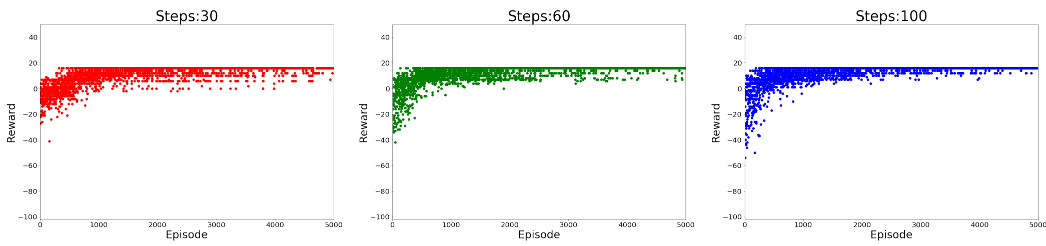
by $\gamma = 0.6$ and then $\gamma = 0.90$. Let us now also see the average rewards for every 100 episode to understand this better. From the average rewards per 100 episode graph we can clearly see that for



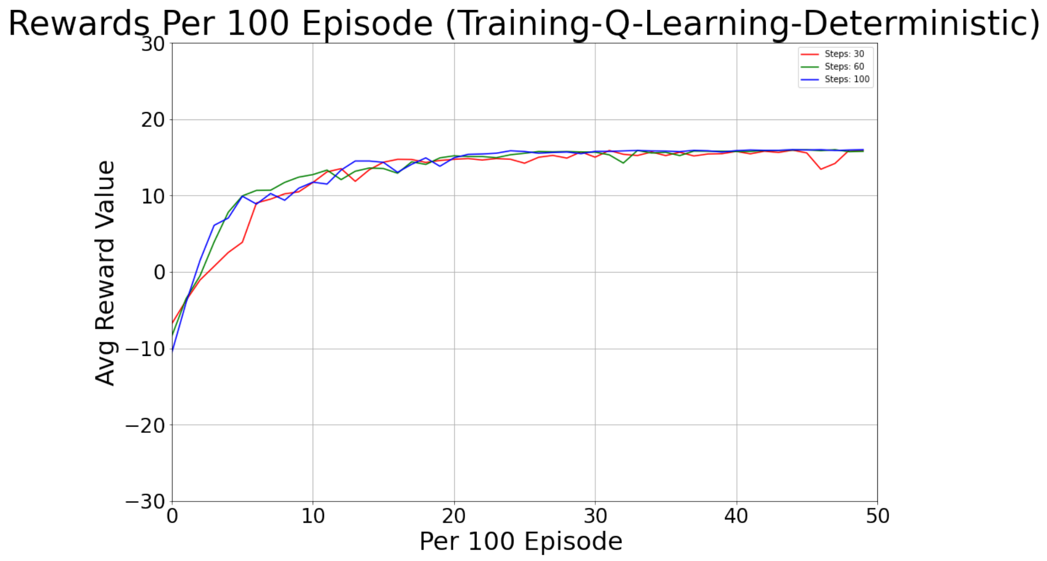
$\gamma = 0.90$ the convergence is the best as we progress with episodes. We can conclude that gamma values closer to 1 motivates the agent to look further and improve rewards in the long-term. For our problem we go with the gamma value 0.90 for training.

2.3.1.2. Step Size - $steps$

For fine-tuning $steps$ i.e the maximum step size per episode, we try with three different values **30, 60, 100**. Following are the results that we observed.



In the above visualization can see the rewards per episode for different values of steps. We can observe that the convergence is very similar with all the three step size. With $steps = 100$ the convergence is best and reward values are good towards the latter episodes. Let us now also see the average rewards for every 100 episode to understand this better. From the average rewards per 100



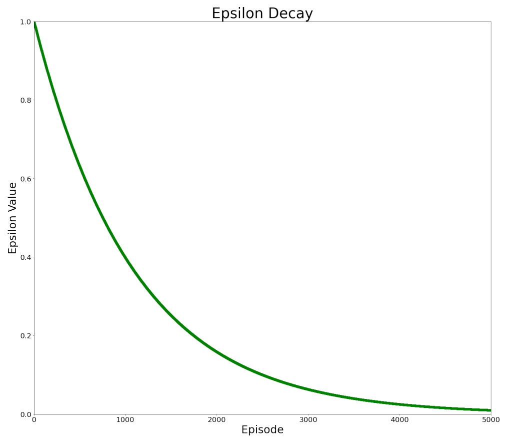
episode graph we can clearly see that the average rewards for $steps = 100$ consistent and the best with slight difference with respect to other step size. For our problem we select the steps size 100 for training.

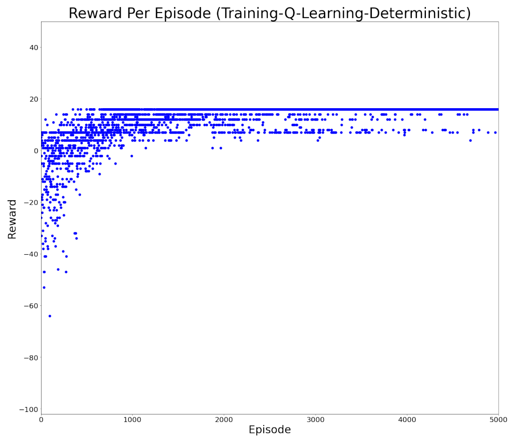
3.4.2 Training

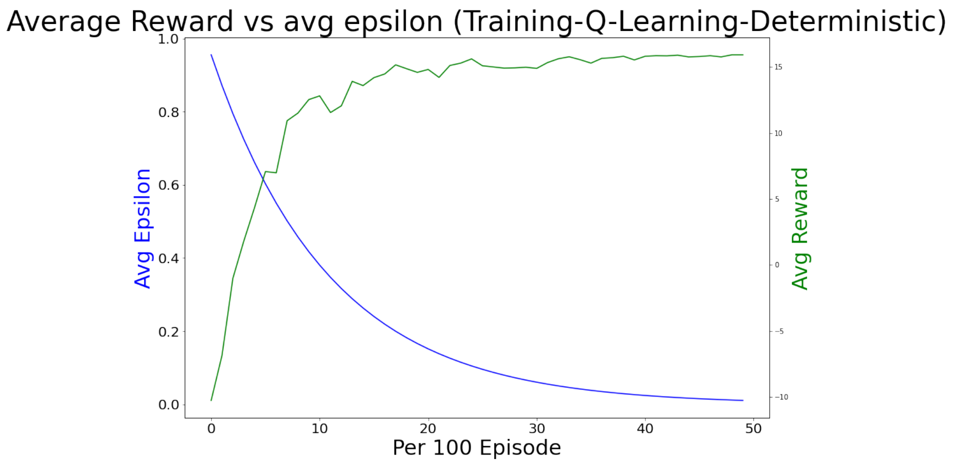
After we tune the parameters as seen in the previous section we move to training the algorithm with the other set of hyperparameters. The final hyperparameters values we use for the training are as follows:

1. $\alpha \leftarrow 0.10$
2. $\gamma \leftarrow 0.90$
3. $\lambda \leftarrow 0.999$
4. $\epsilon \leftarrow 1.0$
5. $episodes \leftarrow 5000$
6. $steps \leftarrow 100$

Following are the results after training Q-Learning for Deterministic Environment. The first graph is of the epsilon decay over time and the second graphs shows the reward received per episode. In the third graph, we can see the average rewards trend with respect to avg epsilon over every 100 episodes. We can clearly see that over time the agent learns the optimal policy.

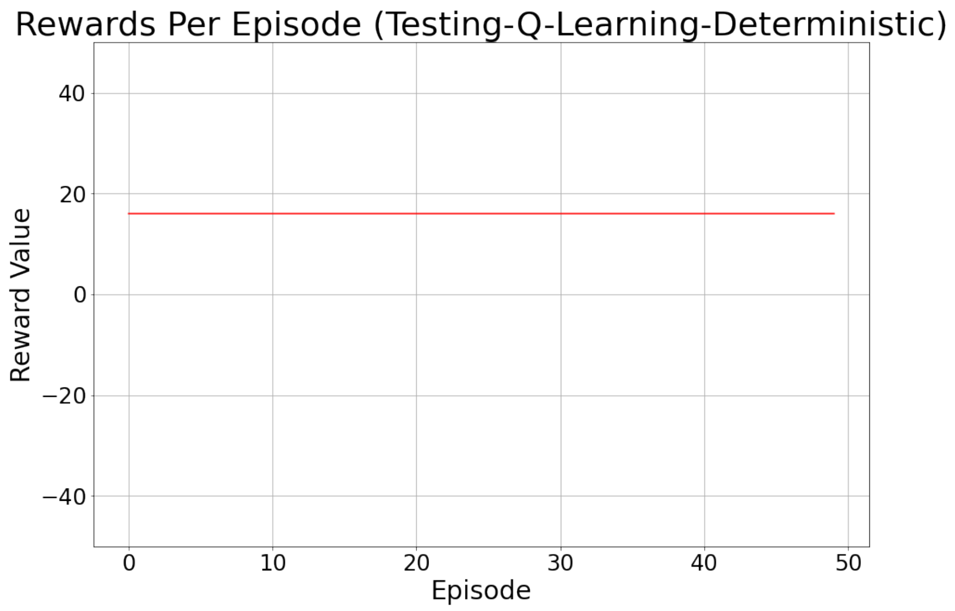






3.4.3 Evaluation

Let us evaluate the policy learned by agent. For evaluation the agent select the greedy action from the learnt policy. Following is the result of running the agent for 25 episodes.



This clearly show our agent has learn the optimal policy.

3.5 Training - Stochastic Environment

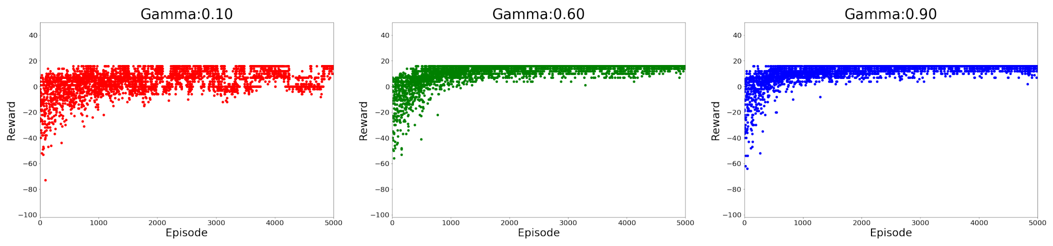
The agent for our environment is trained using the algorithm Q-Learning in the previous section written in Python programming language.

3.5.1 Hyperparameter Tuning

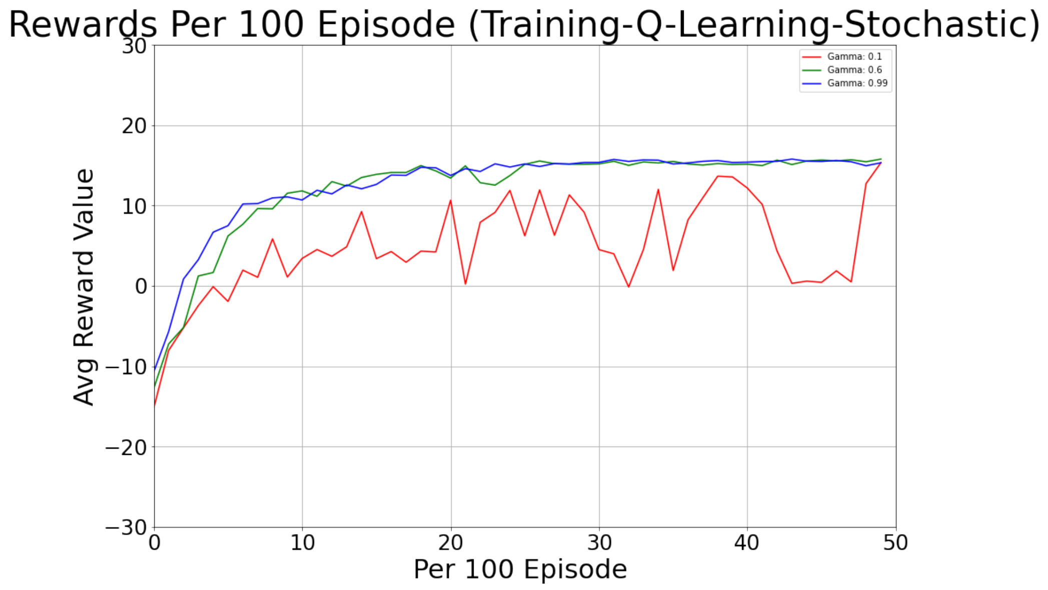
For better training result we do parameter tuning for discount factor and max step size.

2.3.1.1. Discount Factor - γ

For fine-tuning γ we try with three different values bf 0.10, 0.60, 0.90. Following are the results that we observed. In the visualization in the next page we can see the rewards per episode for different values of gamma. We can observe that $\gamma = 0.10$ performs the worst. While $\gamma = 0.6$



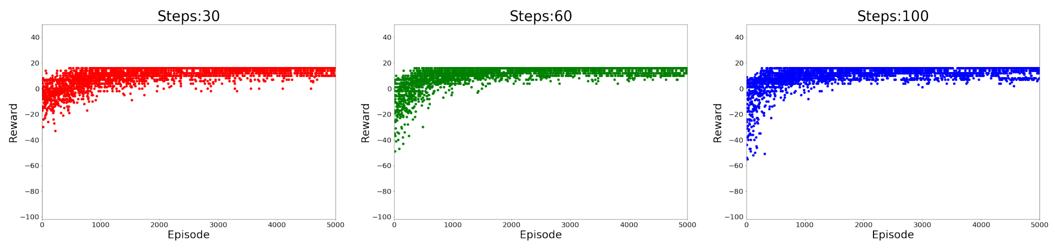
and $\gamma = 0.90$ had very similar performance. Let us now also see the average rewards for every 100 episode to understand this better. From the average rewards per 100 episode graph we can



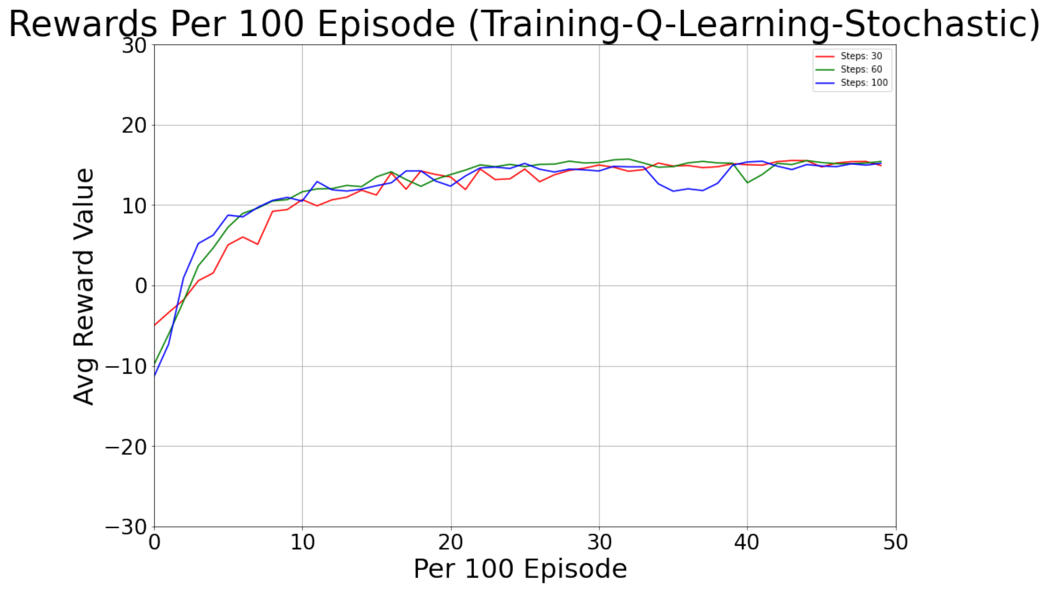
clearly see that for $\gamma = 0.90$ the convergence is the best as we progress with episodes. We can conclude that gamma values closer to 1 motivates the agent to look further and improve rewards in the long-term. $\gamma = 0.10$ performs worst since this made agent to care more for immediate rewards. For our problem we go with the gamma value 0.90 for training.

2.3.1.2. Step Size - $steps$

For fine-tuning $steps$ i.e the maximum step size per episode, we try with three different values **30, 60, 100**. Following are the results that we observed.



In the above visualization can see the rewards per episode for different values of steps. We can observe that the convergence is very similar with all the three step size. Let us now also see the average rewards for every 100 episode to understand this better. From the average rewards per 100



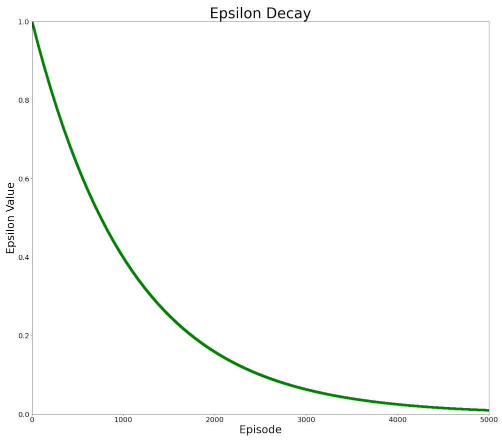
episode graph it is clear that different step size perform fairly same. $steps = 100$ was slightly better since he start, since the agent got more time to explore in every episode. For our problem we select the steps size 100 for training.

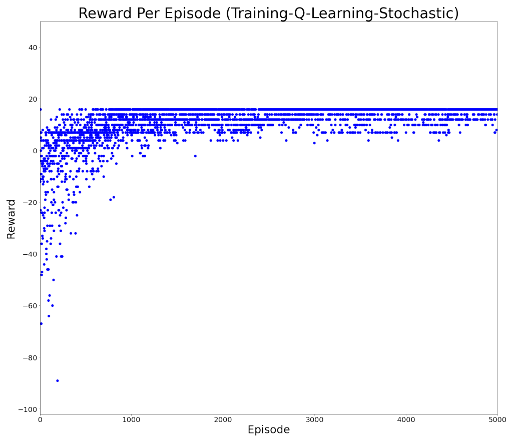
3.5.2 Training

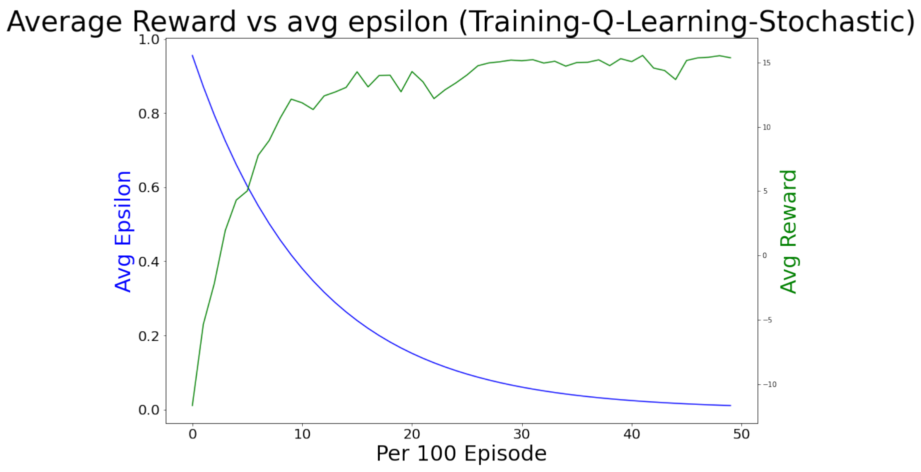
After we tune the parameters as seen in the previous section we move to training the algorithm with the other set of hyperparameters. The final hyperparameters values we use for the training are as follows:

1. $\alpha \leftarrow 0.10$
2. $\gamma \leftarrow 0.90$
3. $\lambda \leftarrow 0.999$
4. $\epsilon \leftarrow 1.0$
5. $episodes \leftarrow 5000$
6. $steps \leftarrow 100$

Following are the results after training Q-Learning for Stochastic Environment. The first graph is of the epsilon decay over time and the second graphs shows the reward received per episode. In the third graph, we can see the average rewards trend with respect to avg epsilon over every 100 episodes. We can clearly see that over time the agent learns the optimal policy. Since the environment is stochastic we see the rewards are more sparse.

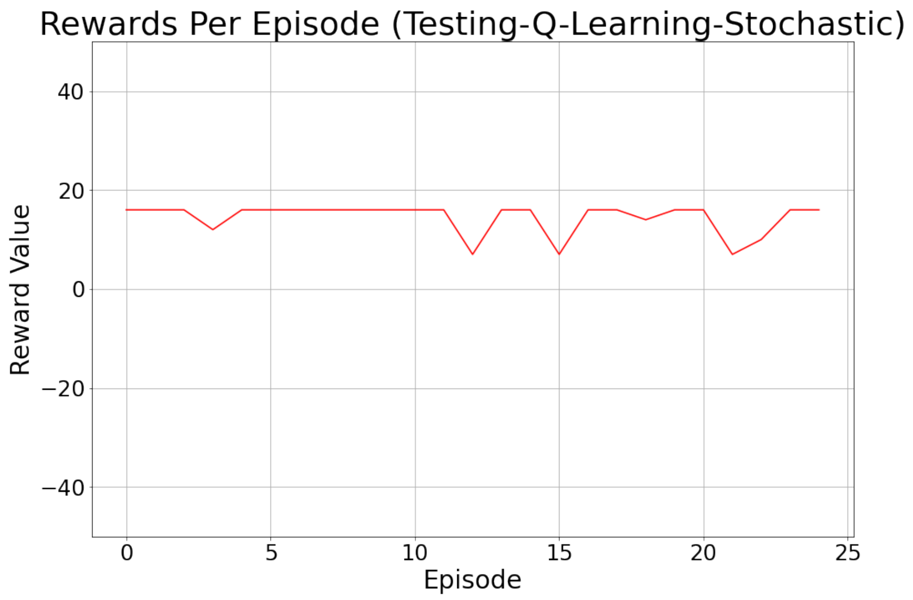






3.5.3 Evaluation

Let us evaluate the policy learned by agent. For evaluation the agent select the greedy action from the learnt policy. Following is the result of running the agent for 25 episodes. This show our agent



has learn the optimal policy, but since the environment is stochastic the reward graph is non linear.

4 Comparative Analysis

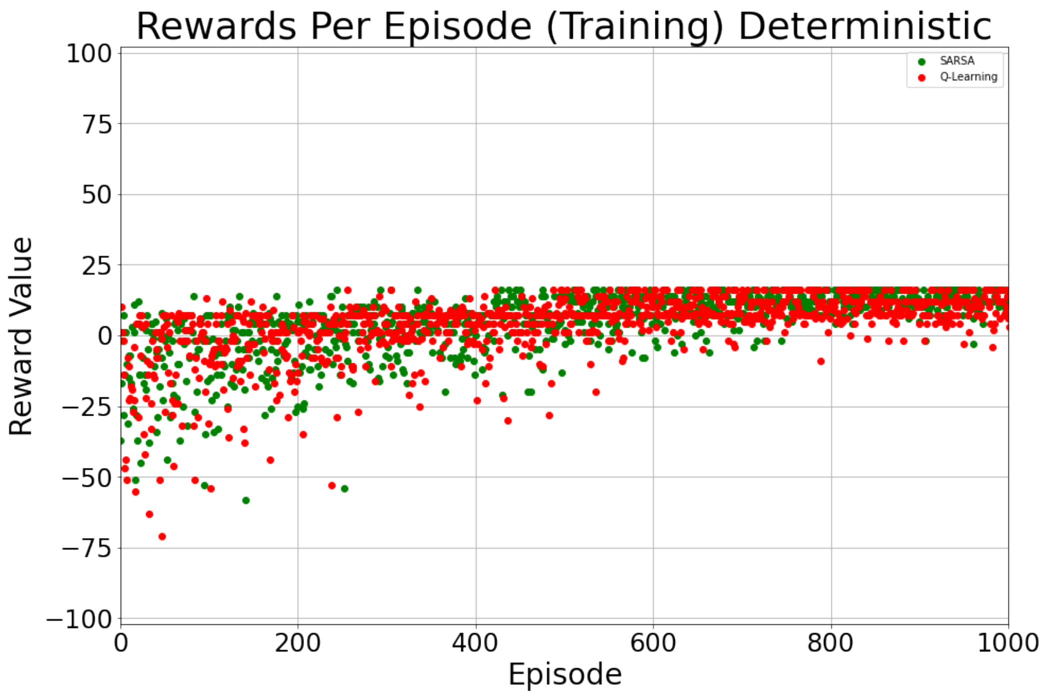
In this section we are going to compare the results from training SARSA and Q-Learning.

4.1 Deterministic Environment

Let us first compare the results for deterministic environment.

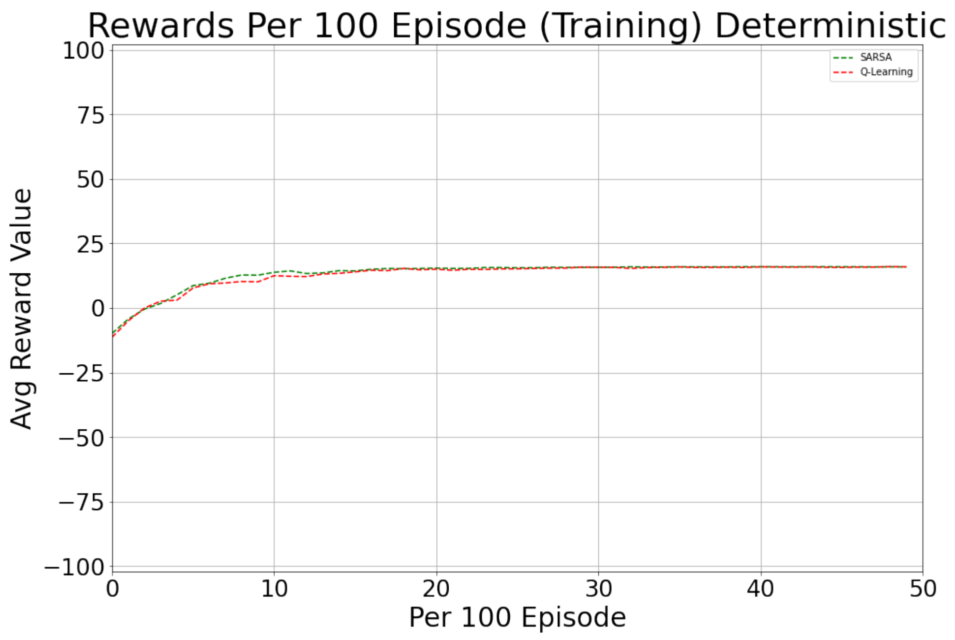
4.1.1 Training Comparison

The first graph we compare is the reward per episode. In the following graph we see that the reward values are very similar for both SARSA and Q-Learning over the Episodes and their convergence is overlapping.



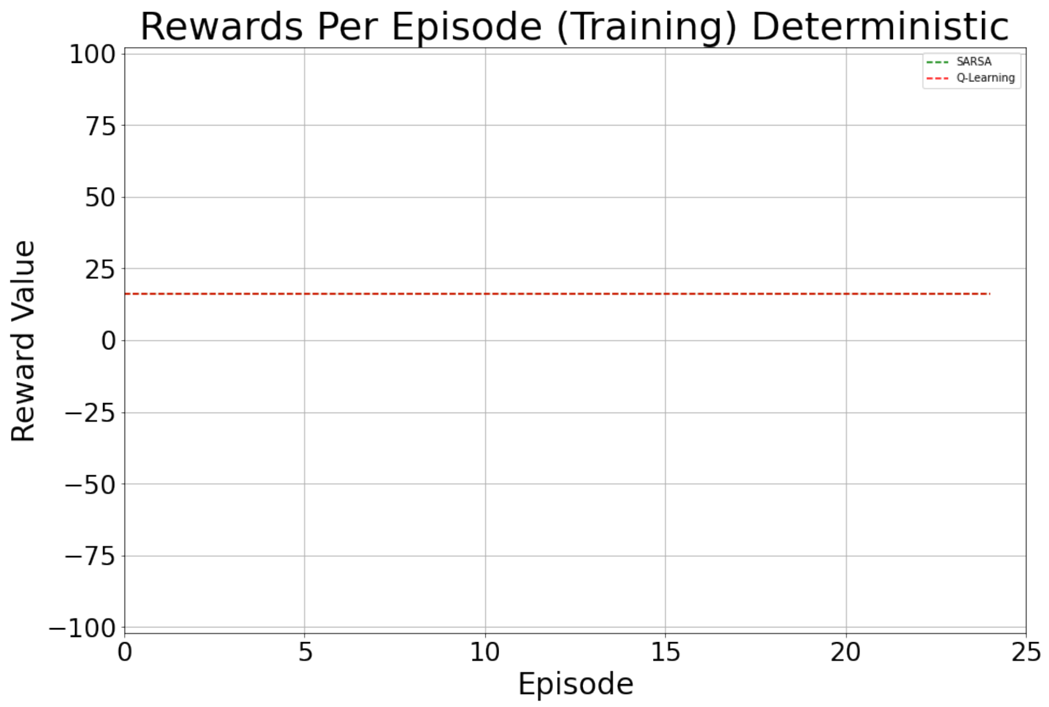
Let us now see the average reward values over per 100 episodes. From this we an conclude that SARSA performs lightly better than the Q-Learning algorithm in terms of convergence. SARSA converges faster. Both the algorithms eventually converge to optimal policy following epsilon-greedy policy. SARSA performing slightly better while training could be because it follows on-policy approach and derives its behavior from the current policy itself.

SARSA uses the Q value from the next state to update the Q value of the current state. Hence the next action selection is taken into account for the current state-action Q-value updation. While Q-Learning being off-policy takes into consideration the Q-value of the next state's best action to update current state-action Q-value.



4.1.2 Evaluation Comparison

During evaluation we do the greedy selection of the action from the learned policy. Below is the graph comparing both the algorithms.



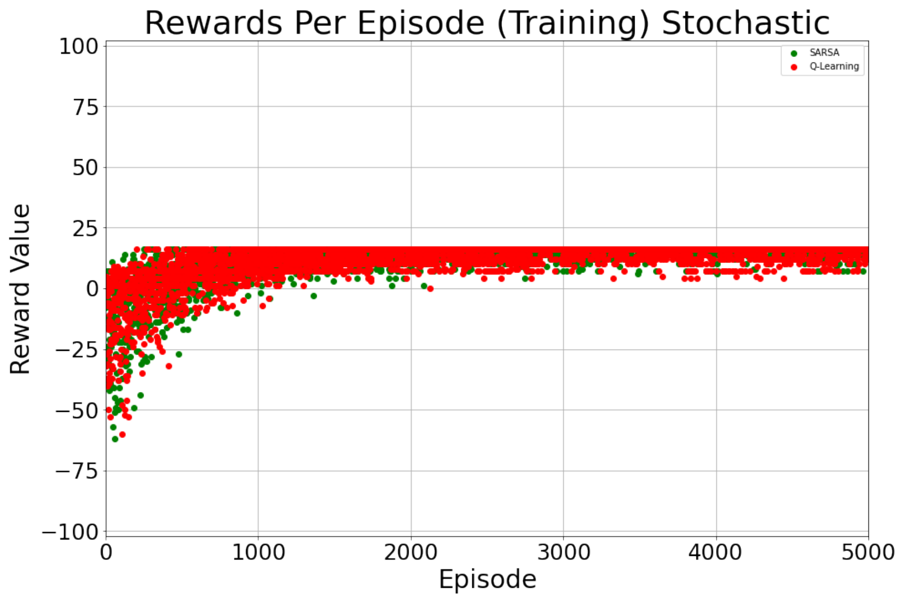
In the above comparison we can see that both the SARSA and Q-Learning Rewards Values overlap each other with the maximum reward. This concludes that both the algorithms have learned the optimal policy successfully despite one being slower than the other during training. The linear line is due to the fact the environment is deterministic.

4.2 Stochastic Environment

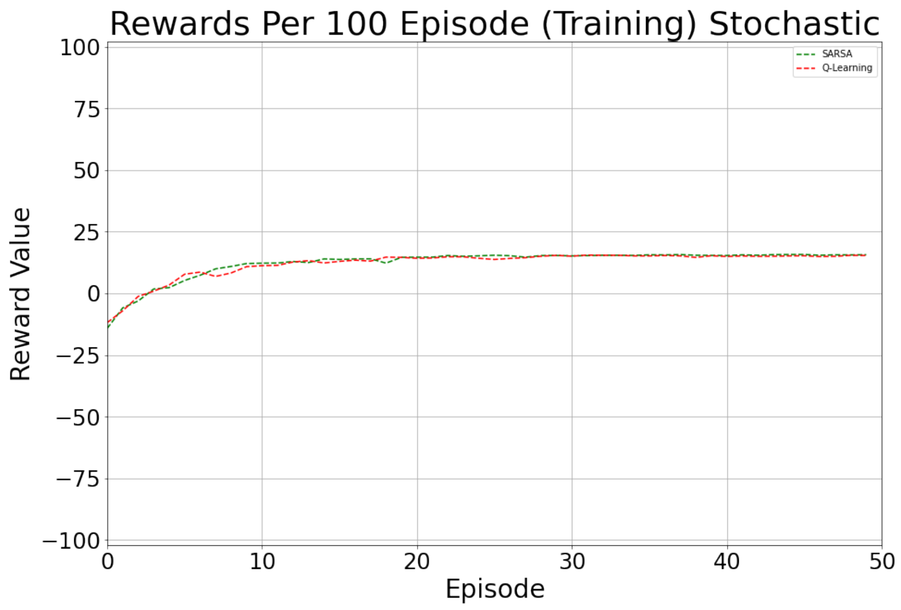
Let us first compare the results for stochastic environment.

4.2.1 Training Comparison

The first graph we compare is the reward per episode. In the following graph we see that the reward values are very similar for both SARSA and Q-Learning over the Episodes and their convergence is overlapping.

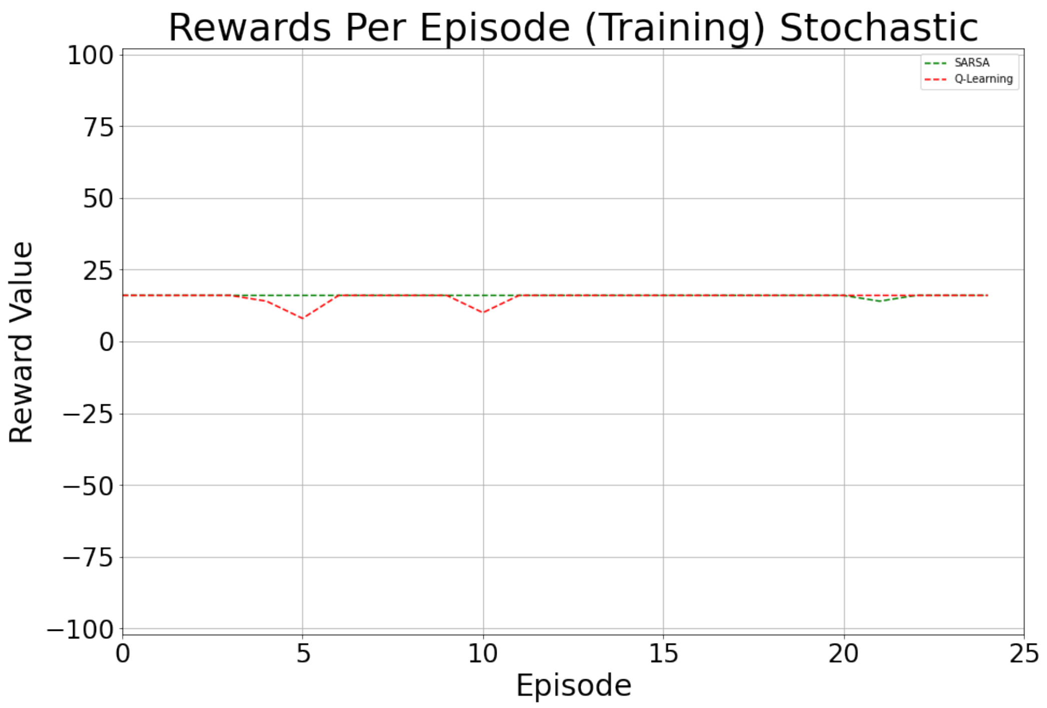


Let us also see the average reward values over per 100 episodes. In this case we confirm both the learning algorithms have converged similarly under stochastic condition.



4.2.2 Evaluation Comparison

Let us now see the evaluation reward values for both the algorithms.



We observe that though for most of the episodes the agent attains the maximum reward of 16 units. But there are cases when it is not. Hence the graph is not linear. This is due to the fact that the environment is stochastic. Despite that it works fairly well in most episodes.

5 Appendix

5.1 Updates to Environment Code

The Environment code was refactored to better suit for training and efficiency. Major change was previously the transition probability to simulate stochasticity was calculated on the fly. Now it is pre calculated during environment object creation and stored for quick reference during training with large episodes and steps. Some more minor changed was with respect to reward map and rendering logic. The states and action space remain same.

6 Reference

[1] Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction.

[2] Lecture Slides

[3] Icons for Rendering credit - Sandro Pereira/ph03nyX, ph03nyx.deviantart.com

[4] gym.openai.com