Assignment 1 - Part 1 Checkpoint submission

CSE546: Reinforcement Learning

Solomon Raj Panduga
spanduga@buffalo.edu

1. Describe the deterministic and stochastic environments, which were defined (set of actions/states/rewards, main objective, etc).

The environment defined is a 4x4 grid where the agent has to reach the goal state from the initial state, the environment defined is identical for both deterministic and stochastic environments.

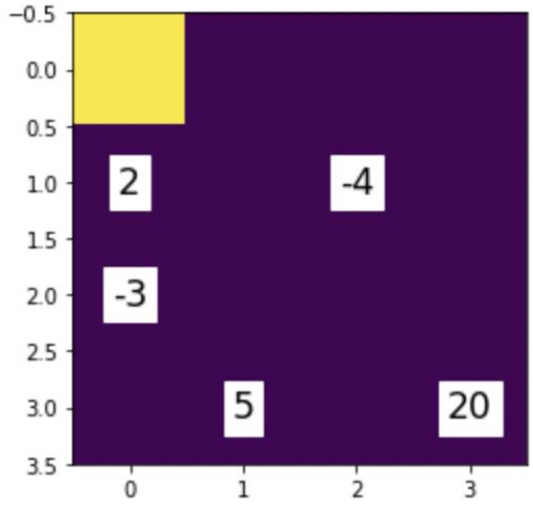
Actions	The agent has 4 possible actions at any given state. up = 0 down = 1 right = 2 left = 3
State	In this 4x4 grid environment there are 16 states
Rewards	4 rewards are defined in this environment, the position and value of the rewards: (2,0): -3, (1,2): -4, (1,0): 2, (3,1): 5, (3,3): 20 The goal position (3,3) has the highest reward value (20)
Objective	To reach the goal state

2. Provide visualizations of your environments.

Deterministic environment:

The render method provides the visualization of the environment with details such as action, reward and current state

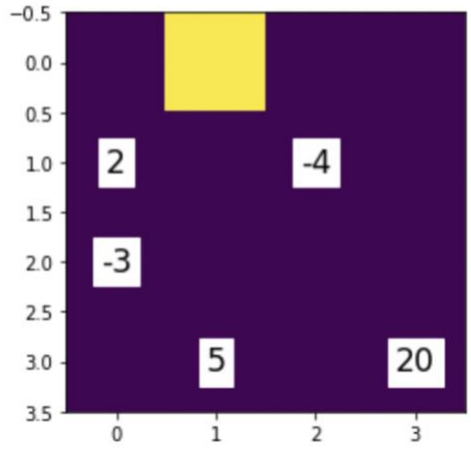
```
observation: [0 0]
reward: 0
action: left
action 3
```



Stochastic environment:

In the stochastic environment, the value random_action indicates if the action is a stochastic action or not

```
observation: [0 1]
reward: 0
action: right
action 2
random_action False
```



3. How did you define the stochastic environment?

Here stochasticity is defined in the step function by choosing between executing the given action or choosing a random action.
The probability of performing the given action is 0.6 and performing a random action is 0.4.

4. What is the difference between the deterministic and stochastic environments?

In a deterministic environment, the agent can determine the next state given the current state and action there is no randomness or probability involved in the environment.
Ex: A game of chess

In a stochastic environment, the agent cannot fully determine the next state given the current state and action.
Ex: Football game

5. Safety in AI: Write a brief review (~ 5 sentences) explaining how you ensure the safety of your environments. E.g. how do you ensure that agent choose only actions that are allowed, that agent is navigating within defined state-space, etc.

Ensuring safety:

1. Defining the allowed actions and performing a check in the step function before executing each action.
2. Using the numpy.clip function to ensure the agent doesn't go out of the defined boundaries.
3. The max_timestamps value ensures that the agent doesn't run forever and if the max value is reached the execution is terminated.

Assignment 1 - Part 2

CSE546: Reinforcement Learning

Solomon Raj Panduga
spanduga@buffalo.edu

In this Assignment we define a deterministic and stochastic environment and implement Q learning and another tabular algorithm of choice to solve the environments

Algorithms implemented:

- Q learning
- SARSA

Changes since the checkpoint submission:

- The reward for goal state is updated to a higher value
- Visualization of the grid world is updated with images for agent and different scenarios

Github link for the jupyter notebook:

- The jupyter notebook is uploaded to GitHub at - https://github.com/solo11/ub-rl
- ub-rl is added as a collaborator
- Notebook along with h5 files and images are uploaded to Github
- **Important** Before running the notebook please upload these file to local path of the notebook :
    - images folder - contains the images used for visualization
    - data_q_table_values.h5 - contains the trained Q-table values
    - data_total_rewards_epsilon.h5 - contains the total rewards and epsilon decay values of training

1 Applying Q-learning to solve the deterministic environment.

For the defined deterministic environment, the Q-learning algorithm is used to solve the environment, the deterministic environment consists of a 4x4 grid world where the agent starts at an initial position (0,0) and needs to reach the goal state (3,3) and should collect rewards on the way, there are negative and positive rewards defined in the grid, the agent should not collect the negative rewards and stay away from them and collect the positive rewards and reach the goal state.

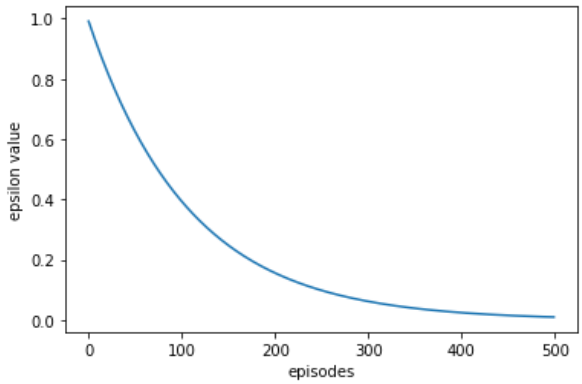
1.1 Epsilon decay

Epsilon value is used to determine whether the agent would explore or exploit. The initial epsilon value is 1 and gradually this value is each episode and by the end of the learning, it's 0 or nearly zero.

Initial epsilon value = 1

The epsilon decay over time in the course of learning is given in Graph 1.1 below, the epsilon value is multiplied by the decay factor for each episode. A minimum epsilon value of 0.01 is used as a limit to ensure the agent always does exploration.

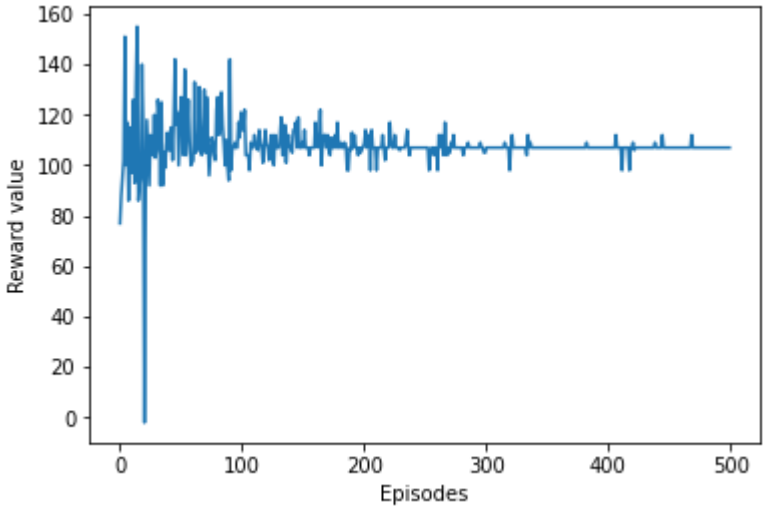
The value is 1 initially to encourage the agent to explore as the values in the q table are unknown but as the agent learns the epsilon value is reduced as now the agent would be able to determine the action that would give the maximum reward.



Graph 1.1 Epsilon decay

1.2 Total rewards per episode

Total rewards per episode is the sum of all the rewards collected by the agent in an episode. An episode is a series of steps taken by the agent to reach the goal state. Here we sample 500 episodes for the agent to learn.



Graph 1.2 Total reward value

1.3 Analysis

- The agent starts with the initial epsilon value of 1, and the epsilon value decreases exponentially as the episodes increase as we see in Graph 1.1
- In Graph 1.2 the reward total is less at the beginning of the learning as the agent is still exploring and the total rewards obtained value is unstable with high fluctuations this is due to the agent

exploring more of the environment due to epsilon value being closer to 1

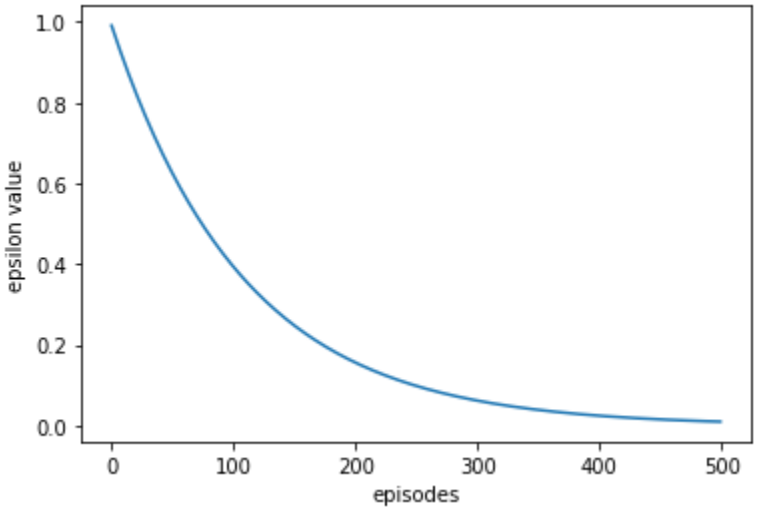
- Till the episodes 300 the total reward value is fluctuating and is stabilized after 300 episodes as the agent populates the q values for the state action pairs and can choose the actions which give the highest reward as epsilon value decreases the agent will start choosing greedy actions
- At the end of the learning (400 - 500 episodes), there is an almost flat line meaning the agent is now able to successfully navigate the grid world to collect the rewards and reach the goal state consistently every time
- There are still small highs and lows in the flat line this is due to the minimum epsilon value 0.01, which makes the agent explore and not choose greedy actions all the time
- Now the agent can navigate the grid world successfully collecting all the positive rewards, staying away from the negative rewards and reaching the goal state

2 Applying Q-learning to solve the stochastic environment.

For the defined deterministic environment, the Q-learning algorithm is used to solve the environment, the deterministic environment consists of a 4x4 grid world where the agent starts at an initial position (0,0) and needs to reach the goal state (3,3) and should collect rewards on the way, there are negative and positive rewards defined in the grid, the agent should not collect the negative rewards and stay away from them and collect the positive rewards and reach the goal state.

2.1 Epsilon decay

For solving the stochastic environment, the epsilon value is used to choose between a random action and greedy action the initial epsilon value is 1 and the value is reduced each episode.

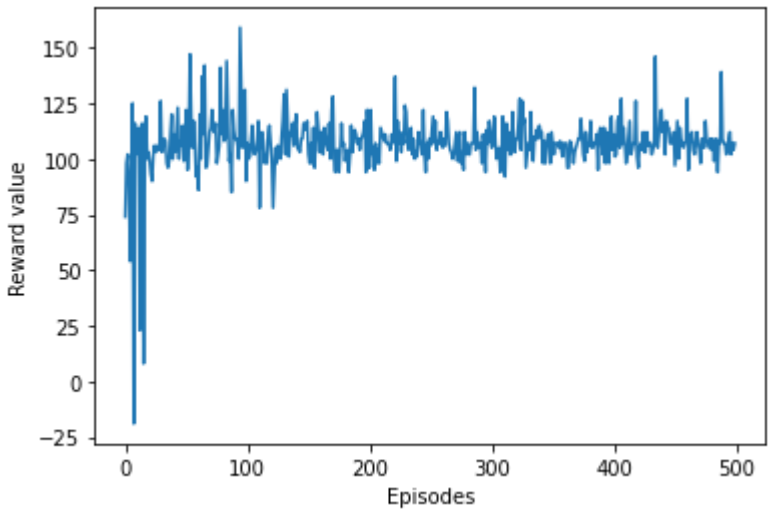


Graph 2.1 Epsilon decay

2.2 Total rewards per episode stochastic environment

Total rewards per episode is the sum of all the rewards collected by the agent in an episode. An episode is a series of steps taken by the agent to reach the goal state. Here we sample 500 episodes for the agent to learn.

The graph helps us to understand how the agent's learning progress has been if the agent can successfully find the optimal path and get consistent rewards or not

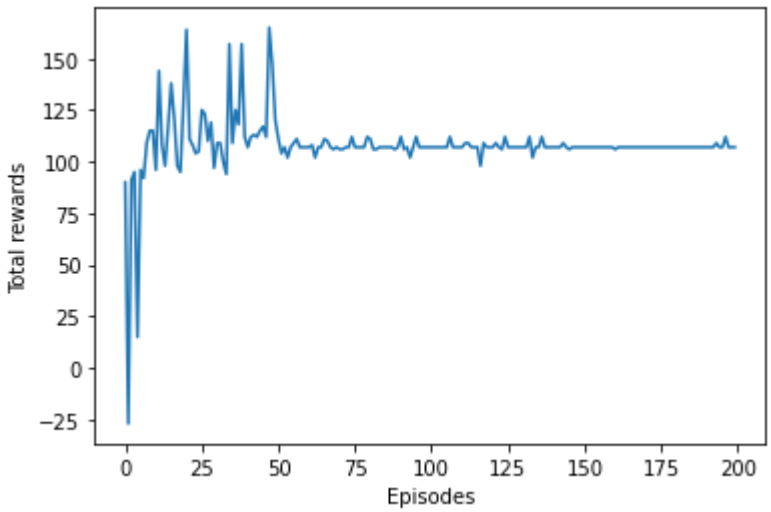


Graph 2.2 Total rewards per episode

2.3 Analysis

- From Graph 2.1 the epsilon value is 1 initially to encourage the agent to explore as the values in the q table is unknown but as the learning progresses the epsilon value is reduced exponentially so the agent can choose greedy actions as the Q table will be populated and the agent will be able to determine the action that would give the maximum reward confidently.
- In the Graph 2.2 the total reward value initially has high fluctuations due to the epsilon value being close to 1 which makes the agent explore and find new paths to reach the goal state and the reward is calculated.
- The graph after 100 episodes seems to have an average total reward of above 100
- But there are still fluctuations in the value which is due to the environment being stochastic, meaning the given action is performed with a probability of 0.6 and the remaining time a random action is chosen by the environment.
- Still the average total reward being the same after 200 episodes mean the agent is able to navigate through the stochastic environment optimally.

3 Applying SARSA algorithm to solve the deterministic environment



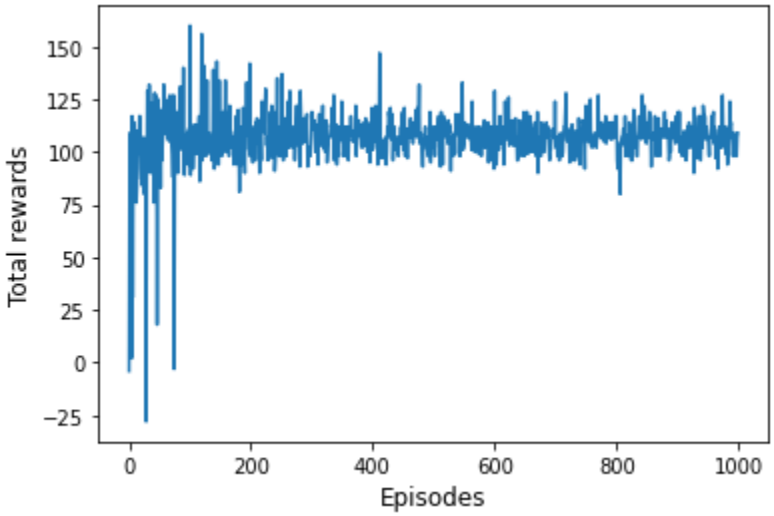
Graph 3.1 Total rewards

3.1 Analysis

- The initial epsilon value is 1 and the value is reduced exponentially as the learning progresses, we train the agent with these parameters:
    - learning rate: 0.1
    - epsilon: 1
    - minimum epsilon value: 0.01
    - no. of episodes: 200
    - gamma: 0.95
    - maximum iterations per episode: 100

- The agent starts off learning with low total reward value till 20 episodes as the epsilon value is closer to 1 the agent is exploring the environment by choosing random actions.
- After 20 episodes the agent is choosing more high reward actions and the total reward value is more this is not necessarily optimal as the agent is collecting the immediate rewards again and again and not navigating to the goal state.
- But as the timesteps progress the agent overcomes this problem due to the discount factor which reduces the immediate reward value and makes the agent focus on navigating to the goal state in less no. of steps because as the time step increases the rewards obtained are also discounted
- After 50 episodes the total rewards obtained is now almost a flat line as the Q table is now populated and epsilon value decreases the agent exploits the environment and chooses greedy actions.
- Due to the minimum epsilon value the agent doesn't always exploit the environment and the little highs and lows in Graph 3.1 towards the end of learning indicate that the agent is choosing greedy actions but also exploring with random action.

4 Applying SARSA algorithm to solve the stochastic environment



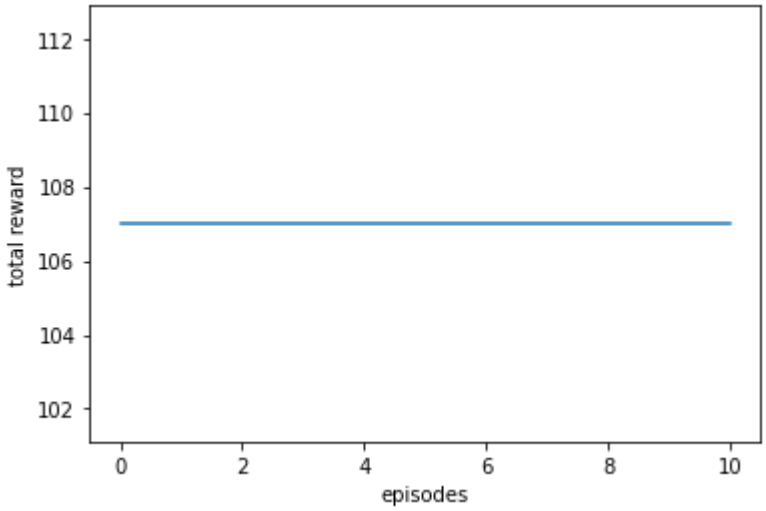
Graph 4.1 Total reward

4.1 Analysis

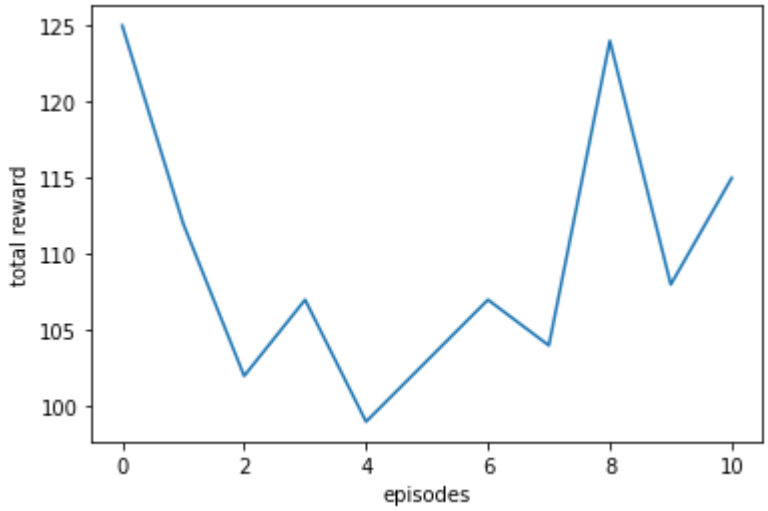
- we use the sarsa algorithm to train the agent to navigate the stochastic environment, the stochasticity of the environment is the given action is executed with a probability the actions performed in the environment have are stochastic meaning 60 percent of the time the given action will be performed and 40% of the time a random action is selected
- The parameter values:
    - number of episodes: 1000
    - discount factor gamma: 0.95
    - learning rate alpha: 0.1
    - initial epsilon value: 1
    - minimum epsilon value: 0.01
    - maximum steps per episode: 100

- We start with the initial value of epsilon 1 and decrease it exponentially
- From the Graph 4.1 the agent initially starts exploring by choosing random actions as epsilon value is closer to 1
- We see high fluctuations at the beginning of the learning up until 200 episodes this is because of the agent choosing to explore.
- As we move further the agent will have the Q table populated with the state values and will begin exploiting more.
- After episode 400 we have an average total reward of above 100 meaning the agent is reaching the goal state and collecting the rewards
- There are still sharp fluctuations in the graph this is due to the environment being stochastic meaning a random action is performed with a probability of 0.4
- The agent has learned to find an optimal path to the goal state from the initial state.

5.1 Evaluating Q-Learning algorithm for 10 episodes



Graph 5.1 Q-Learning deterministic environment

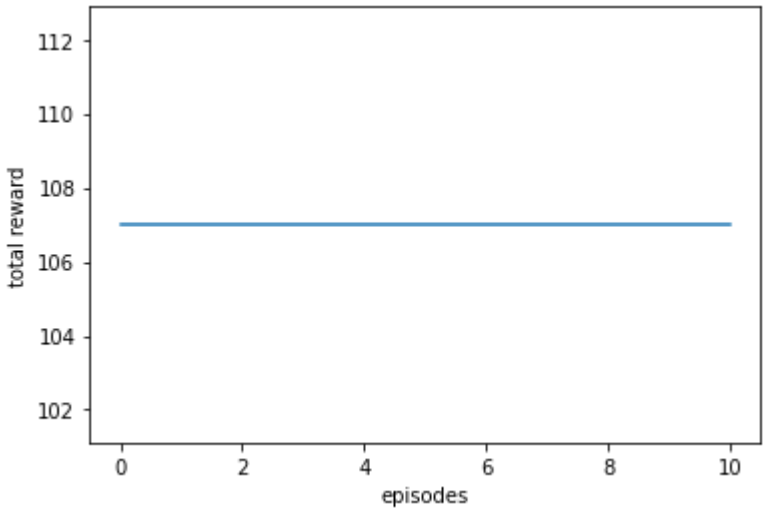


Graph 5.2 Q-Learning stochastic environment

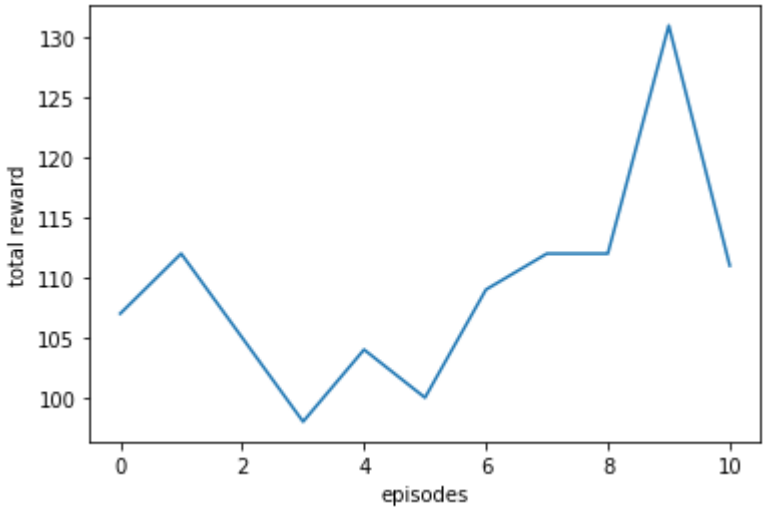
5.1.1 Analysis

- The agent is evaluated on Q-Learning for 10 episodes
- In Graph 5.1 the reward obtained for the deterministic environment is linear and the total reward obtained is maximum for all the episodes
- In Graph 5.2 for the stochastic environment the total reward obtained fluctuates because the environment is stochastic, but the reward obtained is above 100 meaning the agent is successfully reaching the goal state every episode

5.2 Evaluating SARSA algorithm for 10 episodes



Graph 5.3 SARSA deterministic environment

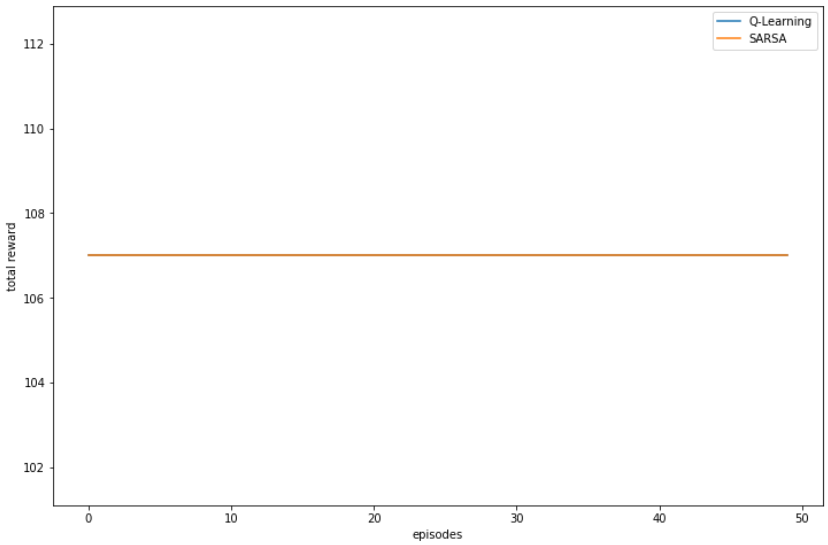


Graph 5.4 SARSA stochastic environment

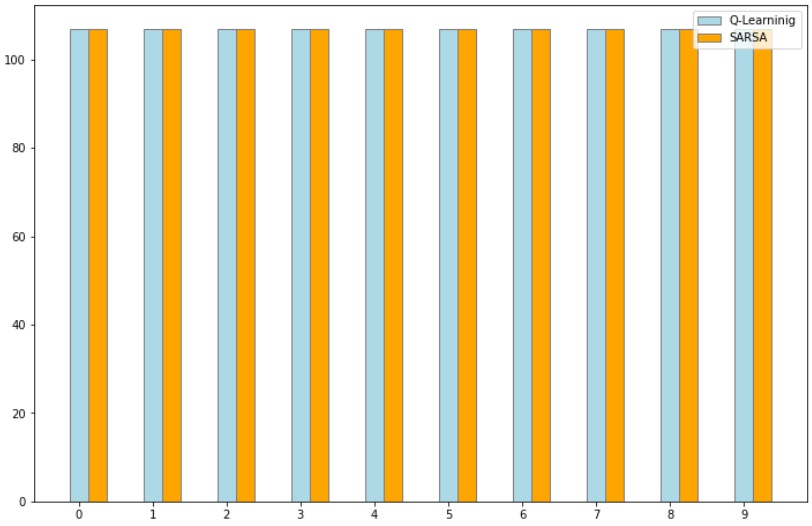
5.2.1 Analysis

- The agent is evaluated on SARSA for 10 episodes
- The results are very similar to Q-Learning
- In Graph 5.3 the reward obtained for the deterministic environment is linear and the total reward obtained is maximum for all the episodes
- In Graph 5.4 for stochastic environment the total reward obtained is fluctuating because the environment is stochastic, but the reward obtained is above 100 meaning the agent is successfully reaching the goal state every episode

6.1 Comparing the algorithms in deterministic environment:



Graph 6.1 Total reward value



Graph 6.2 Total reward value

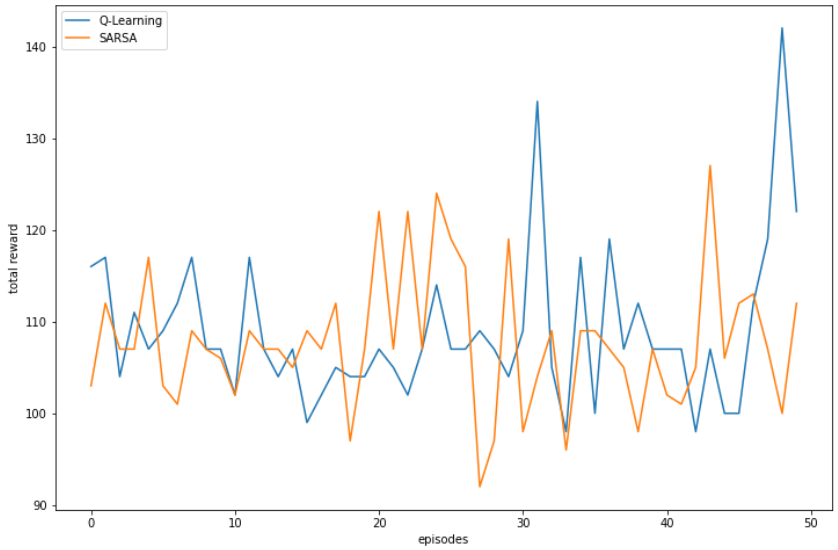
	Q-Learning	**SARSA**
No. of episodes	500	200
Learning rate	0.2	0.1
Discount factor (gamma)	0.97	0.99

Table 6.1

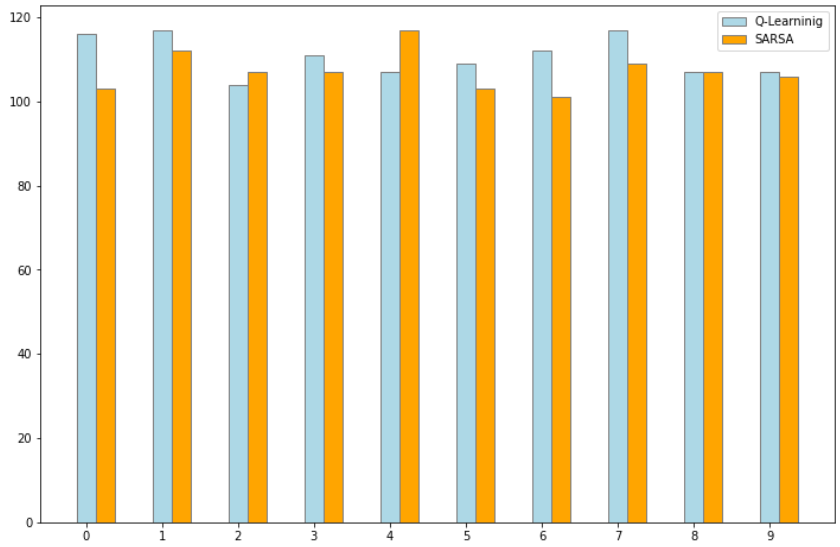
6.1.1 Analysis

- Both the algorithms are able to navigate the deterministic environment successfully, obtaining the highest possible reward each episode
- The algorithms perform equally when evaluated for 50 episodes in Graph 6.1 and also obtain the highest reward
- From the Graph 6.2 we can have a better understanding of algorithms obtaining the same reward
- From the Table 6.1 we see that Q-Learning took more no. of episodes to train and to get a liner graph while training, but SARA was able to achieve it in 200 episodes

6.2 Comparing the algorithms in stochastic environment



Graph 6.3 Total reward value



Graph 6.4 Total reward value

	Q-Learning	**SARSA**
No. of episodes	500	1000
Learning rate	0.1	0.1
Discount factor (gamma)	0.8	0.95

Table 6.2

6.2.1 Analysis

- As the environment is stochastic the rewards obtained are fluctuating in each episode as we see in graphs 6.3 and graph 6.4
- But on an average the total rewards obtained are over 100 from the bar chart Graph 6.4 which means our agent is able to reach the goal state every episode
- From the bar chart Q-Learning is able to gather more rewards than SARSA and beats SARSA in average total no. of rewards
- In Table 6.2 the no. of episodes required to train Q-learning is 500 and SARSA is 1000 but still Q-Learning is able to perform better than SARSA in less no. of episodes
- From the analysis in stochastic environment Q-learning is performing better than SARSA

7.1 Q-Learning - A brief overview

- Q-Learning is a tabular method used to train using reinforcement learning
- It maintains a Q-table and updates it with the value of a state action pair
- The update function of Q-Learning is given below

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))$$

- Action a is choose following an epsilon greedy policy
- The target ***max Q(S_{t+1}, a`)*** gives the action with maximum value in the state ***S_{t+1}***
- Q-Learning is also known as off-policy algorithm as the update function in the target always chooses the action with maximum Q value and is not affected by the agent policy
- The update function implemented in the assignment:

```
q_table[current_state,action] = q_table[current_state,action] + lr * ((reward + gamma *
max(q_table[next_pos,:])) - q_table[current_state,action])
```

  - current_state: the current state where the agent is in the environment
  - action: the action to be performed from the current state, given by the epsilon greedy policy
  - lr: the learning rate
  - reward: the reward received after performing the action
  - gamma: the discount factor value
  - next_pos: the next state where the agent ended up after performing the action from the current state

7.2 SARSA - A brief overview

- SARSA is also one of the tabular methods which can be implemented to solve a reinforcement learning problem
- SARSA is similar to Q-Learning, and it also maintains a Q-table and updates the Q values
- SARSA derives its name from how the algorithm update function works which is state, action, reward, state, action
- The update function is given below

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

- SARSA is an on policy algorithm, meaning the update function chooses the next action based on the epsilon greedy policy defined
- Initially the algorithm chooses random actions and as it learns it chooses greedy actions
- The algorithm chooses the next action ***A_{t+1}*** from the policy
- The implementation of the algorithm in the assignment:

```
  q_table[current_state,action] = q_table[current_state,action] + lr * (reward + gamma *
q_table[next_pos,next_action] - q_table[current_state,action])
```

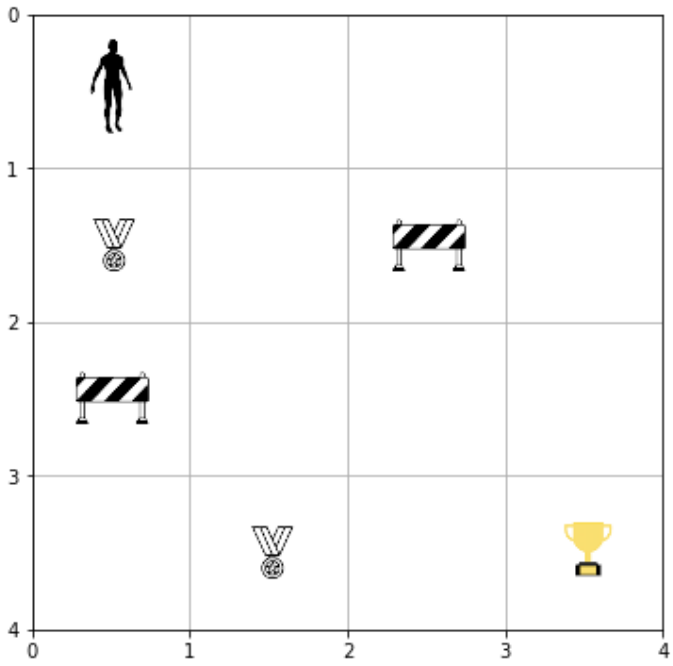
- current_state: the current state where the agent is in the environment
- action: the action to be performed from the current state, given by the epsilon greedy policy
- lr: learning rate
- reward: reward received after performing the action
- gamma: discount factor value
- next_pos: the next state where the agent ended up after performing the action from the current state
- next_action: the next action to be performed in the next state the next action is given by the epsilon greedy policy if the policy chooses to exploit the action maximum value for the next state in the Q-table is chosen

8 Bonus points

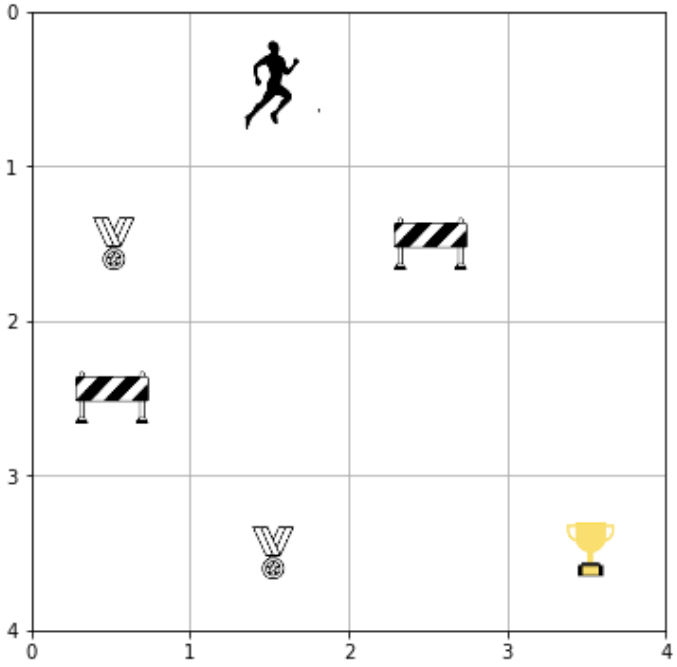
8.1 Visualizing the Grid world scenario:

In a 4x4 grid world initial position of the agent is (0,0) and the goal state is (3,3) the negative rewards are represented with a blocked sign and positive rewards with a medal and the goal state with a golden cup

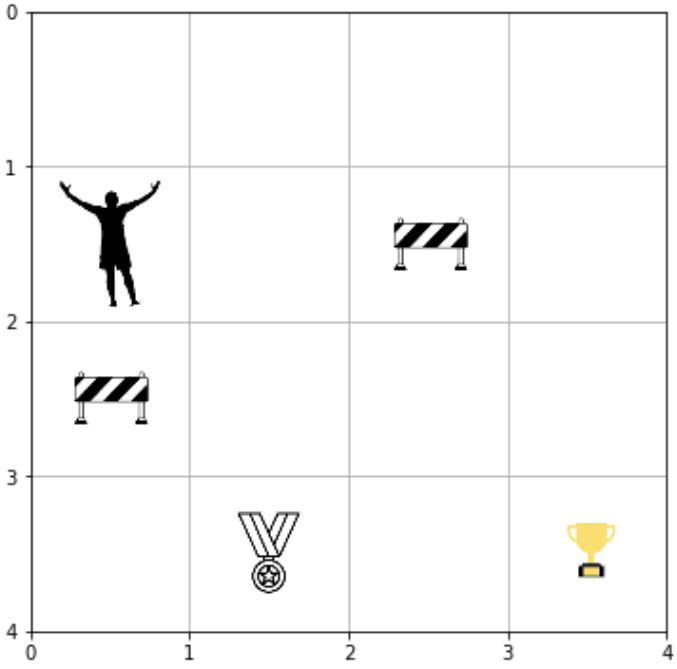
Initial grid world:



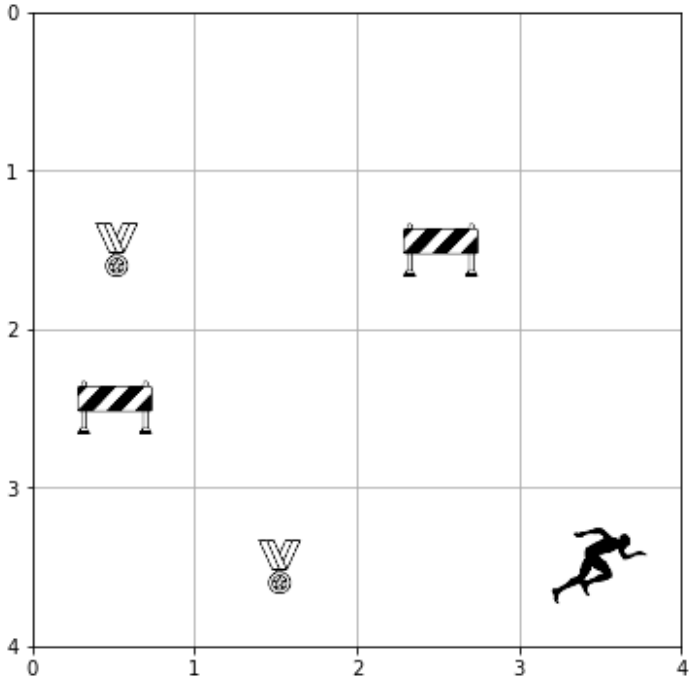
When the agent is navigating:



When agent collects a reward:



When the agent reaches goal state:



8.2 Hyperparameter tuning

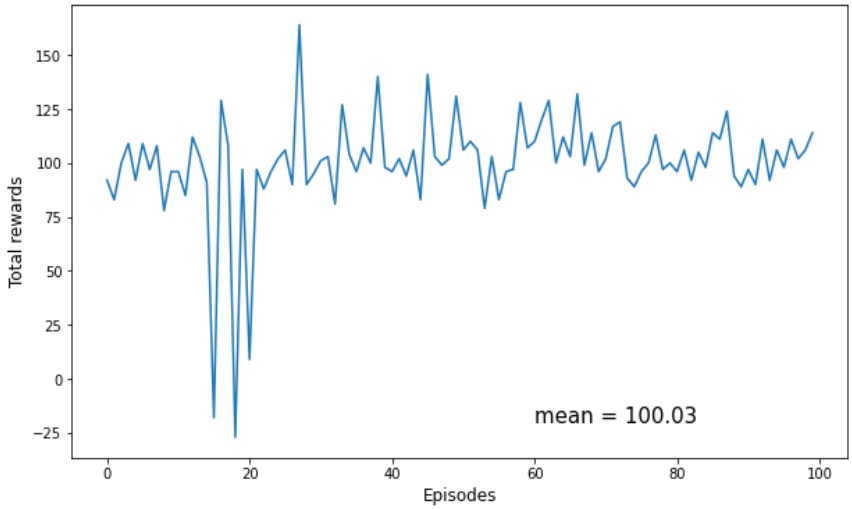
Tuning the hyperparameters:

- Number of episodes
- Maximum timesteps

8.2.1 Tuning the discount factor

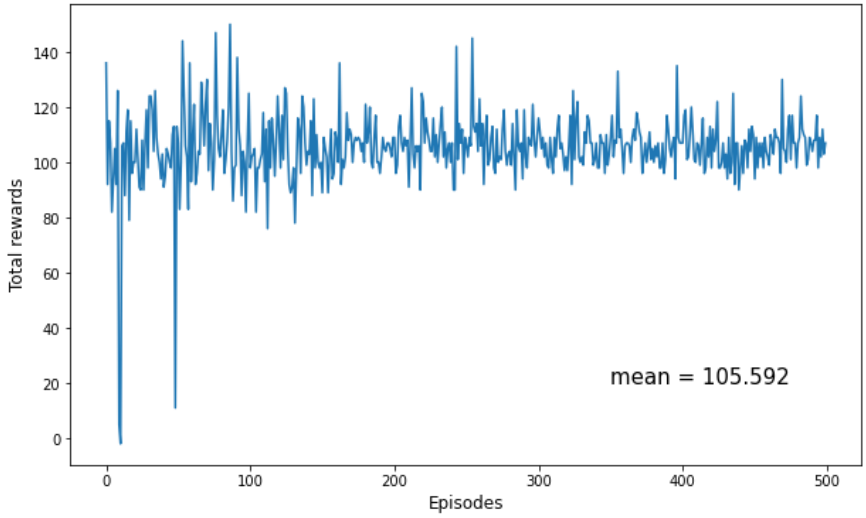
Tuning the discount factor while implementing SARSA for the stochastic environment:

With no. of episodes = 100



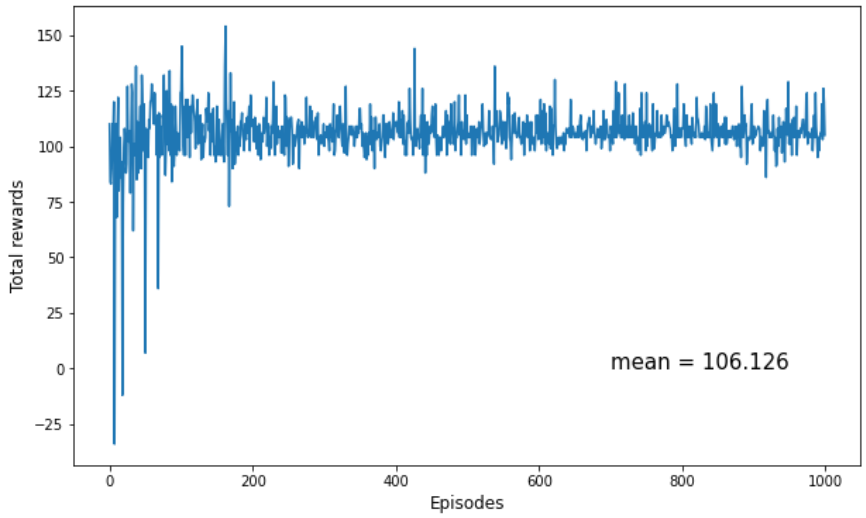
Graph 8.1

With no. of episodes = 500



Graph 8.2

With no of episodes = 1000



Graph 8.3

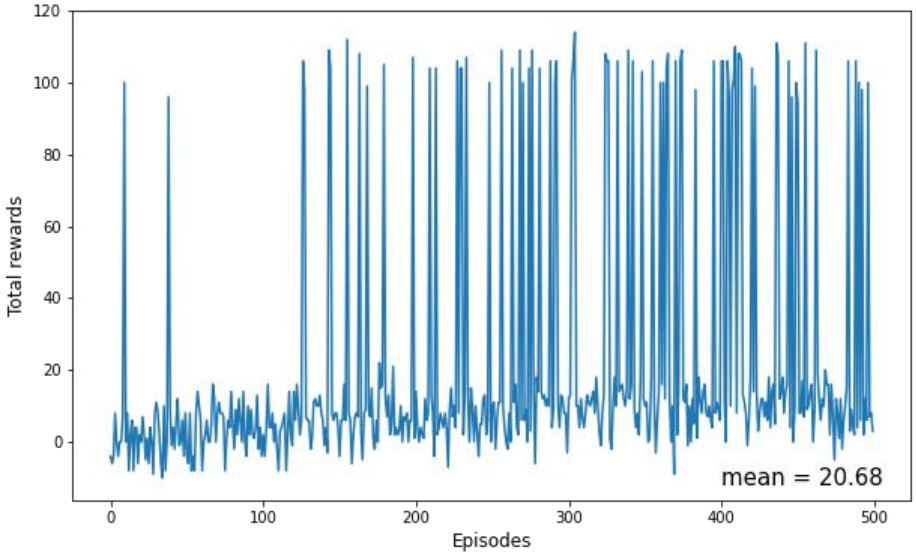
Analysis

- The graphs for total rewards against different values of no. of episodes is given above for SARSA stochastic environment
- For 100 episodes the mean reward value is 100 which is lower than the maximum reward the agent can get which is 107
- In Graph 8.2 when we train for 500 episodes the mean total reward value is 105.592 which is an improvement
- In Graph 8.3 when we train for 1000 episodes the mean total reward value is 106.126 which is much closer to the maximum reward 107
- Keeping the no. of episodes as 1000 will be efficient as the agent is able to collect the maximum reward

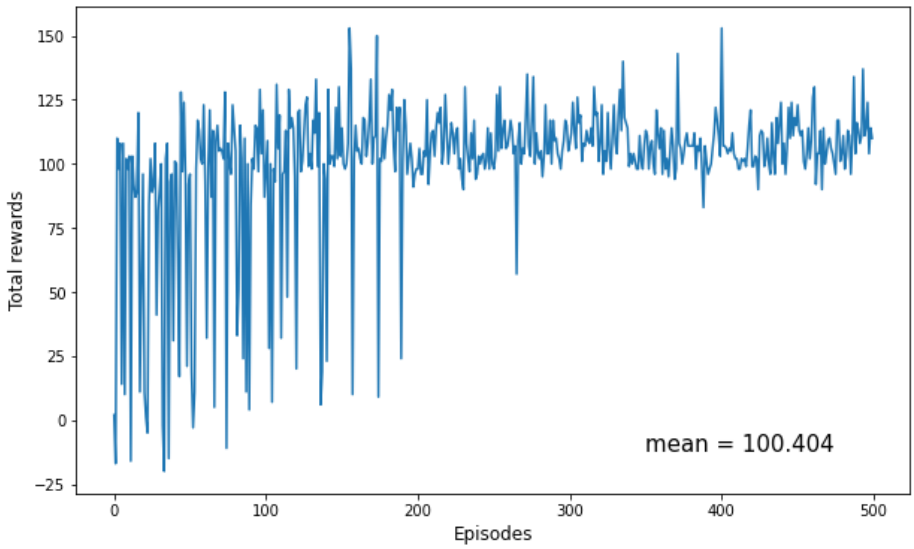
8.2.2 Tuning maximum timesteps value

For stochastic environment when training for 500 episodes, trying different maximum time steps value to see if we can improve the total no. of rewards

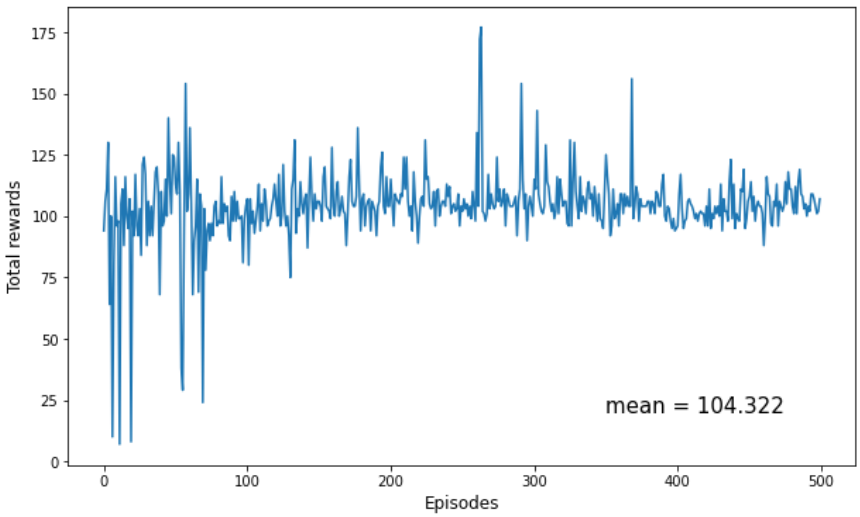
Max timesteps = 10:



Max timesteps = 50:



Max timesteps = 100



Analysis

- From the above graphs we can analyze the mean total no. of rewards obtained against different values of max timesteps
- We start with max time steps = 10 and we get a mean total reward = 20
- As we increase the max timesteps = 100 we get the mean total reward = 104
- By increasing the max timesteps we observe that the total reward value also increases because this helps when the agent is exploring to sample more states and evaluate the rewards

8.3 Project uploaded to GitHub

- The jupyter notebook is uploaded to GitHub at - https://github.com/solo11/ub-rl
- ub-rl is added as a collaborator
- Notebook along with h5 files and images are uploaded to GitHub
- *spanduga_assignment1_checkpoint.ipynb* - Jupyter notebook for assignment part 1
- *spanduga_assignment1_final.ipynb* - Jupyter notebook for assignment part 2
- **Important** Before running the notebook please upload these file to local path of the notebook :
    - images folder - contains the images used for visualization
    - data_q_table_values.h5 - contains the trained Q-table values
    - data_total_rewards_epsilon.h5 - contains the total rewards and epsilon decay values of training