Part 1: Defining RL Environment

1. The environment that we have considered for our study is called Finding Nemo. The environment is a region of ocean confined within a 4*4 matrix. The agent is Marlin(Nemo's father) and the goal of the process is to find Nemo in the most efficient way. By efficient, we mean the path with less number of timesteps or episode length and maximum rewards (less hurdles and more positive rewarding states). Time taken is also a function of these parameters which determines the performance of a path.

Action Space: North, South, East, West

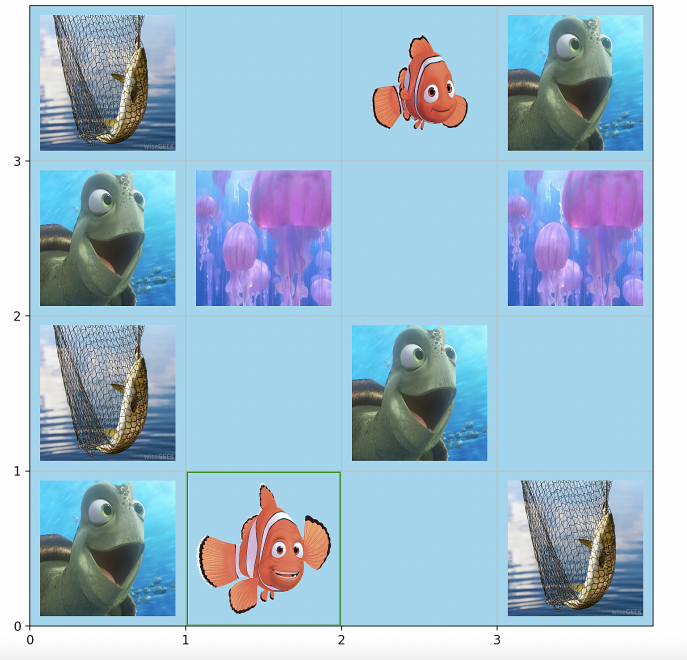
States and their Rewards: Nemo(Goal state): +50, Dory(Gives clue): +20, Turtle(Give direction and a ride): +10, Jelly Fishes: -10, Fishing Net(Trap): -50, Plain water: 0

Terminating Conditions that cause environment reset: Caught in the Fishing Net, Reached Goal State

We assign positive rewards to the states that accelerate the Search in the right direction and negative rewards to the states that pose challenges to the agent. Since we consider all states to be static, the reward at Dory and turtle state can be acquired only once in an episode. This is because taking the same clue and guidance will not be of any use to the agent.

We define accurately as to what would happen to the agent in each state and action. Hence this behaves as a deterministic environment.

2. Shown below is a visual representation of the environment consisting of 16 states. The agent Marlin is at the starting point (coordinate (1,0)) and the goal Nemo is at position (2,3)



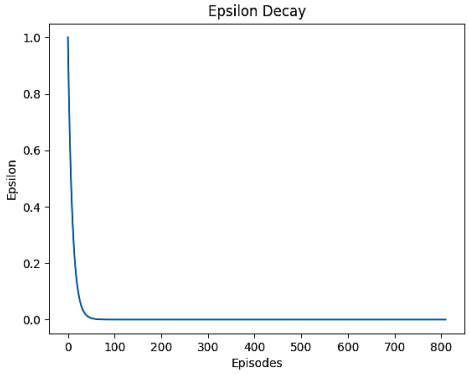
3. When we look at the real case scenario, the probability of attaining a state in this ocean environment with respect to an action is quite unpredictable. For example, There is a slight chance that the agent might escape the fishing net (say 30%), the level of how challenging it is to cross the jelly fish colony will vary with the number of jelly fishes at the time, the speed at which they travel or the agent may die out of suffocation. Also the usefulness of Dory's clue and Turtles' guidance is not completely sure. Hence, the probability cannot be predicted precisely and may lie between 0 and 1. Hence, this forms a stochastic environment where the dynamics of the system are unknown and there is no guarantee that the agent will end up in a particular state if he performs a particular action.
In our code, we have added probability to the states that have uncertainty, to get an idea about how the results differ from a deterministic environment.

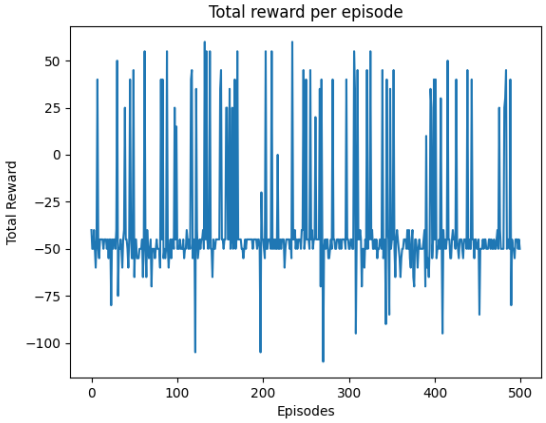
4. In a deterministic environment, the predictions can be made precisely since the states are static and the next state of the agent can be completely determined by its current state. Whereas in a Stochastic environment, the prediction is not precise since randomness exists in the states and the next state cannot be completely determined by the agent. However, through a thorough training of appropriate datasets which covers as many cases possible, we can get close to the actual value. The pattern model should be designed in such a way that all deterministic factors are taken into consideration.

5. We ensure the safety of our environment by holding the agent in the same state when attempting to move across the state-space borders. In case of arriving at the terminal states, we reset the environment and take the agent to its original state. We ensure that the agent takes only once a one-time reward by keeping an array of flags mapped with these states. We explicitly print our invalid actions in all cases.

Part 2: Applying Tabular Methods : *Q-Learning and SARSA*

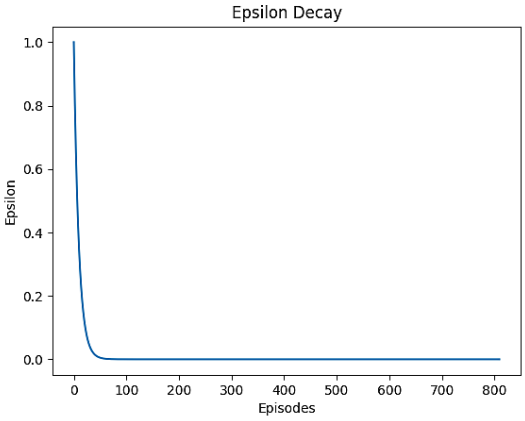
- *Applying Q-Learning to solve the deterministic environment*

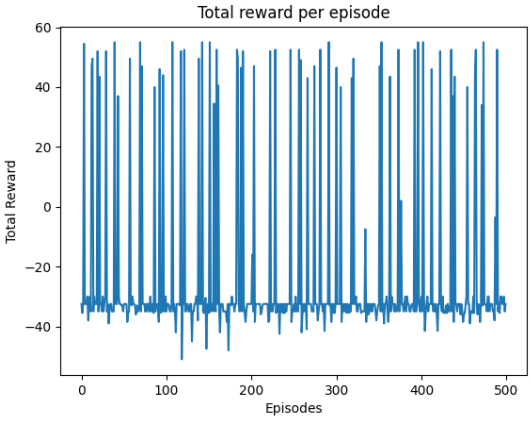




We can observe that episodes or paths navigated with positive rewards increases and negative rewards decreases at later stages as training progresses. However, the learning is not consistent.

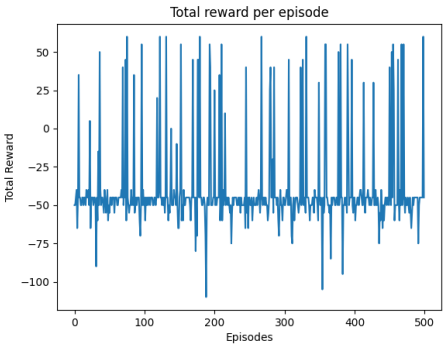
- *Applying Q-Learning to solve the stochastic environment*





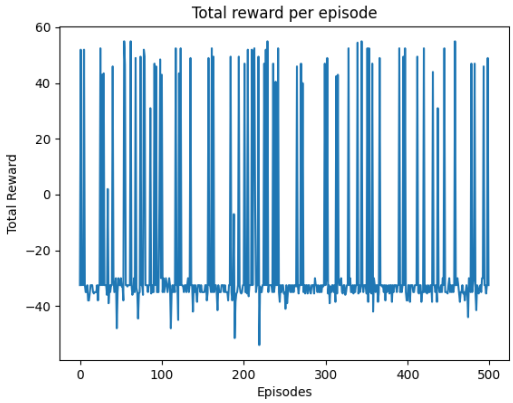
We can observe that in our environment, there is not much improvement in the training because of the randomness brought by the stochastic property of the environment and the risk factor of states ignored by the Q-learning's evaluation during the training, which could have taken the learning in the wrong direction.

- *Applying SARSA to solve the deterministic environment*



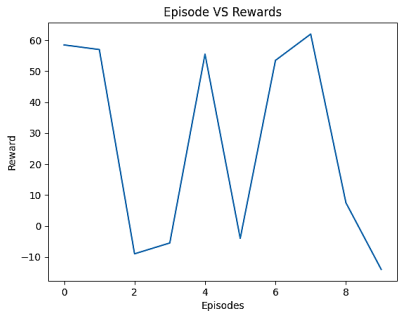
We can see that as training progresses, the number of positive reward hits are increasing and negative hits are decreasing. This is converging to the optimal value function. The learning is consistent.

- Applying SARSA to solve the stochastic environment



We can observe that in our environment, there is only slight improvement in the training because of the randomness brought by the stochastic property of the environment and SARSA takes longer time to converge, it might take much more training of episodes to be able to see any progress.

- Evaluation



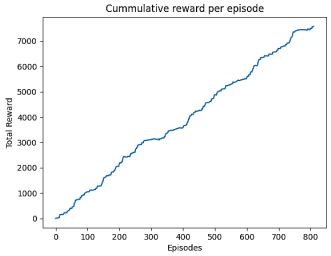
We have received 5 successful episode out of the 10 episodes using the learnt policy after training 800 episodes. The agent learns slowly over the period of training and gradually starts making better decisions and moves towards consistent positive decisions. We find that the learning is dependent on the stochasticity of the environment, the measure of risk factor taken into account during Q value evaluation. The congestion of hurdles or traps in the environment makes the learning more difficult.

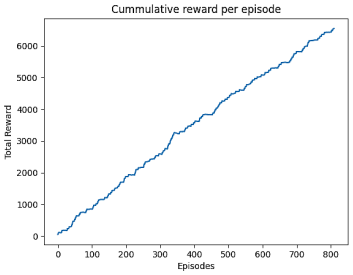
2. SARSA requires more experience to improve the estimates, but grows linearly. Q-learning requires less experience, but is not consistent with the growth. We can see that Q-Learning improves till the 170th episode and suddenly drops during the training. But its not the case with SARSA.

3. SARSA performs better than Q-Learning in stochastic environment since Q-learning misses the risk factor brought by the stochastic environment. SARSA's learning is very slow in stochastic environment when compared to deterministic environment, however, SARSA appears to be the better option among the two.

4. Q-Learning - It is an off-policy algorithm because the new action is taken as greedy and not using the current policy. It picks the maximum action value from the next state and ignores the remaining actions irrespective of how dangerous they are. Q-learning converges to optimal policy. It uses past experience in the old state of the policy to improve the q-values of current policy. Requires less experience to learn, however is not stable. Update function used is:
   $Q(st,at) \leftarrow Q(st,at)+\alpha[rt+1+\gamma \max a' Q(st+1,a')-Q(st,at)]$ where
   $Q(s_{t+1},a_{t+1}) = \max_a Q(s_{t+1},a)$
   SARSA - It is an on-policy algorithm since new action is taken using the same greedy policy as that of the old action. It randomly picks action from the next state. It will tend to avoid a dangerous optimal path and slowly learn to use when epsilon reduces. We reduce our epsilon by 0.9 times the value each episode. Update function used is :
   $Q(st,at) \leftarrow Q(st,at)+\alpha[rt+1+\gamma Q(st+1,at+1)-Q(st,at)]$ where
   $Q(s_{t+1},a_{t+1}) = \varepsilon \cdot \text{mean}_a Q(s_{t+1},a) + (1-\varepsilon) \cdot \max_a Q(s_{t+1},a)$

Cummulative reward per episode of deterministic and stochastic environment



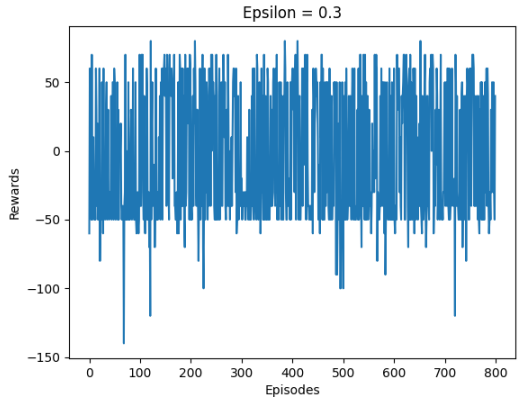


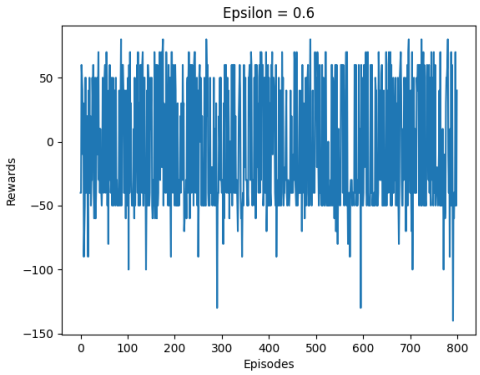
Total reward per episode during training : 16.7

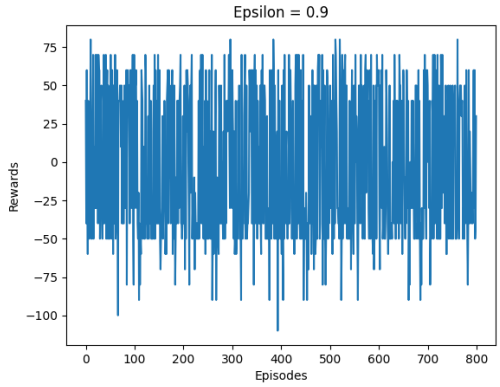
Total reward per episode during evaluation : 30

Percentage of episodes in which the agent achieves the goal : 25%

Hyperparameter epsilon = 0.3, 0.6 and 0.9







For our environment, lower epsilon rate gives less number of negative rewarded episodes since the learning is slow and needs more gradual transition to greedy action function.

REFERENCES:

- Richard S. Sutton and Andrew G. Barto, "Reinforcement learning: An introduction"
- https://stats.stackexchange.com/questions/316447/can-q-learning-or-sarsa-be-used-to-find-an-stochastic-policy
- Nithin Kulkarni's demonstration resource