ESE680-005: Reinforcement Learning
2019 Final

1)

a) The reward function **rew_calc** is modified and the reward values are returned as follows.

1.Draw, Lose, Not Done (Reward=0)

2. Win (Reward=1)

The agent receives a **zero reward at every step when the game's not done**. This is because there's no valid conclusion during the game and the outcome can be determined only at the end of every game. In Tic-tac-toe, the game is to be improved after training and the number of losses must be reduced over the training time when a proper learning algorithm is implemented. In my case, I've avoided using the negative reward for the matches lost.

Whenever, a win is encountered, the reward value is updated by 1 for an episode and the Q value is calculated after every action and state. The rest of the time, the reward value is zero. So, the action just before winning is provided with a higher Q value adding to the previous set of actions which resulted in that and will be used frequently for the next matches.

Also, the states which have been occupied already in the past is made sure that it's not updated during a game.

b) I've used a **Q-Learning** algorithm for the learning process for discrete state space. The structure of the tabular Q function was designed using a Python dictionary with Encoded State Values (a 3*3 matrix flattened to a 1D array and converted to a string.) as Keys and the Actions (a numpy 1D array of 9 values) as Values. The reason for Q-Learning to be used is that it helps to select the best action from a state. The Q-function takes two inputs and returns the expected future reward of that action at that state. Q function scrolls through the Q-table to find the line associated with state, and the column associated with action. It returns the Q value from the matching cell. The learning agent overtime learns to maximize these rewards to behave optimally at any given state it is in and iteratively improves the behaviour of the learning agent.

The **epsilon greedy policy** with epsilon value equal to (2-0.005) was selected to select random actions from 1 to 9 when random probability value is less than epsilon. This policy is adopted to minimize the possibility of overfitting during evaluation Using epsilon of 0 is a fully exploitative and there's no scope for exploration. There can be cases where the agent gets stuck to a location while following the greedy policy by greedily choosing the Max Q value action. So, a little amount of exploration in its policy allows it to get out of such states.

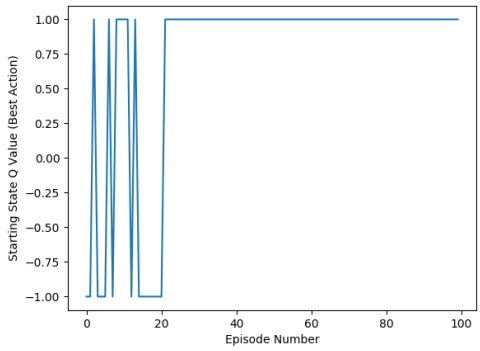
Also, the number of possible states for the Q-Table is calculated to be 3^9(19683) ('X','O',' ') and the number of actions as 9.

3) The fully trained agent doesn't always win particularly during the initial training period. Also, there's a little space for exploration as epsilon is initialised as 0.05 and hence there are

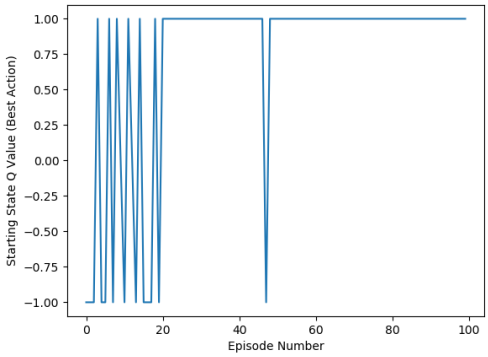
possibilities to select bad moves with low Q values. In order to make the agent always win, we must use reduce epsilon over time close to 0 during training and limit exploration.

It's possible to outperform my agent using a better algorithm or with the same case using a lower epsilon value. After various trials, I was able to outperform my own training convergence by reducing epsilon values from 0.2 to 0.005. The training was done for 100 episodes and convergence could be seen for epsilon 0.005 and epsilon 0.01.

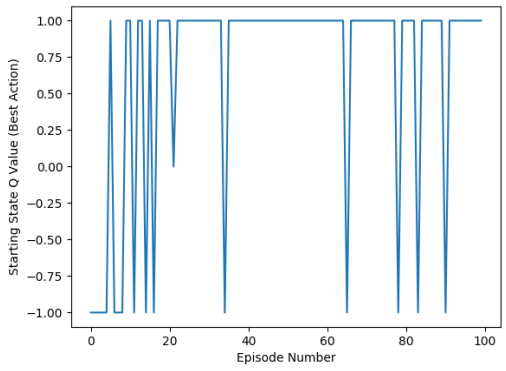
Epsilon 0.005



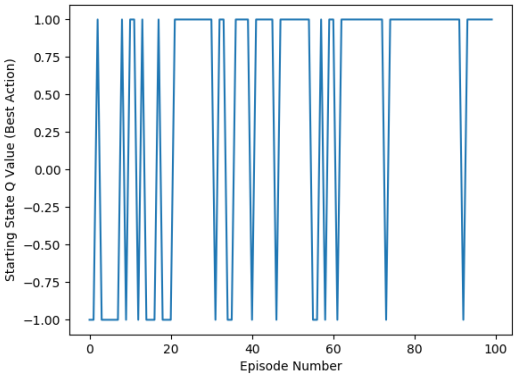
Epsilon 0.01



Epsilon 0.05



Epsilon 0.2



d)

As per the question, we can know whether the agent has learned the globally optimal policy if it's made to play with a copy of its own agent and if one of the agents wins over the other, then it's not a globally optimal policy. (it should always be a draw if the agents with global optimal policy play with each other.) I was able to play only a single match as the training

wasn’t done properly. But since, in that single match, one of them won once and hence I can conclude that it didn’t achieve global optimal policy.

Two functions step1() and step2() (copies of step() function)were added to the environment file of tictactoe for generating X and O for two agents. (Removed ‘getopponent()’ function

```
---------------
| O || X ||   |
---------------
|   ||   || X |
---------------
| O ||   ||   |
---------------
---------------
| O || X ||   |
---------------
|   ||   || X |
---------------
| O || O ||   |
---------------
---------------
| O || X || X |
---------------
|   ||   || X |
---------------
| O || O ||   |
---------------
---------------
| O || X || X |
---------------
|   || O || X |
---------------
| O || O ||   |
---------------
---------------
| O || X || X |
---------------
|   || O || X |
---------------
| O || O || X |
---------------
Win
```