Q-Learning for Connect-4 Board Game

Devanahalli Sunil Archana(2019H1030519P)
Sanchita Badkas(2019H1030520P)

Introduction

The aim of this assignment is to simulate a 2 player board game using Q learning. The two players, simple agents or q learning agents, play against each other multiple times with different models of learning. The results of the simulation, analysis and conclusions are discussed in the further sections. The project has been implemented in Java.

Board Game

The game that was chosen for this assignment is Connect-4. This is a two-player game where each player has to drop the discs of their colours in the columns of the board and if a series of 4 discs of the same colour is formed vertically, horizontally or diagonally, the player of that colour wins the game. If the board gets full or if there is no scope of obtaining a four, the game draws. We have implemented this board game in Java. The details of the agents and their strategies are elaborated below.

Agents

An agent is a player that uses a strategy to the play against its opponent.
Three types of agents have been implemented in this assignment: Random, Simple and Q-learning.

- **Random**
  The random agent selects the next move randomly from the set of possible/valid moves that it can make.

- **Simple Agent**
  The simple agent uses a fixed strategy independent of the state in which the player is in. The state here means the current configuration of the board after the opponent has played. In this assignment, our simple agent always places the disc in the leftmost available spot amongst the valid moves.

- **Q-Learning Agent**
  A Q-learning agent uses reinforcement learning to learn the best possible move for it to take based on the state it is in through experience. The description of the Q-learning algorithm and its implementation details is given below.

MDP(Markov Decision Process)

To understand Q-learning we need to first understand the Markov decision process. Q-Learning uses MDP to model the environment.
The MDP is modelled as 5-tuple : **(S, A, P R, Y)**
where,
S - State-space
A- action space
P- transition probabilities
R- reward function
Y- discount factor.
When P and R of MDP are unknown, MDP is reduced to a Reinforcement learning problem.

Q-Learning algorithm

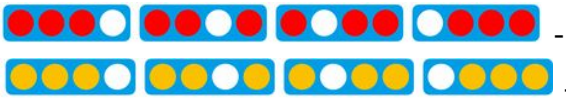
Q- learning is a type of reinforcement learning. The agent interacts with the environment and based on the optimal policy, responds to it. Since the problem cannot be supervised, it is difficult to learn the optimal solution directly. The agent learns the best policy over time through experiences like how humans learn to do tasks. Based on the response received upon doing an action, the agent receives a reward (which could be positive or negative). These rewards are stored along with the state in which the agent was and the action that it took. These are called Q values and these values are updated over time during the learning process. The details of the reward function, policy and other details of the algorithm chosen for the assignment are given below.

- **Reward function (Heuristic)**
  The Q-learning algorithm uses a reward function, to give feedback to the agent after it makes a move. This function is not defined and is based on a heuristic which is calculated based on the state and action. There are two strategies for giving a reward to an agent. The reward could be immediate or a pay-off at the end.
  One-step lookahead heuristic
  For our board game, we chose a one-step lookahead heuristic to give immediate feedback to the agent. The function checks the configuration of the board and plays a one step more after taking an action and if the player has a winning configuration then it is given a higher value based on the following formula :

  score = (num_threes) - 1.2*(num_threes_opp) + 1.5*(num_fours) - 1.8*(num_fours_opp);

  The winning configurations are if there are 3 in a row or four in a row for the player. If the opponent has 3 in a row or 4 in a row after playing that move then a negative weightage is given to such configurations.

   - 4 in a row (num_fours)

 - 3 in a row (num_threes)

- 3 in a row for opponent (num_threes_opp)

- **Policy**
  The optimal move for the agent is chosen based on the Q- values stored in the state action matrix. Three action selection strategies have been implemented: greedy, e-greedy and softmax.
  Greedy strategy always chooses the action which has the highest q value for the agent in a particular state. Always selecting the immediate best solution might not be optimal hence the agent must also be able to explore the other paths.
  In e-greedy, the agent chooses the action with the highest Q value with a probability of 1-e and remaining n actions with a probability of e/n each. The drawback of this approach is that the agent might end up choosing the worst possible action with an equal probability as better options.
  Softmax strategy solves this problem by creating a probability distribution for the set of actions based on the goodness of the move(Q values). So a move with better q value gets a better probability value.

- **State action matrix and updation**
  A Q-learning agent creates a state action matrix which contains Q value for each state-action pair. Whenever an agent revisits a state over the course of playing multiple games, the agent updates the Q-value for the current state action pair based on the following formula :

$$Q(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{\text{old Q-value}} + \underbrace{\alpha}_{\text{learning rate}} \times \overbrace{[\underbrace{r_{t+1}}_{\text{feedback}} + \underbrace{\gamma}_{\text{discount factor}} \underbrace{\max_a Q(s_{t+1},a)}_{\text{max future Q-value}}}^{\text{expected discounted feedback}} - \overbrace{Q(s_t,a_t)}^{\text{old Q-value}}]$$

  The 3 parameters used in the above updation formula are explained below along with their effect on the learning process.

- **Parameters**
  - **Learning rate(α)**: Learning rate gives how much the agent must consider future actions while learning. If the value is 0, the agent takes an action just based on current knowledge and doesn't learn anything.

  - **Discount factor(γ)**: As a state may be visited multiple times, giving the same reward always after visiting that state may increase the q value drastically and also doesn't depict the learning process accurately. Hence the value of reward changes with time. So if a good action is visited multiple times, each time the reward give to it changes with a factor of γ. If the discount factor is 0 then no reward is given for future actions and the same reward is given for each future action if it is 1.

- **Exploration quotient(ε)**: This parameter gives the probability with which the agent must choose an action other than the best option available currently ie, the action with highest Q-value for that state.

Results

The game was run for the following combinations of agents and their statistics are given below.

Player 1	Player 2	Agent Produced	Player1WinRate
Q learning agent	Random	Q1	0.67749
Q learning agent	Q learning agent	Q2	0.992395
Q learning agent	Simple agent	Q3	0.991615
Q1	Q learning agent	Q4	0.002205
Q2	Random	Q5	0.677345
Q4	Q5	Q6	0.99971
Q3	Q1	Q7	0.002205
Q1	Q2	Q8	0.99971

Statistics of different cases averaged over 10 runs. With parameters for Q learning agents as α =0.8 ,γ= 0.2, ε=0(Greedy), number of games = 2*10^5.

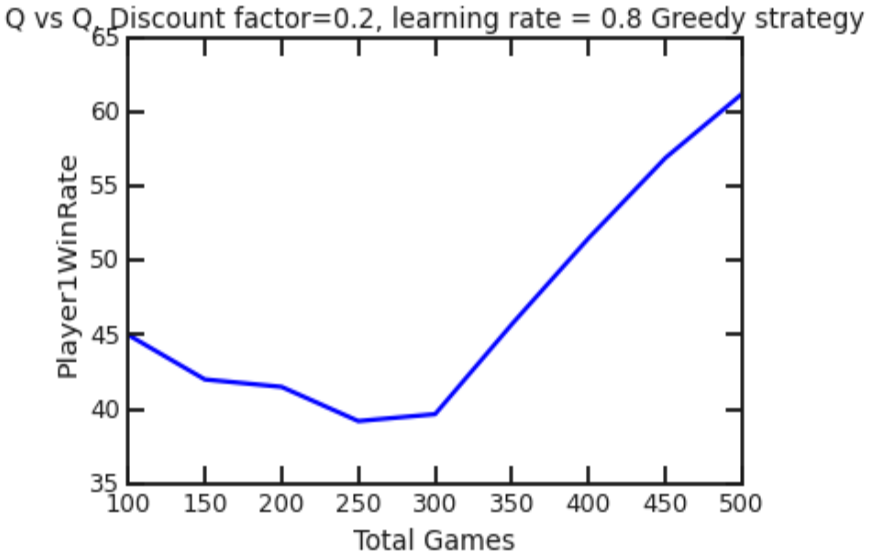
Observations:

- When an agent using Q-learning plays against a random agent, it eventually gets better. Resulting in it winning about 67% of the games. A small amount of percentage leads to a draw. As the opponent plays random moves, the Q-learning explores various configurations of the board resulting in a good Q-matrix.
- When an agent using Q-learning plays against a simple agent it wins almost all games as the simple agent follows the only rule - play to the leftmost column. Thus, the Q-learning agent quickly learns this pattern resulting in wins.
- When a Q-learning agent plays against another Q-learning agent, a huge number of games result in a draw as both agents keep playing the same set of moves

after a while as they both are simultaneously updating their Q-matrix. After a point, it starts to overfit as the same combinations of configurations keep occurring in every game.

- In Q3 vs Q1, Q3 tends to lose most of the games as it's initially gotten trained while playing against a simple agent while Q1 is trained against a random agent. Q3 has only explored a small set of combinations of the boards compared to Q1 as a result of which Q1 beats Q3 most of the time.
- In Q1 vs Q2, Q1 tends to win most of the games as Q2 is trained against a Q-learning agent. Both player and opponent started out with the same matrix and got similar updates as they were saturated by playing the same set of moves in every game. Thus, Q2 tends to overfit.
- In Q4 vs Q5, Q4 has gotten trained against another q learning agent. Contrary to the previous case, here the player had a better trained matrix compared to his opponent. Thus, Q4 wins most of the games as it has been through various states possible in the connect 4 board.
- In Q1 vs Q, Q1 uses the matrix it updated while playing against a random agent. While Q , starts from a new matrix. Q agent in a few games understands the strategy followed by Q1 and updates it's matrix accordingly leading to eventually winning most of the games.

For agent Q7, overfitting was a huge problem. Since, the total games played by the agents were 2*10^5 the matrix update gets saturated. We tried to explore if the trend varies if the agent is trained just for 500 games. The following were the results of the same,

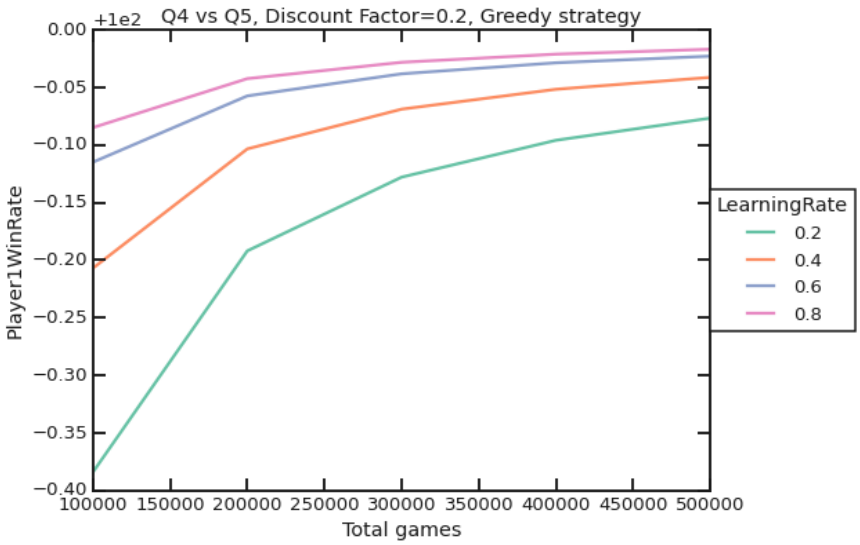


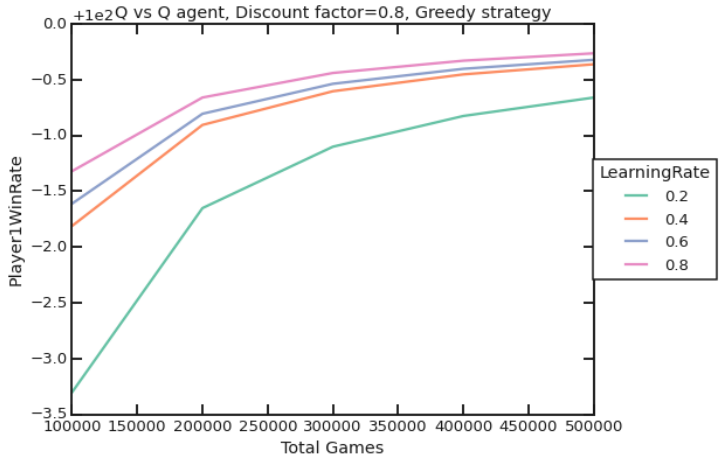
The above graph shows that the win rate keeps increasing after 300 games. However, after a lot of iterations the matrix gets saturated.

Varying the parameters for Q-learning:
These were the overall statistics for the games played amongst the agents.
We further explore how varying various parameters like learning rate, discount factor and epsilon affects the way the agents play.

Learning Rate:

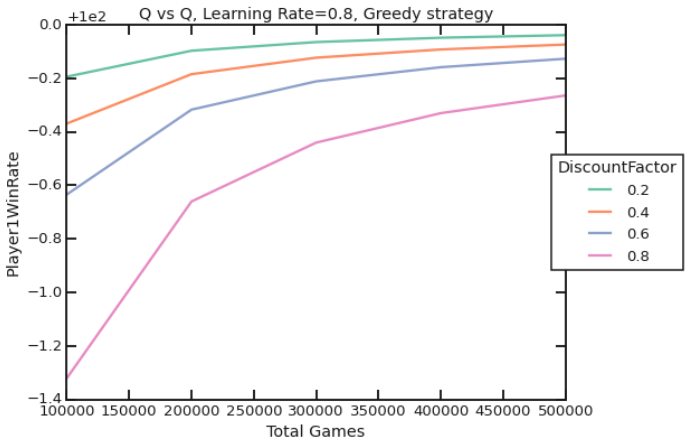
Following graphs show that for different agents, the best learning rate is 0.8. For different discount factors of 0.5 and 0.8 the same trend is followed.

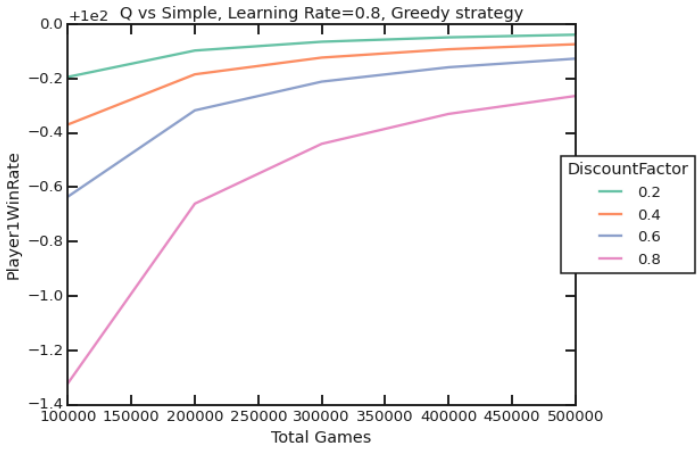


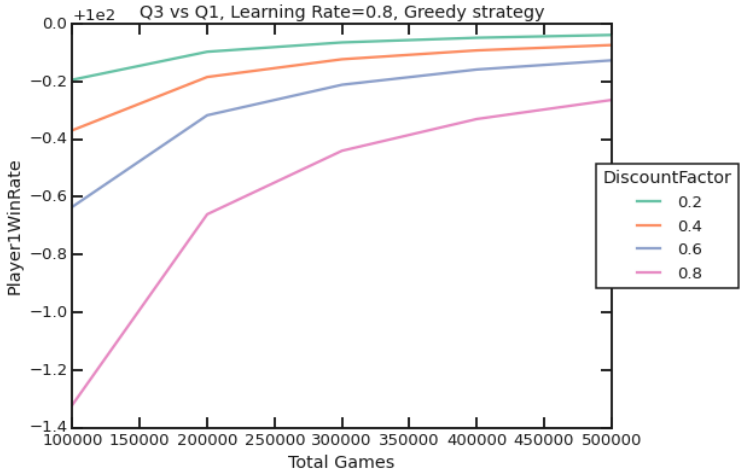


Discount Factor:

From the various graphs below displaying statistics for matches played between different agents we observe that a discount factor of 0.2 gives the best results.

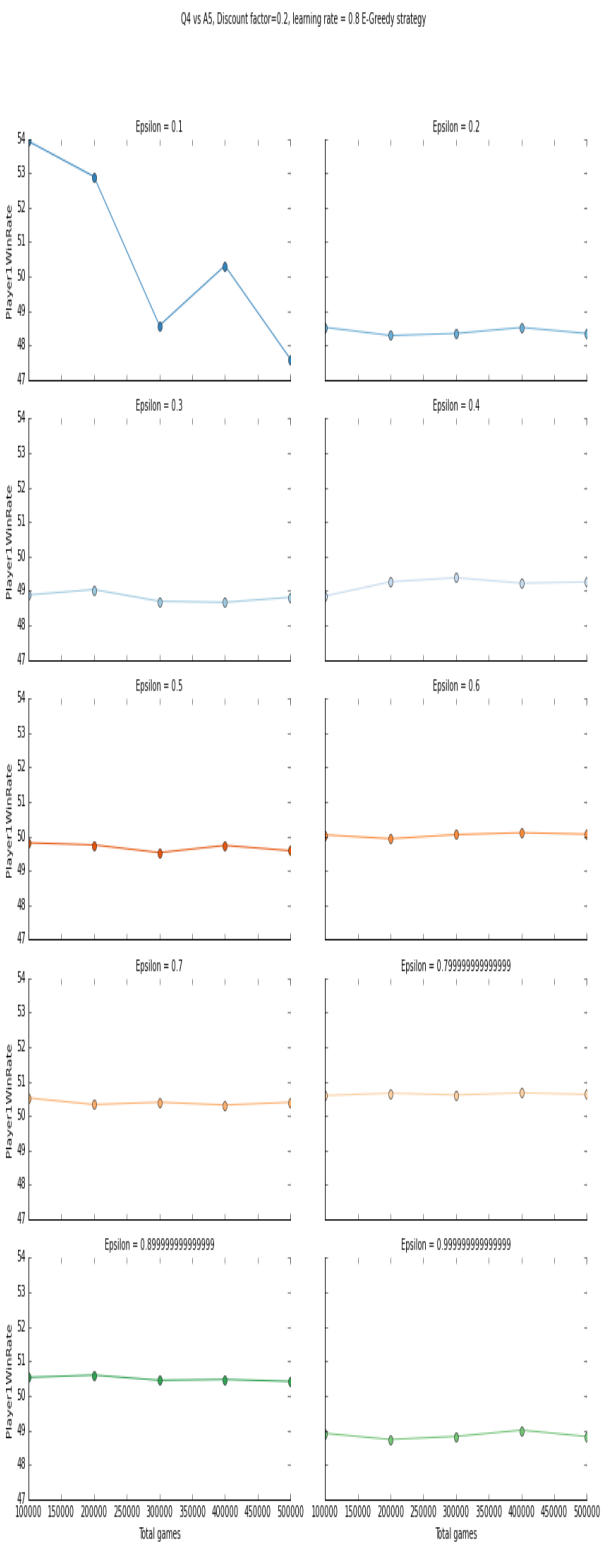


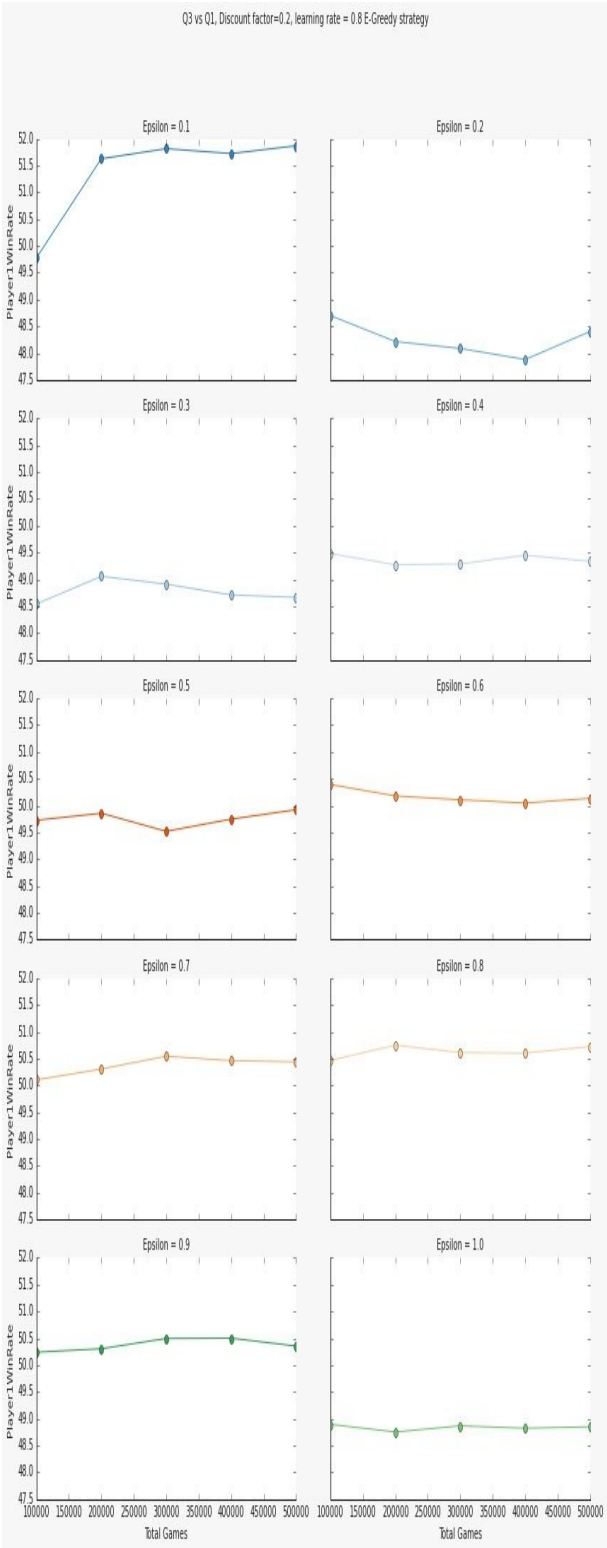


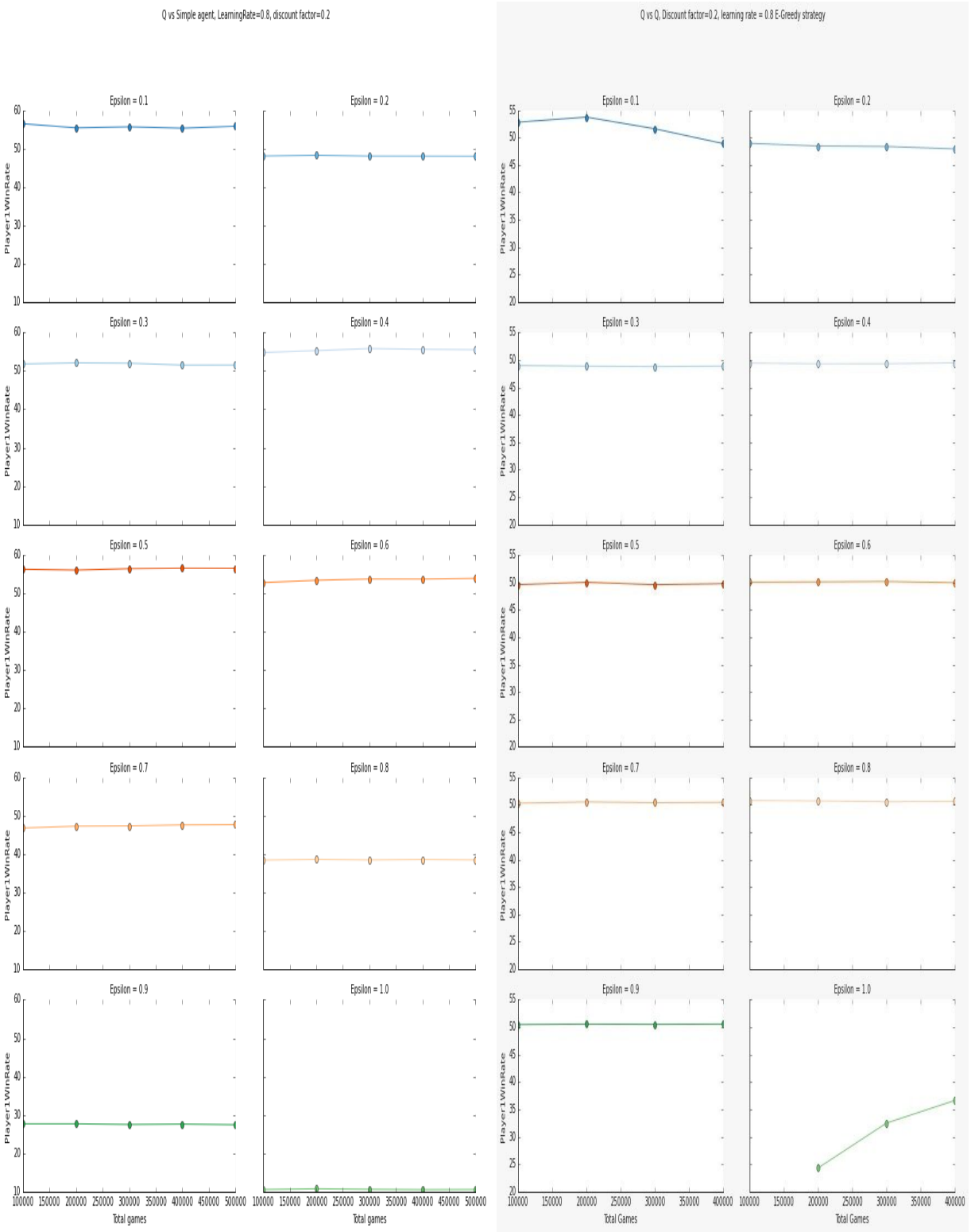


Epsilon:

From the below graphs for games played between various agents, we observe that lower the exploration factor better the results.


Q4 vs A5, Discount factor=0.2, learning rate = 0.8 E-Greedy strategy
Epsilon = 0.1
Epsilon = 0.2
Epsilon = 0.3
Epsilon = 0.4
Epsilon = 0.5
Epsilon = 0.6
Epsilon = 0.7
Epsilon = 0.799999999999999
Epsilon = 0.899999999999999
Epsilon = 0.999999999999999
Player1WinRate
Total games


Q3 vs Q1, Discount factor=0.2, learning rate = 0.8 E-Greedy strategy
Epsilon = 0.1
Epsilon = 0.2
Epsilon = 0.3
Epsilon = 0.4
Epsilon = 0.5
Epsilon = 0.6
Epsilon = 0.7
Epsilon = 0.8
Epsilon = 0.9
Epsilon = 1.0
Player1WinRate
Total Games


Q vs Simple agent, LearningRate=0.8, discount factor=0.2
Epsilon = 0.1
Epsilon = 0.2
Epsilon = 0.3
Epsilon = 0.4
Epsilon = 0.5
Epsilon = 0.6
Epsilon = 0.7
Epsilon = 0.8
Epsilon = 0.9
Epsilon = 1.0
Player1WinRate
Total games
Q vs Q, Discount factor=0.2, learning rate = 0.8 E-Greedy strategy
Epsilon = 0.1
Epsilon = 0.2
Epsilon = 0.3
Epsilon = 0.4
Epsilon = 0.5
Epsilon = 0.6
Epsilon = 0.7
Epsilon = 0.8
Epsilon = 0.9
Epsilon = 1.0
Player1WinRate
Total Games

Analysis:

Since the overall complexity of the game is quite less i.e for 5*6 board for two players, we have 3^30 different combinations. However, in connect 4 most of those configurations would never occur as the game would end before it reaches a certain combination. Due to this, a high learning rate allows the agent to get trained faster. This is especially effective in case of a fresh Q-learning playing against an experienced opponent. The fresh Q agent learns faster due to higher learning rate.

On similar lines of low complexity of the game, a low discount factor gives quicker results as with a low discount factor the Q-learning agent is optimized for choosing a fast way to win before a more secure strategy. Since the overall combinations are low, and possible moves are limited by the number of columns in the board this strategy works well.

For epsilon, we observed that as the exploration factor increases the winning rate of Q-agent decreases. This is because as the exploration factor increases the probability of picking the best Q-value move reduces. Due to low complexity we can safely assume that the number of local maximas is not high as the case of an agent getting stuck in it doesn't occur irrespective of low or zero exploration.

However, a major concern in such a game is overfitting. As the number of iterations increases the Q-matrix tends to get saturated and similar combinations of games are played over and over.

Also, from the first table we understand that, as the agents keep playing against stronger opponents the better they get trained. It's necessary to introduce some randomness in the processes when two Q-learning agents are playing against each other as they both update their Q-matrices in similar fashion leading to saturation. This randomness can be added in the form of an exploratory factor (epsilon).

Due to time constraints, we couldn't get the statistics for the bigger board which we believe would help reduce the overfitting issue.

References

**[1] S. Marsland,Machine Learning: An Algorithmic Perspective.
New York, NY, USA: Taylor & Francis, 2011.**

[2] https://www.kaggle.com/alexisbcook/one-step-lookahead