PROGRAMMING ASSIGNMENT 4:

1. Environment:

I have used the grid world python file provided and modified it such that the reward for choosing an action that moves the bot to the terminal state is 5.0. I have implemented multiprocessing so that 500 experiments can be run over 500 episodes to plot the graph.

The script accepts command line arguments with flags given below. All flags except alg are optional:

--alg or --algorithm: q denotes q learning and s denotes sarsa
--size or --world_size: the size of the grid world. Default is 5
--gamma or --discount_factor: The discount factor. Default is 0.99
--exps or –experiments: The number of experiments. Default is 500
--eps" or --episodes: The number of episodes. Default is 500
--epsilon" or --greedy_epsilon: The epsilon for the epsilon greedy selection of action. Default is 0.1
--alpha or --learning_rate: The learning rate of the agent. Default=0.1
--lambda or –sarsa_parameter: The lambda for sarsa lambda. Default=0

In my program, the class Gridworld represents the grid world and can be initialized by providing the size of the world. The agent starts from the bottom left corner and the only terminal state is when the agent moves to the terminal state. An episode continues until the reaches the terminal state. The agent gets a reward of –0.1 moving to any state other than the terminal state and an action making the agent to move to the terminal state gets a reward of 5.0.

The class ReinforcementLearning is used to contain the methods used by both Q_learning and Sarsa. The reset_Q method is used to initialize the q values to a random value and set the q values of the terminal state to be 0. The epsilon greedy method is used to determine the next action based on the policy.

The class QLearning inherits ReinforcementLearning and has the method run_episode which initializes the agent and an episode continues until the agent reaches the terminal state. The steps taken by the episode is returned.

The class Sarsa inherits ReinforcementLearning and has reset_e which is used to reset the eligibility traces between experiments and the run_episode method to train the agent.

The steps returned from the agents and the maximum initial q values are averaged and the results is plotted for each episode. Some of the graphs are posted below and the effect of various parameters like learning rate, epsilon and lambda are discussed.

If more than one action has the same maximum value, then one of the maximum value is chosen at random. This is done to improve the learning of the agent so that it does not keep choosing the first action with maximum value.

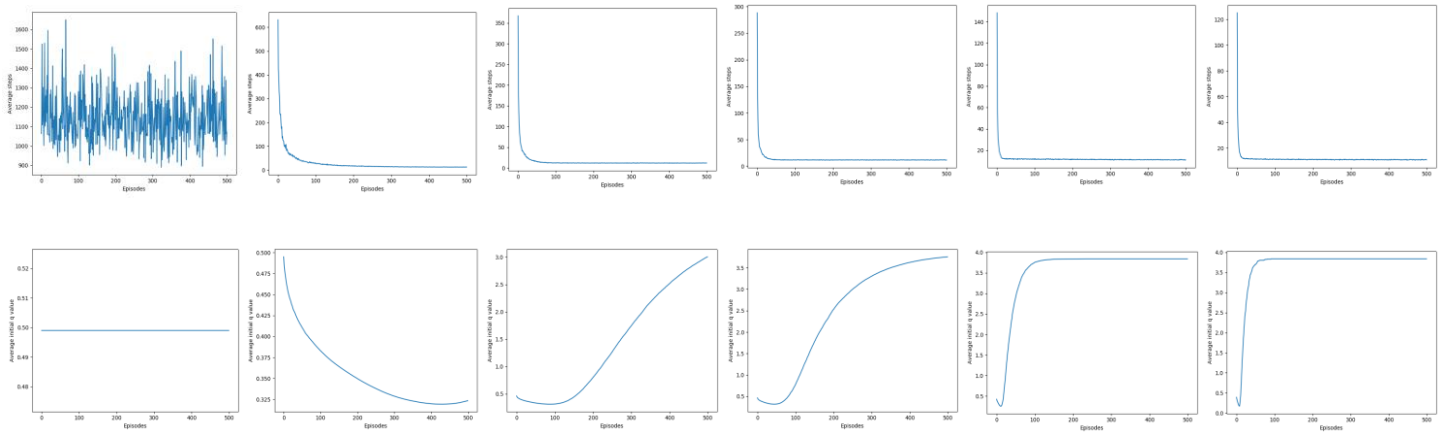
2. Effect of learning rate alpha on Q Learning:

The learning rate is used to determine the degree with which the Q value changes as given by the equation:

$$Q(s,a) = Q(s,a) + \ alpha * (reward + gamma * \max(Q(s',a') - Q(s,a)))$$

If alpha is 0 then the agent never learns from the rewards it took by taking an action from a state and hence no learning happens, and the agent does not provide an optimal path and continues to act randomly. The q values remain the same. If alpha is near 1, then the agent converges very soon but each action by the agent will change the Q value by a lot. If the environment has a random component, then the q values will diverge since the random component will cause the q values to keep changing. Our environment does not have any random components and the rewards and next states are deterministic. Hence the q values converge very soon when the learning rate is set as 1 but it changes the q values drastically and hence they do not produce optimal results.

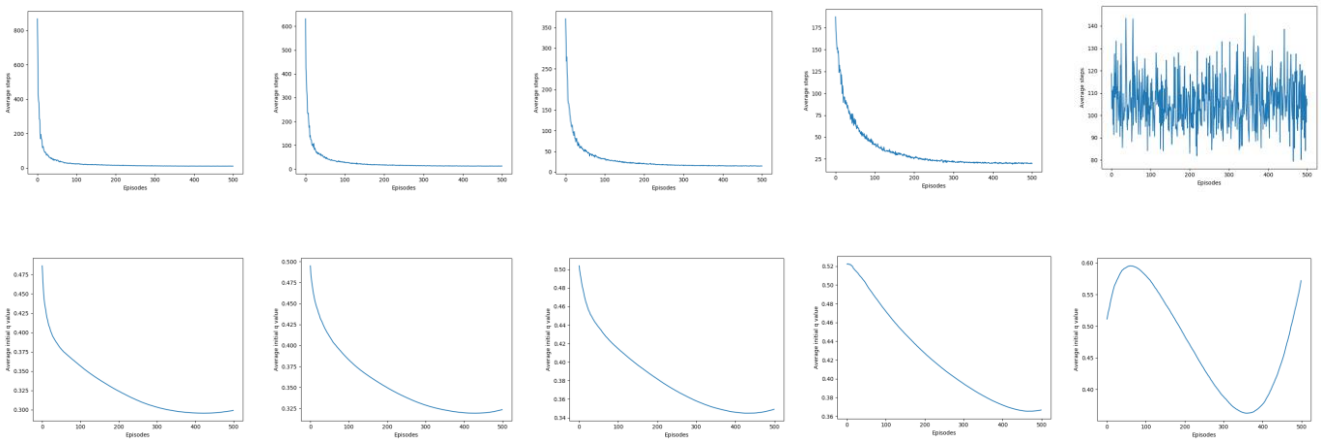
The graphs represent the average step taken by a q learning agent to reach the terminal state and the graphs below then represent the maximum initial q value at each episode for various alpha. The training environment is a 5 x 5 grid world with epsilon as 0.1, discount factor as 0.99 and the agents is trained over 500 episodes and the experiment is repeated 500 times.
The learning rates are 0, 0.01, 0.05, 0.1, 0.5, 1 (Left to Right)



As we can see from the graph, when alpha is zero, the q values are never updated and hence remain constant and the steps never converge since there is no learning. As the learning rate increases, step size converges soon, and the initial q values also increase. The initial Q values first drop and then increase for sufficiently large learning rates and this is because the agent is still learning and has not explored all the states and since the initial q values are random, the agent has to adjust its values. For a high learning rate since the q values are adjusted significantly, the convergence of the initial q value is also fast since the initial random values are adjusted and hence their effect is less. With lower learning rates, the agent takes some time to adjust the q values to provide an optimal path.

3. Effect of epsilon on Q learning:

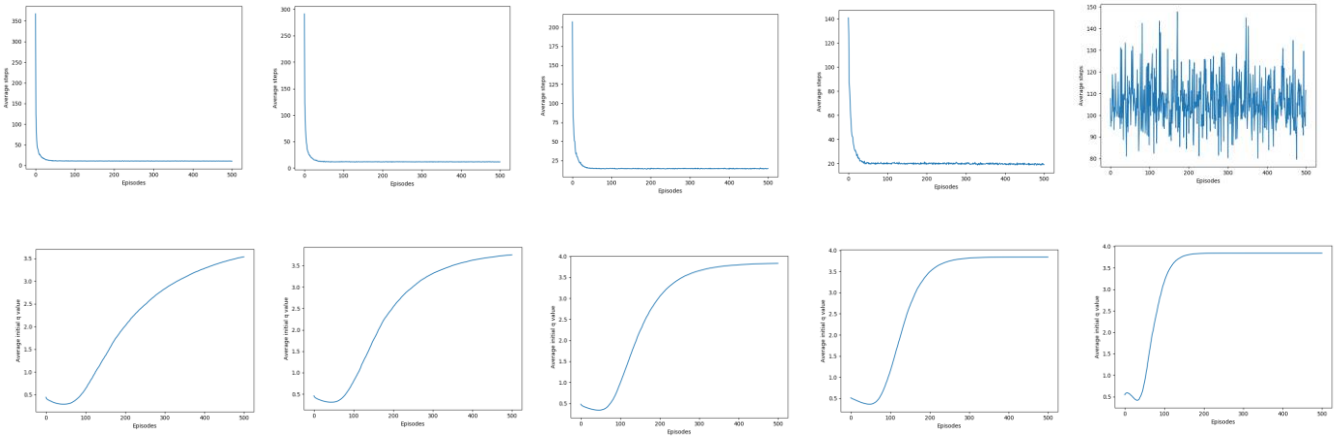
Epsilon is used in the epsilon greedy strategy to determine how much the agent can explore by abandoning the learnt maximum optimal path.
With a probability of epsilon, the agent does not take the action specified by the maximum Q value and chooses a random action (including the action with the maximum Q value). This is used to ensure that the agent explores to find new paths which could lead to discovering an optimal path. Instead of exploiting the already learnt values, the agent explores new paths. If epsilon is zero, then the agent does not learn any new path and hence will keep choosing the same path. The given graph is when alpha is 0.01 and the training environment is a 5 x 5 grid world with discount factor as 0.99 and the agents is trained over 500 episodes and the experiment is repeated 500 times.
The epsilons are 0, 0.1, 0.25, 0.5, 1 (Left to Right)



As can be noticed from the graph, the initial q values increase a little as we increase the epsilon, but it does not improve a lot since the learning rate is very low. The q values increase a little since the agent explores more options and hence explores more path. As the epsilon increases, the agent prefers to choose a random path and explore rather than exploit the knowledge obtained by the previous episodes. Hence the agents take more time to converge. When the epsilon is set to 1, the agent always prefers to explore, and previously calculated q values are never used and hence the agent never converges to produce an optimal path. It never uses to q values it learnt so the graph is similar to that of having no learning (setting learning rate to 0). The initial q values keep changing in this case though because, updating on q value happens.

Learning rate set at 0.1

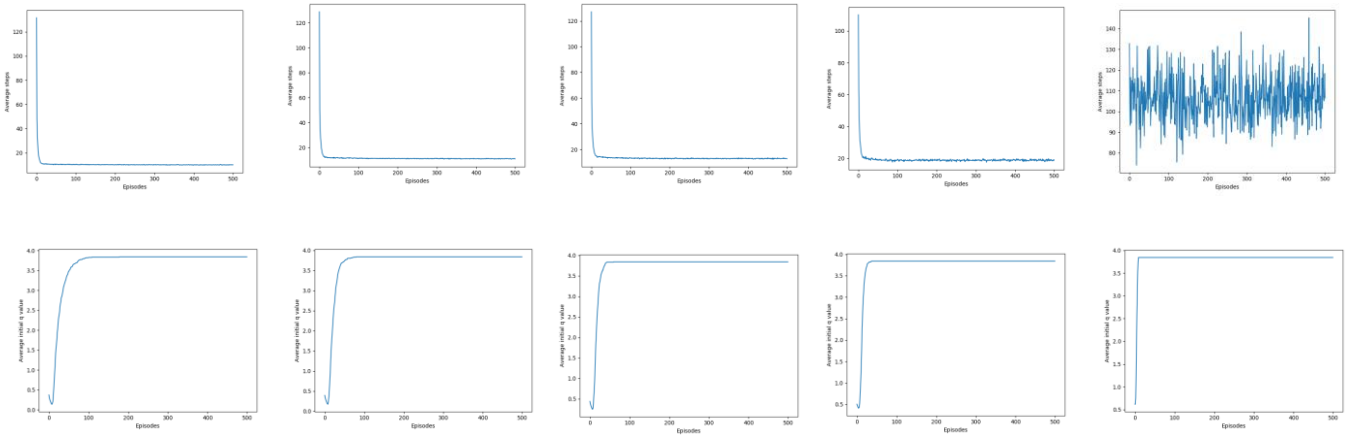
The graphs when the learning rate is set at 0.1 is given below.



As we increase the learning rate, the agent learns from its previous experience and hence uses it when it is not exploring. Similar to having a lower learning rate while training, having epsilon as zero, irrespective of the learning rate causes the agent to keep choosing the same path and hence the number of steps converge very soon, and the initial Q value also increases significantly due to higher learning rate. But the agent never explores and always takes the same path and hence finding an optimal path is not guaranteed. Similarly, when epsilon is one, the agent keeps exploring and hence the steps are erratic. But the q values get updated and hence due to higher learning rate, the initial q value also increases. For values in between, the trade-off between exploration and exploitation is set by epsilon and hence the agent converges after a point.

Learning rate set at 1

The graphs when the learning rate is 1 and across various epsilon is given below. The learning rate does not change the observation when epsilon is 0 or when epsilon is 1. When epsilon varies from 0 to 1, the begins to explore more than exploit the q values already learned. Hence, when epsilon increases, the steps taken to reach the terminal state converges later since the agents tends to explore more. The initial Q values however reach a maximum sooner when epsilon is higher and since the agent explores optimal paths.



4. SARSA:

Effect of Learning rate

The effect of learning rate on SARSA agent is similar to that of Q learning agent.

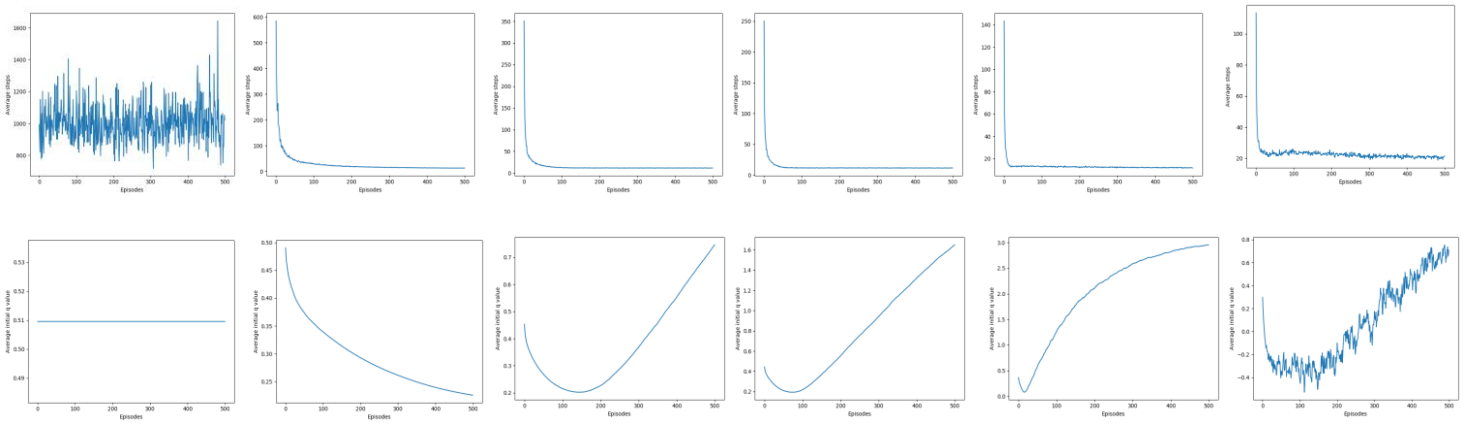
$$Q(s,a) = Q(s,a) + alpha * delta * e(s,a)$$

Where,

$$e(s,a) = gamma * lambda * e(s,a)$$
$$delta = reward + \big(gamma * Q(s',a')\big) - Q(s,a)$$

Setting the learning rate, alpha, to 0 will cause the agent to learn nothing. Hence, the q value remains constant as show in the first graph and the agent never converges, learning rate decides how much effect it has on the q value. If learning rate is too high then the q value varies very much at each step and hence for low q value learning is too small and for high learning rate, the q values are unstable hence again an optimal path may not be reached. Choosing a learning rate such that it leverages the old q value and new computer q value will provide an effective learning. Hence a learning rate of 0.5 as shown in the following graphs, provide the best result since it provides an overall balance in the value.

The following graphs have the training environment is a 5 x 5 grid world with epsilon as 0.1, discount factor as 0.99 and the agents is trained over 500 episodes and the experiment is repeated 500 times. The alphas are 0, 0.01, 0.05, 0.1 and 1.

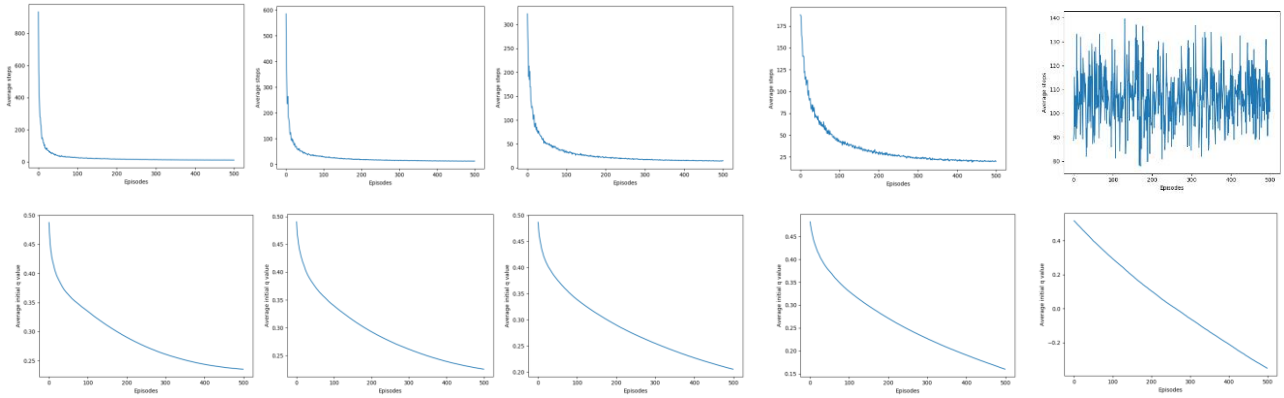


EFFECT OF EPSILON:

The effect of epsilon is similar to the observation made for Q learning. provide an effective learning. Hence a learning rate of 0.5 as shown in the following graphs, provide the best result since it provides an overall balance in the value.

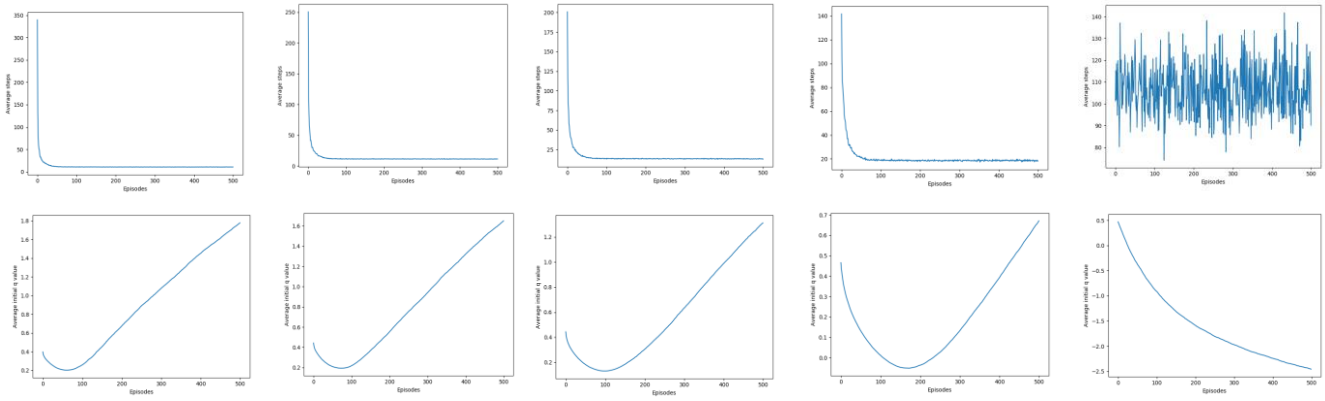
The following graphs have the training environment is a 5 x 5 grid world with discount factor as 0.99 and the agents is trained over 500 episodes and the experiment is repeated 500 times. The alpha is set to 0.01.

The learning rate is very small hence, the q values are updated in small steps. Due to the small learning rate the effect of changing epsilon is negligible, and the agent is not able to find the optimal path. Hence, the maximum initial q values are low, and they keep reducing over the episodes. Hence, we cannot be conclusive about the effect of epsilon from these graphs.



Learning Rate as 0.1

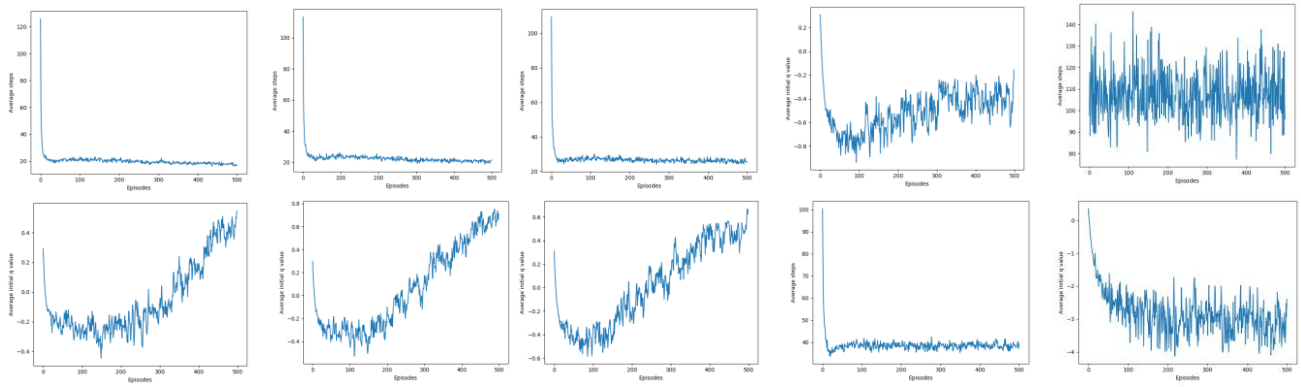
Setting the learning rate to 0.1 updates the q value significantly. When epsilon increases the agent prefers to explore rather than choose the action based on the previous observations which are present in the q values.



As can be seen from the graph for lower epsilon values like 0.1, 0.25, the agent performs well. But, as epsilon increases, ignoring the q values learned the agent chooses random action which causes the initial q value to reduce. Epsilon values around 0.1 and 0.5 work well for the given situation as it leverages a good amount of exploration and exploitation for the learning rate 0.1.

Learning rate as 1:

When the learning rate is 1, each update to q value, results in a substantial change. Hence, if our agent explores more then it will update q for each such random act and hence will result in a lot of distortion in the q values and the steps. The more our agent explores, the more our q values will vary. The effect of epsilon is more profound when learning rate is higher because more random acts tend to distort the q value from optimizing and will result in more steps. As can be seen from the graph, for a 5 x 5 grid the optimal path is of 8 steps. But the agent takes around 20-25 steps since each update of q value distorts it. This observation is clear from the second set of graphs which plot the maximum q values at the initial state. The fluctuations in the q values help us to identify that setting the learning rate to 1 does not provide good results. As epsilon increases, the randomness increases, and the q values are more distorted and tend to drop or stay the same.



Effect of lambda:

Lambda has a direct effect on the q values. When updating the Q and the eligibility traces matrix for all the state and action pair, we use:

$$Q(s,a) = Q(s,a) + alpha * delta * e(s,a)$$

Where,

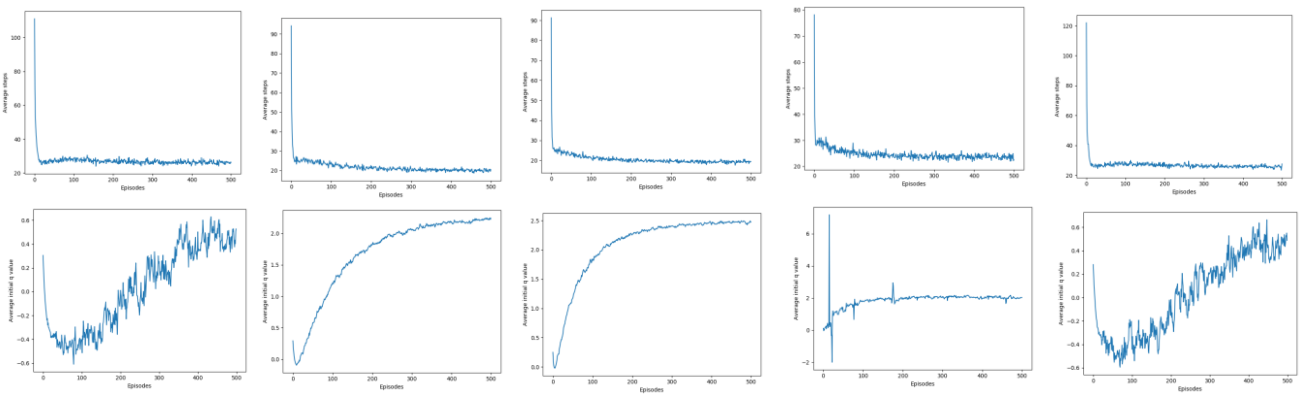
$$e(s,a) = gamma * lambda * e(s,a)$$
$$delta = reward + \big(gamma * Q(s',a')\big) - Q(s,a)$$

Based on the equation shown above, when lambda = 0, eligibility traces matrix will be a zero matrix. This results in q value not being updated, i.e., the agent will never learn from its previous observations. This leads to problem not to be converged.

As lambda increases, the eligibility traces matrix increases by the factor of (lambda * gamma), this results in change in Q(s, a). As lambda increases, it influences the amount of changes in Q(s, a). Hence, for very large lambda value, the agent will look forward too much and might not optimize well.

The training environment is a 5 x 5 grid world with learning rate as 1, epsilon as 0.25, discount factor as 0.99 and the agents is trained over 500 episodes and the experiment is repeated 500 times.

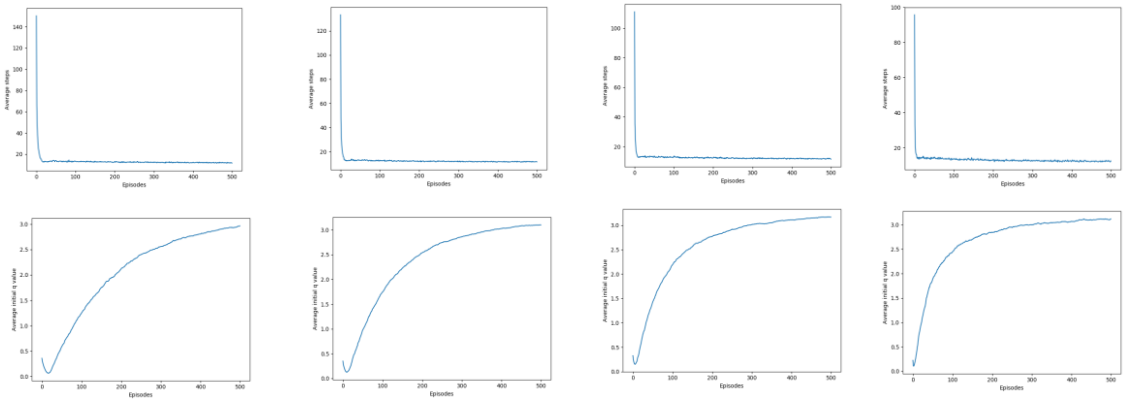
The lambda values are 0, 0.25, 0.5, 0.75, 1 (Left to Right)



Since setting the learning rate to 1 is a bad practice, as observed from the previous graphs, the similar experiment is run for learning rate = 0.5 and epsilon = 0.1

The lambda values are 0, 0.25, 0.5, 0.75 (Left to Right).

The graphs are shown below.

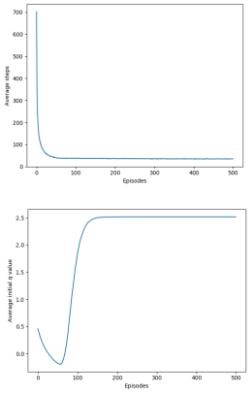


5. Choosing the right parameters for 10x10 grid world – Q Learning:

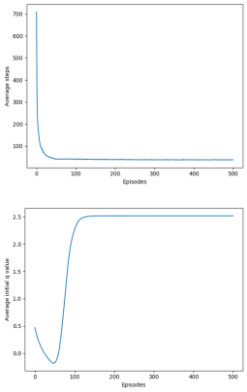
From the previous observations choosing extreme values for alpha and epsilon do not produce optimized results. Hence, for the 10x10 grid world, I experimented with alpha values around 0.5 and epsilon around 0.4. The higher epsilon rate is used so that, the agent can find the optimal paths by taking random action more frequently to explore the larger grid world. The learning rate is chosen to be around 0.5 so that, the new q values contribute as much as the existing ones. This helps the agent to converge the q values faster.

Following are some of the graphs obtained by training the agent using q learning on a 10x10 grid world.

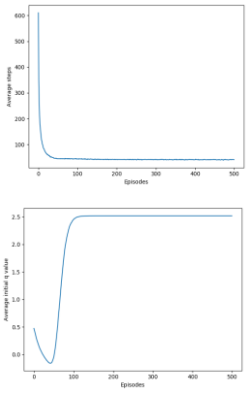
alpha=0.4
epsilon=0.35



alpha=0.45
epsilon=0.4



alpha=0.5
epsilon=0.45

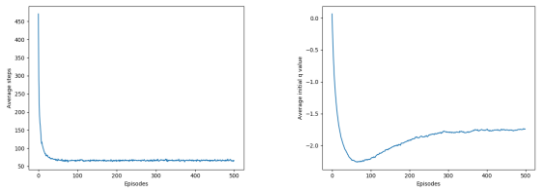


All the three combinations provide optimal solution. Concluding this, the following parameters seem to provide optimal solution for the 10x10 grid world.
alpha = 0.45
epsilon = 0.4

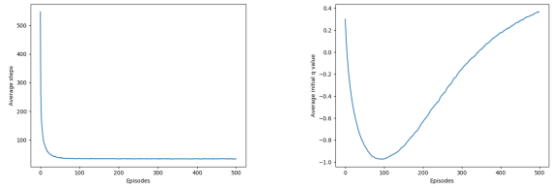
6. Choosing the right parameters for 10x10 grid world – Sarsa(λ) Learning:

From the previous observations choosing extreme values for alpha, epsilon and lambda do not produce optimized results. Hence, for the 10x10 grid world, I experimented with alpha values from 0.1 to 0.6, epsilon from 0.1 to 0.6 and lambda from 0.1 to 0.8. The results from experimenting around these values are given below:

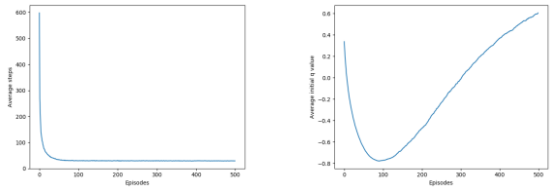
α = 0.6, λ = 0.75, ε = 0.6



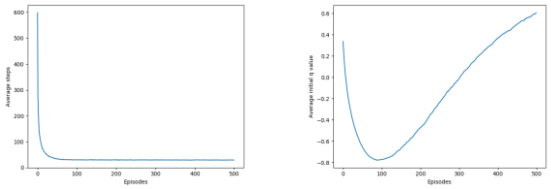
α = 0.5, λ = 0.4, ε = 0.3



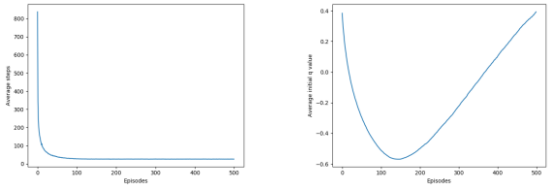
α = 0.5, λ = 0.25, ε = 0.2



α = 0.4, λ = 0.1, ε = 0.1



α = 0.3, λ = 0.1, ε = 0.1



From the graphs obtained, for higher the values of α, λ and ε, the agent is performing worse. On the other hand, as the values decrease the agent shows better performance. From the range of values, the parameter values α = 0.4, λ = 0.1 and ε = 0.1 provide the optimal solution. It takes 50 episodes for the agent to converge to the optimal policy.

7. Choosing α, ε and λ for Q – Learning and Sarsa (λ):

Choosing an optimal alpha, epsilon and lambda is crucial for finding the optimal path using Q – Learning and Sarsa (λ). If alpha is chosen to be 0, then no learning happens and hence there will no convergence. If alpha is chosen to be too high, then the q values are updated by huge values at each episode and hence the q values will be distorted, and the agent will not be able to derive a good solution. Likewise, if epsilon is set to be 0, then no exploration happens and hence the agent keeps using a fixed path and does not try to discover alternate paths. Setting epsilon to be too high will be cause the agent to never use the q values learnt. Choosing a learning rate of around 0.1 and epsilon depending on the size of the problem (usually around 0.3). The lambda for Sarsa is used to specify the degree with which the Q values are updated. It denotes the how many states must be considered while updating the q value. Setting a very large or very small lambda value will result in a distorted q value.