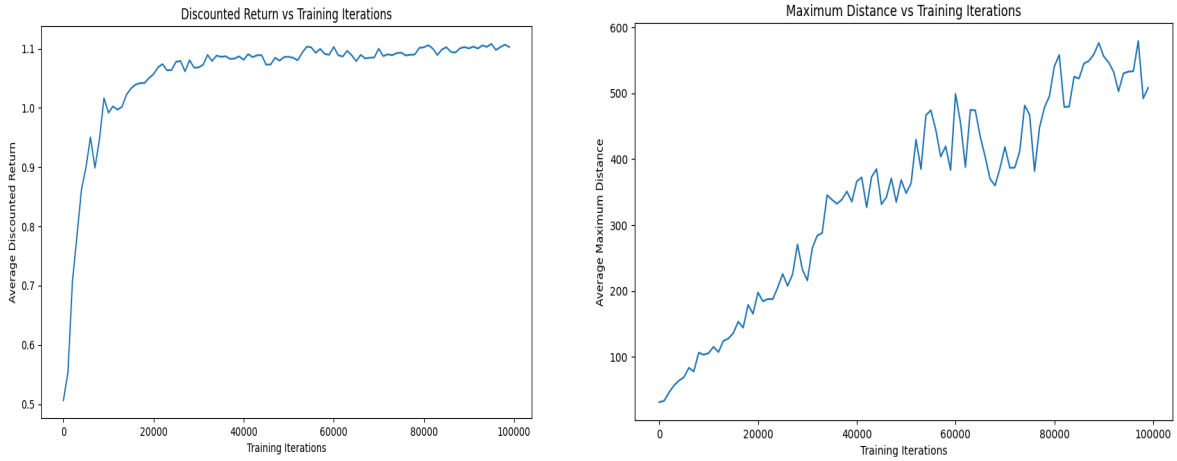
# COL778:Principles of Autonomous Systems
# Assignment-II

Savudam Sai Sathvik 2024JRB2022
Parthasaradhi Reddy N 2024JRB2028

## Part 1 : Tabular Q Learning

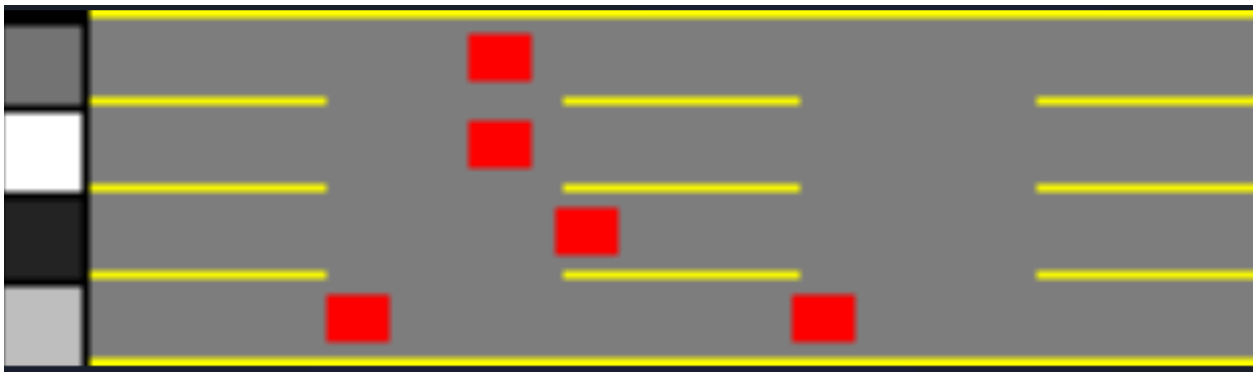
**a)**With learning rate to alpha = 0.1, epsilon = 0.75, the discount factor gamma = 0.9 and trained for 100000 iterations. These are the plots that are obtained for discounted return from the start state and the maximum distance traveled from the start state by control car averaged over 1000 trajectories using learned policy.

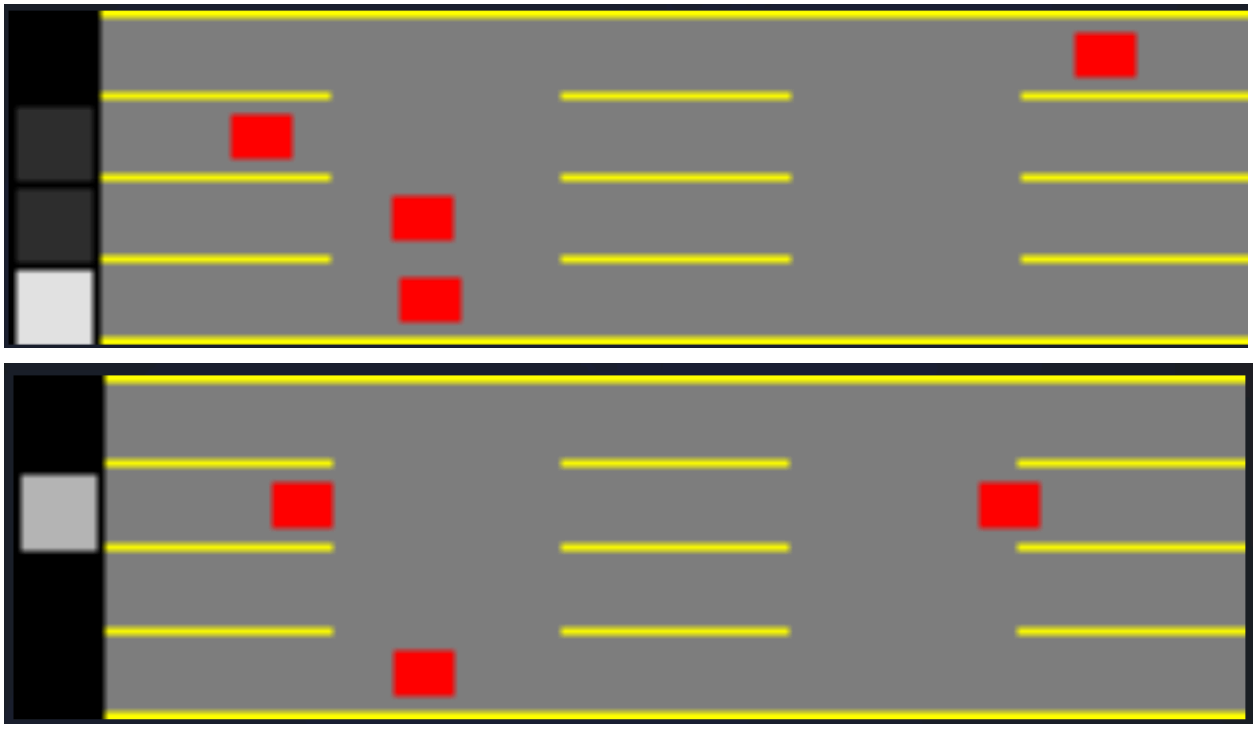



**Plots of return and average maximum distance**

**Visualization of Lane Value:**

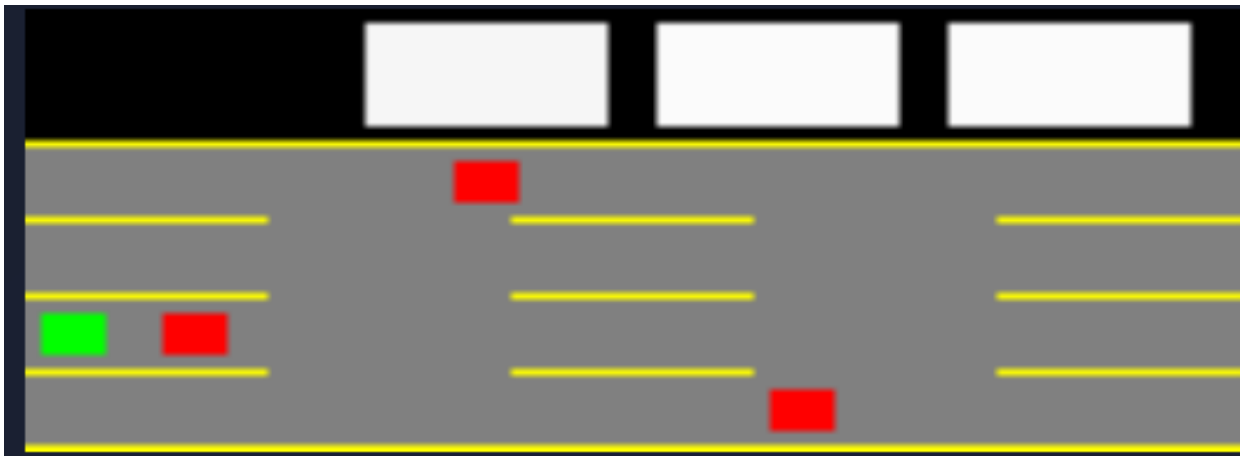
Visualization was performed for each lane by averaging the Q(s,a) values across all speeds. It was observed that a favorable lane for staying was colored black, while an unfavorable lane for the control car in a given state was shaded from gray to black, depending on the severity of its undesirability.

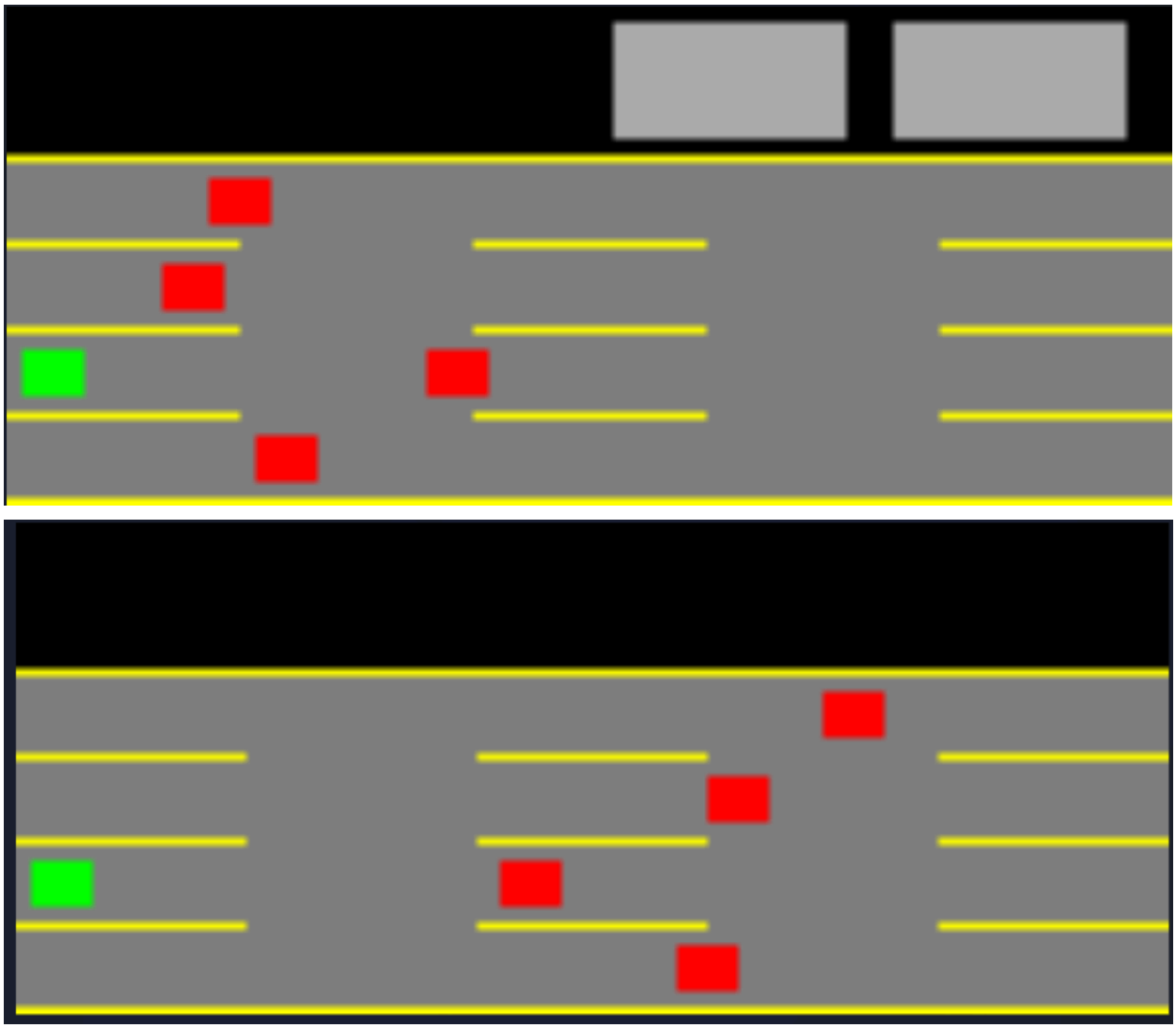




**Visualization of Speed Value**

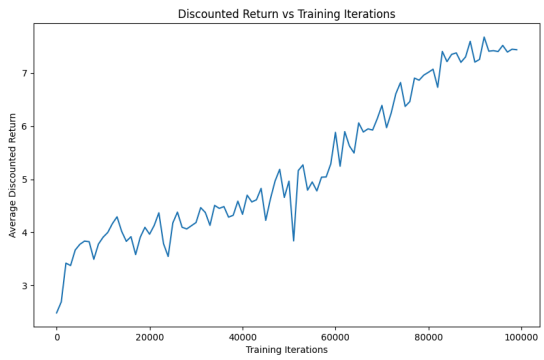
Visualization of speed values for each lane was performed, revealing that if a particular speed is unfavorable, it is shaded from gray to white based on its severity, whereas a favorable speed is marked black. Thus, if high speed is undesirable, it appears white, and vice versa.





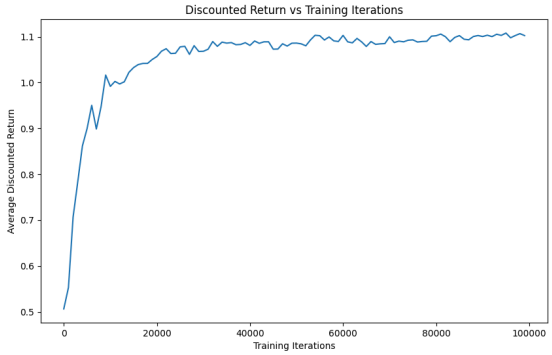
## b) Variation of Gama

By varying the discount factor gama these are the plots obtained.Model has been trained on gama=0.99, gama=0.9 and gama=0.8.



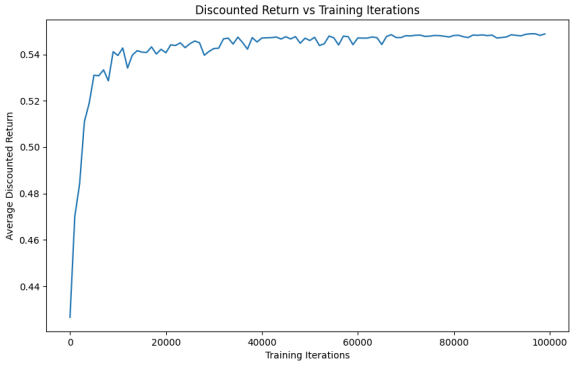


**Plots of return and average maximum distance with gama=0.99**





**Plots of return and average maximum distance with gama=0.9**





**Plots of return and average maximum distance with gama=0.8**

**Observations:**

Distance travelled for different discount factors

| Gamma | Iteration 10000 | Iteration 50000 | Iteration 100000 |
|---|---|---|---|
| 0.99 | 330.4581 | 412.6359 | 556.6542 |
| 0.9 | 103.1595 | 368.4735 | 508.3971 |
| 0.8 | 111.5334 | 313.9518 | 486.1536 |

Average Rewards for different discount factors

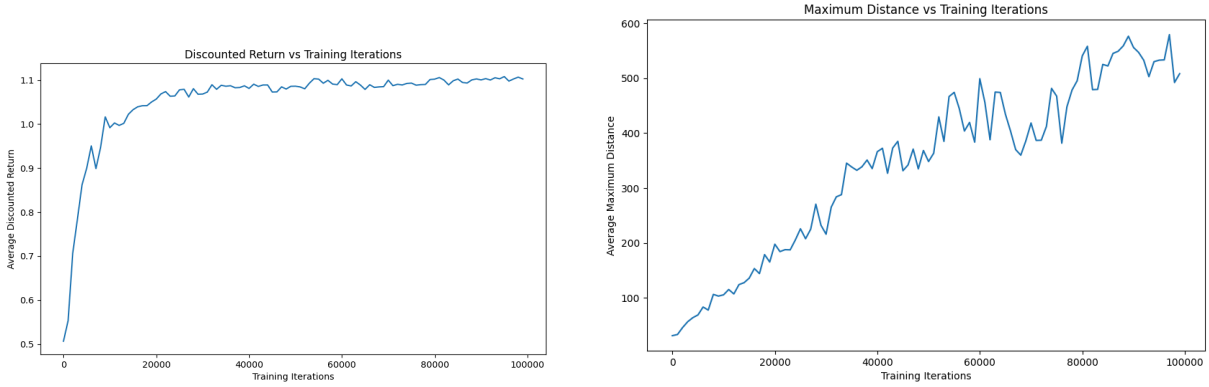
| Discount factor | 0.99 | 0.9 | 0.8 |
|---|---|---|---|
| Reward | 7.43 | 1.1 | 0.55 |

From the graphs of three different discount factor values(0.99,0.9,0.8), the plots show a similar trend of increasing average reward and average distance travelled.

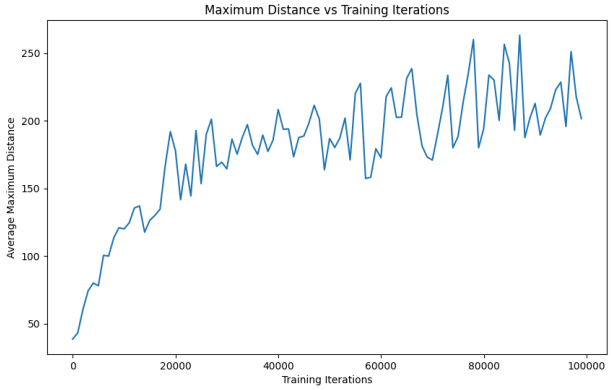
- The average reward and average distance travelled is higher for discount factor 0.99, when compared to 0.9 and 0.8.
- This indicates that the agent is trying to get immediate rewards over long term rewards as we decrease the discount factor.
- Concentrating on immediate rewards makes the agent take risks(overtake cars faster), when the other cars are only changing lanes rapidly then there is a chance of collision giving a negative reward(-5). So the average reward decreases as we decrease the discount factor.

## c) Variation of Alpha

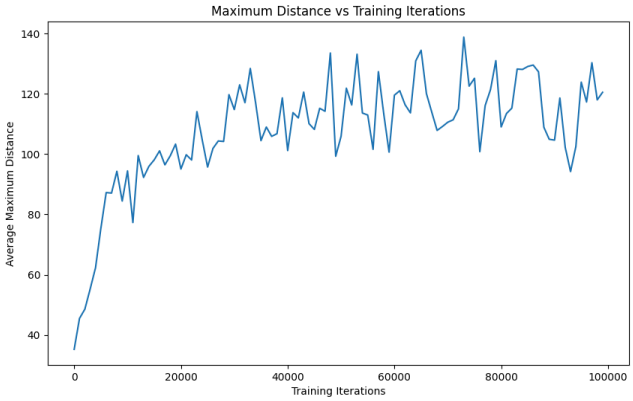
By varying the discount factor gama these are the plots obtained.Model has been trained on alpha=0.3 and alpha=0.5.




**Plots of return and average maximum distance with alpha=0.1**


**Plots of return and average maximum distance with alpha=0.3**


**Plots of return and average maximum distance with alpha=0.5**

# Observations on the Effect of alpha in Tabular Q-learning

Average distance travelled for different alpha values

| alpha | Iteration 10000 | Iteration 50000 | Iteration 100000 |
|---|---|---|---|
| 0.1 | 103.1595 | 368.4735 | 508.3971 |
| 0.3 | 120.9717 | 163.8021 | 201.5418 |
| 0.5 | 94.4256 | 133.5609 | 120.5250 |

- alpha=0.1: Best overall performance, with distances increasing steadily. The small learning rate ensures stable, gradual updates, leading to a near-optimal policy.
- alpha=0.3 : Strong early performance but it does not increase rapidly, indicating faster initial learning but potential overfitting or premature convergence.

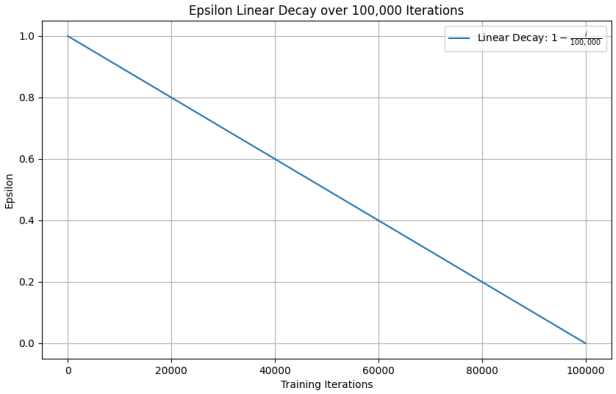
- alpha=0.5: Worst performance. High alpha causes instability, leading to poor convergence.

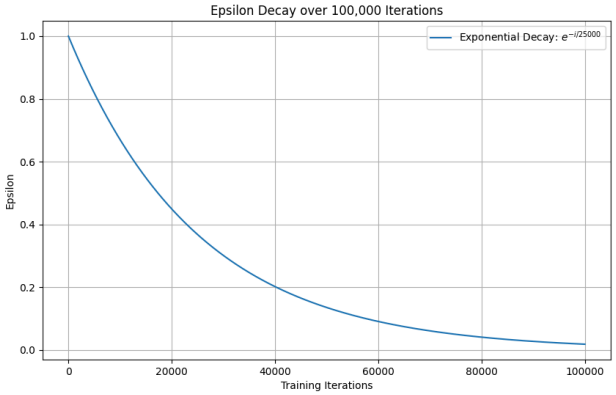
**Q(s,a) = Q(s,a)+ alpha * TD_error**

The reason for this trend is that the alpha value indicates the trust that we believe in the TD error update. Since we know the TD error can fluctuate a lot, then these fluctuations also make the updates really fast. That is the reason we can observe a lot of fluctuations in the distance graphs for alpha = 0.5, and alpha = 0.3 when compared with alpha = 0.1.

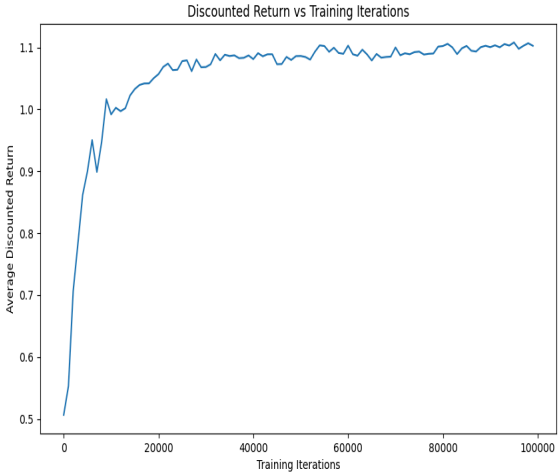
## d)Variation of Epsilon

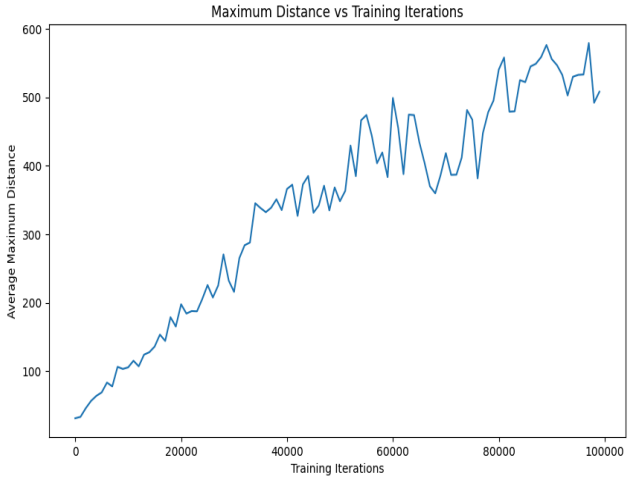
By varying the discount factor gama these are the plots obtained.Model has been trained with constant ,Linear decay and exponential decay of epsilon value.



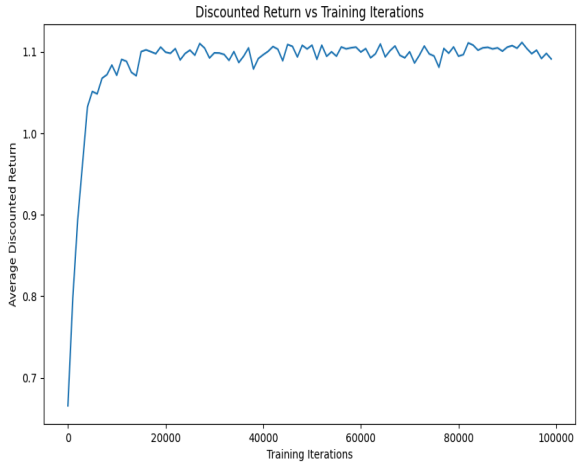

**Epsilon=0.75**

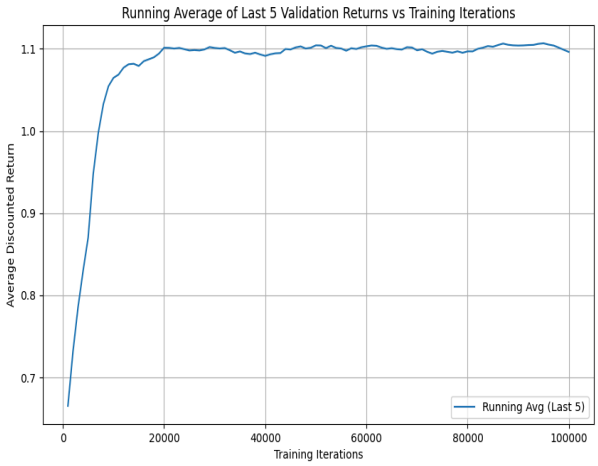



**Plots of return and average maximum distance for epsilon=0.75**

## Epsilon=0.25

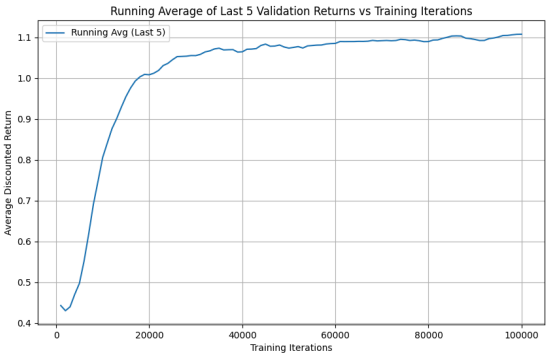


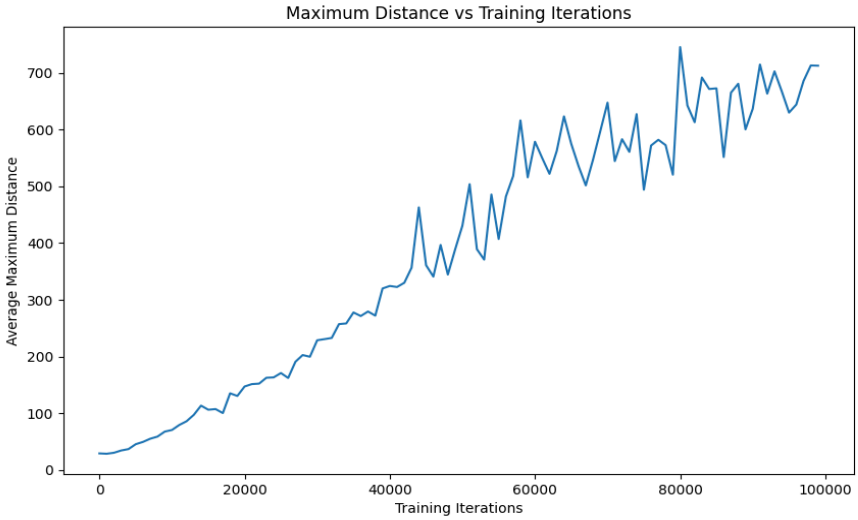

**Plots of return and running average return with epsilon=0.25**


**Plot of average maximum distance**

## Epsilon=1 and Linear Decay

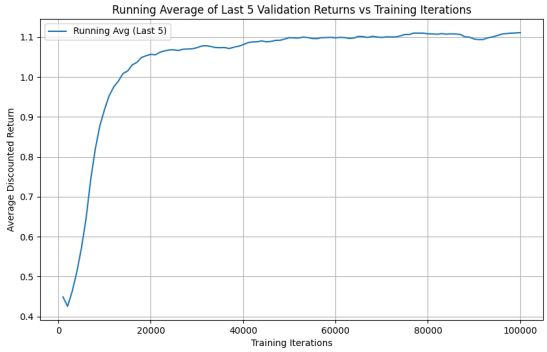



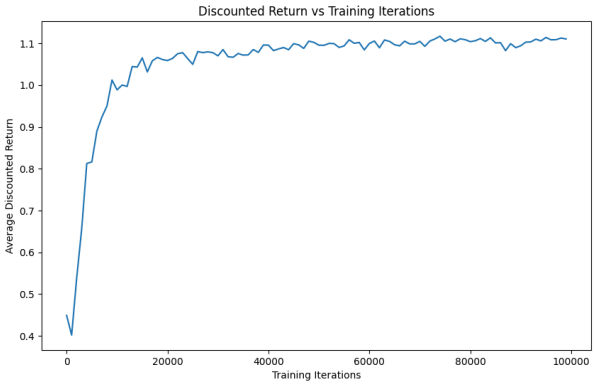
## Plots of return and running average return with linear epsilon decay




## Plot of average maximum distance

## Epsilon=1 and Exponential Decay




## Plots of return and running average return with exponential epsilon decay


**Plot of average maximum distance**

## Observations on the Effect of epsilon in Tabular Q-learning

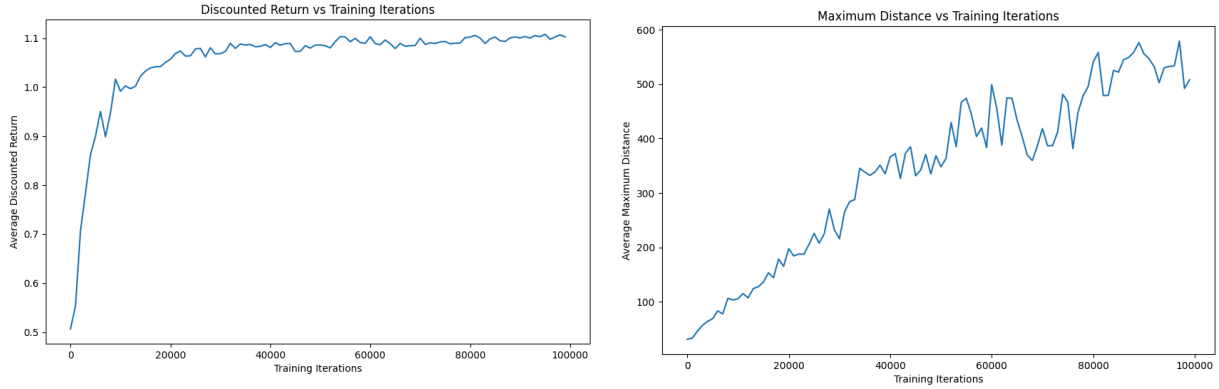
Average distance travelled for various exploration strategies

| Epsilon | Iteration 10000 | Iteration 50000 | Iteration 100000 |
|---|---|---|---|
| 0.25 | 208.6539 | 208.6539 | 520 |
| 0.75 | 103.1595 | 368.4735 | 508.3971 |
| Start with 1 and Linear decay(1-i/iterations) | 67.5165 | 388.7247 | 703.1025 |
| Start with 1 and Exp decay (exp{-i/(iterations/4)} | 110.0682 | 549.4656 | 621.2877 |

- **epsilon=0.25 :** Low exploration limits early learning. But by 100000 iterations, the distance improves to 520, showing better exploitation of the learned policy.
- **epsilon=0.75 :** Higher exploration leads to slower initial progress , but steady improvement, though persistent exploration may cap performance.
- **Linear Decay** : Starts with high exploration, improves significantly and achieves the highest distance (703.10 at 100,000) by balancing exploration and exploitation.
- **Exponential Decay**: Moderate early exploration, strong mid-term gains, and high final distance (621.29 at 100,000), benefiting from faster decay to exploitation.

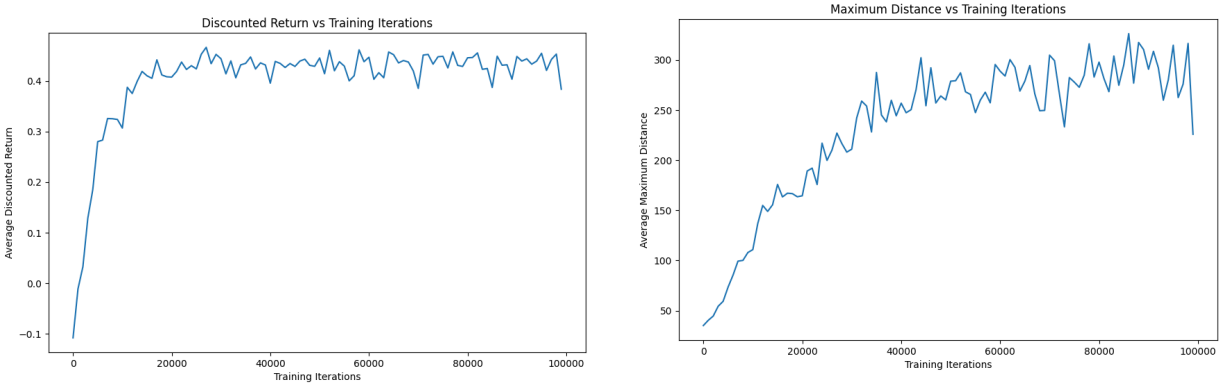
- **Trend**: Decay strategies outperform fixed epsilon, with linear decay achieving the best result (703.10) by gradually shifting to exploitation. Fixed epsilon=0.25 limits early learning, while epsilon=0.75 may over-explore.

## e) Variation of  Reward model and distance quantization

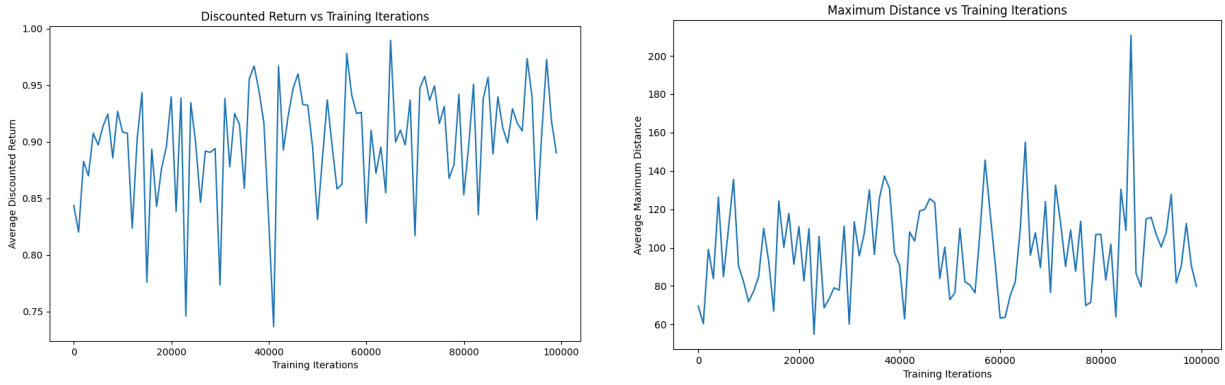



**Plots for maximum distance as reward and distance quantization =5 values(Base Model)**

By varying the reward model these are the plots obtained.




**Plots for number of overtakes as a reward model**




**Plots for distance quantization = 3 values**

**Observations**

| alpha | Iteration 10000 | Iteration 50000 | Iteration 100000 |
|---|---|---|---|
| Base | 103.1595 | 368.4735 | 508.3971 |
| Overtakes | 107.9958 | 260.2233 | 225.9339 |
| Obs = 3 | 82.2411 | 100.2546 | 79.7757 |

**Reward Type**

- As you can see in the table, when we change the reward to the number of cars overtaken from the total distance traveled, the performance is degraded.
- It is because in order to maximize the reward in overtaking the other cars, this can be dangerous as the other cars can change position while you are overtaking the car, then collide with them, this increases the probability of collision.
- When the reward type is distance travelled, in order to maximize the reward, it has to survive and move faster, survival becomes priority in the long term, so it takes less risks(overtakes).

**Distance quantization (5 and 3)**

- When we decrease the observation state discretization number to 3 from 5, we can see that the performance was really bad.
- This is simply because it thinks that it is safe to move forward when the others are located in the 3rd discrete state, and moves to the end of the 2nd state discrete state. Now when it wants to maneuver to other states, it has very far less time to react.
- In most cases it won't get enough reaction time to take action and eventually collides. Leading to negative reward, and decreasing performance.
- As you can see it the plots the Q values were not converging to a fixed values, it rapidly changing because of the above reason

# Part 2: DQN Learning

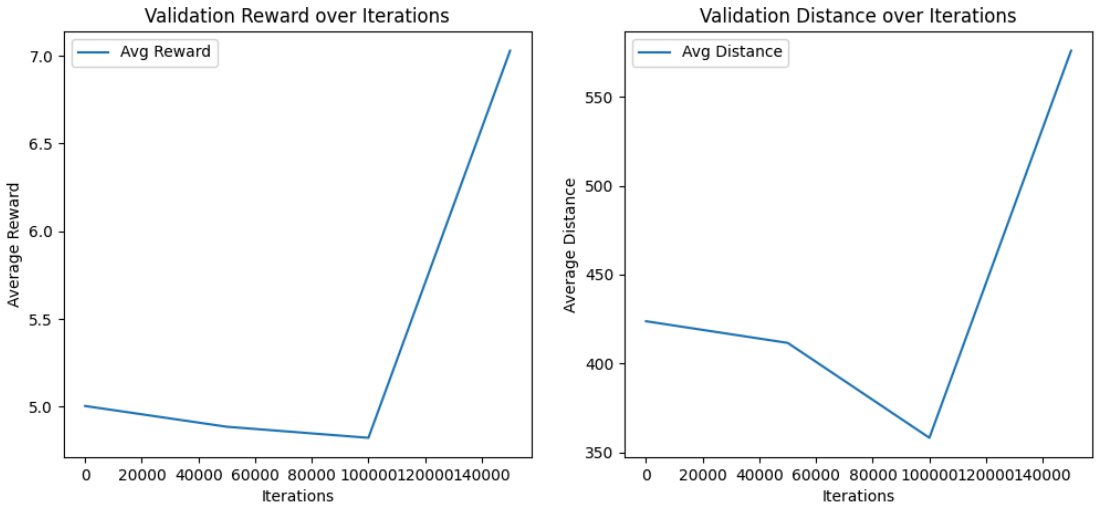
**Normal DQN:**

- First we implement a basic single DQN network, for each episode we collected the experience, calculated TD error, and did back propagation to adjust weights.
- Later we realised that, in normal supervised learning the labels are fixed, and we try to optimise our predictions based on the labels.
- In RL the labels, and predictions are from the same network, if both are varying continuously how will the DQN model converge to a particular solution.

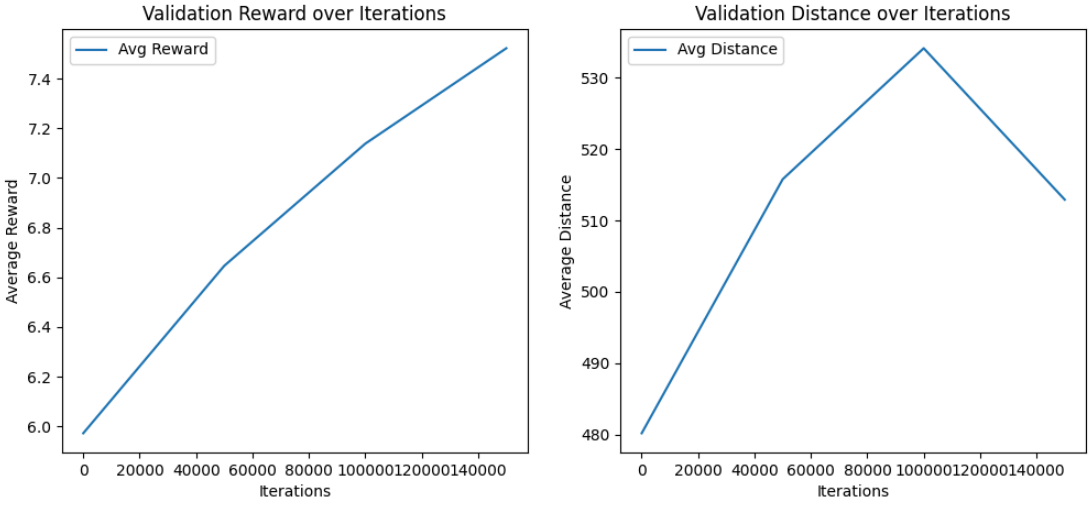
**Two DQN(target_model, predition_model):**

- Here we maintain two networks, target network and prediction network. Unlike the single DQN we fix the target model for some time, adjust the weights of the prediction model according to the target model, and later update the target model.
- By doing this we can stabilize the model, it eventually converges to an optimal solution.
- Experience replay buffer is used to store TD(0) trajectories, this breaks the correlation between trajectory(episode) points.

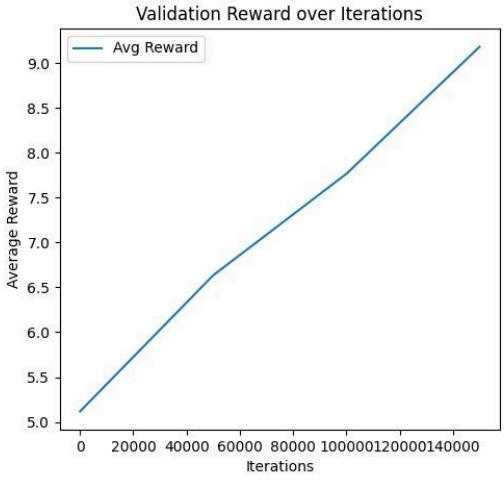
**Discrete states Input and discrete actions DQN:**

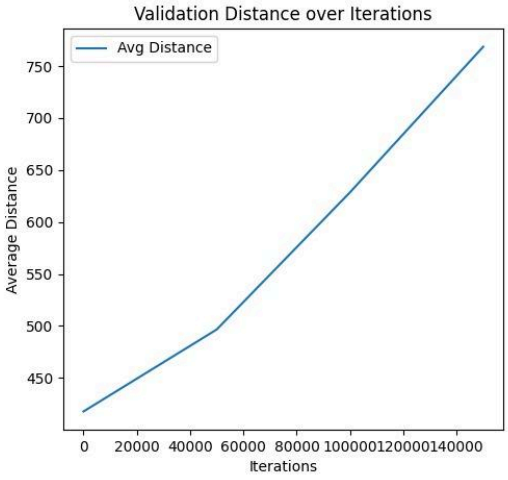



**Continuous states input and discrete action DQN**:

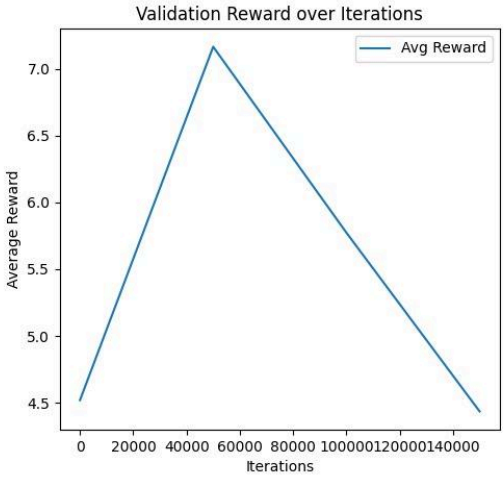


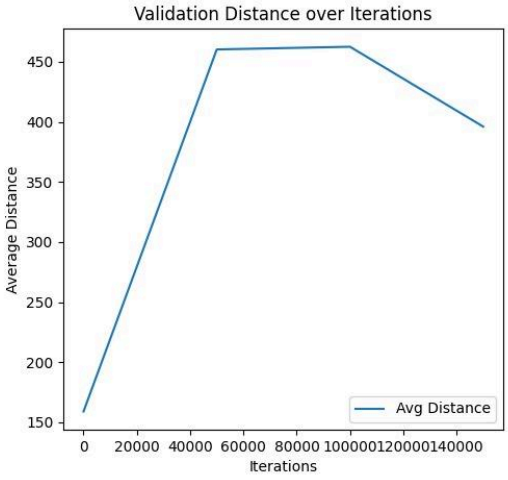
## Every step discrete



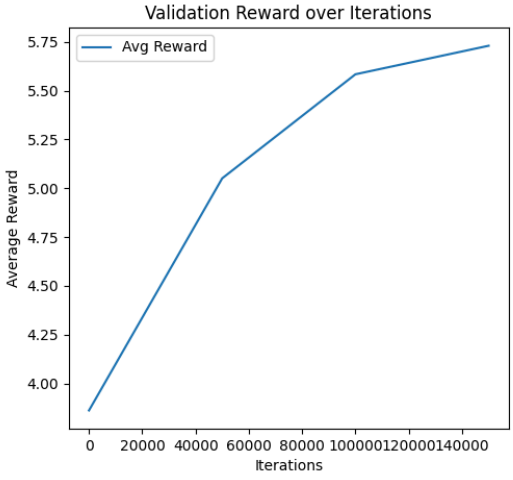



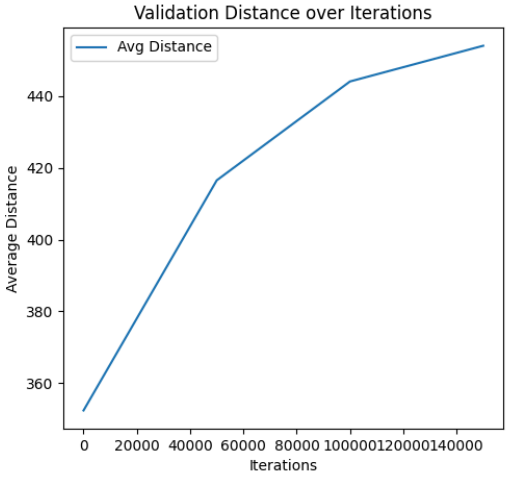

## Every step Continuous




## Terminal update



**Observations:**

Experience replay algorithm modifications

| DQN Learning | Every step | Every 20 steps | At the end of the episode |
|---|---|---|---|
| Average Distance travelled | 768 | 576 | 450 |

- According to the original algorithm: we have to sample experiences from the replay buffer every step in a trajectory and train the model until it terminates and repeat it for the 200,000 episodes. But it was taking a lot of time to train, roughly around 10 hours.
- We didn't go with this algorithm, we modified the algorithm by we train the model for every 20 steps in a episode(sample and train), it was taking 2 hours 35 minutes
- We even reduced it to update at the end of the trajectory, it took 2 hours.
- As we can see the performance of the algorithm decreases as we don't learn frequently, this is because we might miss the history as it will get deleted if you wait till the end of the episode. It's a wasted effort in storing the experiences and deleting it without even using it once.

By comparing the results with continuous cases, we can say that the direct implementation of DQN may not yield good results.

- For the discrete case there might be a huge statesace, in this case it is around 10,000. But with exploration we might visit a large portion of state space, and approximate the Q values for the states it never visited by the learned network.
- In the continuous state space is very large(infinite possibilities), but we have just learned about a few states. The one major advantage is that in discrete cases we put hard constraints on the states, while in continuous cases there are no such constraints. So it can generalize Q values better for the states it has not visited.
- Even a small change in state values can cause a huge Q value update, which might be erroneous.
- Here the actions are discrete, the DQN can't be implemented for continuous action space, since we have to take argmax of outputs, so the outputs must be infinite as actions are infinite. Which is impartial to implement.

Conclusion:

- DQN really works well for discrete state space and discrete action space, but can't be used for continuous action space.
- We can Implement DQN for continuous states and discrete actions, achieve good results but it requires tuning the parameters and improving algorithms.

## Part 3: Improving DQN Learning

**Improvement Ideas:**

**Eligibility traces:**

- Initialize eligibility trace values for each state and action and update the value when state and action is revisited by 1, or else decay the rate.
- By doing this we prioritize the most recent and most frequent ones over others, multiply the eligibility values to the loss, so that based on the values the gradient changes.
- 10,000x5 table update each step is very computationally expensive, it is taking a lot of time.

**TD(lamda):**

- Instead of just taking one step, take the N step return, and add (present state, action, Lambda return,nthstate) to the buffer and do experience replay.

**Prioritised replay buffer:**

- What happens with the experience replay buffer is that there are many points which have no useful information to extract, for example there might be many zero reward state transitions.
- If you sample randomly then there might be cases where you might sample many un useful points from the buffer. So to avoid this we add a priority list based on the TD error.
- The ones with maximum TD error either positive or negative have more probability of getting sampled. P_j = TD_error

$$\text{Sample transition } j \sim P(j) = p_j^\alpha / \textstyle\sum_i p_i^\alpha$$
$$\text{Compute importance-sampling weight } w_i = (N \cdot P(j))^{-\beta} / \max_i w_i$$

**Double DQN:**

- In the DQN replay buffer, we have two networks one is target and one is for training policy. In the case of DQN, we take a maximum of Q_hat(next_state, a, theta_hat), but it is too optimistic, maybe it might be worse, and we move the gradients in that direction by blindly overestimating the targets.

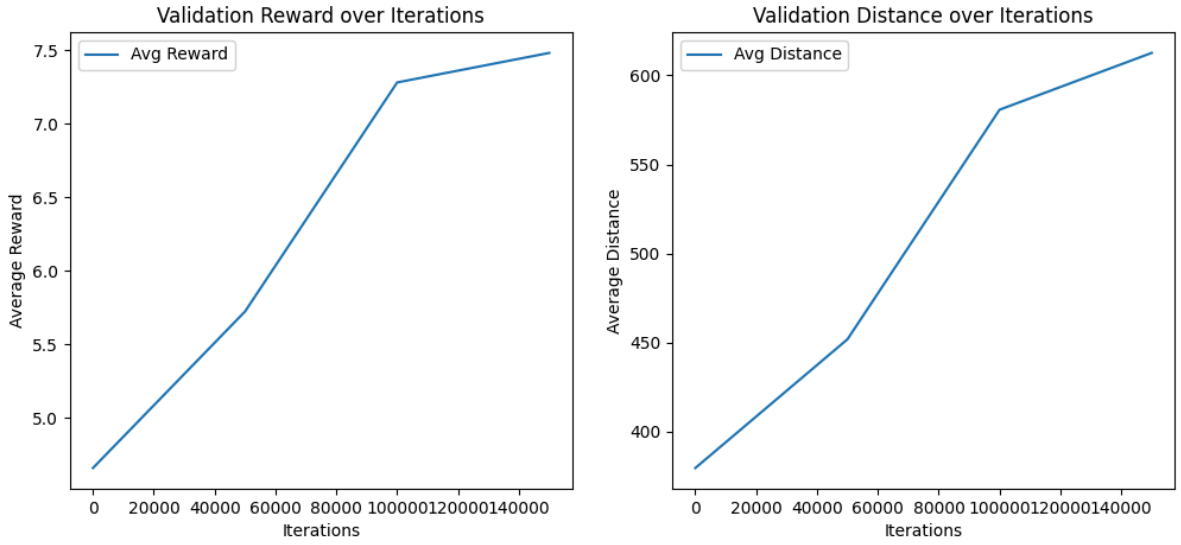
$$Q_{target} = Q_{\hat{\theta}}(s', a_{pred}), \quad where \quad a_{pred} = \arg\max_a Q_\theta(s', a)$$

$$\text{TD-error } \delta_j = R_j + \gamma_j Q_{\text{target}}\left(S_j, \arg\max_a Q(S_j, a)\right) - Q(S_{j-1}, A_{j-1})$$

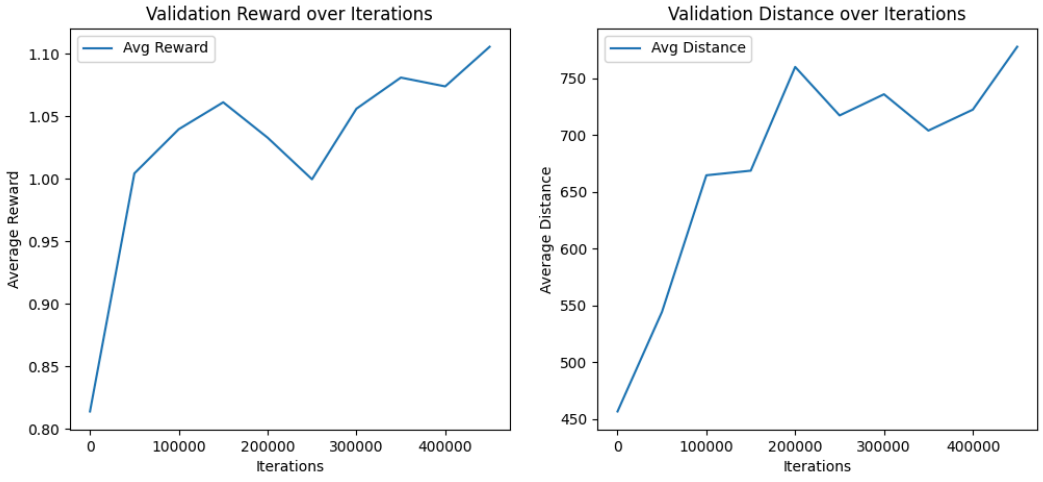
r

**Final Algorithm: Double DQN with prioritised replay**

- By combining experience replay and DDQN we can achieve good performance.




For DDQN priority experience replay 200000 iterations(gamma = 0.99)




For DDQN priority experience replay 500000 iterations(gamma = 0.9)

**Observations:**

| Algorithm | Average distance travelled |
|---|---|
| DQN with experience replay(20 steps) | 576 |
| DDQN with priority experience replay(20 steps)-200000, df = 0.99 | 612 |
| DDQN with priority experience replay(20 steps)-500000, df = 0.9 | 777 |

- Here when we keep all the other parameters same and just change the algorithm, we can see that the agent travelled a large distance 612