[COL778] Assignment 2: Q-Learning

Abhinav Khanna - 2022CS11964, Hemanshu Garg - 2022CS11090

The GIFs and visualizations for parts I and II are provided `here`

Part I: Tabular Q-Learning

a. Training the tabular Q-learning Agent

$\alpha = 0.1$, $\epsilon = 0.75$, $\gamma = 0.9$, 100000 iterations.

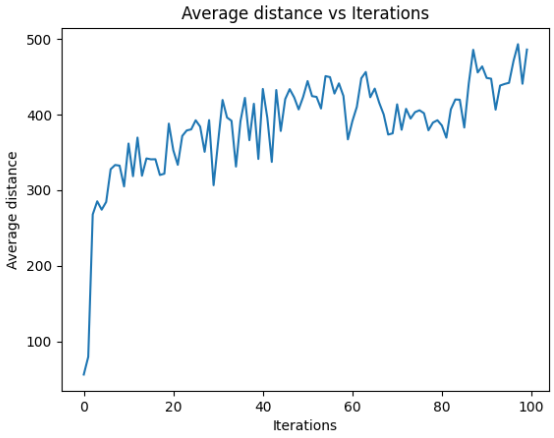

Figure 1: Avg Distance over 1000 runs vs every 1000th iterations

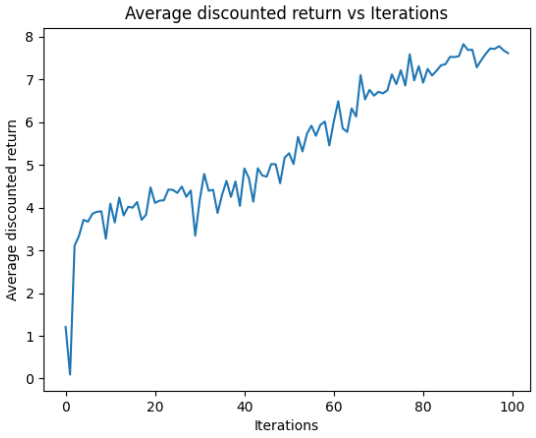

Figure 2: Avg Rewards over 1000 runs vs every 1000th iterations

Observations:

- The average rewards and distances travelled increases with the number of iterations.
- The agent learns a policy for navigating the road and maximizing the reward.
- It first seemed to learn about the strategy of moving at constant speed of 1, which enables it to never collide with any car however as the number of steps by the environment is finite it realizes increasing speed can lead to better rewards.
- It further then adapts to the fact of how to maintain speed as high as possible without colliding.
- From the Gifs it seemed to learn the policy that **when the other cars are far or the nearby lane is empty, maintain the max speed else slow down and wait for sufficient gap to form between cars so as to overtake from there.**
- This rewards and distance travelled hasn't actually platued (verfied by running it for 1 million iterations and getting about 700-800 units of distance covered)

Value for staying in a lane

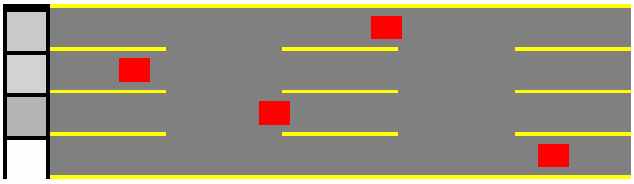

Figure 3: Lane Value

Observations:

- We see that the model learns to not stay in the first three lanes as the cars are closer and the top priority is the bottom most lane where the car is the furthest away, as is expected.

Value for maintaining a speed

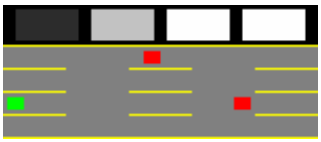

Figure 4: Speed Value Fast

Figure 5: Speed Value Slow

Observations:

- We see that the model learns to go fast when the cars are far away (it highly prioritizes this) to maximize reward and when the cars are nearby it relies on going at lower speeds to avoid collisions (here it would be after a lane change by a red car).

b. Varying the discount factor

Avg Distance and Reward over 1000 runs vs every 1000th iterations

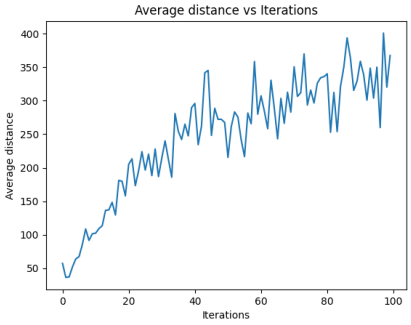

Figure 6: Discount Factor = 0.8

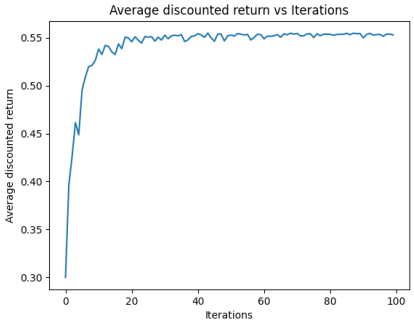

Figure 7: Discount Factor = 0.8

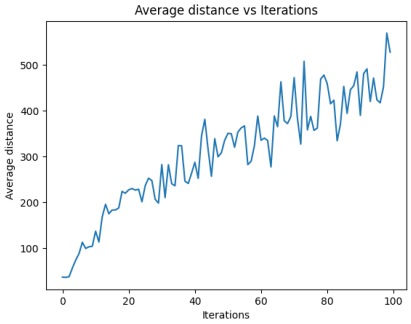

Figure 8: Discount Factor = 0.9

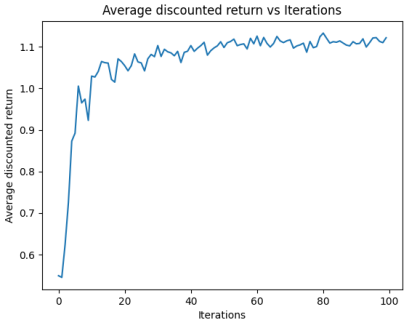

Figure 9: Discount Factor = 0.9

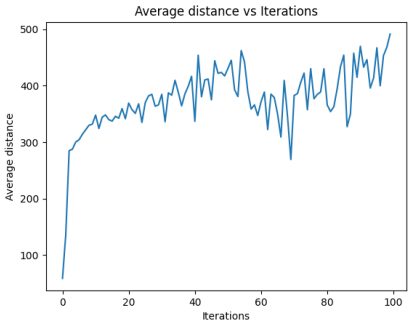

Figure 10: Discount Factor = 0.99

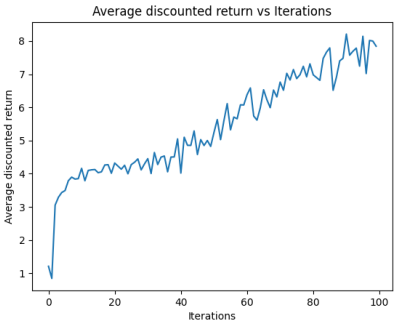

Figure 11: Discount Factor = 0.99

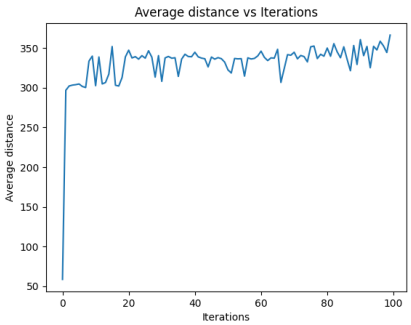

Figure 12: Discount Factor = 0.999

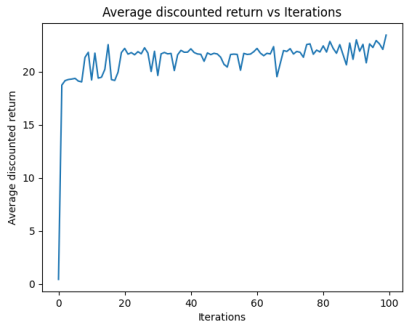

Figure 13: Discount Factor = 0.999

Observations:

- Higher reward isnt necesserily more distance as evident in df=0.999. This is because of the fact higher discount factor would simply collect more reward out of all the states it has seen if even if they are suboptimal.
- Having discount factor too high causes stability issues it seems where rewards for any episode is high enough that either it becomes hard to improve upon this local policy significantly or also the fact that at higher discount also can be interpreted how optimisitic is agent about getting that future reward, because there is inherent stochasticity wether our action is executed in practice this can also make it harder distinguish between episodes at such high rewards values.
- Too low of a discount factor is not a good idea because it can make the agent "myopic" or short-sighted about the rewards. Therefore it is unable to learn long term strategies. Evident as here df = 0.8 agent strategy seems to be constant speed of 1 (sometimes 2) and its rewards already saturated.

c. Varying the learning rate

Avg Distance and Reward over 1000 runs vs every 1000th iterations

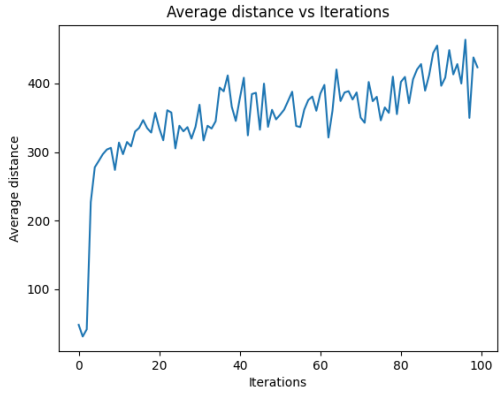

Figure 14: Learning Rate = 0.05 (Distance)

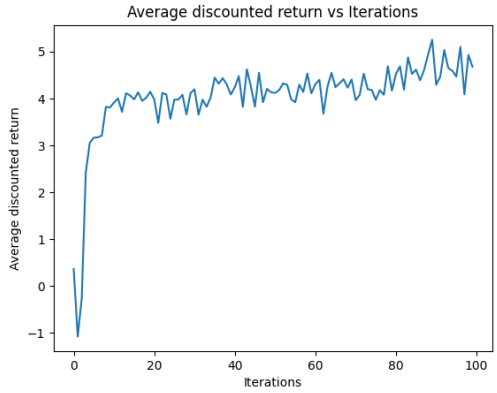

Figure 15: Learning Rate = 0.05 (Reward)

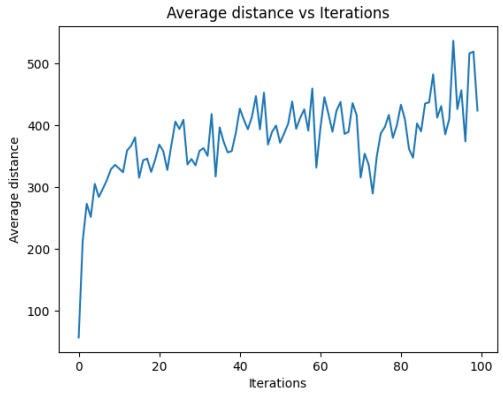

Figure 16: Learning Rate = 0.1 (Distance)

Figure 17: Learning Rate = 0.1 (Reward)

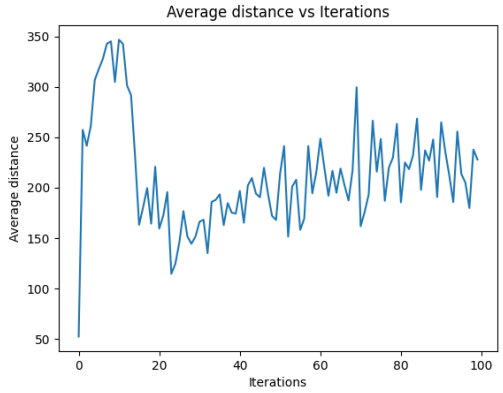

Figure 18: Learning Rate = 0.3 (Distance)

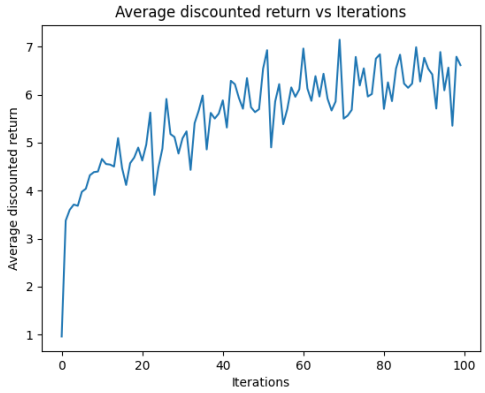

Figure 19: Learning Rate = 0.3 (Reward)

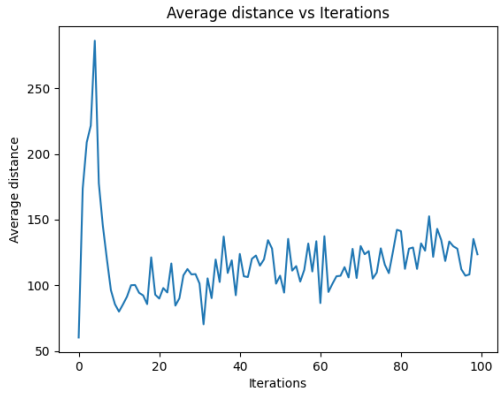

Figure 20: Learning Rate = 0.5 (Distance)

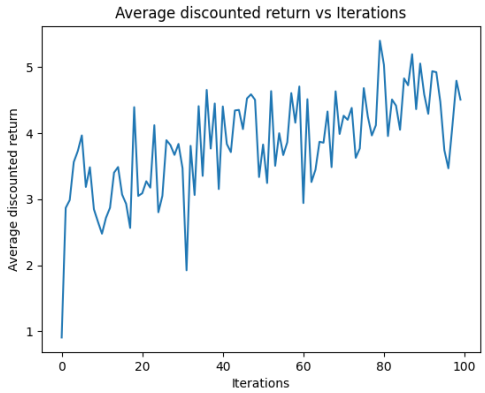

Figure 21: Learning Rate = 0.5 (Reward)

- The learning rate or step size determines to what extent newly acquired information overrides old information. A learning rate of 1 would imply we completely remove the old information, this is could be disastrous in a stochastic environment as it may never execute that new information.
- Too high of learning rate and we see in the graphs the average distance in fact being less.
- This is inherently because our environment is stochastic, as we get close enough to convergence, a stochastic environment would make it impossible to converge if the learning rate is too high. Also the Q-values at higher learning rates could jump around erratically.
- Too low of learning rate and we risk getting stuck in a local minima.

d. Varying the exploration strategy

Avg Distance and Reward over 1000 runs vs every 1000th iterations. Epsilon varies at every 1000 iterations.

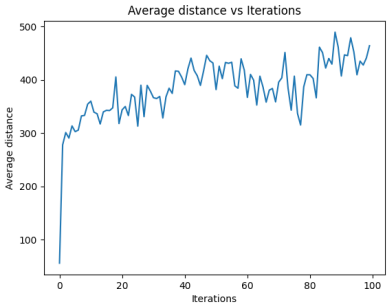

Figure 22: Constant ε

Figure 23: Constant ε

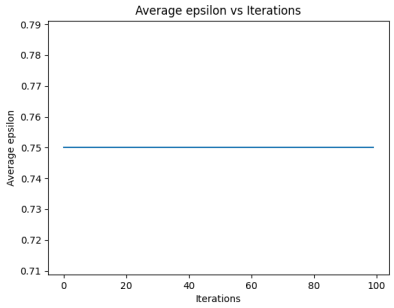

Figure 24: Constant ε

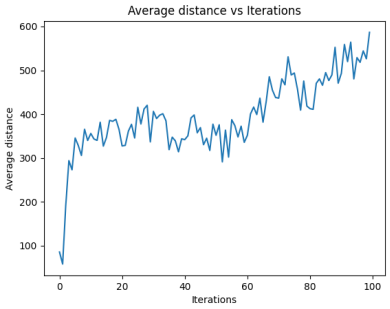

Figure 25: Linearly decreasing ε with rate 0.007

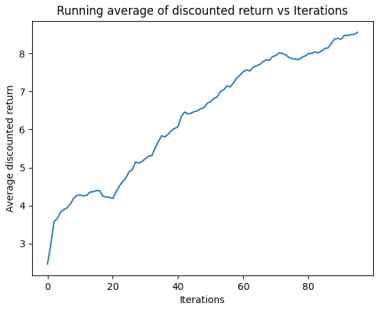

Figure 26: Linearly decreasing ε with rate 0.007

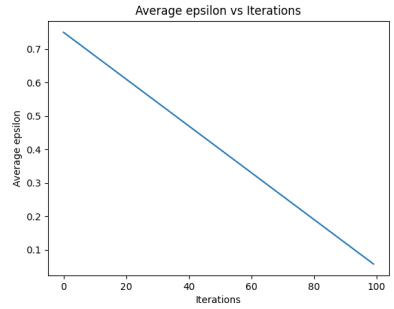

Figure 27: Linearly decreasing ε with rate 0.007

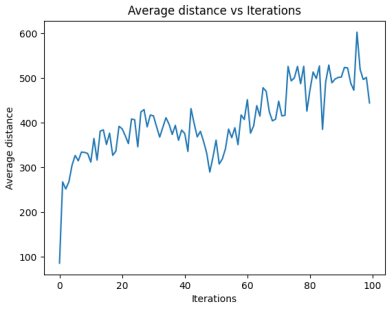

Figure 28: Exponentially decreasing ε with rate 0.99

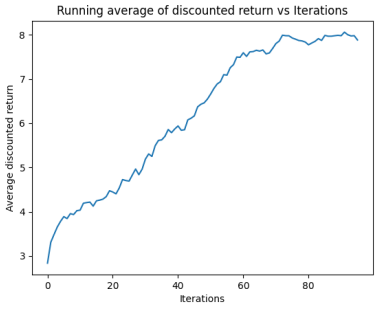

Figure 29: Exponentially decreasing ε with rate 0.99

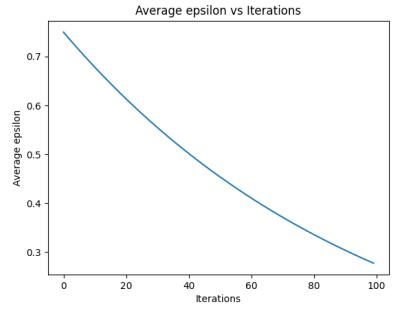

Figure 30: Exponentially decreasing ε with rate 0.99

Observations:

- Epsilon marks the trade-off between exploration and exploitation. At the beginning, we want epsilon to be high so that you take big leaps and learn things. As we learn about future rewards, epsilon should decay so that we can exploit the higher Q-values we've found.
- We can observe from the figures above decaying epsilon indeed improves model's exploitation at higher rewards and leads to higher average distance values and moving average return.

e. Varying reward model and quantization level of the State

(a) Changing Reward Model

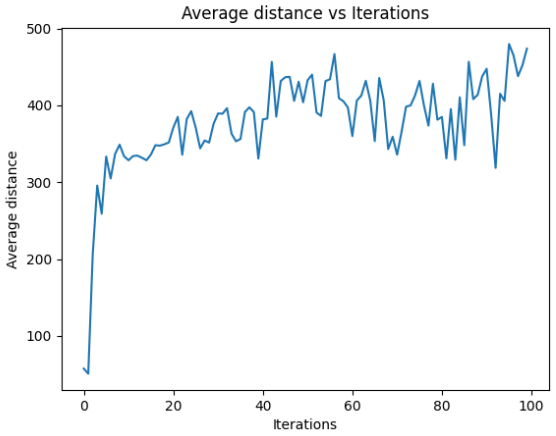

Figure 31: Reward Model = Distance travelled

Figure 32: Reward Model = Distance travelled

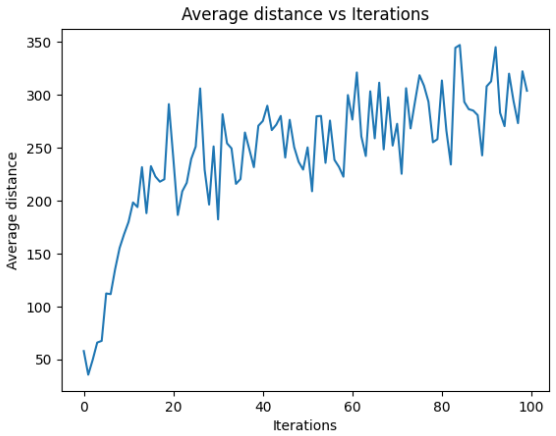

Figure 33: Reward Model = Number of overtakes

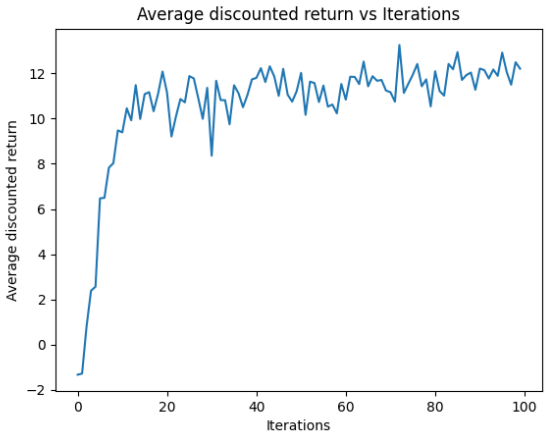

Figure 34: Reward Model = Number of overtakes

Observations:

- We see that for average distance travelled the 'dist' model seems to perform better.
- The amount of reward collected is higher for the 'overtakes' model in comparision.
- This can be explained because in the second model the agent prioritizes effective overtaking, and we know the speeds of all cars is either 1 or 2, therfore the second model quickly can figure out a strategy which has higher chance of getting stuck in some local minima by just having enough speed to reach a car and mostly using its action to change lane and not increase speed (and we have finite number of actions we can take overall as restricted by the environment, therefore for a lot of steps car remains in lower levels of speed).

(b) Changing Quantization level of State

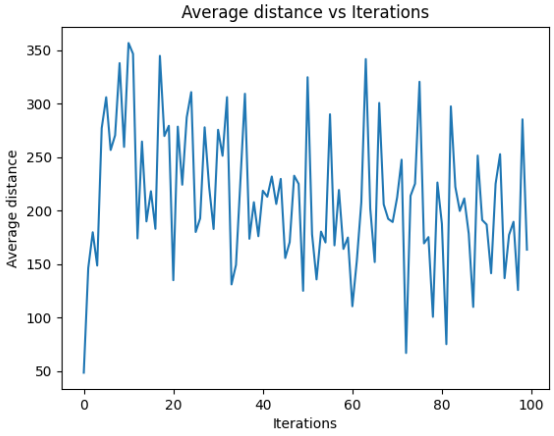

Figure 35: Quantization level : 0 to 2

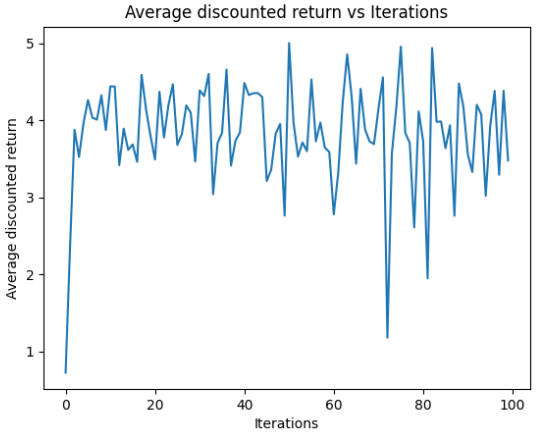

Figure 36: Quantization level : 0 to 2

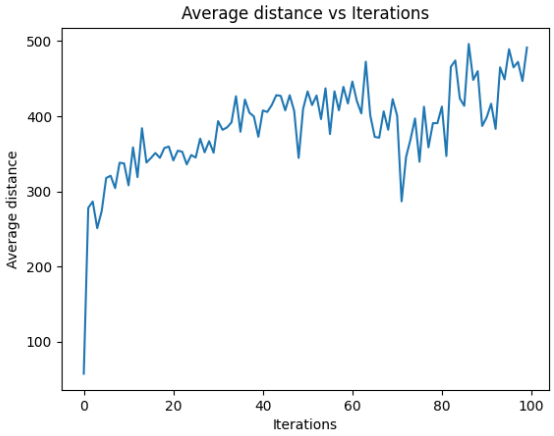

Figure 37: Quantization level : 0 to 4

Figure 38: Quantization level : 0 to 4

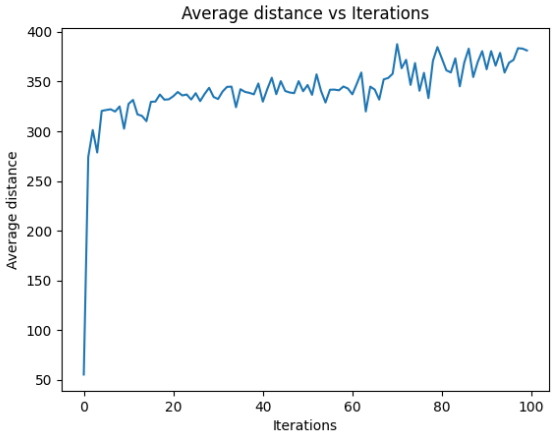

Figure 39: Quantization level : 0 to 6

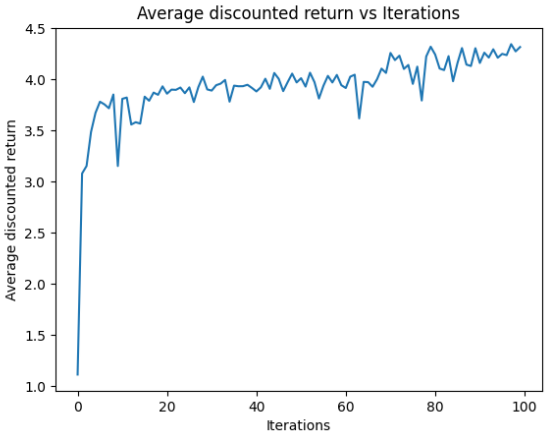

Figure 40: Quantization level : 0 to 6

- We are able to observe effect of granularity of state space from the above graphs.
- Too Coarse (0-2): State representation is inadequate; critical information is lost, preventing effective learning.

- Too Fine (0-6): State space becomes too large; learning is slow and requires excessive data / time, leading to a worse distance traveled, although it also has the best distance/(reward collected) ratio.
- Just Right (0-4): Represents a sweet spot, capturing enough detail for good performance without making the state space unmanageably large for tabular Q-learning within the given iterations

Part II: Deep Q-Learning

a. Training the DQN-learning Agent

The following hyperparameters were used as the default, lr = 0.0001, replay buffer size = 100,000, discount factor = 0.9 and $\epsilon = 0.75$ (constant). (Scale: X-axis, 1 unit = 1,000 iterations)

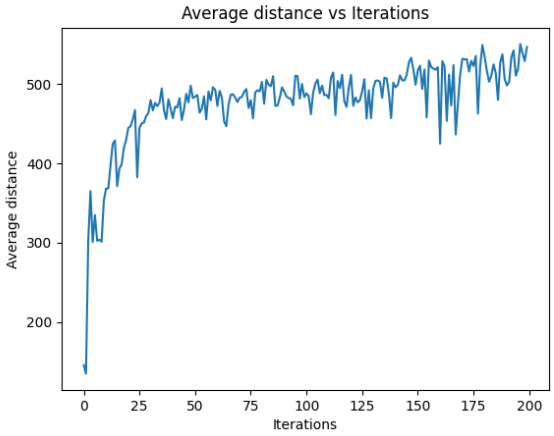

Figure 41: Avg Distance over 1000 runs vs every 1000 iterations

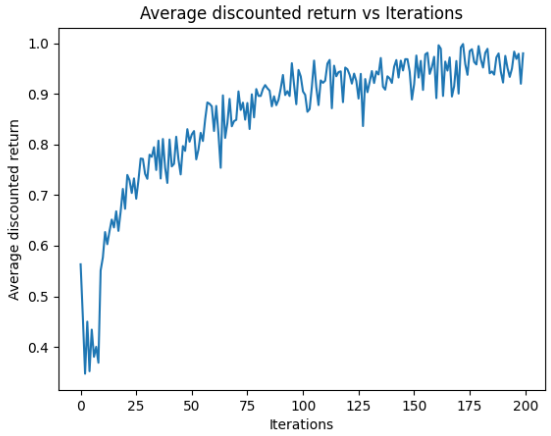

Figure 42: Avg Rewards over 1000 runs vs every 1000 iterations

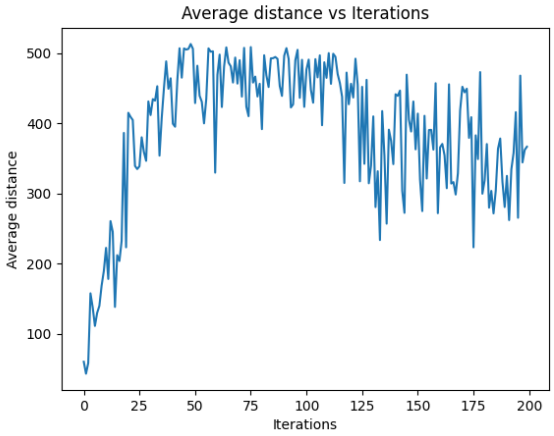

Figure 43: Exponentially decaying ϵ

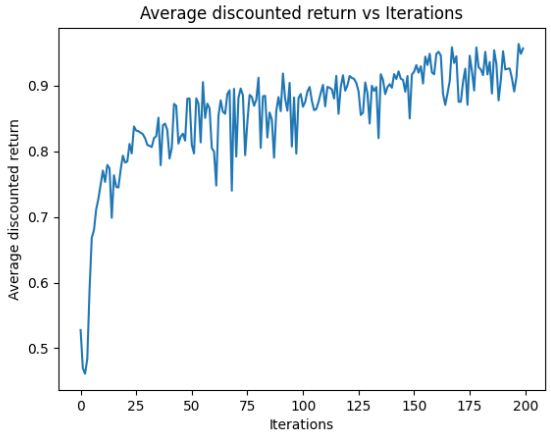

Figure 44: Exponentially decaying ϵ

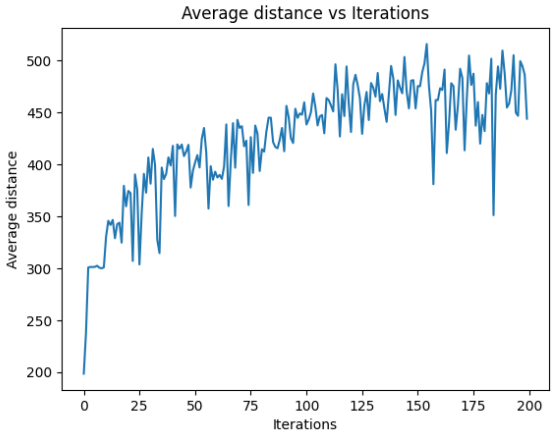

Figure 45: buffer size = 10,000

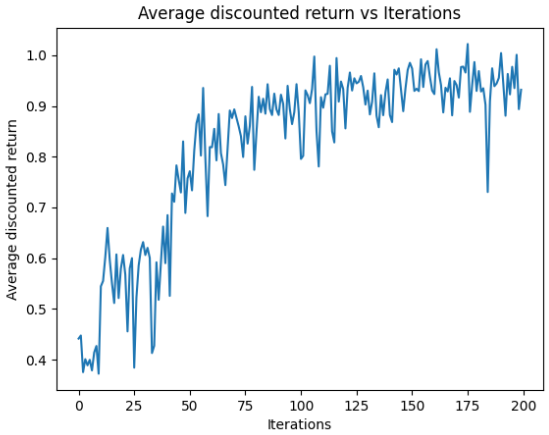

Figure 46: buffer size = 10,000

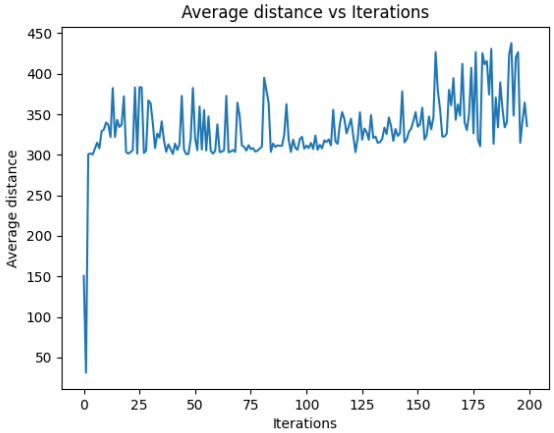

Figure 47: discount factor = 0.99

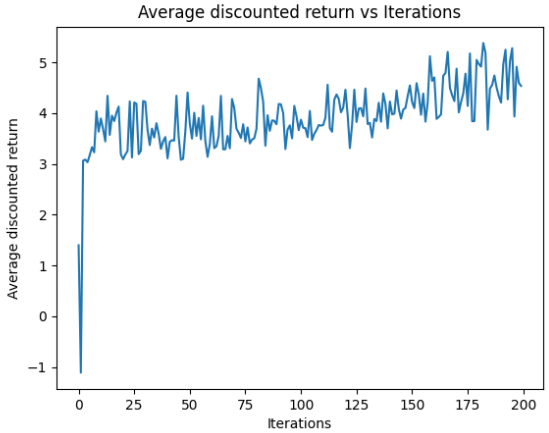

Figure 48: discount factor = 0.99

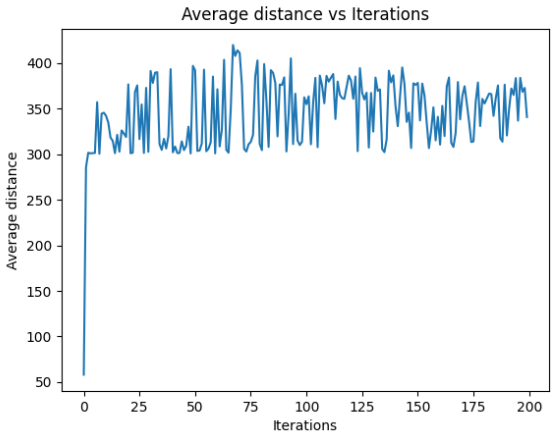

Figure 49: lr = 0.0003

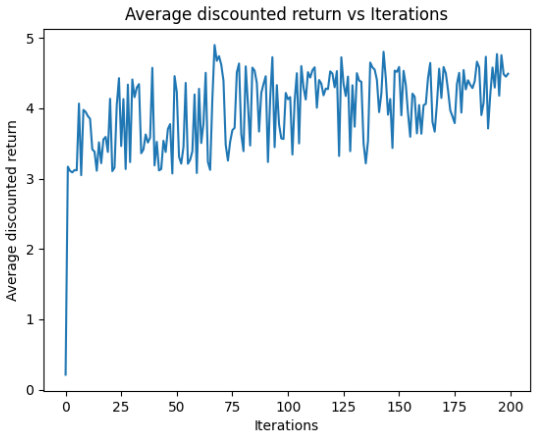

Figure 50: lr = 0.0003

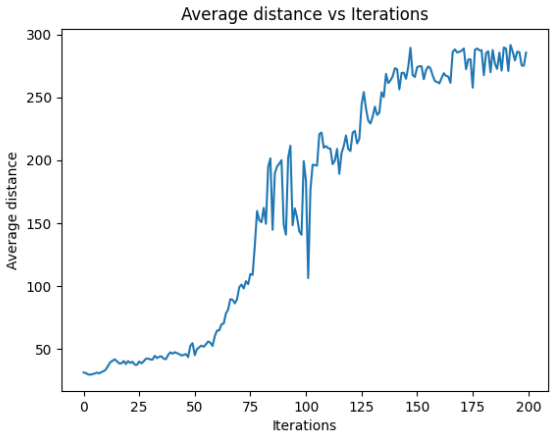

Figure 51: Using SGD

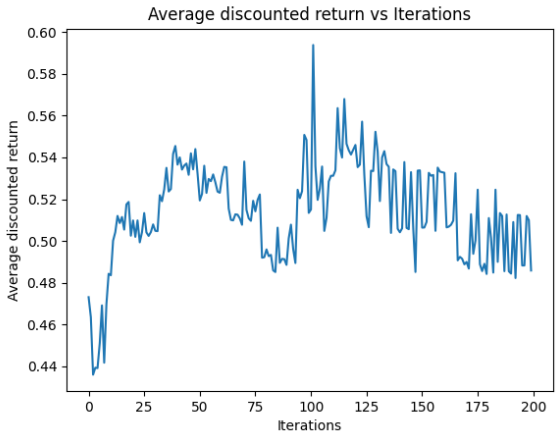

Figure 52: Using SGD

Observations:

- We see that decaying ϵ (by 0.99) leads to a decrease in the average distance traveled, likely due to not enough exploration occurring leading to the DQN getting fixed on a policy.
- On decreasing the buffer size we see that the average distance traveled and the reward obtained drops likely due to it forgetting the runs made initially as there are 200,000 iterations (each episode in turn produces 350-400 tuples) but only 10,000 spots in the replay buffer
- With a higher discount factor we see that the policy begins to saturate at around 300, this is expected as this can be obtained by just going at speed 1 (you'd never collide) and the negative reward of hitting a car would greatly affect runs with higher discount factors.
- With an increase in the learning rate we see that the average distance saturates very quickly at about 350, much lower than the distance for lr = 0.0001.
- On using SGD (with momentum = 0.5) we see that the learning happens very slowly and around 200,000 iterations it begins to reach a distance of 300 showing that the machine learning aspect (Adam optimizer) of the algorithm has a significant impact.

Value for staying in a lane

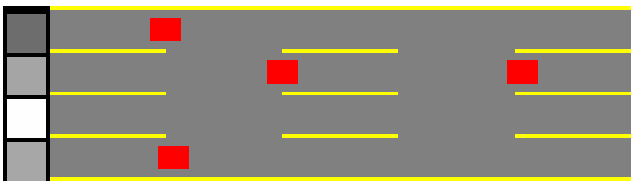

Figure 53: Lane Value

Observations:

- We see that the model learns to not stay in lanes 1, 2 and 4 as the cars are close there and the top priority is lane three where there is no red car, as is expected.

Value for maintaining a speed

Figure 54: Speed Value Slow

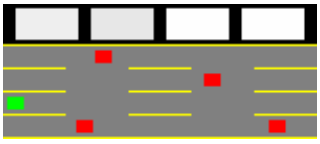

Figure 55: Speed Value Fast

Observations:

- We see that the model learns to go fast when the cars are far away (it prioritizes this) to maximize reward and when the cars are nearby it relies on going at lower speeds to avoid collisions (here it would be because the red car is right in front of it).

b. DQN agent on the continuous state representation of environment.

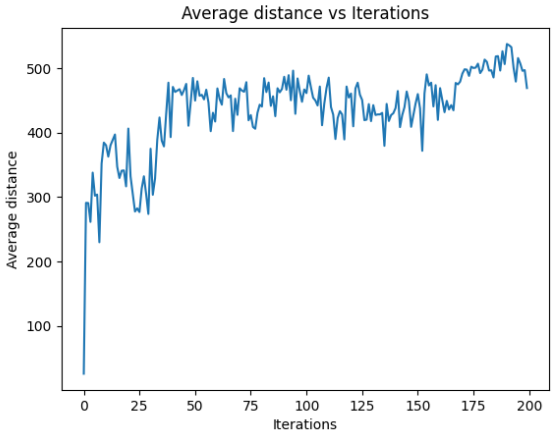

Figure 56: Avg Distance over 1000 runs vs every 1000 iterations

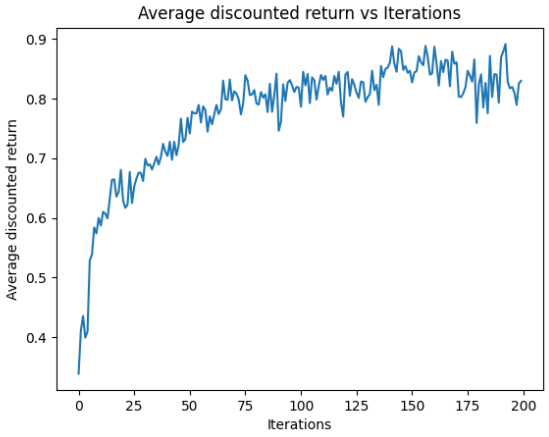

Figure 57: Avg Rewards over 1000 runs vs every 1000 iterations

- Analogous to the discrete domain, the model didn't need to be changed as only the distances in the state are continuous which wouldn't affect the argmax taken over the q values for all the actions
- We see that the model went to about 500 after the 200,000 iterations.

Value for staying in a lane

Figure 58: Lane Value

Observations:

- We see that the model learns to not stay in the first three lanes (lane 3 isn't preferred as the car in lane 2 could switch lanes) as the cars are closer and the top priority is the bottom most lane where there is no red car, as is expected.

Value for maintaining a speed

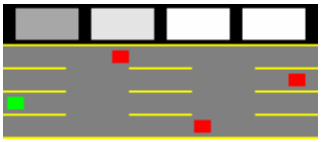

Figure 59: Speed Value Fast

Figure 60: Speed Value Slow

Observations:

- We see that the model learns to go fast when the cars are far away (it highly prioritizes this) to maximize reward and when the cars are nearby it relies on going at lower speeds to avoid collisions (here it would be because the red car is right in front of it).

Part III: Best Model

a. DQN improvement

One strategy we used was to improve upon the default architecture and parameters provided in part II. The architecture used is (each of the fully connected layers are followed by ReLU activations),

Layer	Description
Fully Connected 1	Input: 6, Output: 256, followed by ReLU
Dropout 1	Dropout probability: $p = 0.2$
Fully Connected 2	Input: 256, Output: 128, followed by ReLU
Fully Connected 3	Input: 128, Output: 64, followed by ReLU
Dropout 2	Dropout probability: $p = 0.1$
Fully Connected 4	Input: 64, Output: 32, followed by ReLU
Fully Connected 5	Input: 32, Output: 5, followed by ReLU

Table 1: Neural network architecture used in our experiments.

We also tried a slightly larger model with 1 more fully connected layer (Input = 256, Output = 256) between fc1 and fc2 but there wasn't any improvement on average (over 1000 seeds) at the expense of an increased training time.
The hyper parameters used are, lr = 3e-4 (did better than 1e-4 in this case), buffer size = 100,000, discount factor = 0.9 (did better than 0.99 in this case) and $\epsilon = 0.75$ (exponentially decaying with a minimum of 0.3). The dropout layers in the above model significantly improved the average distance traveled (over 1000 seeds) due to regularization of the learning. We also tried to increase the dropout probability to 0.4 and 0.3 for the 2 dropout layers respectively but this also caused a significant drop in the average distance traveled.
The model provided when run on 1000 seeds (1 - 1000) gave an average distance traveled of 880.2.

b. Tabular Q Improvement

Also tried to improve Tabular Q-Agent to its limit for potentially best model. After learning from the effects of hyper paramters in part 1. Using exponential decay of 0.99 with min 0.3 and df = 0.9 and increasing the number of iterations, managed to get it to improve to about 800 units but not as much as the DQN improvement nor as stable for various seeds.
Best Model Chosen: Therefore the model chosen after consideration is a.) The DQN agent.