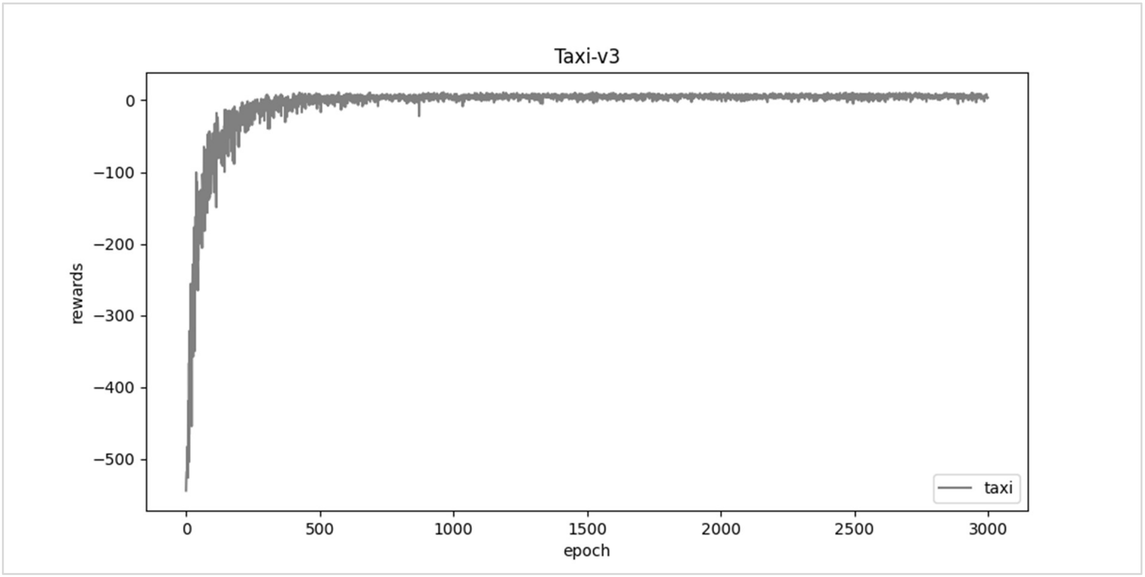
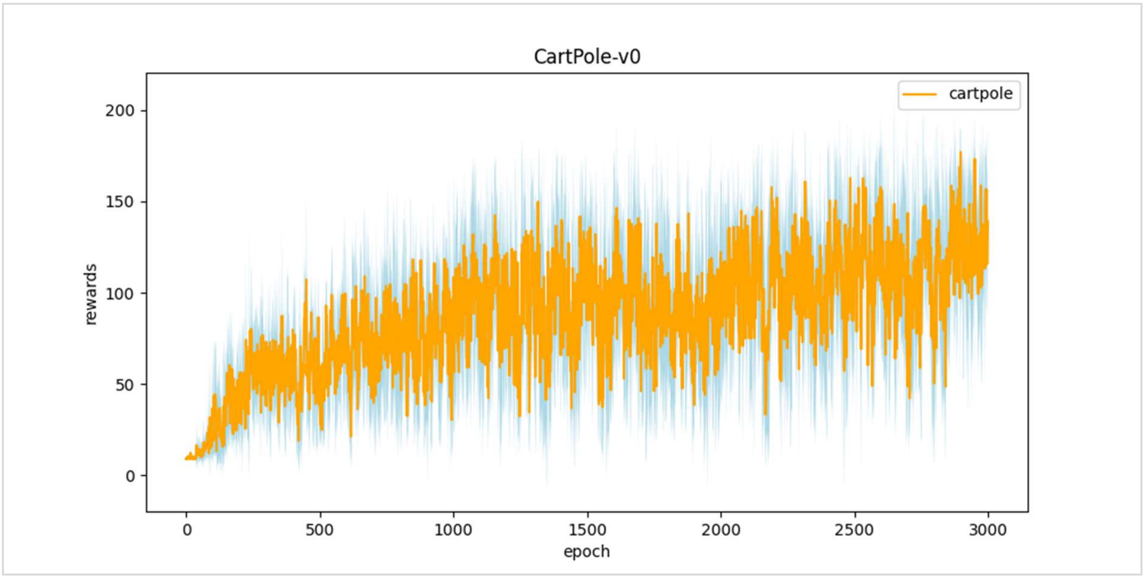
Homework 4: Reinforcement Learning Report

Part I. Experiment Results:

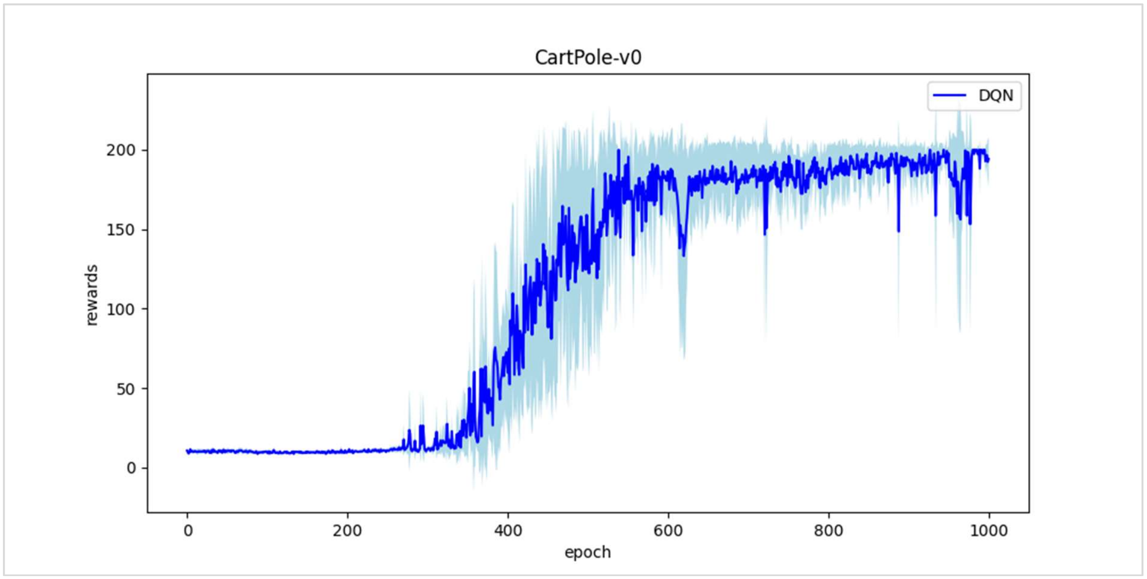
1. taxi.png:



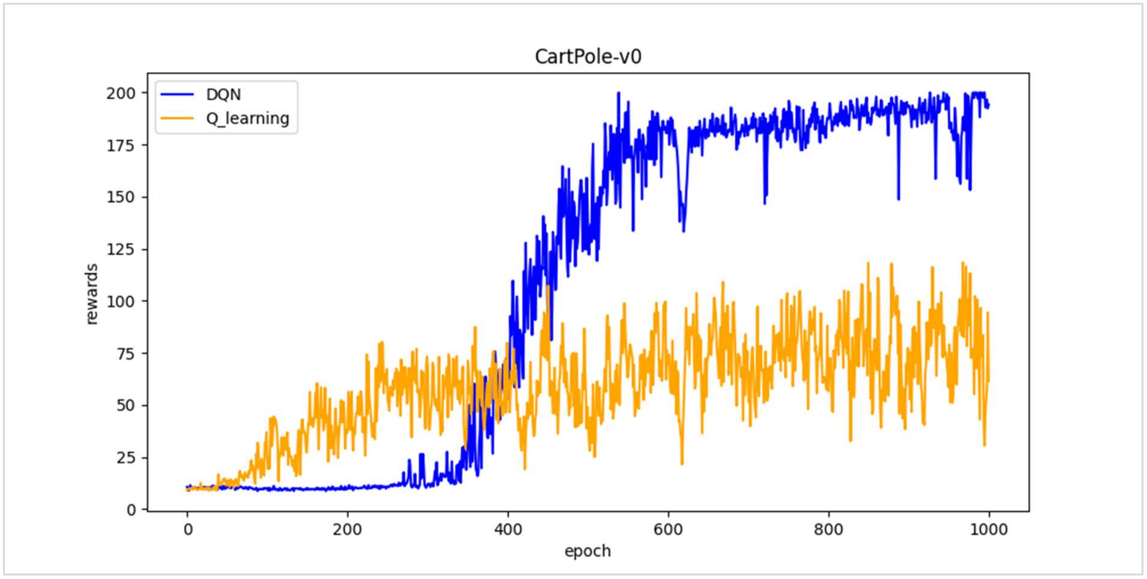
2. cartpole.png:



3. DQN.png:



4. compare.png:

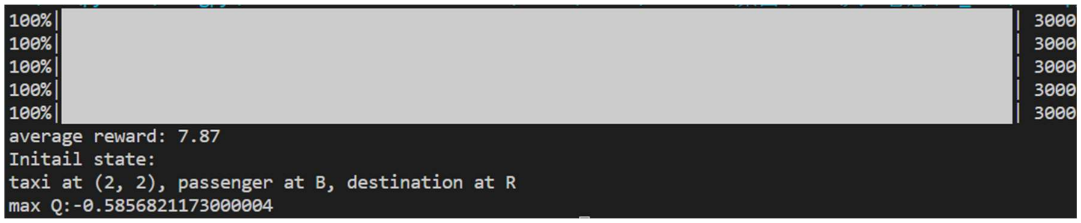


Part II. Question Answering (50%):

1. **Calculate the optimal Q-value of a given state in Taxi-v3 (the state is assigned in google sheet), and compare with the Q-value you learned (Please screenshot the result of the "check_max_Q" function to show the Q-value you learned). (4%):**

   Withγ= 0.9, and the taxi agent takes minimal step which is totally 12 actions to complete the task (take passenger from B to R):

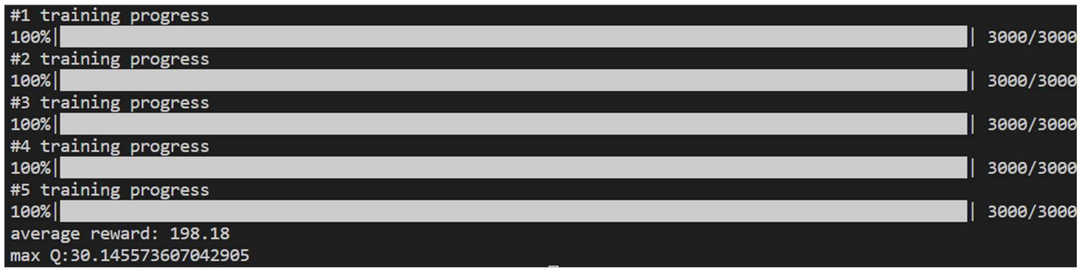
Optimal Q-value = (-1) + 0.9 * (-1) + $(0.9)^2$ * (-1) + … + $(0.9)^{10}$ * (-1) + $(0.9)^{11}$ * 20 = -0.899492713390001



```
100%|                                                                    | 3000
100%|                                                                    | 3000
100%|                                                                    | 3000
100%|                                                                    | 3000
100%|                                                                    | 3000
average reward: 7.87
Initail state:
taxi at (2, 2), passenger at B, destination at R
max Q:-0.5856821173000004
```

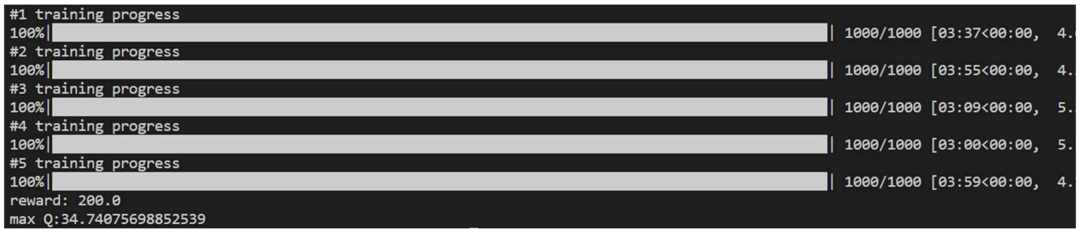
2. **Calculate the max Q-value of the initial state in CartPole-v0, and compare with the Q-value you learned. (Please screenshot the result of the“check_max_Q” function to show the Q-value you learned) (4%):**

Part2: Withγ= 0.97, and the cartpole agent takes action to prevent pole from falling over and get highest reward:
Optimal Q-value = 1 + 0.97 * 1 + $(0.97)^2$ * 1 + … + $(0.97)^{199}$ * 1
= 33.25795863300013



```
#1 training progress
100%|                                                              | 3000/3000
#2 training progress
100%|                                                              | 3000/3000
#3 training progress
100%|                                                              | 3000/3000
#4 training progress
100%|                                                              | 3000/3000
#5 training progress
100%|                                                              | 3000/3000
average reward: 198.18
max Q:30.145573607042905
```

Part3: Withγ= 0.97
Optimal Q-value = 1 + 0.97 * 1 + $(0.97)^2$ * 1 + … + $(0.97)^{199}$ * 1
= 33.25795863300013



```
#1 training progress
100%|                                                    | 1000/1000 [03:37<00:00,  4.
#2 training progress
100%|                                                    | 1000/1000 [03:55<00:00,  4.
#3 training progress
100%|                                                    | 1000/1000 [03:09<00:00,  5.
#4 training progress
100%|                                                    | 1000/1000 [03:00<00:00,  5.
#5 training progress
100%|                                                    | 1000/1000 [03:59<00:00,  4.
reward: 200.0
max Q:34.74075698852539
```

3.
   a. **Why do we need to discretize the observation in Part 2? (2%)**

   I think the reason is we need an integer value of state information to access and index a Q-table. In this game, every state involves velocity, angle…, which has infinite and continuous values and information. Thus, if we want to use Q-table, we should discretize them into different intervals and label them with an integer. So that we can use the each “label” of them to access the Q-table.

In my opinion, the concept of "**discretization"** is just like the concept of sampling analog signals (infinite and continuous) into digital signals (0, 1, 2, …).

b. **How do you expect the performance will be if we increase "num_bins"? (2%)**

In my expectation, the increasing of "num_bins" will improve the performance of the game agent. The reason is the more "num_bins" it has, the data will be more like original state that before we do discretization.

c. **Is there any concern if we increase "num_bins"? (2%)**

The one concern of increasing of "num_bins" I came up with is that will spend more time to implement discretization process.

The other concern is if the "num_bins" is out of the range of any data type we use to label, it will not work and cause overflow problem. Eg: Int data type can only store the label from – (2^31) to (2^31 – 1), when "num_bins" exceeds 2^32, it will fail.

4. **Which model (DQN, discretized Q learning) performs better in Cartpole-v0, and what are the reasons? (3%)**

DQN performs better in Cartpole-v0.

In Cartpole-v0, every state space is infinite, this is not a suitable condition for Q learning which use Q-table unless we do discretization. For the problem, using function approximation done by deep neural network to generalize across states is a better way to solve it.

Besides, from compare.png above, we can see that DQN agent has faster speed converging into higher performance, that also means DQN is better.

5.

a. **What is the purpose of using the epsilon greedy algorithm while choosing an action? (2%)**

In RL, the agent has none or limited knowledge about the environment at the beginning. And we meet the balance of choosing an action between "exploration" and "exploitation" by setting parameter "epsilon".

Thus, the agent can choose to explore by selecting an action with an unknown outcome, to get more information about the environment. Or, it can choose to exploit and choose an action based on its prior knowledge of the environment to get a good reward.

b. **What will happen, if we don't use the epsilon greedy algorithm in the CartPole-v0 environment? (3%)**

If we don't use the epsilon greedy algorithm, the agent will only take the best action from previous experience and will not try the other action which has unknown reward. That is, if the agent initially pushes the cart to the right side and thus get a good reward. Next, it will never push the cart to the left due to the risk and uncertainty.

c. **Is it possible to achieve the same performance without the epsilon greedy algorithm in the CartPole-v0 environment? Why or Why not? (3%)**

I think it is impossible to achieve the same performance without the epsilon greedy algorithm in the CartPole-v0 environment. The reasons are that we initialize the all values of every states to 0 in the Q-table, once a game or an episode is over, in the next game, the agent will only follow the actions of last time because the Q-value is greater than 0. So, without the choosing action randomly, the agent will get same total reward as first time forever.

d. **Why don't we need the epsilon greedy algorithm during the testing section? (2%)**

Because randomness is unnecessary in testing section, all the thing during the testing section need to do is verifying the model we trained is good or not. Thus, in testing section, the tested agent only needs to follow the action which has optimal Q-value at every state.

6. **Why is there "with torch.no_grad():" in the "choose_action" function in DQN? (3%)**

"with torch.no_grad():" perform inference without gradient calculation and deactivates autograd engine. Eventually it will reduce the memory usage and speed up computations. And when the agent chooses an action, ""with torch.no_grad():" should being used to let the evaluation network stop doing gradient descent computation. So that the agent can get an action from stable situation.

7.

a. **Is it necessary to have two networks when implementing DQN? (1%)**

Yes, when implementing DQN, two networks are necessary.
One for predicting the appropriate action. The second one for predicting the target Q values for calculating the error.

b. **What are the advantages of having two networks? (3%)**

Using two networks which updated every steps with a copy of the latest learned parameters, helps prevent the bias from dominating the system numerically, causing the estimated Q values to diverge.

c. **What are the disadvantages? (2%)**

The disadvantage of having two networks is they may take long execution time to compute the loss function and gradient descent operation every moment. I think this problem is unfriendly to someone who want to implement DQN and doesn't have good CPU or GPU.

8.

a. **What is a replay buffer(memory)? Is it necessary to implement a replay buffer?**

The replay buffer contains a collection of experience tuples (Sate, Action, Reward, Next state). The tuples are added into the buffer gradually when the agent interacts with the environment.

Yes, if we want to train a good model by implementing DQN, the replay buffer is necessary because it can sample the experience tuples randomly and reduce the correlation between data.

b. **What are the advantages of implementing a replay buffer? (5%)**

One of the advantages of implementing a replay buffer is its randomness of sampling experiences can reduce the correlation of data, and then, reduce the MSE loss during training process.

And other advantage is its independent and identically distributed output data would prevent the model from falling into partial optima. Besides, it also can reach faster speed of convergence.

c. **Why do we need batch size? (3%)**

Batch size controls the accuracy of the estimation of the error gradient when training neural networks. Besides, the batch size impacts how quickly a model learns and the stability of the learning process.

d. **Is there any effect if we adjust the size of the replay buffer(memory) or**

batch size? Please list some advantages and disadvantages. (2%)

Pros:
(1) Using smaller batch sizes have been empirically shown to have faster convergence to "good" solutions.
(2) Use higher batch size can let the model converge to the global optima faster and easier.

Cons:
(1) The downside of using a smaller batch size is that the model is not guaranteed to converge to the global optima.
(2) Higher batch size is a loading for computing devices like GPU.

9.

a. What is the condition that you save your neural network? (1%)

When the max q-value of the current evaluation network is best among the training episodes, I will save my neural network.

b. What are the reasons? (2%)

Reason: When the performance of predicted network we train is highest, I think that is the best and suitable timing to save the neural network.

10. What have you learned in the homework? (2%)

In HW4, I have learned that how to implement RL with Q learning and Q-table and how to use the concept of discretization to let the continuous state information can be mapped into Q-table.

Most important part I have learned is that the realization and implementation of DQN with PyTorch, and thus from this homework I got the idea of our final project.