Homework 4:

Reinforcement Learning

Part I. Implementation:

TAXI

Choose_action:

```python
# Begin your code
# TODO
"""
Choose the best action for the given state and epsilon value.
This function implements the epsilon-greedy policy where the agent either explores a random action with
probability epsilon or exploits the best known action with probability 1 - epsilon.
"""
if np.random.uniform(0,1)<=self.epsilon:
    move=env.action_space.sample()
else:
    move=np.argmax(self.qtable[state])
return move
# End your code
```

Learn:

```python
# Begin your code
# TODO
"""
caculate the Q-value update Qtable,if it is done target = rewaed
"""
if not done:
    target_q = reward + self.gamma * np.max(self.qtable[next_state])
else:
    target_q = reward

current_q = self.qtable[state, action]

self.qtable[state, action] += self.learning_rate * (target_q - current_q)
# End your code
np.save("./Tables/taxi_table.npy", self.qtable)
```

check_max_Q:

```python
# Begin your code
# TODO
"""
Return the max Q-value
"""
max_q=np.max(self.qtable[state])
return max_q
# End your code
```

Cartpole

init_bins

```python
# Begin your code
# TODO
"""
Divide the range between the specified lower_bound and upper_bound into
num_bins equal segments.Since np.linspace() includes the lower_bound in
its result, return the list starting from the second element, which omits
the lower_bound.
"""
return np.linspace(lower_bound, upper_bound, num_bins, endpoint=False)[1:]
# End your code
```

discretize_value

```python
# Begin your code
# TODO
"""
Categorize the value into specific intervals defined by the bins array.
Utilize np.digitize() to ascertain the interval in which the value falls.
"""
return np.digitize(value, bins, right=False)
# End your code
```

discretize_observation

```python
# Begin your code
# TODO
"""
Convert the continuous observation into discrete categories. Use the "discretize_value()"
function to transform each of the four features in the observation into discrete data.
"""
discretized_features = [
    self.discretize_value(observation[0], self.bins[0]),
    self.discretize_value(observation[1], self.bins[1]),
    self.discretize_value(observation[2], self.bins[2]),
    self.discretize_value(observation[3], self.bins[3])
]
return discretized_features
# End your code
```

choose_action

```python
"""
# Begin your code
# TODO
"""
Generate random number between 0 and 1.if this number is not exceed epsilon
choose an action at random or select the action accroding to highest Q-value
for current state from qtable.the essilon is change 0.95 and 0.05
"""
if np.random.uniform(0,1)<=self.epsilon:
    move=env.action_space.sample()
else:
    move=np.argmax(self.qtable[tuple(state)])
return move
# End your code
```

Learn

```python
"""
# Begin your code
# TODO
"""
caculate the Q-value update Qtable,if it is done target = rewaed
"""
if not done:
    target_q = reward + self.gamma * np.max(self.qtable[tuple(next_state)])
else:
    target_q = reward

current_q = self.qtable[tuple(state)][action]

self.qtable[tuple(state)][action] += self.learning_rate * (target_q - current_q)
# End your code
np.save("./Tables/cartpole_table.npy", self.qtable)
```

check_max_Q

```python
# Begin your code
# TODO
"""
Dicretized initial state first.Return the max Q-value
"""
Q_values = self.discretize_observation(self.env.reset())
max_q=np.max(self.qtable[tuple(Q_values)])
return max_q
# End your code
```

DQN

learn

```python
def learn(self):
    # Begin your code
    # TODO
    """
    Retrieve a batch of trajectory data from the replay buffer using
    the 'sample' function of the 'replay_buffer' class. After sampling,
    this function converts these data points—states, actions, rewards,
    next states, and termination flags (done)—into tensors for further processing.
    """
    sample = self.buffer.sample(self.batch_size)
    states = torch.tensor(np.array(sample[0]), dtype=torch.float)
    actions = torch.tensor(sample[1], dtype=torch.long).unsqueeze(1)
    rewards = torch.tensor(sample[2], dtype=torch.float)
    next_states = torch.tensor(np.array(sample[3]), dtype=torch.float)
    done = torch.tensor(sample[4], dtype=torch.bool)

    """
    Process data through both networks. 'current_q_values' are predicted
    values from the evaluate network, indexed by 'actions'.
    'next_q_values' from the target network are adjusted for terminal states
    and detached to prevent gradient updates.
    'max_next_q_values' extracts the highest Q-value for non-terminal states.
    'target_q' calculates expected Q-values using the discount factor 'gamma'.
    """
    current_q_values = self.evaluate_net(states).gather(1, actions)
    next_q_values = self.target_net(next_states).detach()*(~done).unsqueeze(-1)
    max_next_q_values = next_q_values.max(1)[0].view(self.batch_size, 1)

    target_q=rewards.unsqueeze(-1)+ self.gamma *max_next_q_values
    """
    Calculate the mean squared error loss to evaluate the difference between the
    predicted and target Q-values.
    """
    lose_fuc=nn.MSELoss()
    loss = lose_fuc(current_q_values, target_q)
    """
    Reset gradients to zero, perform backpropagation to calculate gradients,
    and update the model's weights.
    """
    self.optimizer.zero_grad()
    loss.backward()
    self.optimizer.step()

    # End your code
```

choose_action

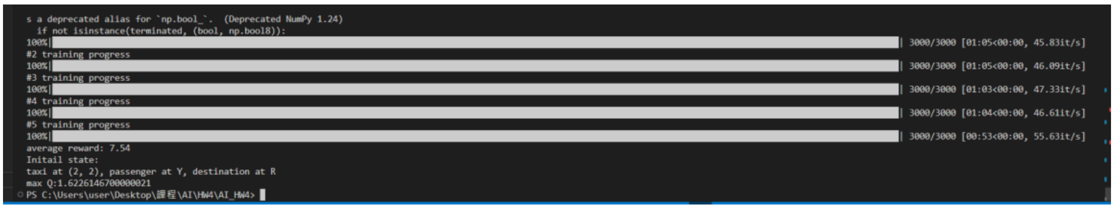
```python
# Begin your code
# TODO
"""
Use the epsilon-greedy strategy to determine the action.
With probability epsilon, choose a random action to encourage exploration.
With probability 1-epsilon, choose the best-known action (exploitation) based
on the maximum Q-value predicted by the evaluate network.
"""
if np.random.uniform(0,1)<=self.epsilon:
    action=env.action_space.sample()
else:
    action=torch.argmax(
        self.evaluate_net(torch.tensor(state, dtype=torch.float))
    ).item()
# End your code
```

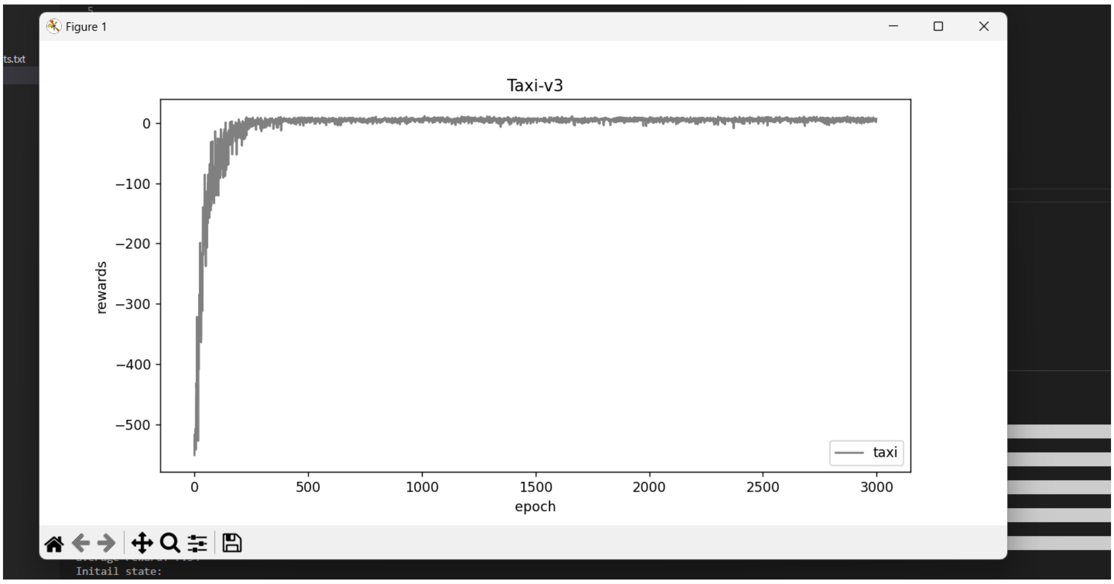
check_max_Q

```python
# Begin your code
# TODO
"""
Reset the environment and obtain the initial state. Pass the state through the
target network to compute the action values.
Return the maximum action value from the computed action values.
"""
x = torch.unsqueeze(torch.tensor(self.env.reset(), dtype=torch.float), 0)
action_values = self.target_net(x)
max=torch.max(action_values).item()
return max
# End your code
```

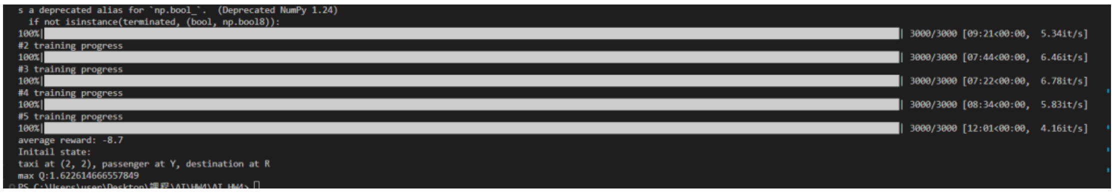
Part II. Experiment Results:

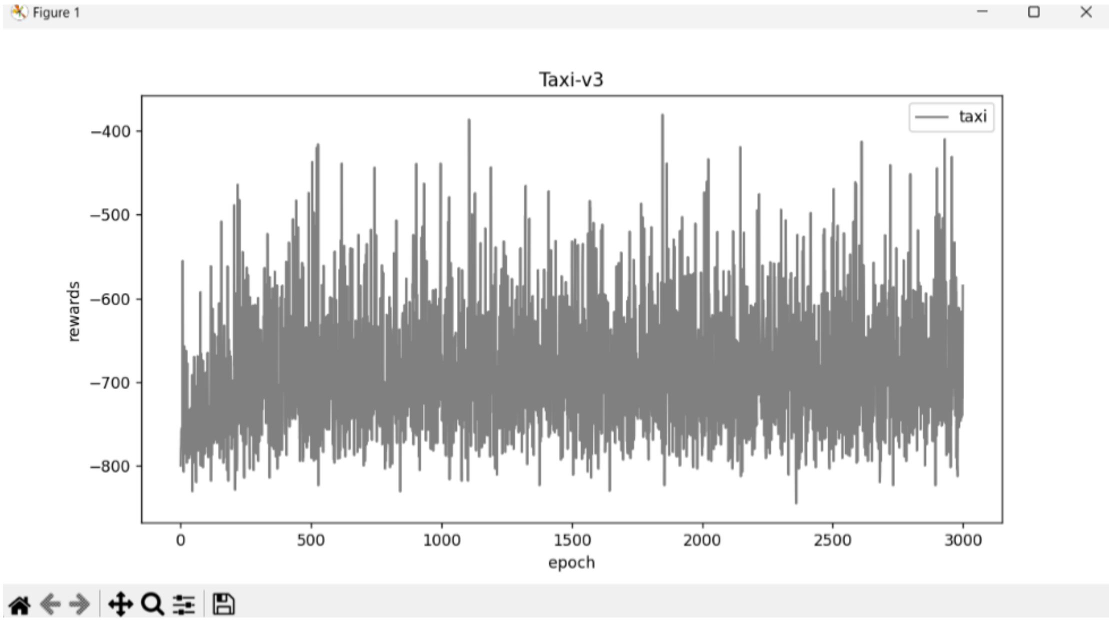
Taxi(epsilon=0.05)





Taxi(epsilon=0.95)

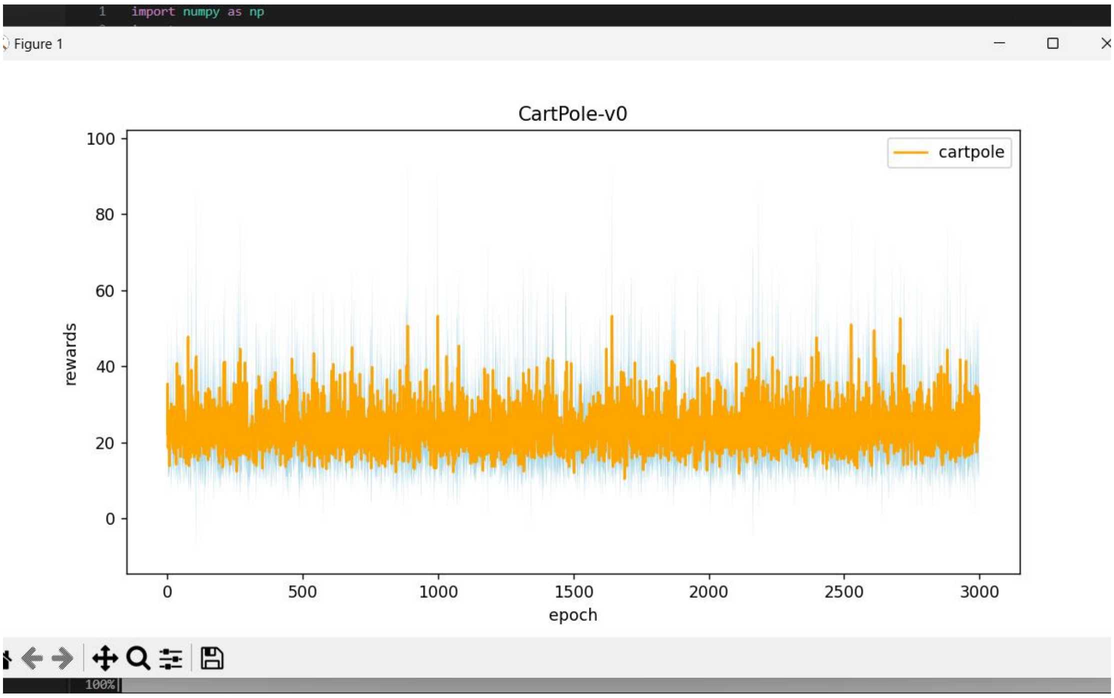




Cartpole (epsilon=0.95)

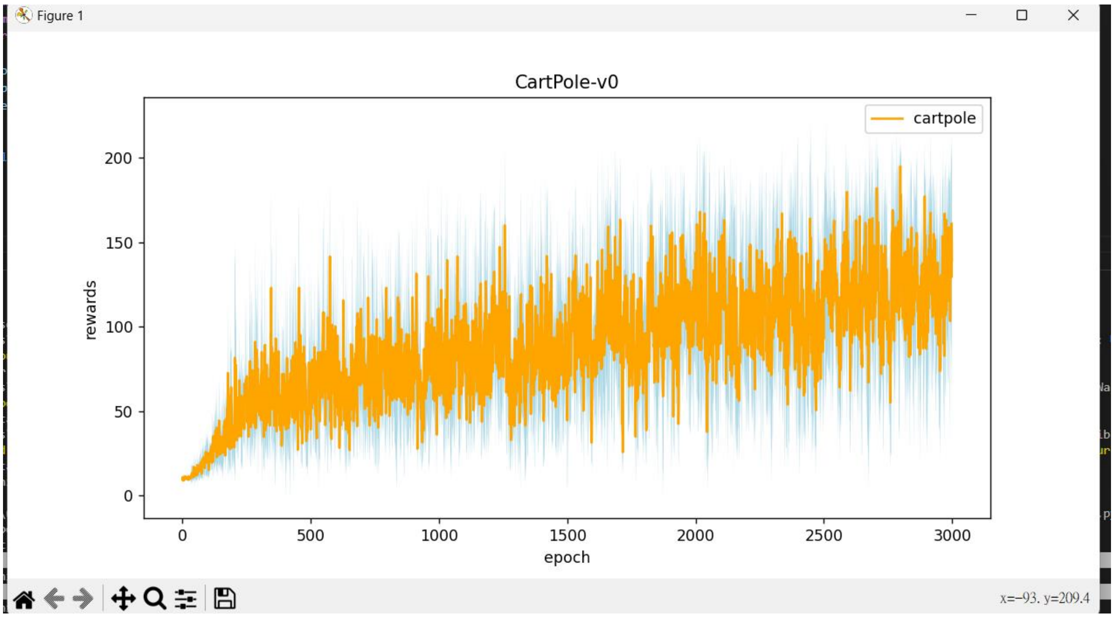
```
  0%|                                                                                  | 0/3000 [00:00<?, ?it/s]C
:\Users\user\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-packages\gym\utils\passive_env_checker.py:241: DeprecationWarning: `np.bool8` i
s a deprecated alias for `np.bool_`.  (Deprecated NumPy 1.24)
  if not isinstance(terminated, (bool, np.bool8)):
100%|██████████| 3000/3000 [00:58<00:00, 51.66it/s]
#2 training progress
100%|██████████| 3000/3000 [01:08<00:00, 43.52it/s]
#3 training progress
100%|██████████| 3000/3000 [01:29<00:00, 33.50it/s]
#4 training progress
100%|██████████| 3000/3000 [01:04<00:00, 46.69it/s]
#5 training progress
100%|██████████| 3000/3000 [01:03<00:00, 47.55it/s]
average reward: 92.33
max Q:32.66337751340888
PS C:\Users\user\Desktop\課程\AI\HW4\AI_HW4>
```

```
import numpy as np
```

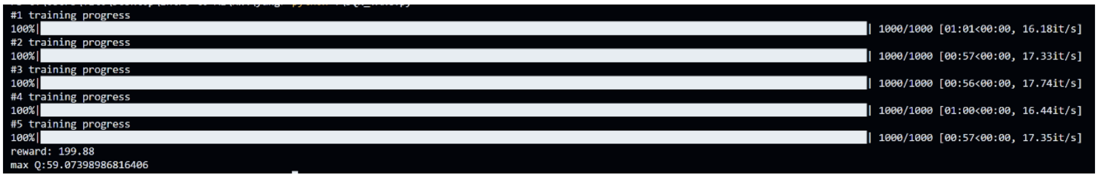


Cartpole (epsilon=0.05)

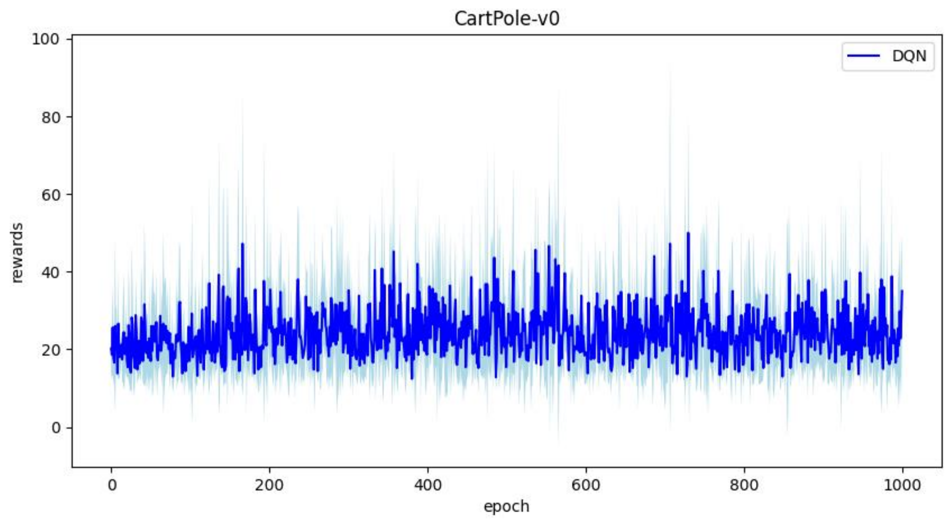
```
  0%|                                                                     | 0/3000 [00:00<?, ?it/s]C
:\Users\user\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-packages\gym\utils\passive_env_checker.py:241: DeprecationWarning: `np.bool8` is a deprecated alias fo
r `np.bool_`.  (Deprecated NumPy 1.24)
  if not isinstance(terminated, (bool, np.bool8)):
100%|| 3000/3000 [06:10<00:00,  8.09it/s]
#2 training progress
100%|| 3000/3000 [06:30<00:00,  7.69it/s]
#3 training progress
100%|| 3000/3000 [06:26<00:00,  7.76it/s]
#4 training progress
100%|| 3000/3000 [08:36<00:00,  5.81it/s]
#5 training progress
100%|| 3000/3000 [07:46<00:00,  6.43it/s]
average reward: 62.63
max Q:29.75247767630704
PS C:\Users\user\Desktop\課程\AI\HW4\AI_HW4>
```



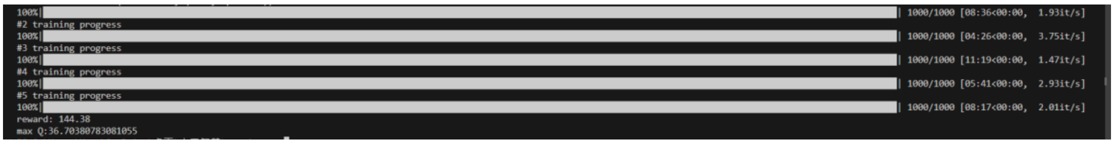
DQN (epsilon=0.95)



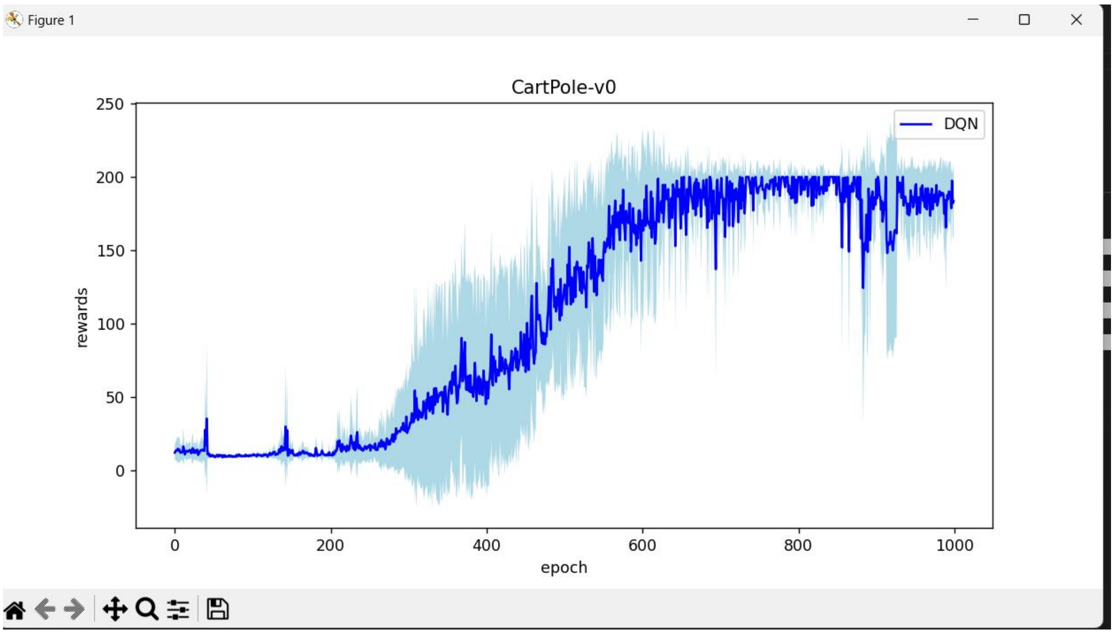
```
#1 training progress
100%|| 1000/1000 [01:01<00:00, 16.18it/s]
#2 training progress
100%|| 1000/1000 [00:57<00:00, 17.33it/s]
#3 training progress
100%|| 1000/1000 [00:56<00:00, 17.74it/s]
#4 training progress
100%|| 1000/1000 [01:00<00:00, 16.44it/s]
#5 training progress
100%|| 1000/1000 [00:57<00:00, 17.35it/s]
reward: 199.88
max Q:59.07398986816406
```



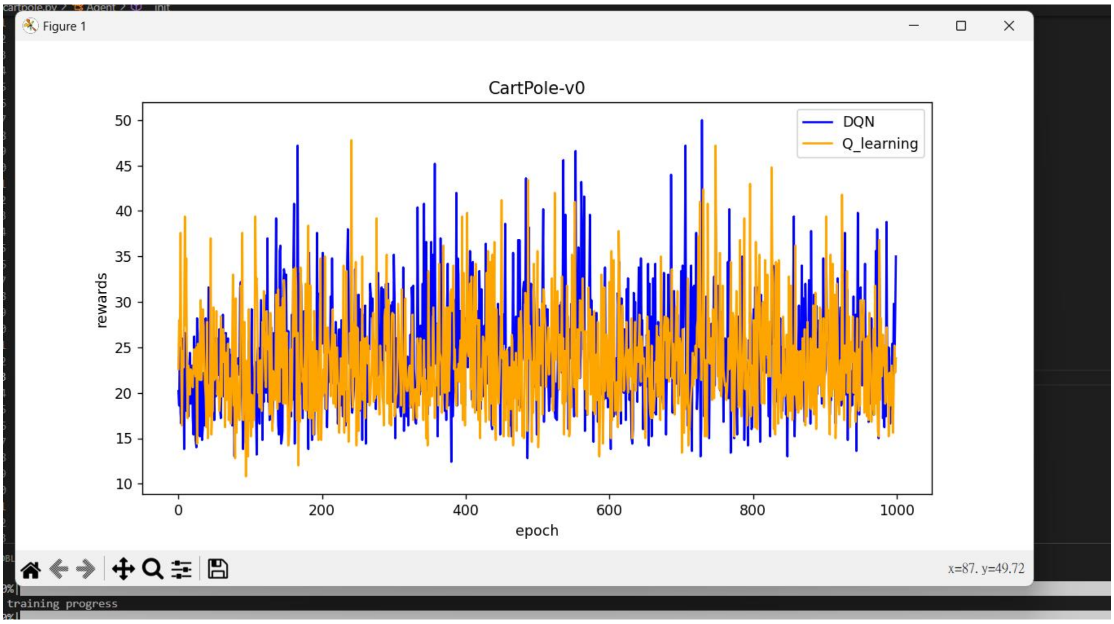
DQN (epsilon=0.05)



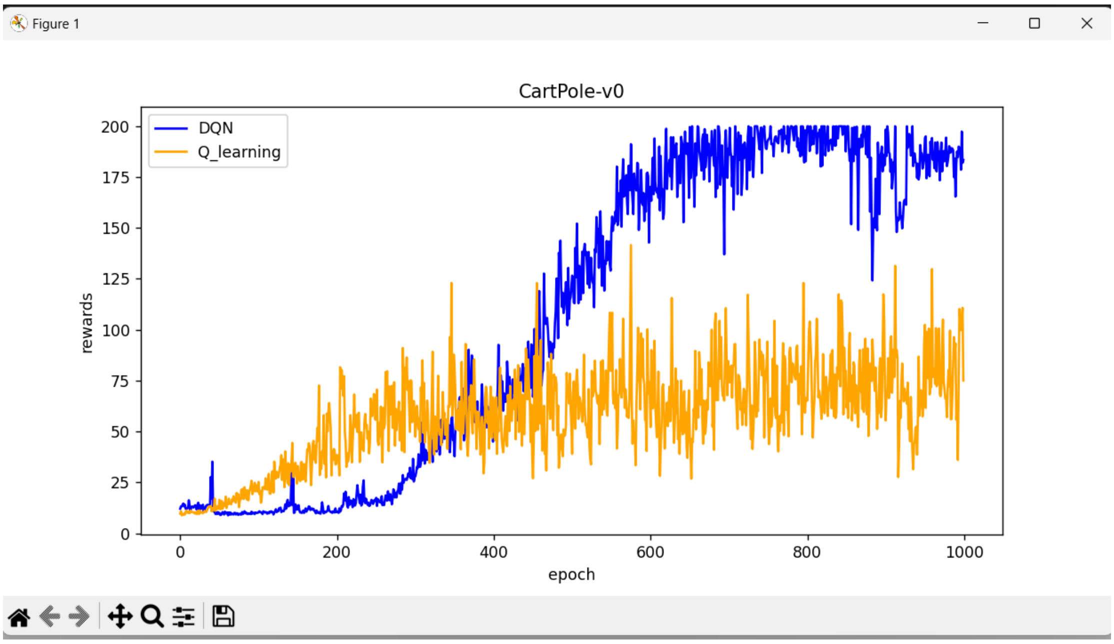
```
100%|████████████████████████████████████████| 1000/1000 [08:36<00:00,  1.93it/s]
#2 training progress
100%|████████████████████████████████████████| 1000/1000 [04:26<00:00,  3.75it/s]
#3 training progress
100%|████████████████████████████████████████| 1000/1000 [11:19<00:00,  1.47it/s]
#4 training progress
100%|████████████████████████████████████████| 1000/1000 [05:41<00:00,  2.93it/s]
#5 training progress
100%|████████████████████████████████████████| 1000/1000 [08:17<00:00,  2.01it/s]
reward: 144.38
max Q:36.70380783081055
```



Compare (epsilon=0.95)



Compare (epsilon=0.05)



Part III. Question Answering (50%):

1. Calculate the optimal Q-value of a given state in Taxi-v3, 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). (10%)

Taxi is at(2,2)

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

left->left->down->down->pick->up->up->up->up->drop except for last step with reward(-1),last step reward(20)

Optimal Qvalue=-1*(1-gamma^9)/1-gamma+20*gamma^9(gamma=0.9)

Therefore,Optimal Qvalue=1.6226....

```
max Q:1.6226146700000021
PS C:\Users\user\Desktop\課程\AI\HW4\AI_HW4>
```

```
taxi at (2, 2), passenger at Y, destination at R
max Q:1.622614666557849
PS C:\Users\user\Desktop\課程\AI\HW4\AI_HW4>
```

Epsilon in0.05 and 0.95 is both close to the Optimal value

2. Calculate the optimal Q-value of the initial state in CartPole-v0, and compare with the Q-value you learned(both cartpole.py and DQN.py). (Please screenshot the result of the "check_max_Q" function to show the Q-value you learned) (10%)

The Optimal Qvalue=1-gamma^average_re/1-gamma(gamma=0.97)

which is close to 1/1-0.97=33.33....

Qlearning:

```
max Q:29.75247767630704
```

DQN:

```
max Q:36.70380783081055
```

We compare with epsilon is 0.05

DQN is more closer.

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

    Since the states are continuous, we must first discretize the observations to simplify the Q-learning process and improve its efficiency.

    b. How do you expect the performance will be if we increase “num_bins”? (3%)

    I think is can increase the State resolution, it will perform better.

    c. Is there any concern if we increase “num_bins”? (3%)

    It slower the convergence because the complexity and computational cost is increase.

4. Which model (DQN, discretized Q learning) performs better in Cartpole-v0, and what are the reasons? (5%)

    On average,DQN performs better in Cartpole-v0

    Reason:

    1.DQN can directly process continuous state spaces without needing discretization, maintaining the integrity and detail of the state information.

    2. DQN uses neural networks that can generalize across states, making it more effective for environments with large or infinite state spaces.

    3. DQN includes features like experience replay and target networks, which stabilize and improve the learning process.

5. a. What is the purpose of using the epsilon greedy algorithm while choosing an action? (3%)

    In the beginning, since the agent lacks knowledge about the environment, it's essential to encourage exploration. However, it's equally important for the agent to leverage what it has learned for optimal decision-making. The epsilon-greedy algorithm is used to strike a balance between these two needs—exploring to gather more information and exploiting known information to maximize rewards.

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

If the epsilon greedy algorithm is not used in the CartPole-v0 environment, the agent will rely solely on known information to make decisions, potentially missing out on some unknown high-performance conditions. In other words, the lack of exploration may prevent the agent from discovering superior strategies, thus limiting its performance optimization.

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

Yes, it is possible.there are some others algorithm can change the epsilon-greedy.Such as,Boltzmann Eploration

d. Why don't we need the epsilon greedy algorithm during the testing section? (3%)

Because the agent has enough information of the encironment, and it is no need to exploration.

6. Why does "with torch.no_grad():" do inside the "choose_action" function in DQN? (4%)

   In this step, we use a neural network to estimate the next possible actions and select one, so there's no need to calculate gradients. Additionally, using torch.no_grad() helps optimize memory usage, speed up computations, and ensure computational correctness.