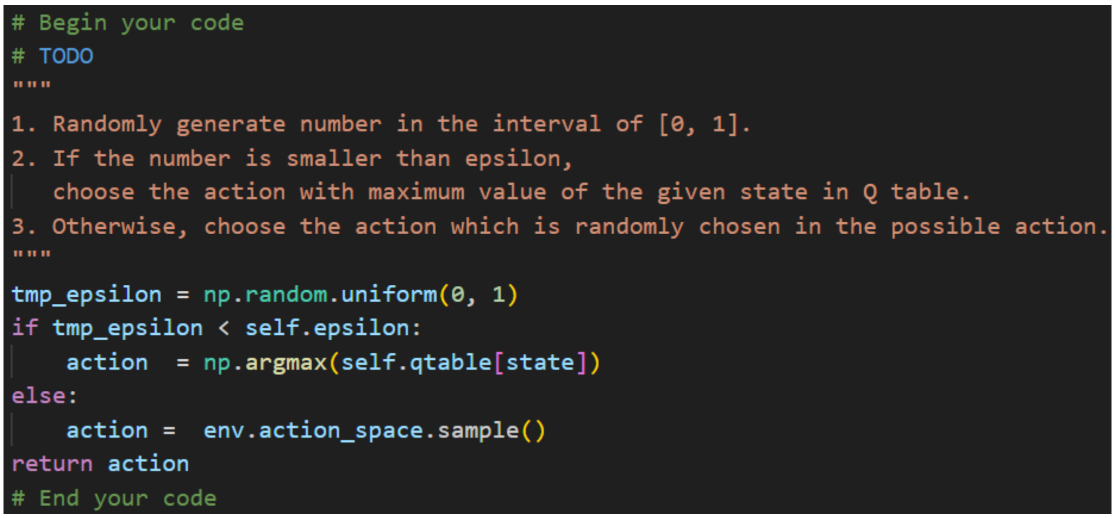
HW 4: Reinforcement Learning

Student ID: 111550100 Name: 邱振源

Part I. Implementation

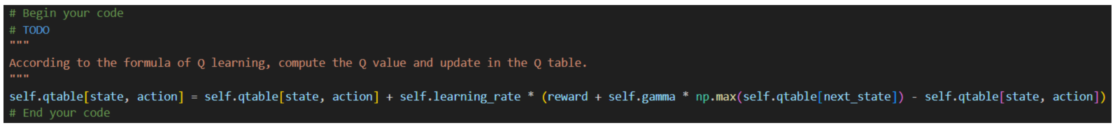
Part 1: Q learning in Taxi-v3

- choose_action



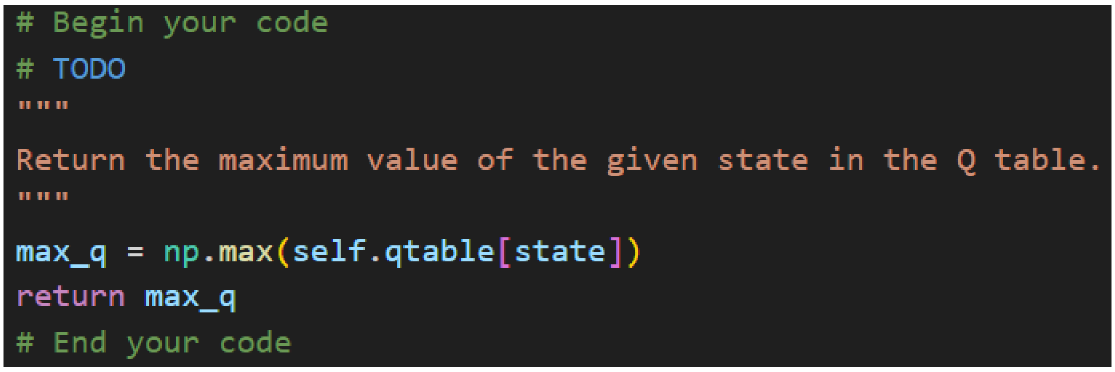
```
# Begin your code
# TODO
"""
1. Randomly generate number in the interval of [0, 1].
2. If the number is smaller than epsilon,
    choose the action with maximum value of the given state in Q table.
3. Otherwise, choose the action which is randomly chosen in the possible action.
"""
tmp_epsilon = np.random.uniform(0, 1)
if tmp_epsilon < self.epsilon:
    action  = np.argmax(self.qtable[state])
else:
    action =  env.action_space.sample()
return action
# End your code
```

- learn



```
# Begin your code
# TODO
"""
According to the formula of Q learning, compute the Q value and update in the Q table.
"""
self.qtable[state, action] = self.qtable[state, action] + self.learning_rate * (reward + self.gamma * np.max(self.qtable[next_state]) - self.qtable[state, action])
# End your code
```

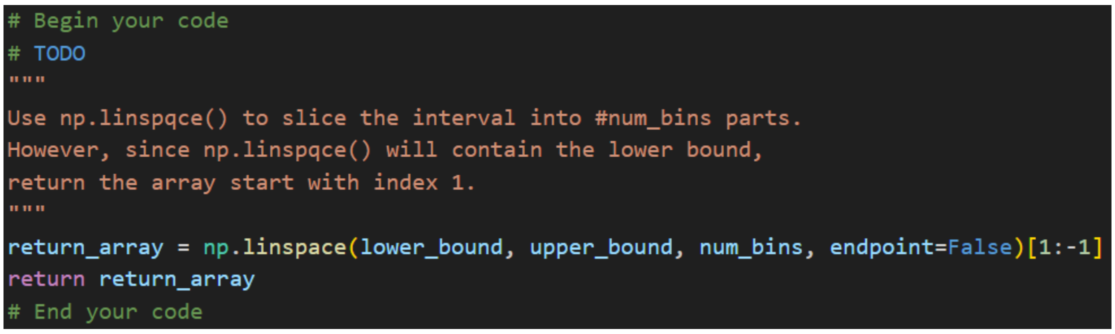
- check_max_Q



```
# Begin your code
# TODO
"""
Return the maximum value of the given state in the Q table.
"""
max_q = np.max(self.qtable[state])
return max_q
# End your code
```

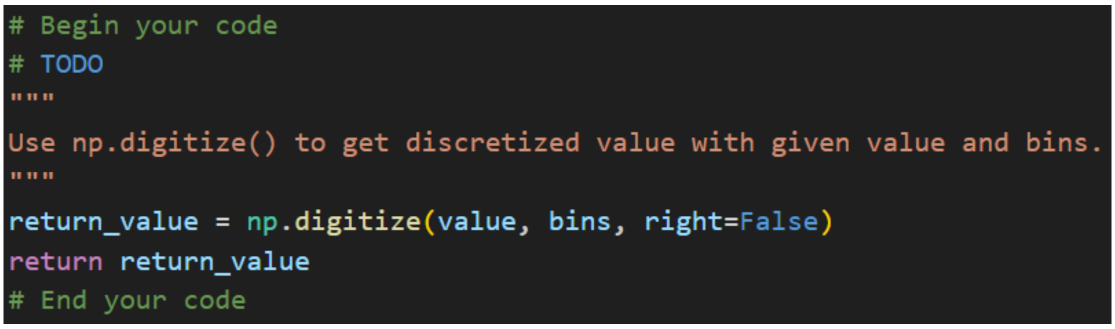
Part 2: Q learning in CartPole-v0

- init_bins



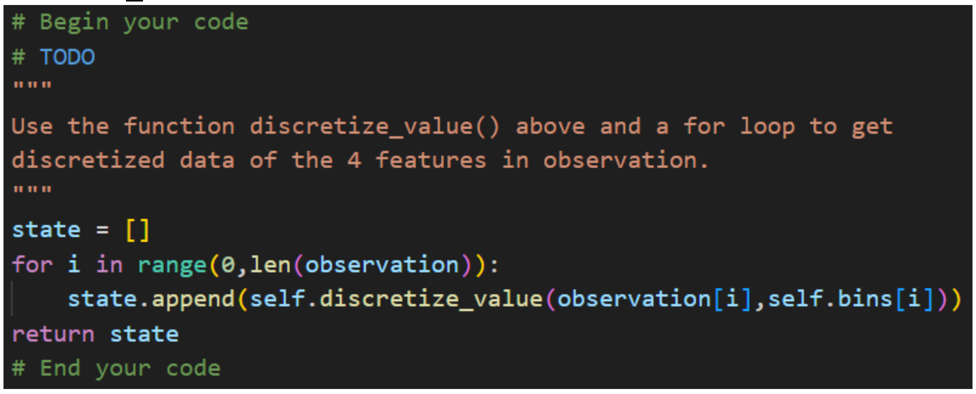
```
# Begin your code
# TODO
"""
Use np.linspqce() to slice the interval into #num_bins parts.
However, since np.linspqce() will contain the lower bound,
return the array start with index 1.
"""
return_array = np.linspace(lower_bound, upper_bound, num_bins, endpoint=False)[1:-1]
return return_array
# End your code
```

- discretize_value



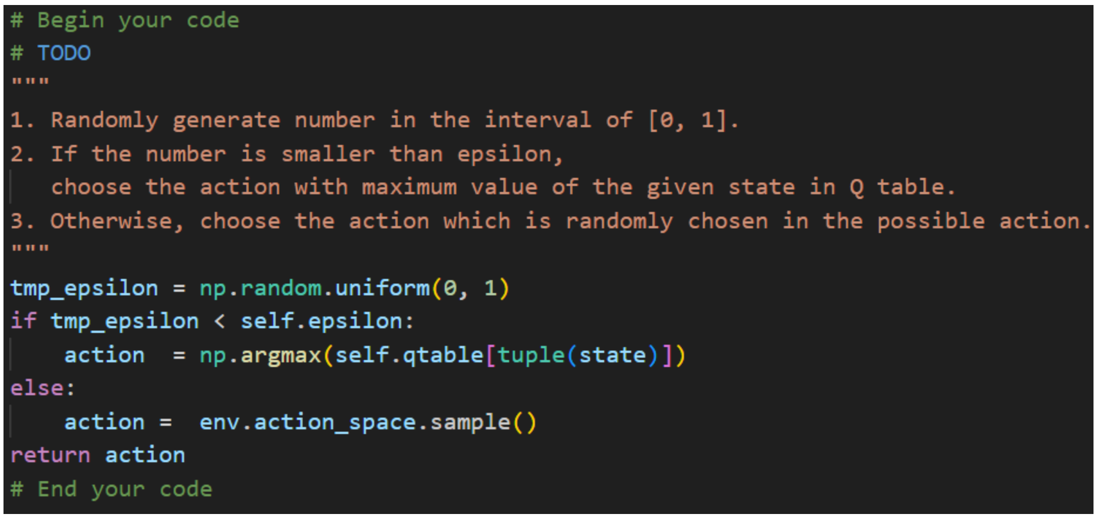
```
# Begin your code
# TODO
"""
Use np.digitize() to get discretized value with given value and bins.
"""
return_value = np.digitize(value, bins, right=False)
return return_value
# End your code
```

- discretize_observation



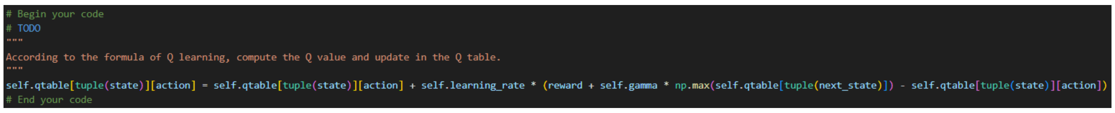
```
# Begin your code
# TODO
"""
Use the function discretize_value() above and a for loop to get
discretized data of the 4 features in observation.
"""
state = []
for i in range(0,len(observation)):
    state.append(self.discretize_value(observation[i],self.bins[i]))
return state
# End your code
```

- choose_action



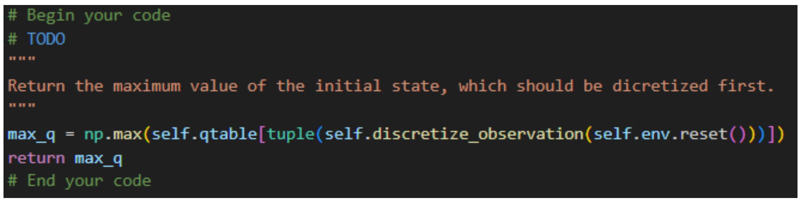
```
# Begin your code
# TODO
"""
1. Randomly generate number in the interval of [0, 1].
2. If the number is smaller than epsilon,
    choose the action with maximum value of the given state in Q table.
3. Otherwise, choose the action which is randomly chosen in the possible action.
"""
tmp_epsilon = np.random.uniform(0, 1)
if tmp_epsilon < self.epsilon:
    action  = np.argmax(self.qtable[tuple(state)])
else:
    action =  env.action_space.sample()
return action
# End your code
```

- learn



```python
# Begin your code
# TODO
"""
According to the formula of Q learning, compute the Q value and update in the Q table.
"""
self.qtable[tuple(state)][action] = self.qtable[tuple(state)][action] + self.learning_rate * (reward + self.gamma * np.max(self.qtable[tuple(next_state)]) - self.qtable[tuple(state)][action])
# End your code
```

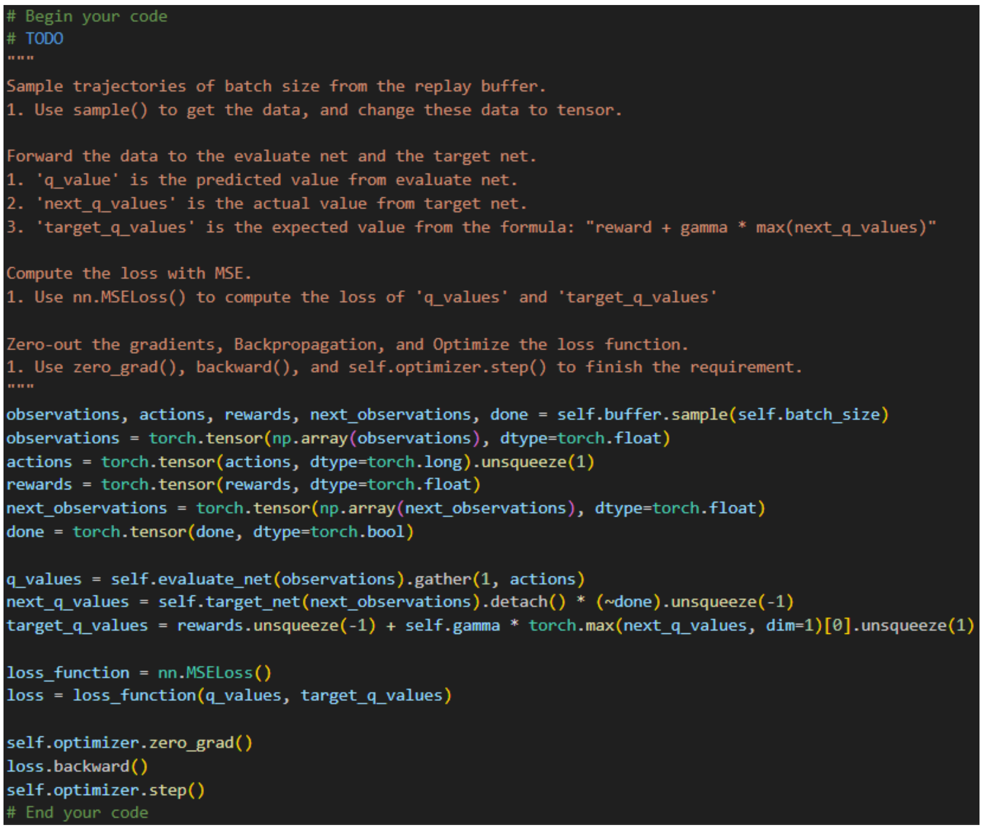
- check_max_Q



```python
# Begin your code
# TODO
"""
Return the maximum value of the initial state, which should be dicretized first.
"""
max_q = np.max(self.qtable[tuple(self.discretize_observation(self.env.reset()))])
return max_q
# End your code
```

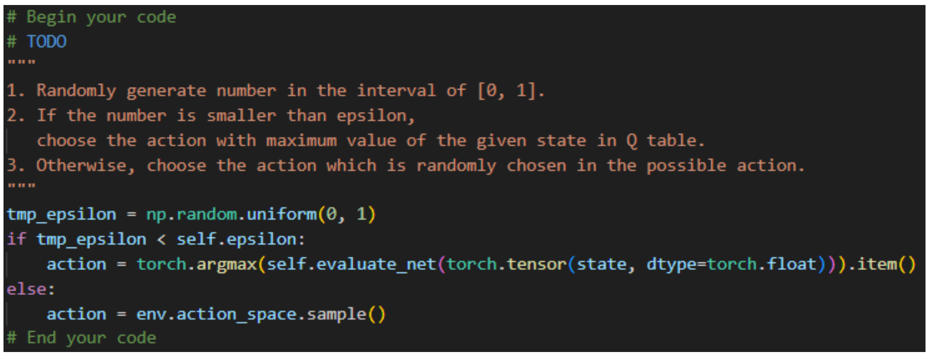
Part 3: DQN in CartPole-v0

- learn



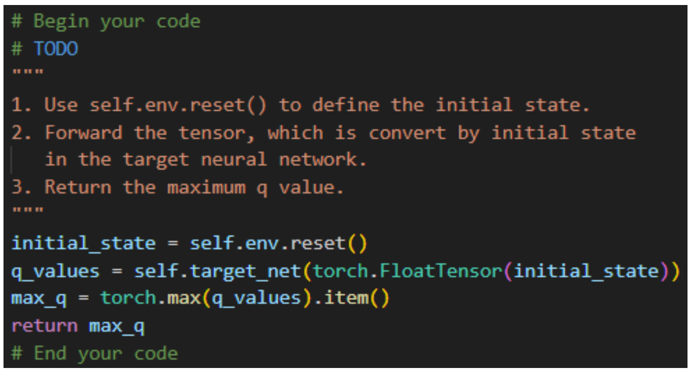
```python
# Begin your code
# TODO
"""
Sample trajectories of batch size from the replay buffer.
1. Use sample() to get the data, and change these data to tensor.

Forward the data to the evaluate net and the target net.
1. 'q_value' is the predicted value from evaluate net.
2. 'next_q_values' is the actual value from target net.
3. 'target_q_values' is the expected value from the formula: "reward + gamma * max(next_q_values)"

Compute the loss with MSE.
1. Use nn.MSELoss() to compute the loss of 'q_values' and 'target_q_values'

Zero-out the gradients, Backpropagation, and Optimize the loss function.
1. Use zero_grad(), backward(), and self.optimizer.step() to finish the requirement.
"""
observations, actions, rewards, next_observations, done = self.buffer.sample(self.batch_size)
observations = torch.tensor(np.array(observations), dtype=torch.float)
actions = torch.tensor(actions, dtype=torch.long).unsqueeze(1)
rewards = torch.tensor(rewards, dtype=torch.float)
next_observations = torch.tensor(np.array(next_observations), dtype=torch.float)
done = torch.tensor(done, dtype=torch.bool)

q_values = self.evaluate_net(observations).gather(1, actions)
next_q_values = self.target_net(next_observations).detach() * (~done).unsqueeze(-1)
target_q_values = rewards.unsqueeze(-1) + self.gamma * torch.max(next_q_values, dim=1)[0].unsqueeze(1)

loss_function = nn.MSELoss()
loss = loss_function(q_values, target_q_values)

self.optimizer.zero_grad()
loss.backward()
self.optimizer.step()
# End your code
```

- choose_action



```python
# Begin your code
# TODO
"""
1. Randomly generate number in the interval of [0, 1].
2. If the number is smaller than epsilon,
   choose the action with maximum value of the given state in Q table.
3. Otherwise, choose the action which is randomly chosen in the possible action.
"""
tmp_epsilon = np.random.uniform(0, 1)
if tmp_epsilon < self.epsilon:
    action = torch.argmax(self.evaluate_net(torch.tensor(state, dtype=torch.float))).item()
else:
    action = env.action_space.sample()
# End your code
```

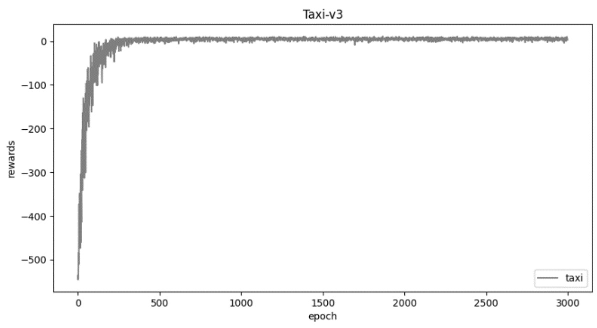
- check_max_Q



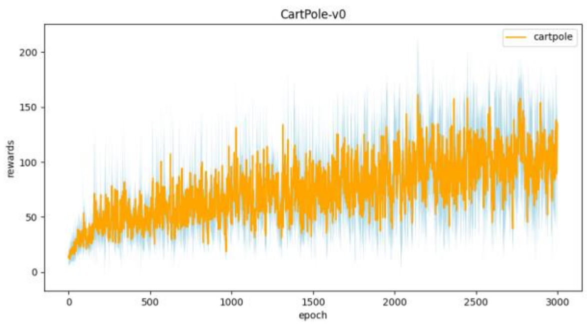
```
# Begin your code
# TODO
"""
1. Use self.env.reset() to define the initial state.
2. Forward the tensor, which is convert by initial state
   in the target neural network.
3. Return the maximum q value.
"""
initial_state = self.env.reset()
q_values = self.target_net(torch.FloatTensor(initial_state))
max_q = torch.max(q_values).item()
return max_q
# End your code
```

Part II. Experiment Results:

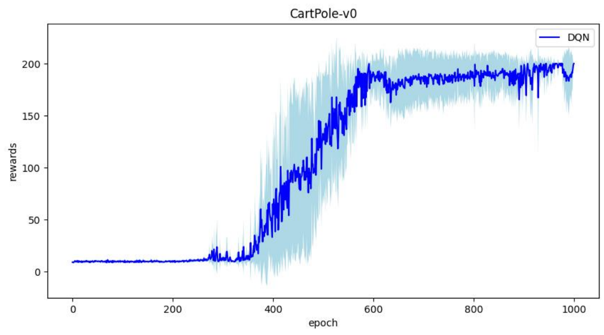
1. taxi.png



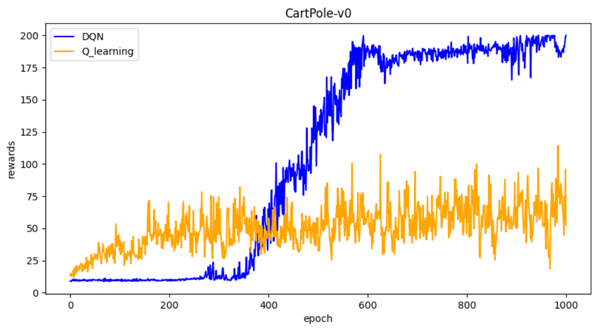
2. cartpole.png



3. DQN.png

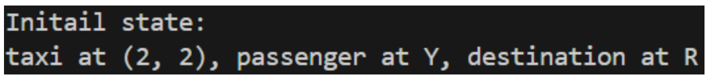


4. compare.png



Part III. Answer the questions

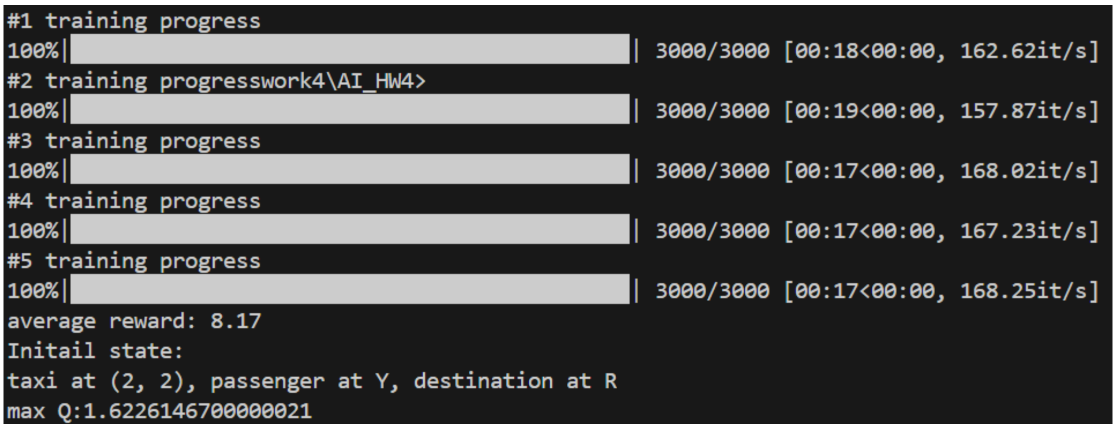
1. **Calculate the optimal Q-value of a given state in Taxi-v3, and compare with the Q-value you learned (Please screenshot the result of the "check_max_Q" function to show the Q-value you learned). (10%)**



```
Initail state:
taxi at (2, 2), passenger at Y, destination at R
```

By the above picture and the map given by the website, we can know the step should be left→left→down→down→pick→up→up→up→up→drop, so the rewards will be 9 steps of -1 and 1 steps of 20.

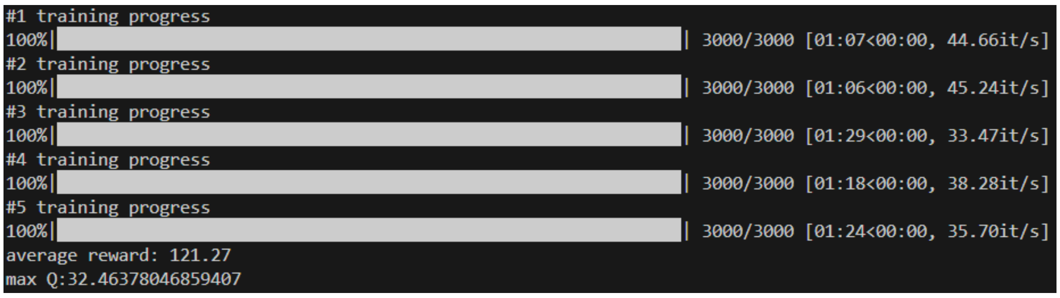
$$\text{Optimal Q} - \text{value: } \frac{(-1) \times (1-\gamma^9)}{1-\gamma} + 20 \times \gamma^9 = \frac{(-1) \times (1-0.9^9)}{1-0.9} + 20 \times 0.9^9 = 1.622614\ldots$$



```
#1 training progress
100%|██████████████████████████████████████| 3000/3000 [00:18<00:00, 162.62it/s]
#2 training progresswork4\AI_HW4>
100%|██████████████████████████████████████| 3000/3000 [00:19<00:00, 157.87it/s]
#3 training progress
100%|██████████████████████████████████████| 3000/3000 [00:17<00:00, 168.02it/s]
#4 training progress
100%|██████████████████████████████████████| 3000/3000 [00:17<00:00, 167.23it/s]
#5 training progress
100%|██████████████████████████████████████| 3000/3000 [00:17<00:00, 168.25it/s]
average reward: 8.17
Initail state:
taxi at (2, 2), passenger at Y, destination at R
max Q:1.6226146700000021
```

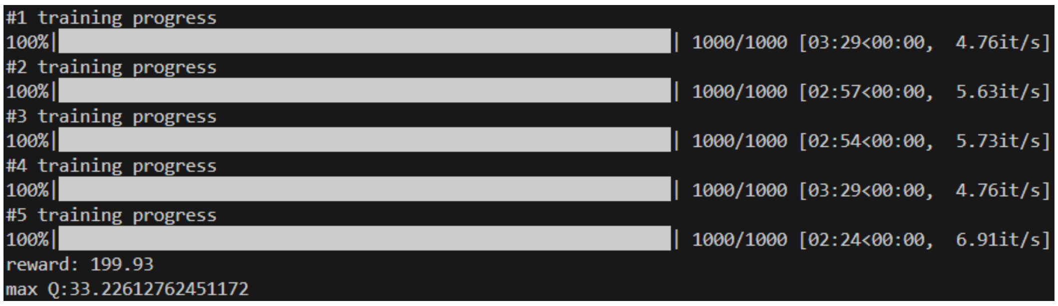
Compare with the max Q, which is the Q-value I learned, the result is really closed.

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%)**



```
#1 training progress
100%|████████████████████████████████████████| 3000/3000 [01:07<00:00, 44.66it/s]
#2 training progress
100%|████████████████████████████████████████| 3000/3000 [01:06<00:00, 45.24it/s]
#3 training progress
100%|████████████████████████████████████████| 3000/3000 [01:29<00:00, 33.47it/s]
#4 training progress
100%|████████████████████████████████████████| 3000/3000 [01:18<00:00, 38.28it/s]
#5 training progress
100%|████████████████████████████████████████| 3000/3000 [01:24<00:00, 35.70it/s]
average reward: 121.27
max Q:32.46378046859407
```

cartpole.py



```
#1 training progress
100%|███████████████████████████████████████| 1000/1000 [03:29<00:00,  4.76it/s]
#2 training progress
100%|███████████████████████████████████████| 1000/1000 [02:57<00:00,  5.63it/s]
#3 training progress
100%|███████████████████████████████████████| 1000/1000 [02:54<00:00,  5.73it/s]
#4 training progress
100%|███████████████████████████████████████| 1000/1000 [03:29<00:00,  4.76it/s]
#5 training progress
100%|███████████████████████████████████████| 1000/1000 [02:24<00:00,  6.91it/s]
reward: 199.93
max Q:33.22612762451172
```

DQN.py

Since reward will +1 for each step, and the truncation is executed for episode length greater than 200, we can calculate the optimal Q-value as below with gamma equals 0.97.

$$\text{Optimal Q} - \text{value: } 1 + 0.97 + 0.97^2 + \cdots + 0.97^{199} = \frac{1 - 0.97^{200}}{1 - 0.97} = \frac{1 - 0.97^{200}}{0.03} = 33.2579\cdots$$

Compare with the max Q in cartpole.py and DQN.py, it is clear that the max Q in DQN.py is closer to the optimal Q-value.

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

a. Because the state is continuous, we need to discretize the observation to build a Q-table. Otherwise, when implementing Q-learning, the Q-table would become too large, resulting in inefficient methods.

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

b. The performance will become better, since the discretized data would be more accurate to the real data, i.e., similar to the observation. And, the difference between similar data could be noticed.

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

c. If we increase it too much, the Q-table will be too large which may increase computational time and space complextity.

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

DQN performs better in Cartpole-v0. And here are the reasons:

1. DQN could directly use the continuous data. To compare with, while applying

Q-learning, it is needed to discretize data first. However, this will cause the loss of information.

2. DQN make the training process more stable by using the neural network, while Q-learning will cause overestimate.

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

   a. To balance the exploration and exploitation. By using epsilon greedy, it can choose the random action to explore and get more information, or choose the best action with max Q-value to exploit.

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

   b. If we only use exploration which randomly choose the actions, the convergence would occur too fast and we cannot get the best solution since we would not choose the best action in the Q-table.
   If we only use exploitation, we will choose the actions that is known, then we may sometimes miss the unknown but having high performance conditions.

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

   c. Yes, it is possible. If we can find some algorithm which can balance the exploration and exploitation, we can achieve the same performance without the epsilon greedy algorithm. For example, Boltzmann Exploration can do this.

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

   d. In the testing section, since it is to check the performance, there is no need to explore more. Thus, epsilon greedy algorithm is not needed.

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

   Since we do not need to compute gradient when choosing actions, we use torch.no_grad() to disable gradient checking and ensure using Q-value function without affecting the performance.