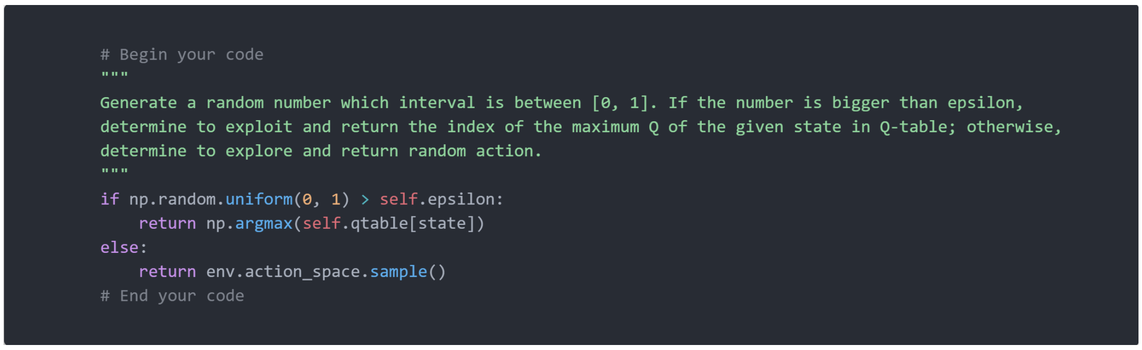
HW4: Reinforcement Learning

110550093 蔡師睿

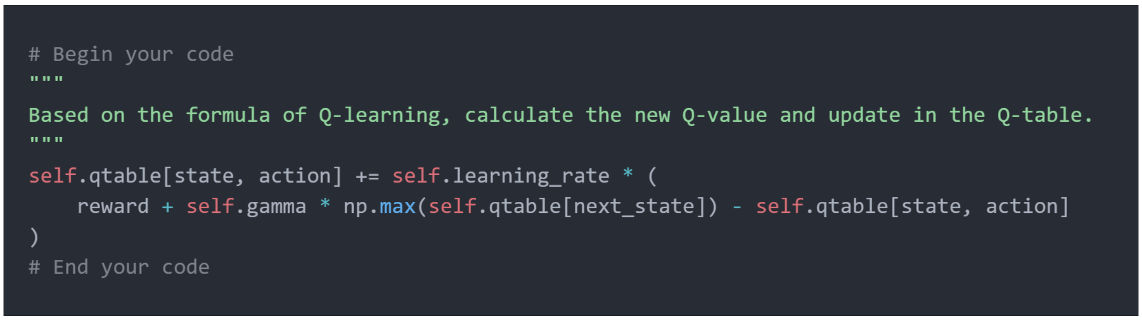
Part I. Implementation

Part 1: Q-learning in Taxi-v3



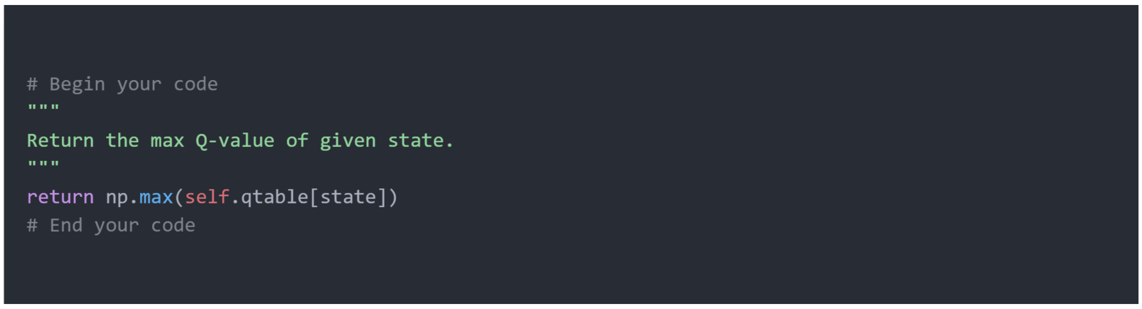
```python
# Begin your code
"""
Generate a random number which interval is between [0, 1]. If the number is bigger than epsilon,
determine to exploit and return the index of the maximum Q of the given state in Q-table; otherwise,
determine to explore and return random action.
"""
if np.random.uniform(0, 1) > self.epsilon:
    return np.argmax(self.qtable[state])
else:
    return env.action_space.sample()
# End your code
```

$choose\_action(self, state)$



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

$learn(self, state, action, reward, next\_state, done)$



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

$choose\_max\_Q(self, state)$

Part 2: Q-learning in Cartpole-v0

```python
# Begin your code
"""
Slice the interval based on the given lower_bound, upper_bound, and num_bins. Because np.linspace()
will contain the lower_bound, return the list started from index 1.
"""
return np.linspace(lower_bound, upper_bound, num_bins, endpoint=False)[1:]
# End your code
```

$$init\_bins(self, lower\_bound, upper\_bound, num\_bins)$$

```python
# Begin your code
"""
Discretize the value with given bins. Using np.digitize() to determine the value in which interval of bins.
"""
return np.digitize(value, bins, right=False)
# End your code
```

$$discretize\_value(self, value, bins)$$

```python
# Begin your code
"""
Discretize the continuous observation. Using the function "discretize_value()" to get the discretized
data of the 4 features in observation.
"""
return [self.discretize_value(observation[i], self.bins[i]) for i in range(4)]
# End your code
```

$$discretize\_observation(self, observation)$$

```python
# Begin your code
"""
Generate a random number which interval is between [0, 1]. If the number is bigger than epsilon,
determine to exploit and return the index of the maximum Q of the given state in Q-table; otherwise,
determine to explore and return random action.
"""
if np.random.uniform(0, 1) > self.epsilon:
    return np.argmax(self.qtable[tuple(state)])
else:
    return env.action_space.sample()
# End your code
```

$$choose\_action(self, state)$$

```python
# Begin your code
"""
Based on the formula of Q-learning, calculate the new Q-value and update in the Q-table.
"""
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
```

$$learn(self, state, action, reward, next\_state, done)$$

```
# Begin your code
"""
Return the max Q-value of given state, and before returning the max Q-value, the initial state should be dicretized first.
"""
return np.max(self.qtable[tuple(self.discretize_observation(self.env.reset()))])
# End your code
```

$check\_max\_Q(self)$

Part 3: DQN in Cartpole-v0

```
# Begin your code
"""
Sample trajectories of batch size from the replay buffer. Use function "sample" defined in class
"replay_buffer" to get the sampled data. Then, convert these sampled data into tensor.
"""
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)

"""
Forward the data to the evaluate net and the target net. Variable "q_eval" is predicted values from
evaluate network which is extracted based on "action". Variable "q_next" is actual values from target
network. Variable "q_target" is the expected Q-values obtained from the formula "reward + gamma * max(q_next)".
"""
q_eval = self.evaluate_net(states).gather(1, actions)
q_next = self.target_net(next_states).detach() * (~done).unsqueeze(-1)
q_target = rewards.unsqueeze(-1) + self.gamma * q_next.max(1)[0].view(self.batch_size, 1)

"""
Compute the loss of "q_eval" and "q_target" with nn.MSELoss().
"""
loss_func = nn.MSELoss()
loss = loss_func(q_eval, q_target)

"""
Zero-out the gradients before doing backpropagation, or the gradient would be a combination of
the old gradient. Next, do backpropagation, and update the parameters.
"""
self.optimizer.zero_grad()
loss.backward()
self.optimizer.step()
# End your code
```

$learn(self)$

```
# Begin your code
"""
Generate a random number which interval is between [0, 1]. If the number is bigger than epsilon,
determine to exploit and return the index of the maximum Q of the given state forwarded by the
evaluate neural network; otherwise, determine to explore and return random action.
"""
if np.random.uniform(0, 1) > self.epsilon:
    action = torch.argmax(
        self.evaluate_net(torch.tensor(state, dtype=torch.float))
    ).item()
else:
    action = env.action_space.sample()
# End your code
```

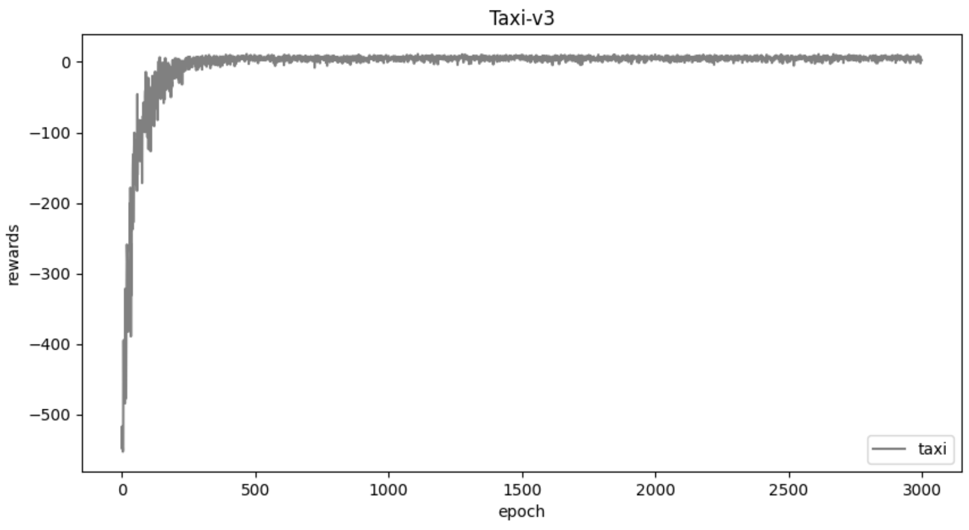
$choose\_action(self, state)$

```
# Begin your code
"""
First, convert the initial state to tensor. Next, forward the tensor in the target neural network.
Then, find the max Q-value and return it.
"""
x = torch.unsqueeze(torch.tensor(self.env.reset(), dtype=torch.float), 0)
return torch.max(self.target_net(x)).item()
# End your code
```

$check\_max\_Q(self)$

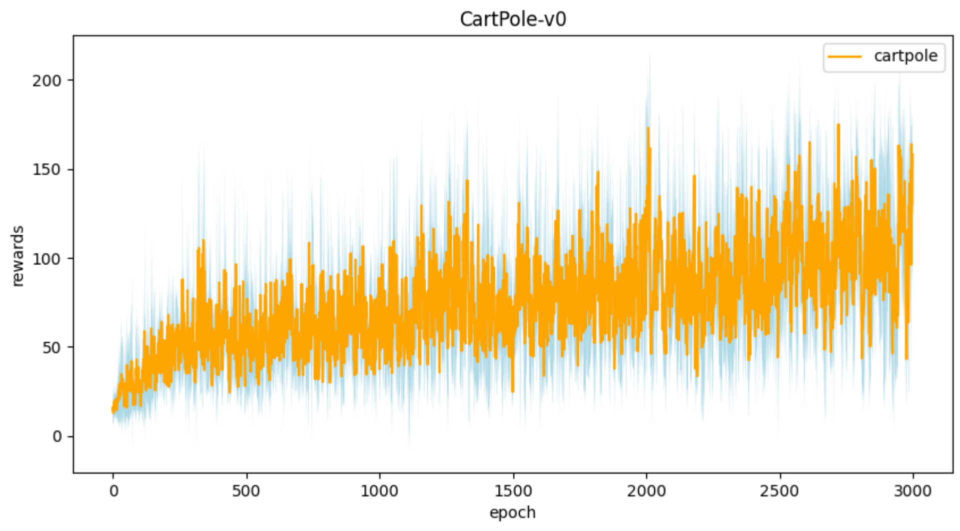
Part II. Experiment Results

Q-learning in Taxi-v3



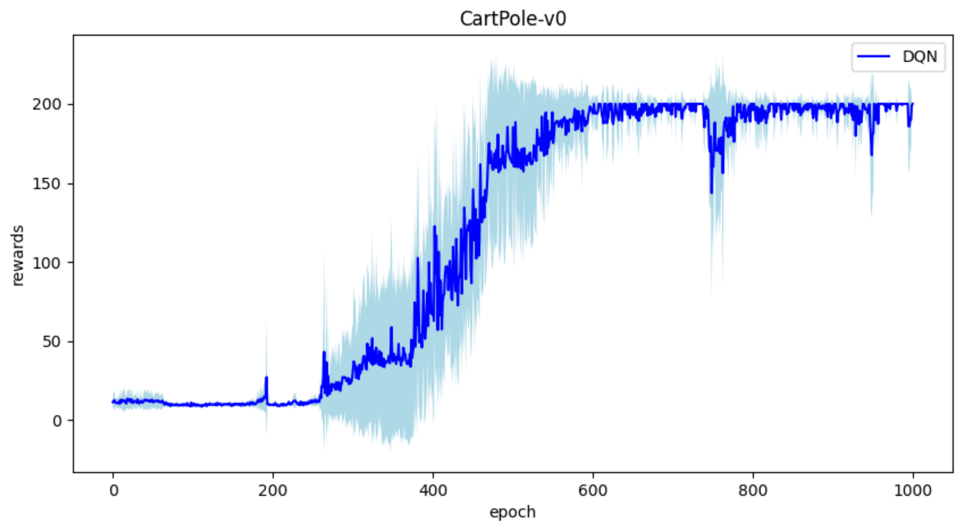
taxi.png

Q-learning in Cartpole-v0



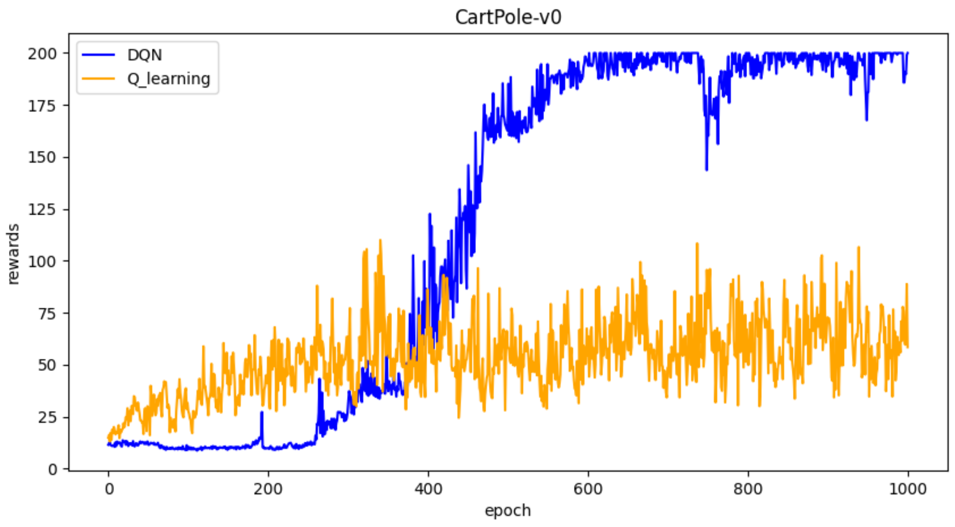
cartpole.png

DQN in Cartpole-v0



DQN.png

Compare Q-learning with DQN

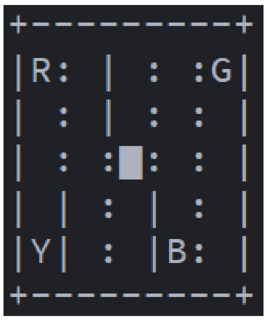


compare.png

Part III. Question Answering

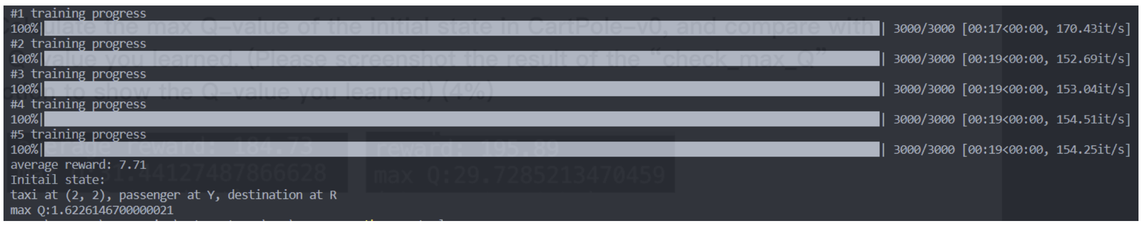
1. Calculate the optimal Q-value of a given state in Taxi-v3, and compare with the Q-value you learned.

Due to taxi at (2, 2), passenger at Y, and destination at R, the step would be left->left->down->down->pick->up->up->up->up->drop. Therefore, the rewards are 9 steps of -1 and 1 step of 20.



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

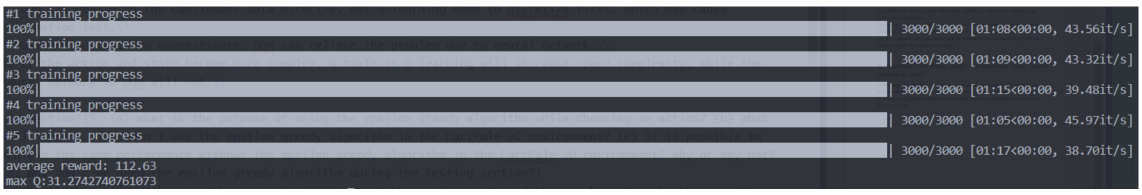
Optimal Q-value: $\frac{-1\times(1-\gamma^9)}{1-\gamma} + 20 \times (\gamma^9) = \frac{-1\times(1-0.9^9)}{1-0.9} + 20 \times (0.9^9) = 1.622614...$, which is close to the max Q.



```
#1 training progress
100%|████████████████████████████████████████| 3000/3000 [00:17<00:00, 170.43it/s]
#2 training progress
100%|████████████████████████████████████████| 3000/3000 [00:19<00:00, 152.69it/s]
#3 training progress
100%|████████████████████████████████████████| 3000/3000 [00:19<00:00, 153.04it/s]
#4 training progress
100%|████████████████████████████████████████| 3000/3000 [00:19<00:00, 154.51it/s]
#5 training progress
100%|████████████████████████████████████████| 3000/3000 [00:19<00:00, 154.25it/s]
average reward: 7.71
Initail state:
taxi at (2, 2), passenger at Y, destination at R
max Q:1.6226146700000021
```

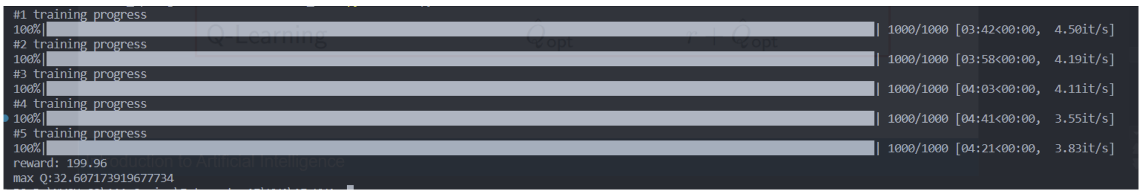
2. Calculate the max Q-value of the initial state in CartPole-v0, and compare with the Q-value you learned.

Optimal Q-value: $\frac{1-\gamma^{average\_\_reward}}{1-\gamma} \approx \frac{1}{1-0.97} = 33.333...$, which is close to the max Q of Q-learning and DQN, yet DQN is more closer.



```
#1 training progress
100%|████████████████████████████████████████| 3000/3000 [01:08<00:00, 43.56it/s]
#2 training progress
100%|████████████████████████████████████████| 3000/3000 [01:09<00:00, 43.32it/s]
#3 training progress
100%|████████████████████████████████████████| 3000/3000 [01:15<00:00, 39.48it/s]
#4 training progress
100%|████████████████████████████████████████| 3000/3000 [01:05<00:00, 45.97it/s]
#5 training progress
100%|████████████████████████████████████████| 3000/3000 [01:17<00:00, 38.70it/s]
average reward: 112.63
max Q:31.2742740761073
```

Q-learning



```
#1 training progress
100%|████████████████████████████████████████| 1000/1000 [03:42<00:00,  4.50it/s]
#2 training progress
100%|████████████████████████████████████████| 1000/1000 [03:58<00:00,  4.19it/s]
#3 training progress
100%|████████████████████████████████████████| 1000/1000 [04:03<00:00,  4.11it/s]
#4 training progress
100%|████████████████████████████████████████| 1000/1000 [04:41<00:00,  3.55it/s]
#5 training progress
100%|████████████████████████████████████████| 1000/1000 [04:21<00:00,  3.83it/s]
reward: 199.96
max Q:32.607173919677734
```

DQN

3. (a) Why do we need to discretize the observation in Part 2? (b) How do you expect the performance will be if we increase "num_bins"? (c) Is there any concern if we increase "num_bins"?

(a) Since the states are continuous, we have to dicretize the observations first; otherwise, our Q-learning will become complex or inefficient.
(b) Become better, because the discretized data will be more similar to the observation.
(c) Oversampling may increase computational complexity and space complexity.

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

DQN performs better. Reasons :
1. DQN can deal with the continuous data directly, but Q-learning need to dicretize first, which may cause information lost.
2. Q-learning tends to overestimate; DQN can relieve the problem due to neural network.
3. If the action and state become more complex, Q-table in Q-learning will increase

space complexity, while the neural network in DQN will not.

5. (a) What is the purpose of using the epsilon greedy algorithm while choosing an action? (b) What will happen, if we don ' t use the epsilon greedy algorithm in the CartPole-v0 environment? (c) Is it possible to achieve the same performance without the epsilon greedy algorithm in the CartPole-v0 environment? Why or Why not? (d) Why don ' t we need the epsilon greedy algorithm during the testing section?

(a) In the beginning, the agent has none knowledge about the environment, and thus we have to make the agent to explore the environment. However, we still need to ensure the agent to exploit the best, so we use epsilon greedy algorithm to ensure our agent able to explore and exploit.
(b) Only work on the known information, which may miss some unknown high-performance conditions.
(c) Yes, it is possible. There are several alternative algorithms that can be used instead of epsilon-greedy: Upper Confidence Bound, Boltzmann Exploration, etc.
(d) After trained, our agent has enough information to the environment and it's no need to explore the environment again.

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

In this step, we are going to select an action using neural network to estimate the next possible actions, so we don't need to calculate the gradient. Besides, using *torch.no_grad()* helps optimize memory usage, speed up computations, and ensure the correctness.