Homework 4:

Reinforcement Learning

110550085房天越

Part I. Implementation (-5 if not explain in detail):

1. taxi.py

```python
# Begin your code
# TODO
'''
Here I use a random r to determine whether
to choose the argmax of the current state
or to randomly return an action in the
action space.
'''
r=np.random.uniform(0,1)
if(r>self.epsilon):
    return np.argmax(self.qtable[state])
else:
    return env.action_space.sample()
# End your code
```

```python
# Begin your code
# TODO
'''
Here I update the Q_table with the function given in the slide, and use (1-done) to control whether to set the gamma part to 0.
'''
self.qtable[state, action]+=self.learning_rate*(reward+self.gamma*(1-done)*np.max(self.qtable[next_state])-self.qtable[state, action])
# End your code
```

```python
# Begin your code
# TODO
'''
Here return the maximum value of the Q_table of the given state.
'''
max_q= max(self.qtable[state])
return max_q
# End your code
```

2. cartpole.py

```python
# Begin your code
# TODO
'''
According to the hint, use linspace function to implement, and delete the 0th element.
'''
bins=np.linspace(upper_bound, lower_bound, num_bins, endpoint=False)
bins=np.delete(bins, 0)
return bins
# End your code
```

```python
# Begin your code
# TODO
'''
According to the hint, use digitize function to implement.
'''
return np.digitize(value, bins)
# raise NotImplementedError("Not implemented yet.")
# End your code
```

```python
# Begin your code
# TODO
'''
Discretize the continuous four features using the two functions
implemented above, append them to a list of state, and then
convert the list to tuple and return.
'''
state=[]
for i in range(len(observation)):
    state.append(self.discretize_value(observation[i], self.bins[i]))
return tuple(state)

# End your code
```

```python
# Begin your code
# TODO
'''
Choose action with the same way as implemented in taxi.py
'''
r=np.random.uniform(0,1)
if(r>self.epsilon):
    return np.argmax(self.qtable[state])
else:
    return self.env.action_space.sample()
# End your code
```

```python
# Begin your code
# TODO
'''
Learn with the same function as in taxi.py
'''
self.qtable[state][action]+=self.learning_rate*(reward+self.gamma*(1-done)*np.max(self.qtable[next_state])-self.qtable[state][action])
# End your code
```

```python
# Begin your code
# TODO
'''
Use the discretize_observation function to get
the state, and return the maximum of the Q_table.
'''
state=self.discretize_observation(self.env.reset())
return np.max(self.qtable[state])
# End your code
```

3. DQN.py

```python
# Begin your code
# TODO
'''
Following the steps, first sample the trajectories of batch size from self.buffer.
Set state, action, reward, next_state, and done to 0, 1, 2, 3, 4th, respectively,
with the most appropriate value type.
Then, recalculate the current Q_value of the given experiences as eva(evaluate_net),
and calculate target net (detach the get rid of training the target net), get the
result target.
Continuously, calculate loss funciton with MSELoss.
Finally, zero-out the gradients, and do back propagation and optimization.
'''
sample = self.buffer.sample(self.batch_size)

state = torch.tensor(np.array(sample[0]), dtype=torch.float32)
action = torch.tensor(sample[1],dtype=torch.long)
reward = torch.tensor(sample[2],dtype=torch.float32)
next_state = torch.tensor(np.array(sample[3]),dtype=torch.float32)
done = sample[4]

eva=self.evaluate_net(state).gather(1, action.unsqueeze(1))
+torch.LongTensor([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
tar=self.target_net(next_state).detach()
for i in range(self.batch_size):
    if(done[i]):
        tar[i]=0
target=reward+self.gamma*tar.max(1).values.unsqueeze(-1)

f_loss=nn.MSELoss()
loss=f_loss(eva.float(), target.float())

self.optimizer.zero_grad()

loss.backward()

self.optimizer.step()

# End your code
```

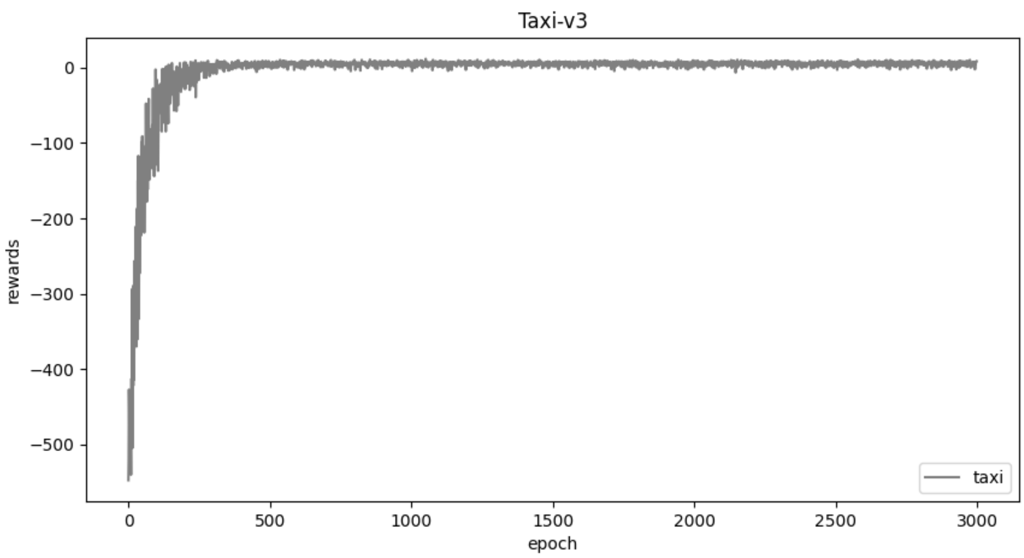
^actually the '+torch.LongTensor(…)' is right after the the gather function, I pressed enter here in order to screenshot.

```python
# Begin your code
# TODO
'''
Use a similar strategy with the other two games.
If the randomly picked number is smaller than epsilon, then
randomly choose an action; If not, choose the maximum of the
given state's evaluate net.
'''
r=np.random.uniform(0,1)
if(r<self.epsilon):
    action=np.random.randint(0, self.n_actions)
else:
    state=torch.as_tensor(state, dtype=torch.float32)
    maxi=torch.argmax(self.evaluate_net.forward(state), dim=0)
    action=int(maxi)
# End your code
```

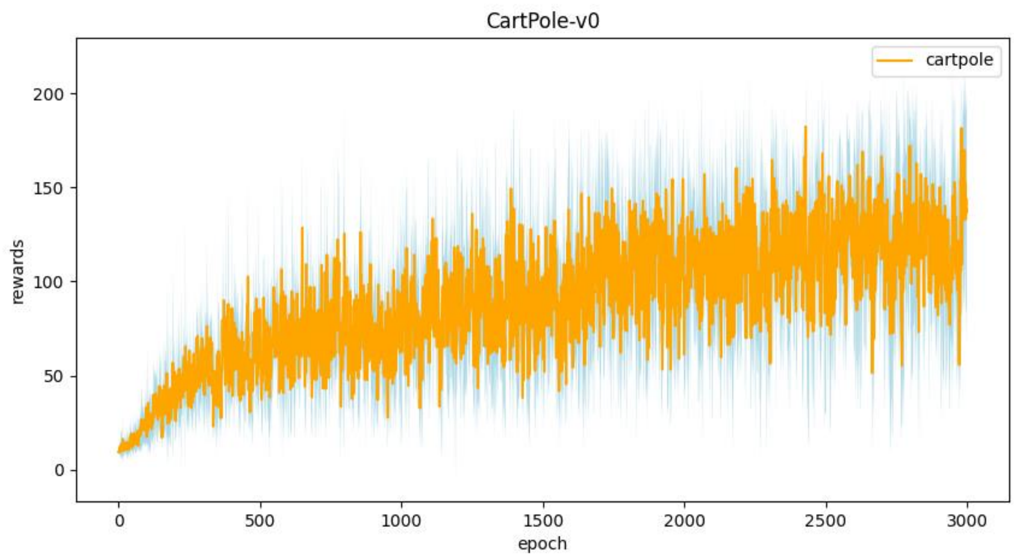
```python
# Begin your code
# TODO
'''
Return max_q, which is the maximum forwarded value of self.evaluate_net.
'''
fl=torch.tensor(env.reset(), dtype=torch.float32)
fw=self.evaluate_net.forward(fl)
max_q=torch.max(fw)
return max_q
# End your code
```

Part II. Experiment Results:

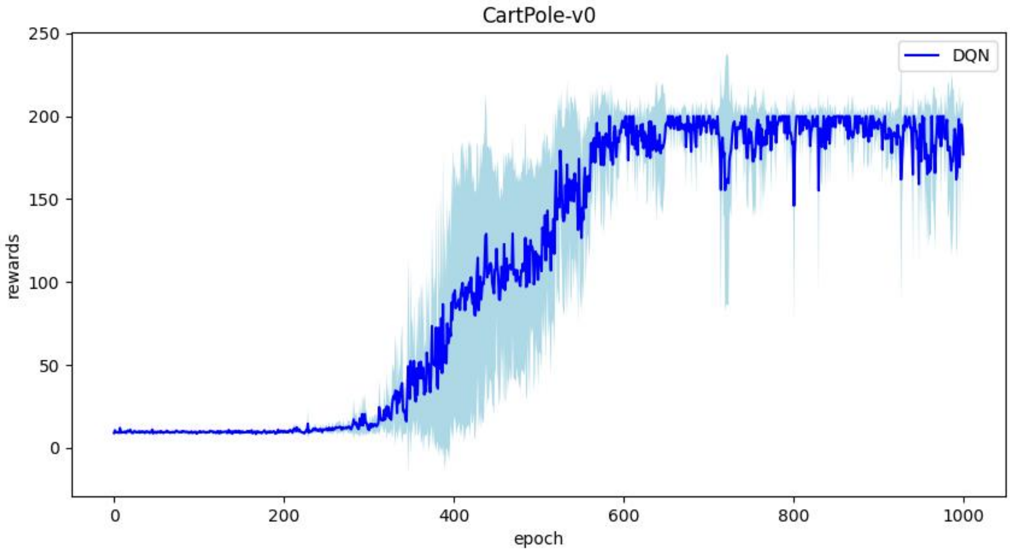
1. taxi.png:



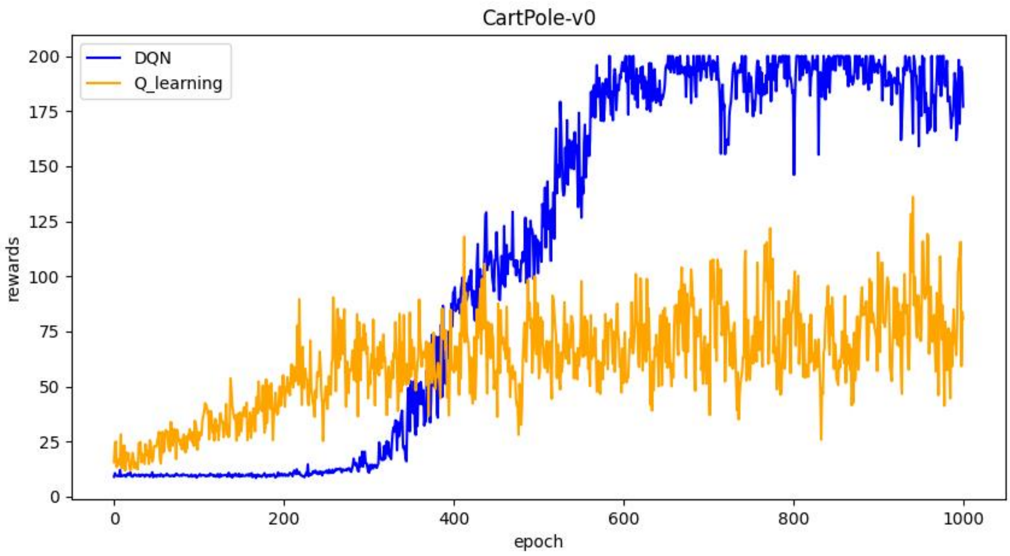
2. cartpole.png



3. DQN.png



4. compare.png



Part III. Question Answering (50%):

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

```
average reward: 8.11
Initail state:
taxi at (2, 2), passenger at Y, destination at R
max Q:1.6226146700000021
```

According to the given state, the optimal solution is left, left, down, down, pick, up, up, up, up drop, with rewards=9*(-1)+20=11.
And the optimal Q value is (-1)*(1-gamma^9)/(1-gamma)+20*gamma^9, with gamma=0.9, the optical value is close to the max Q, which is about 1.6226…

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) **(10%)**

```
(python37) C:\Users\jerem\OneDrive\桌面\AI_HW4>python cartpole.py
#1 training progress
100%|████████████████████████████████| 3000/3000 [02:10<00:00, 22.97it/s]
#2 training progress
100%|████████████████████████████████| 3000/3000 [02:11<00:00, 22.80it/s]
#3 training progress
100%|████████████████████████████████| 3000/3000 [02:04<00:00, 24.11it/s]
#4 training progress
100%|████████████████████████████████| 3000/3000 [02:16<00:00, 22.02it/s]
#5 training progress
100%|████████████████████████████████| 3000/3000 [02:05<00:00, 23.90it/s]
average reward: 180.17
max Q:29.428661566976093
```

```
(python37) C:\Users\jerem\OneDrive\桌面\AI_HW4>python DQN.py
#1 training progress
100%|████████████████████████████████| 1000/1000 [06:05<00:00,  2.74it/s]
#2 training progress
100%|████████████████████████████████| 1000/1000 [04:47<00:00,  3.48it/s]
#3 training progress
100%|████████████████████████████████| 1000/1000 [06:08<00:00,  2.72it/s]
#4 training progress
100%|████████████████████████████████| 1000/1000 [05:52<00:00,  2.84it/s]
#5 training progress
100%|████████████████████████████████| 1000/1000 [05:11<00:00,  3.21it/s]
reward: 200.0
max Q:0.06077925115823746
```

The optimal Q_value is (1-gamma^average_reward)/(1-gamma)~=1/0.03=33.33…
The max_Q I got in cartpole.py is slightly lower than the optimal value, and that for DQN is near 0.

3.

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

      The data we observe are continuous, if we don't discretize, our learning would be inefficient or even impossible.

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

      I think that increasing "num_bins" might make the performance better because by doing so, we can better describe our states.

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

      Yes, increasing num_bins means more space requirement, which would increase the time complexity and space complexity.

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

   DQN performs better than Q-learning.

The reasons may include the following:
First is that DQN can deal with the continuous data, however, in DQN, we have to discretize first, which may lead to leaking of identity. Second is that the learning methods of the two ways are different, one is using limited Q table, while the another one is using neural network, which would not be that limited after a lot of iterations.

5.

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

   The main purpose is to balance between explore and exploit. We can explore randomly and maintaining the maximum choice at the same time.

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

   Without explore, we might lose some possibilites for high performance.
   And without exploit, we are just walking around randomly like a stray sheep.

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 yes, if we can find some other algorithm to maintain the similar ratio of explore and exploit, we might be able to achieve the same performance.

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

   The agent is already trained and do not need randomness anymore.

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

   We don't need to calculate gradient when choosing action, so it helps us skipping the calculation, and speed up the computation.