Experiment 2

211220166 王诚昊

1.实验过程

依据算法原理，最终完成了：完全分布式ddpg_il.py，完全集中式ddpg_centralized.py，以及maddpg.py。并补全了对应的验证脚本，结合TensorBoard提供的数据可视化效果对相应训练结果做出评估。

1. ddpg_il.py

   ```
   for _ in range(20):
       for i in range(2):
           # TODO: update agents
           s, a, r, s_ = self._replay_buffer[i].sample(BATCH_SIZE)
           a_ = self._agent[i].query_target_action(s_)
           self._agent[i].update(s, a, r, s_, a_)
   ```

   更新agents的步骤与初始版本的ddpg算法类似，只是要注意现在有2个agent。第一步是从经验回放池中按照BATCH_SIZE取出一批样本；第二步中的query_target_action()函数实际就是调用目标actor网络，根据输入next_state确定next_action；第三步就是调用更新函数即可。

2. ddpg_centralized.py

   ```
   for _ in range(20):
       # TODO: update agents
       s, a, r, s_ = self._replay_buffer.sample(BATCH_SIZE)
       a_ = self._agent.query_target_action(s_)
       self._agent.update(s, a, r, s_, a_)
   ```

   完全集中式意味着训练和执行都由中央控制来执行，更新agent的操作与初始版本ddpg几乎完全一致，事实上唯一的修改在于把多个agent的奖励加起来作为系统的奖赏值，但这步也已经在框架代码中了:

   ```
   reward = reward.sum()
   ```

3. maddpg.py

   - critic loss

   ```
   # critic loss
   # TODO: add critic_loss
   q_loss_fn = torch.nn.MSELoss()
   q_loss = q_loss_fn(q_hat, q)
   ```

   阅读框架代码，发现已经给出了处理成一维张量的q和q_hat：

   ```
   q = r_tensor[:, ai] + 0.95 * self.target_critic[ai](n_sa_tensor).view(-1)
   ```

   ```
   q_hat = self.critic[ai](sa_tensor).view(-1)
   ```

那么直接调用MSE均方差函数即可。

- actor loss

注意到actor loss的计算是基于：

$$L(\theta^{\mu_i}) = -Q_i(x, a_1, \cdots, a_{i-1}, \mu_i(o_i; \theta^{\mu_i}), a_{i+1}, \cdots, a_n; \theta^{Q_i})$$

由于critic网络直接调用即可，故而关键在于通过**拼接（concatenate）**的方法来构造critic网络的输入，特别需要注意在拼接前调整张量的格式。

又注意到原理式中仅仅改变了：

$$a_i \to \mu_i(o_i; \theta^{\mu_i})$$

所以在循环中单独处理$i == a_i$的情况，使用actor网络来通过输入相应的x来输出得到对应动作；其余的i则将张量格式处理之后直接进行拼接即可。

```
# TODO: add actor_loss
current_state_inputs = [x_tensor]
for i in range(2):
    if i == ai:
        # use actor to get action_i
        new_action = self.actor[ai](x_tensor[:, ai * 3:ai * 3 + 3])
        current_state_inputs.append(new_action)
    else:
        # shape matters
        ith_action = a_tensor[:,i]
        ith_action_shaped = ith_action.view(-1,1)
        current_state_inputs.append(ith_action_shaped)
# concatenate
new_sa_tensor = torch.concatenate(current_state_inputs, dim = 1)
# use critic
a_loss = -self.critic[ai](new_sa_tensor).mean()
```

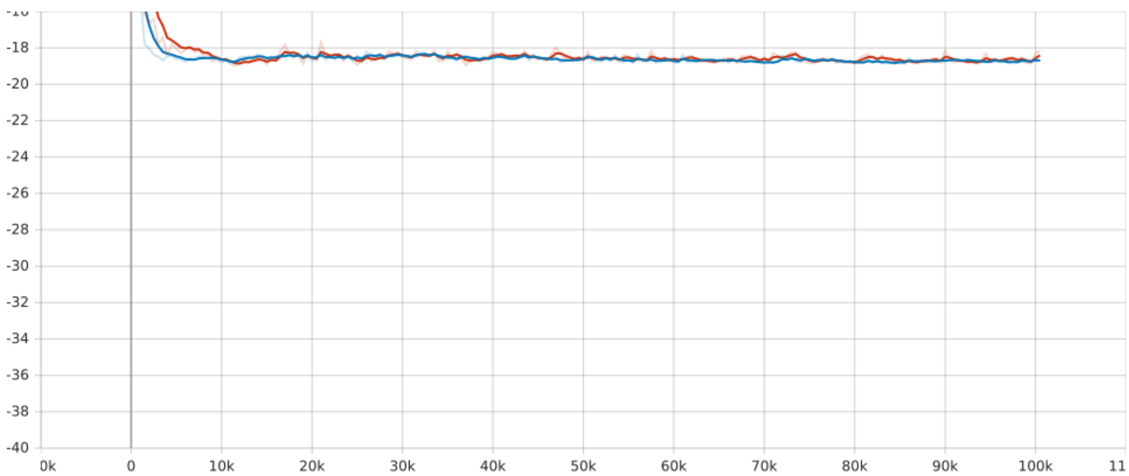
2.训练结果可视化展示

1. ddpg_il.py

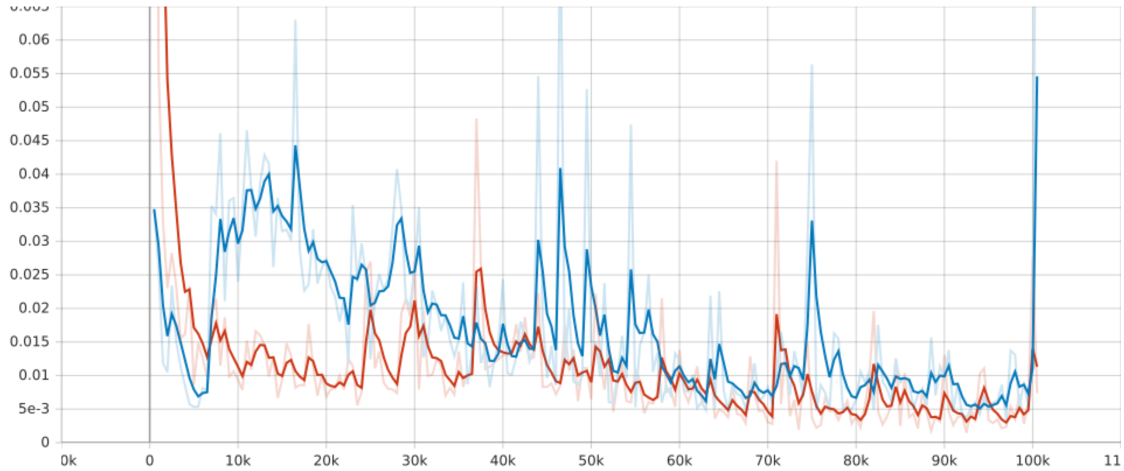
   evaluate:

```
PS D:\NJU_undergraduate\大三下\无人机\homework\Experiment2\src> python .\evaluate_ddpg_il.py
191.2308344748925
```

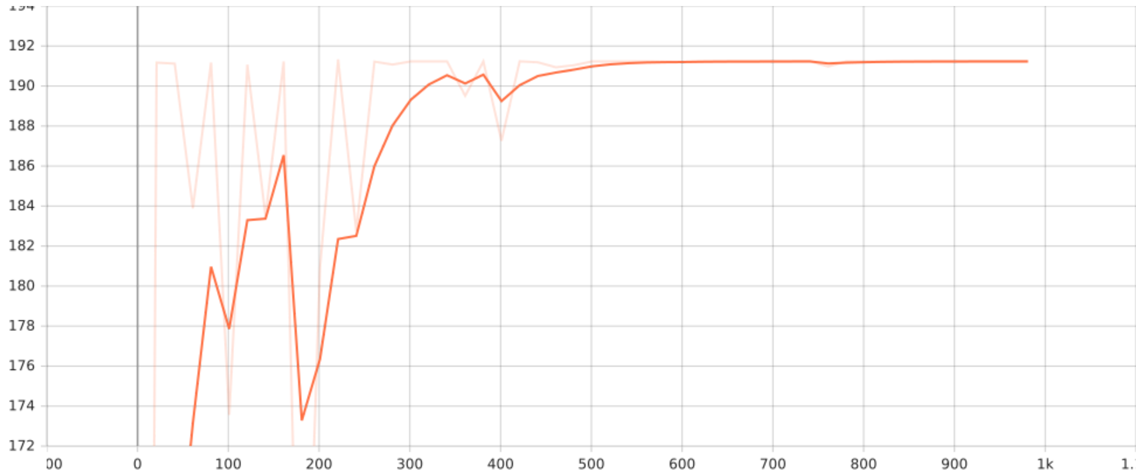
actor_loss:



cirtic_loss:



reward:

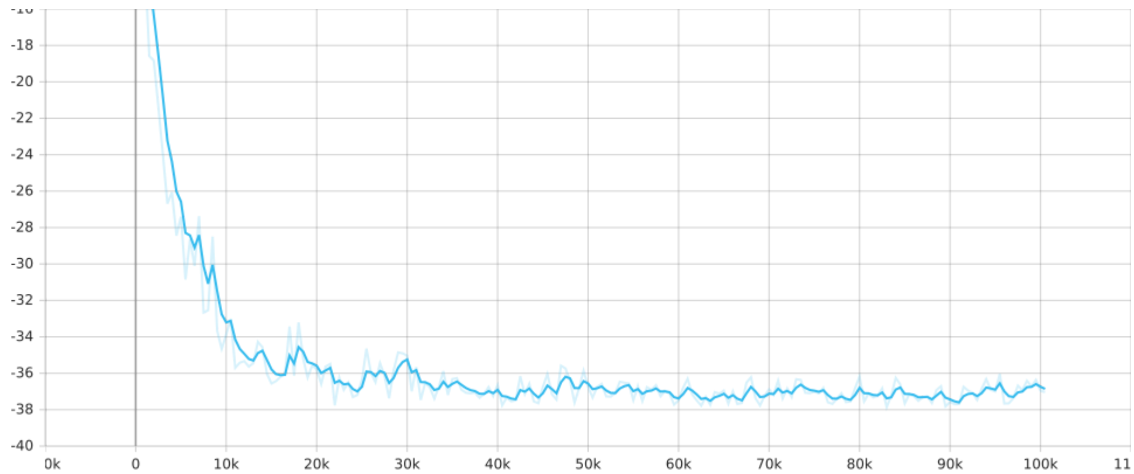


2. ddpg_centralized.py

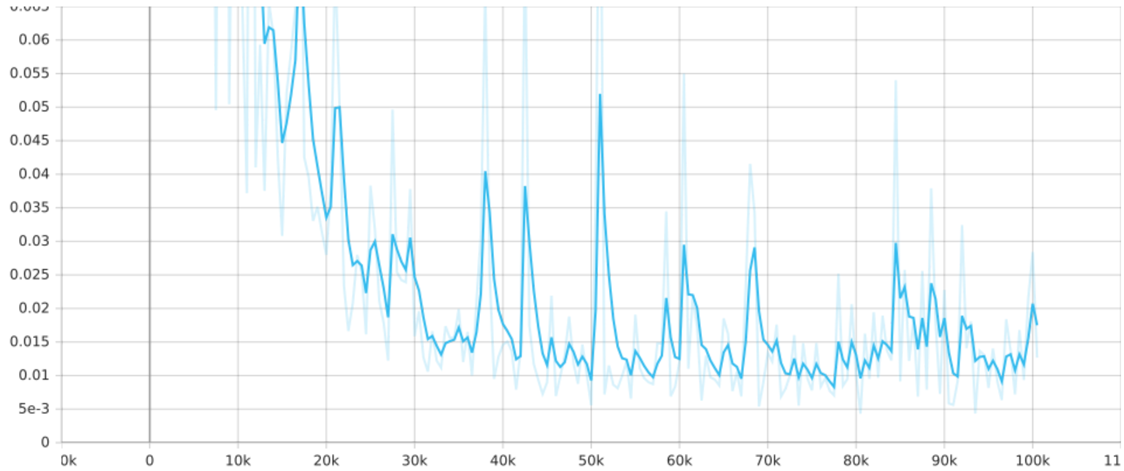
evaluate:

```
PS D:\NJU_undergraduate\大三下\无人机\homework\Experiment2\src> python .\evaluate_ddpg_centralized.py
191.23083112720045
```

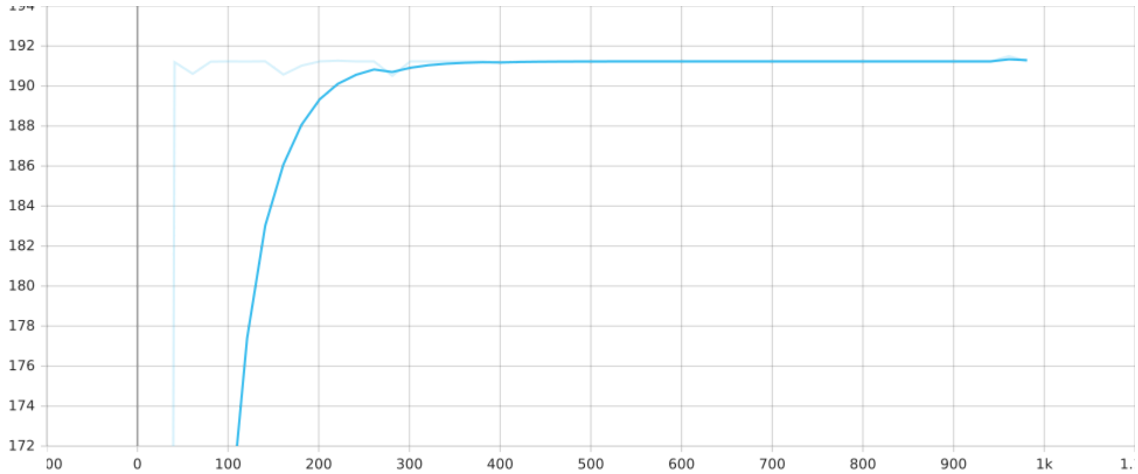
actor_loss:



cirtic_loss:

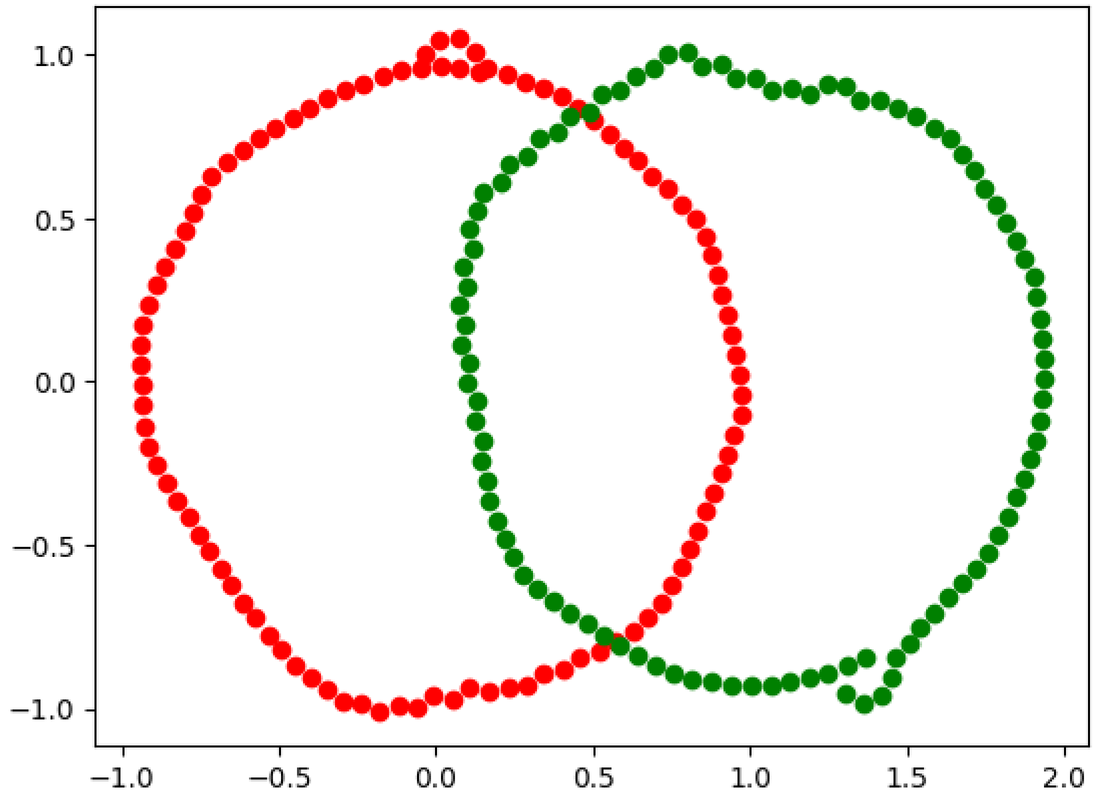


reward:

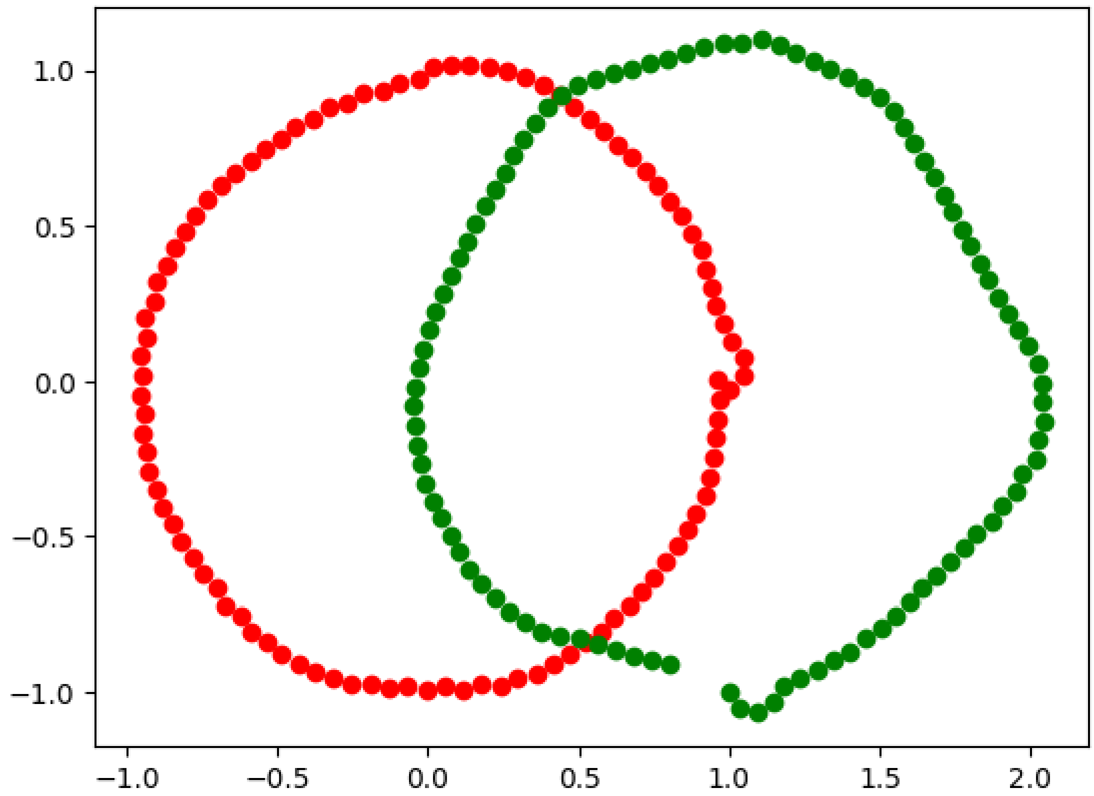


3. maddpg.py

```
models = torch.load(r'maddpg_2\75000.th')
```



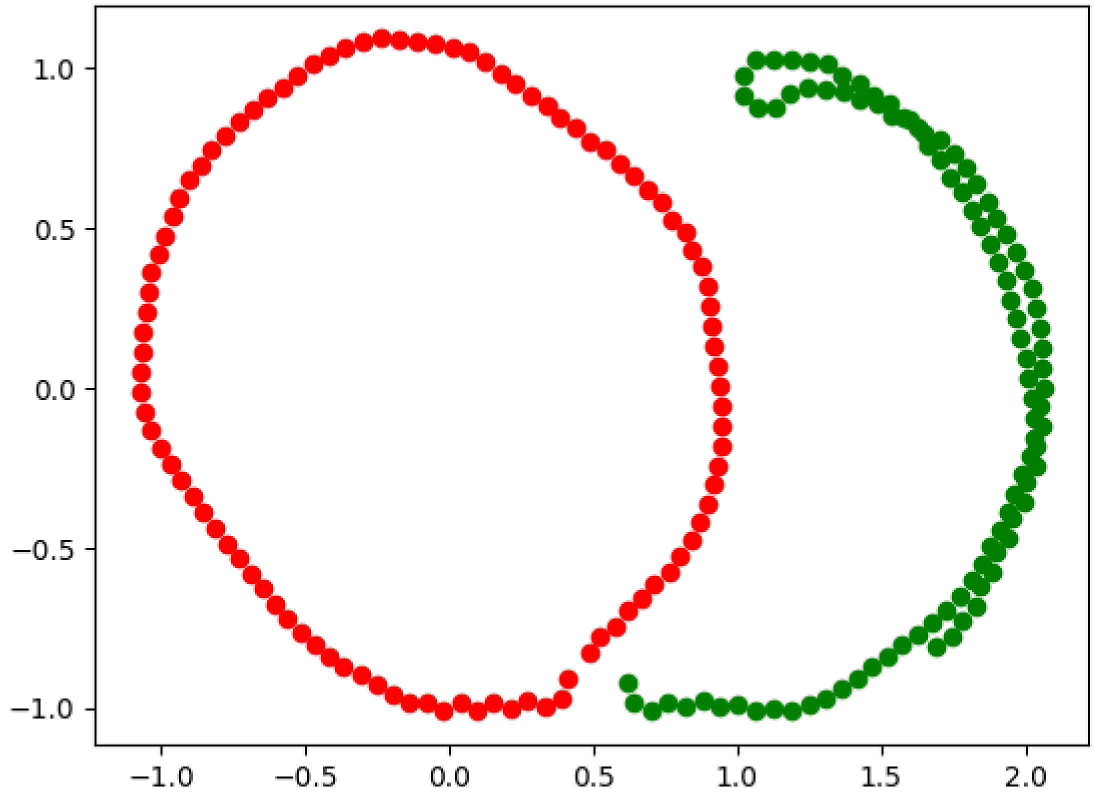

```
models = torch.load(r'maddpg\20000.th')
```



实验表明，训练时间越长效果不一定很好，梯度值在10^{-4}左右就很难再下降了，同时可能还存在过拟合问题。

比如：

```
models = torch.load(r'maddpg_2\80000.th')
```



3.疑问

框架代码中疑似有一处错误，在maddpg.py中，line110附近：

```
for i in range(2):
    with torch.no_grad():
        a_ = self.target_actor[i](x_tensor[:, i * 3:i * 3 + 3])  # hard coded
state -> obs
    next_state_inputs.append(a_)
```

x_tensor应该为nx_tensor，我在实验过程中已经修改。