Final project of Machine Learning of Engineering

基于MADDPG算法的集群机器人强化学习

陈逸飞12010502
龙嘉骏12011624
方艺钧12011301

2022. 12. 14

强化学习(RL)

强化学习是机器学习领域之一，受到行为心理学的启发，主要关注智能体如何在环境中采取不同的行动，以最大限度地提高累积奖励。

集群机器人(Swarm robotics)

涉及大集群机器人的设计、建造和部署，它们能够相互协调并协同解决问题或者执行任务。集群机器人的灵感来源于自然组织系统，例如社交昆虫，鱼类或鸟群

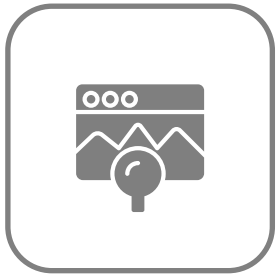
通过多种形式的学习情景来获得经验（竞争/合作/团队竞争/领导）

Our work

基于OpenAi的multiagent-particle-envs环境，我们实现多智能体强化学习MADDPG 算法。通过对论文的算法理解，按照我们的理解实现了算法（pytorch）并且对其进行了修改（即搭建了critic-actor神经网络）。我们所构建的MADDPG的算法部分，能够与我们自行构建的实验场景相对接，实现对于机器人之间合作/竞争/沟通等场景的学习。最后对于集群式机器人执行不同任务的学习效果进行分析和总结。

陈逸飞
（33%）

整体算法框架和研究思路的搭建



龙嘉骏
（33%）

实验仿真情景设计和可视化的实现

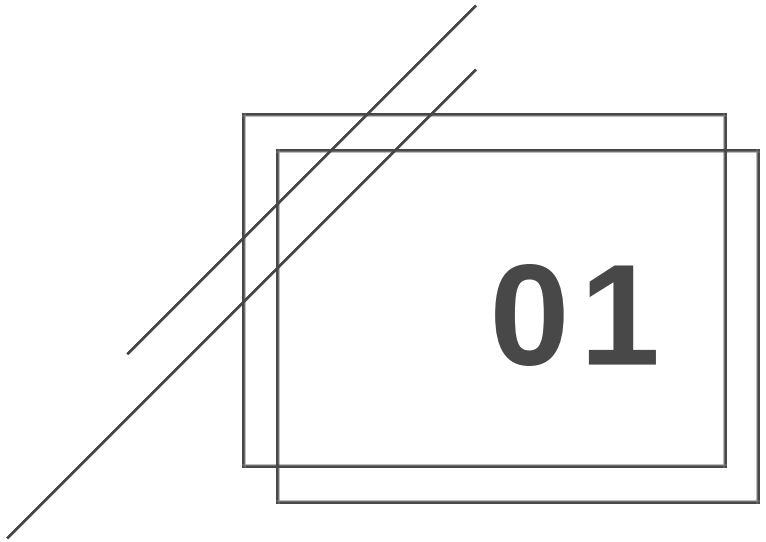
方艺钧
（33%）

算法的具体实现和学习的训练

还有1%感谢ddl

目录

CONTENTS



01

Expiation of the algorithm

算法解释

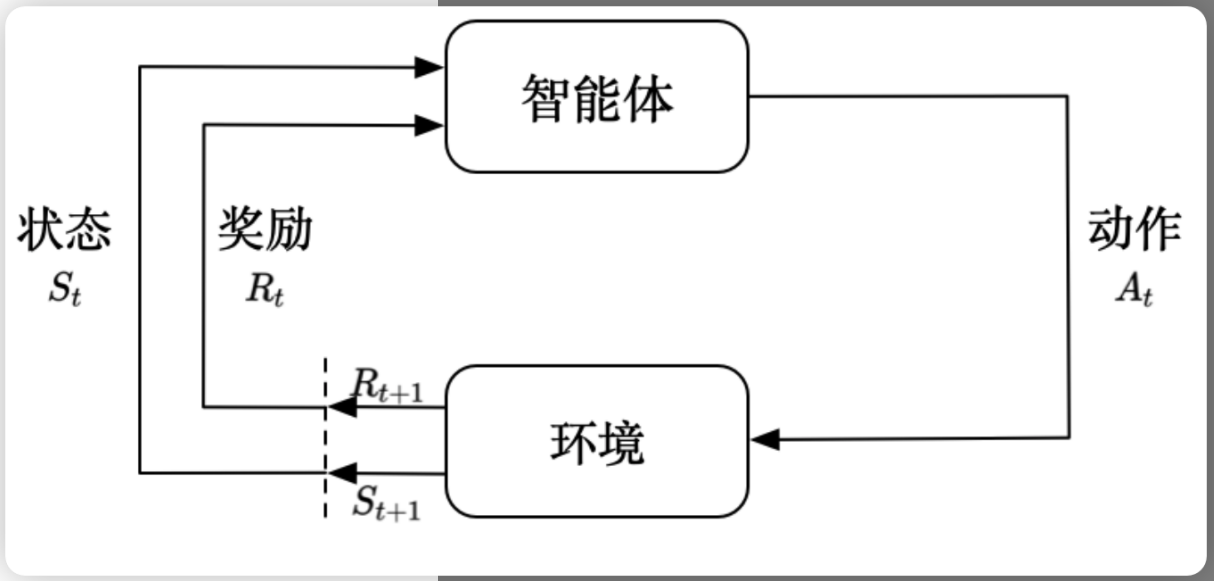
算法解释

Layer 1

第一层：强化学习由两部分组成：智能体(agents)和环境(env)。在强化学习过程中，智能体与环境一直在交互。

Layer 2

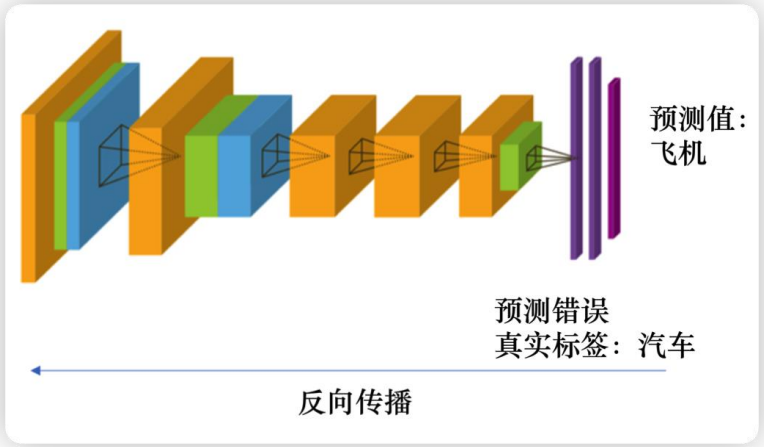
智能体在环境中获取某个状态后，它会利用该状态输出一个动作（action），这个动作也称为决策（decision）。这个动作会在环境中被执行，环境会根据智能体采取的动作，输出下一个状态以及当前这个动作带来的奖励。智能体的目的就是尽可能多地从环境中获取奖励。



算法解释

监督学习

- - 输入的数据（标注的数据）都应是没有关联的。因为如果输入的数据有关联，学习器（learner）是不好学习的。（要求独立同分布）
- - 需要告诉学习器正确的标签是什么，这样它可以通过正确的标签来修正自己的预测。

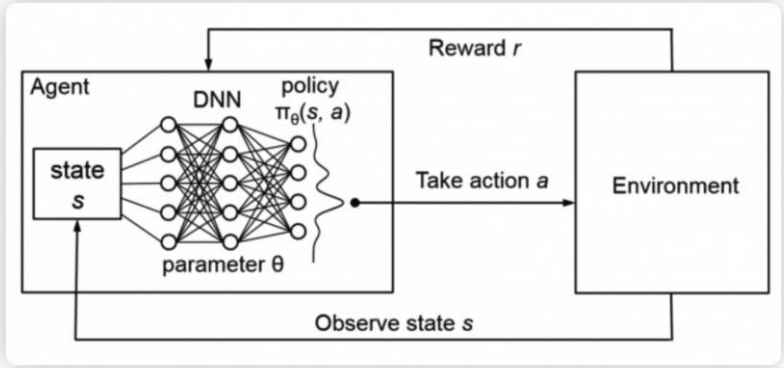


监督学习

强化学习

- 强化学习会试错探索，它通过探索环境来获取对环境的理解。
- 强化学习智能体会从环境里面获得延迟的奖励。
- 在强化学习的训练过程中，时间非常重要。因为我们得到的是有时间关联的数据（sequential data），而不是独立同分布的数据。

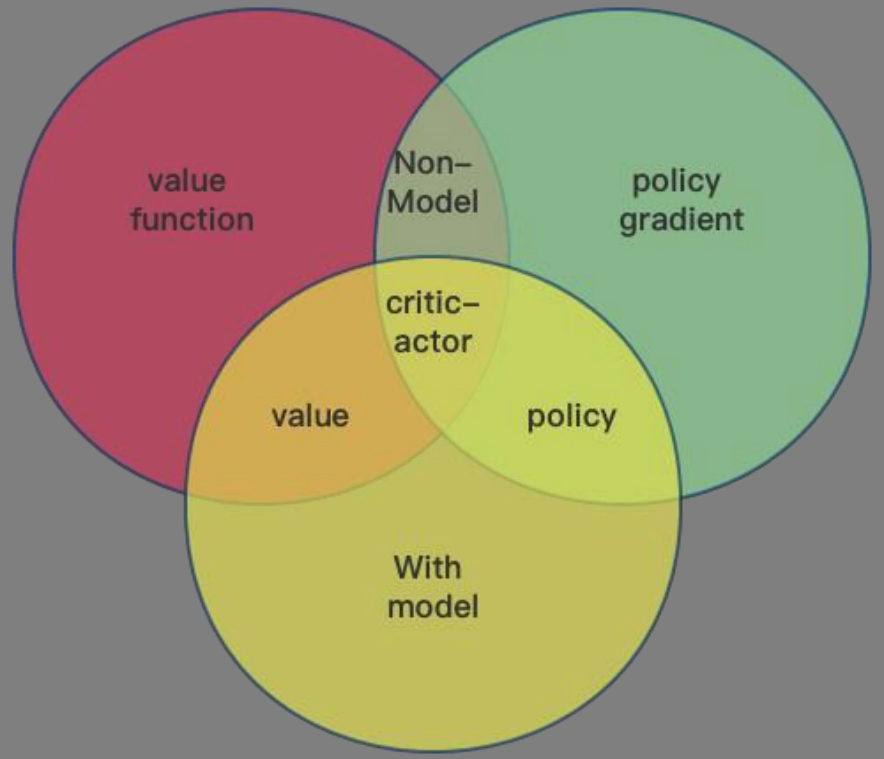
强化学习



算法解释

RL type

- **策略（policy）**。智能体会用策略来选取下一步的动作。
- **价值函数（value function）**。我们用价值函数来对当前状态进行评估。价值函数用于评估智能体进入某个状态后，可以对后面的奖励带来多大的影响。价值函数值越大，说明智能体进入这个状态越有 利。
- **模型（model）**。模型表示智能体对环境的状态进行理解，它决定了环境中世界的运行方式。



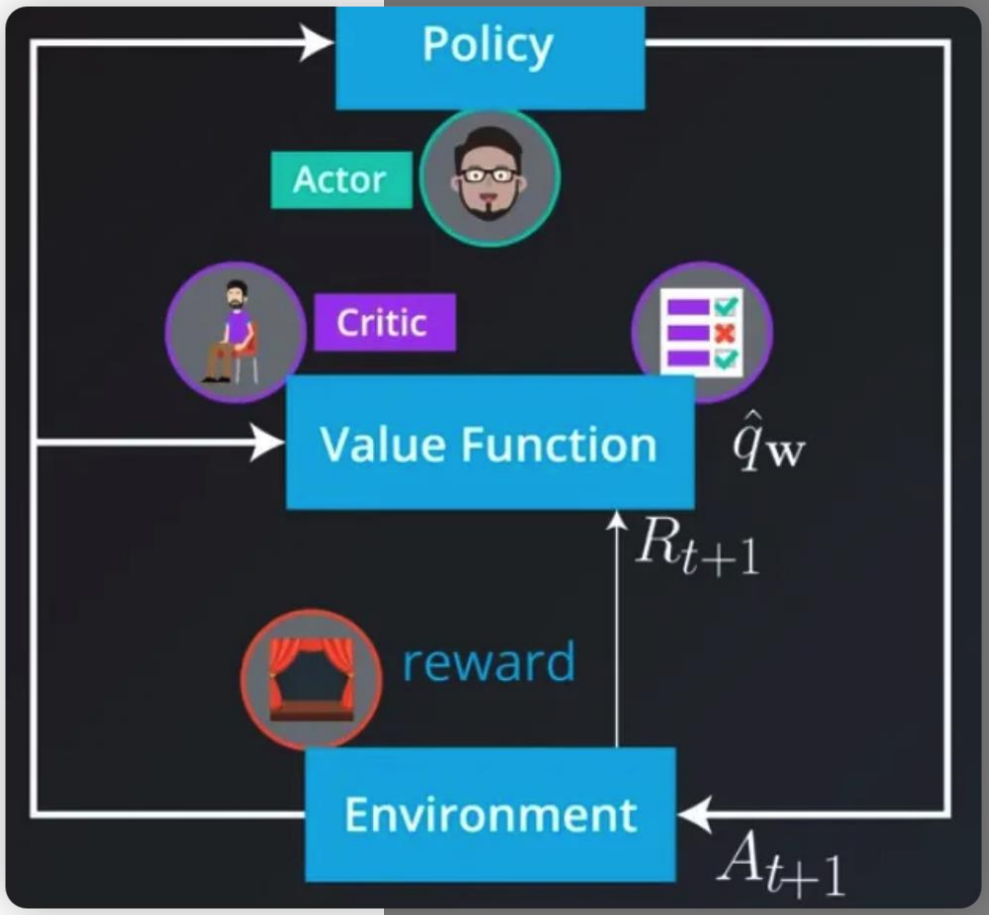
critic-actor

Value

基于价值的智能体（value-based agent）显式地学习价值函数，隐式地学习它的策略。策略是其从学到的价值函数里面推算出来的（价值函数，按照最大化这个函数值的动作来走）

Policy

基于策略的智能体（policy-based agent）直接学习策略（在每一个状态都得到一个最佳的动作）。



critic-actor

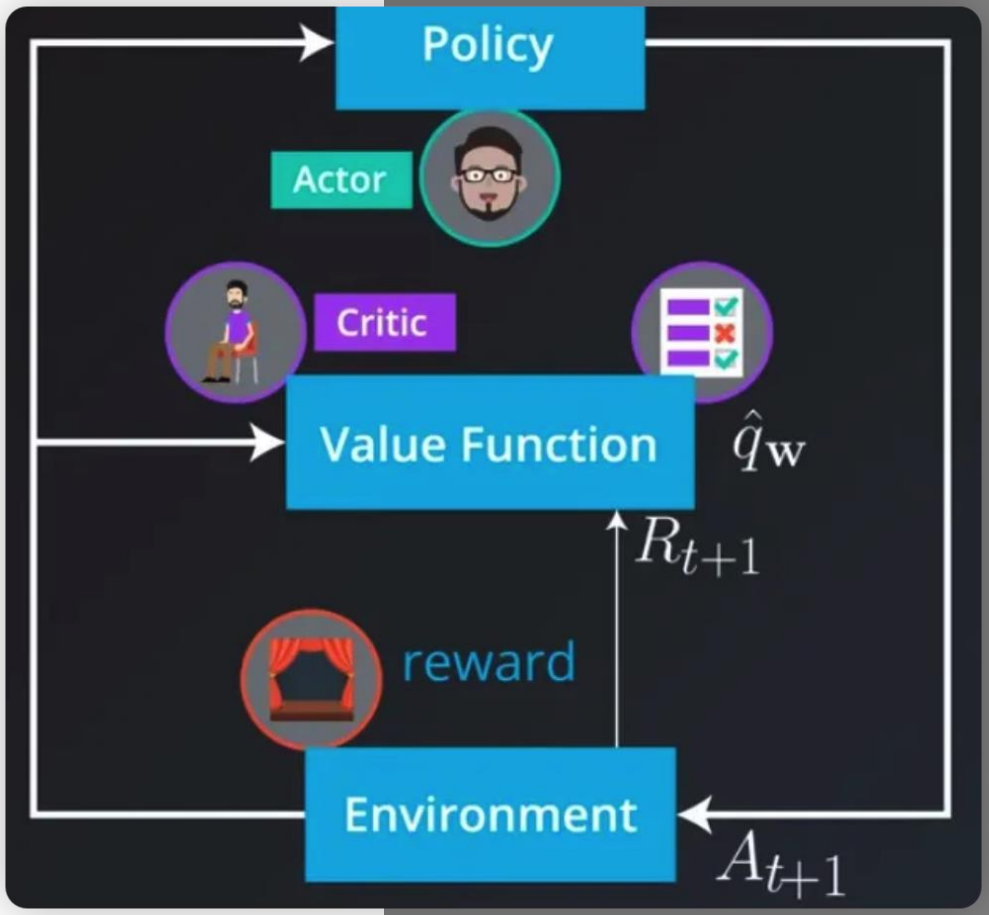
Value

基于价值的智能体（value-based agent）显式地学习价值函数，隐式地学习它的策略。策略是其从学到的价值函数里面推算出来的（价值函数，按照最大化这个函数值的动作来走）

Policy

基于策略的智能体（policy-based agent）直接学习策略（在每一个状态都得到一个最佳的动作）。

actor-critic agent 就是两种的结合。
智能体会根据策略做出动作，而价值函数会对做出的动作给出价值，这样可以在原有的策略梯度算法的基础上加速学习过程，取得更好的效果。



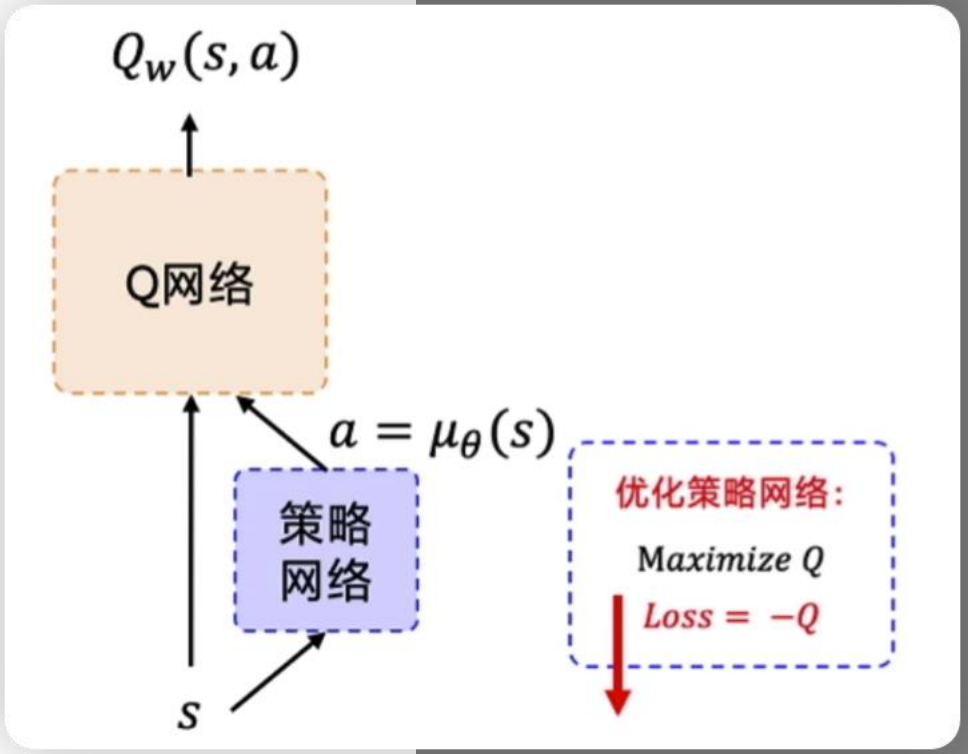
DDPG（deep deterministic policy gradient）

actor——Policy based

策略网络扮演的就是演员的角色，它负责对外展示输出，输出动作。

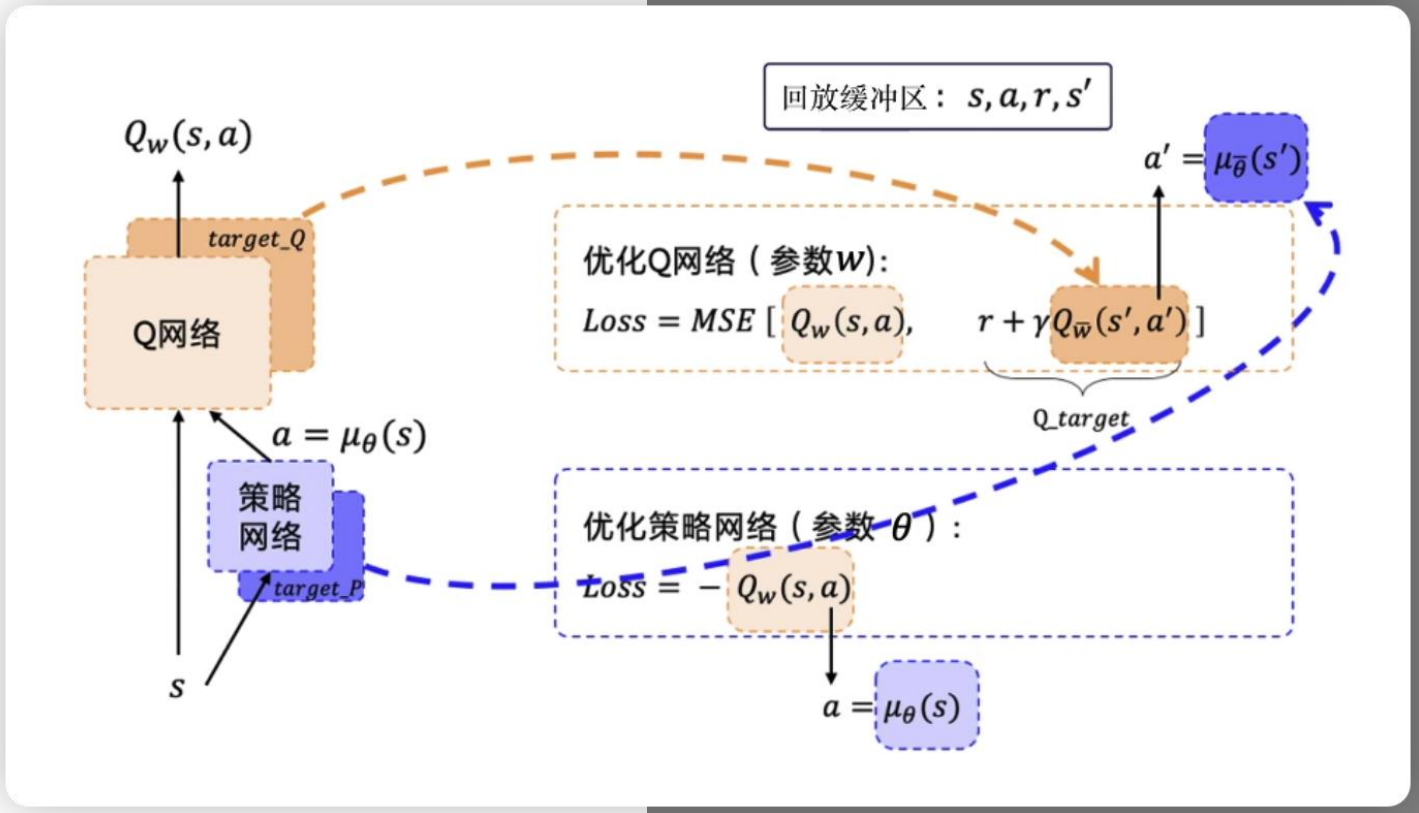
critic——Value based

Q 网络就是评论员，它会在每一个步骤都对演员输出的动作做一个评估，打一个分，估计演员的动作未来能有多少奖励



DDPG（deep deterministic policy gradient）

DDPG 的目的也是求解让 Q 值最大的那个动作。演员只是为了迎合评委的打分而已，所以优化策略网络的梯度就是要最大化这个 Q 值，所以构造的损失函数就是让 Q 取一个负号。我们写代码的时候把这个损失函数放入优化器里面，它就会自动最小化损失，也就是最大化 Q。



MADDPG

（multi-agents deep deterministic policy gradient）

Algorithm 1: Multi-Agent Deep Deterministic Policy Gradient for N agents

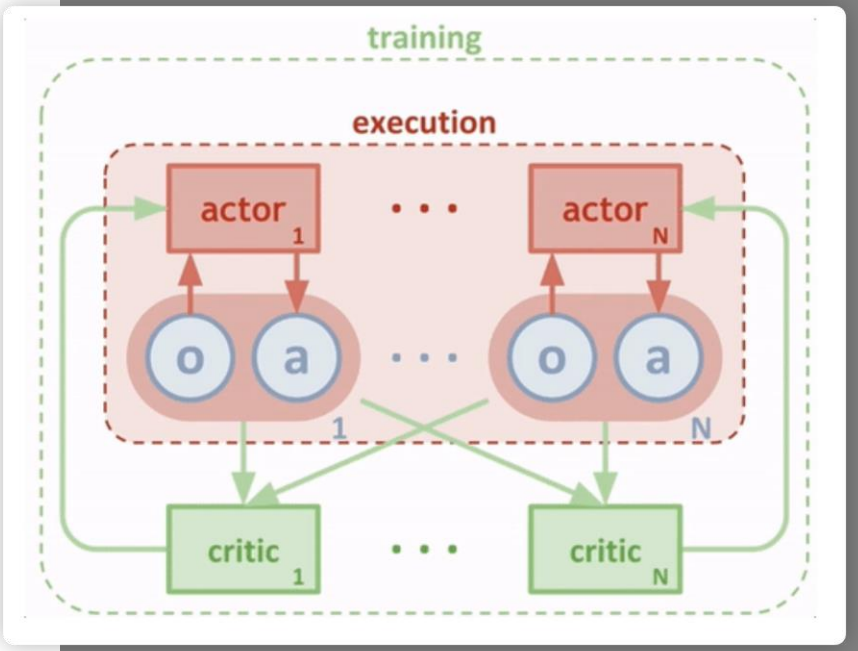
for episode $= 1$ to M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial state $\mathbf{x}$
  for $t = 1$ to max-episode-length **do**
    for each agent i, select action $a_i = \boldsymbol{\mu}_{\theta_i}(o_i) + \mathcal{N}_t$ w.r.t. the current policy and exploration
    Execute actions $a = (a_1, \ldots, a_N)$ and observe reward r and new state $\mathbf{x}'$
    Store $(\mathbf{x}, a, r, \mathbf{x}')$ in replay buffer $\mathcal{D}$
    $\mathbf{x} \leftarrow \mathbf{x}'$
    for agent $i = 1$ to N **do**
      Sample a random minibatch of S samples $(\mathbf{x}^j, a^j, r^j, \mathbf{x}'^j)$ from $\mathcal{D}$
      Set $y^j = r_i^j + \gamma\, Q_i^{\boldsymbol{\mu}'}(\mathbf{x}'^j, a'_1, \ldots, a'_N)\big|_{a'_k = \boldsymbol{\mu}'_k(o_k^j)}$
      Update critic by minimizing the loss $\mathcal{L}(\theta_i) = \frac{1}{S}\sum_j \left(y^j - Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_N^j)\right)^2$
      Update actor using the sampled policy gradient:

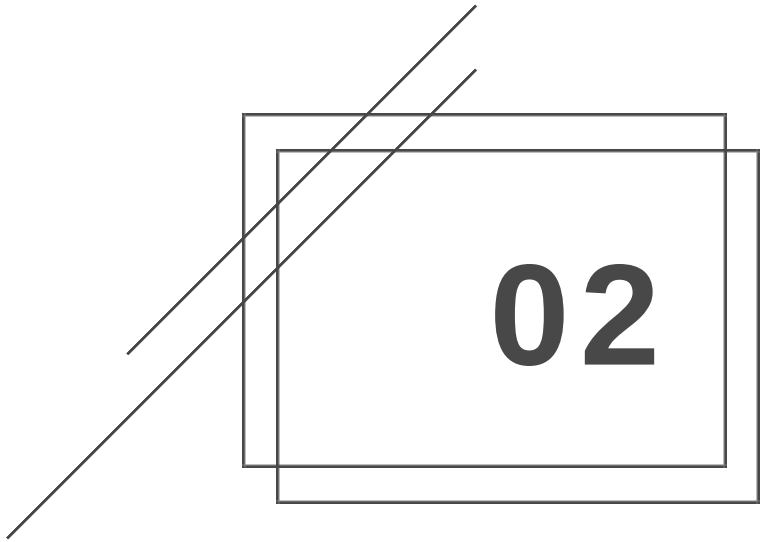
$$\nabla_{\theta_i} J \approx \frac{1}{S}\sum_j \nabla_{\theta_i} \boldsymbol{\mu}_i(o_i^j) \nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_i, \ldots, a_N^j)\big|_{a_i = \boldsymbol{\mu}_i(o_i^j)}$$

    end for
    Update target network parameters for each agent i:

$$\theta_i' \leftarrow \tau\theta_i + (1-\tau)\theta_i'$$

  end for
end for





02

Code structure

代码框架

MADDPG

（multi-agents deep deterministic policy gradient）

Algorithm 1: Multi-Agent Deep Deterministic Policy Gradient for N agents

for episode $= 1$ to M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial state $\mathbf{x}$
  for $t = 1$ to max-episode-length **do**
    for each agent i, select action $a_i = \boldsymbol{\mu}_{\theta_i}(o_i) + \mathcal{N}_t$ w.r.t. the current policy and exploration
    Execute actions $a = (a_1, \ldots, a_N)$ and observe reward r and new state $\mathbf{x}'$
    Store $(\mathbf{x}, a, r, \mathbf{x}')$ in replay buffer $\mathcal{D}$
    $\mathbf{x} \leftarrow \mathbf{x}'$
    for agent $i = 1$ to N **do**
      Sample a random minibatch of S samples $(\mathbf{x}^j, a^j, r^j, \mathbf{x}'^j)$ from $\mathcal{D}$
      Set $y^j = r_i^j + \gamma\, Q_i^{\boldsymbol{\mu}'}(\mathbf{x}'^j, a'_1, \ldots, a'_N)\big|_{a'_k = \boldsymbol{\mu}'_k(o_k^j)}$
      Update critic by minimizing the loss $\mathcal{L}(\theta_i) = \frac{1}{S}\sum_j \left(y^j - Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_N^j)\right)^2$
      Update actor using the sampled policy gradient:

$$\nabla_{\theta_i} J \approx \frac{1}{S}\sum_j \nabla_{\theta_i} \boldsymbol{\mu}_i(o_i^j) \nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_i, \ldots, a_N^j)\big|_{a_i = \boldsymbol{\mu}_i(o_i^j)}$$

    end for
    Update target network parameters for each agent i:

$$\theta_i' \leftarrow \tau\theta_i + (1-\tau)\theta_i'$$

  end for
end for

```
MADDPG
├── buffer.py
├── agent.py
├── main.py
├── networks.py
└── maddpg.py
```

MADDPG

（multi-agents deep deterministic policy gradient）

Algorithm 1: Multi-Agent Deep Deterministic Policy Gradient for N agents

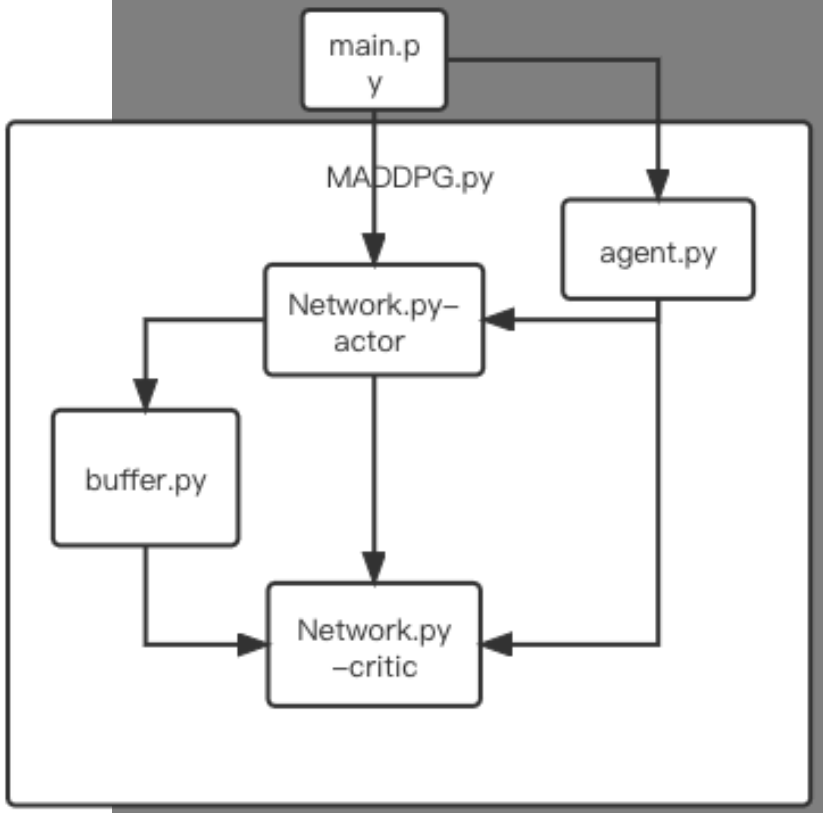
for episode $= 1$ to M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial state $\mathbf{x}$
  for $t = 1$ to max-episode-length **do**
    for each agent i, select action $a_i = \boldsymbol{\mu}_{\theta_i}(o_i) + \mathcal{N}_t$ w.r.t. the current policy and exploration
    Execute actions $a = (a_1, \ldots, a_N)$ and observe reward r and new state $\mathbf{x}'$
    Store $(\mathbf{x}, a, r, \mathbf{x}')$ in replay buffer $\mathcal{D}$
    $\mathbf{x} \leftarrow \mathbf{x}'$
    for agent $i = 1$ to N **do**
      Sample a random minibatch of S samples $(\mathbf{x}^j, a^j, r^j, \mathbf{x}'^j)$ from $\mathcal{D}$
      Set $y^j = r_i^j + \gamma\, Q_i^{\boldsymbol{\mu}'}(\mathbf{x}'^j, a'_1, \ldots, a'_N)\big|_{a'_k = \boldsymbol{\mu}'_k(o_k^j)}$
      Update critic by minimizing the loss $\mathcal{L}(\theta_i) = \frac{1}{S}\sum_j \left(y^j - Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_N^j)\right)^2$
      Update actor using the sampled policy gradient:

$$\nabla_{\theta_i} J \approx \frac{1}{S}\sum_j \nabla_{\theta_i} \boldsymbol{\mu}_i(o_i^j) \nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_i, \ldots, a_N^j)\big|_{a_i = \boldsymbol{\mu}_i(o_i^j)}$$

    end for
    Update target network parameters for each agent i:

$$\theta_i' \leftarrow \tau\theta_i + (1-\tau)\theta_i'$$

  end for
end for



Code

agent.py

```python
class Agent:
    def __init__(self, actor_dims, critic_dims, n_actions, n_agents, agent_idx, chkpt_dir,
                 alpha=0.01, beta=0.01, fc1=64,
                 fc2=64, gamma=0.95, tau=0.01):
        self.gamma = gamma
        self.tau = tau
        self.n_actions = n_actions
        self.agent_name = 'agent_%s' % agent_idx
        self.actor = ActorNetwork(alpha, actor_dims, fc1, fc2, n_actions,
                                  chkpt_dir=chkpt_dir,  name=self.agent_name+'_actor')
        self.critic = CriticNetwork(beta, critic_dims,
                            fc1, fc2, n_agents, n_actions,
                            chkpt_dir=chkpt_dir, name=self.agent_name+'_critic')
        self.target_actor = ActorNetwork(alpha, actor_dims, fc1, fc2, n_actions,
                                         chkpt_dir=chkpt_dir,
                                         name=self.agent_name+'_target_actor')
        self.target_critic = CriticNetwork(beta, critic_dims,
                                           fc1, fc2, n_agents, n_actions,
                                           chkpt_dir=chkpt_dir,
                                           name=self.agent_name+'_target_critic')

        self.update_network_parameters(tau=1)
```

Buffer.py

```python
class MultiAgentReplayBuffer:
    def __init__(self, max_size, critic_dims, actor_dims,
                 n_actions, n_agents, batch_size):
        self.mem_size = max_size
        self.mem_cntr = 0
        self.n_agents = n_agents
        self.actor_dims = actor_dims
        self.batch_size = batch_size
        self.n_actions = n_actions

        self.state_memory = np.zeros((self.mem_size, critic_dims))
        self.new_state_memory = np.zeros((self.mem_size, critic_dims))
        self.reward_memory = np.zeros((self.mem_size, n_agents))
        self.terminal_memory = np.zeros((self.mem_size, n_agents), dtype=bool)

        self.init_actor_memory()
```

Network.py

```python
class ActorNetwork(nn.Module):
    def __init__(self, alpha, input_dims, fc1_dims, fc2_dims,
                 n_actions, name, chkpt_dir):
        super(ActorNetwork, self).__init__()

        self.chkpt_file = os.path.join(chkpt_dir, name)

        self.fc1 = nn.Linear(input_dims, fc1_dims)
        self.fc2 = nn.Linear(fc1_dims, fc2_dims)
        self.pi = nn.Linear(fc2_dims, n_actions)

        self.optimizer = optim.Adam(self.parameters(), lr=alpha)
        self.device = T.device('cuda:0' if T.cuda.is_available() else 'cpu')

        self.to(self.device)
```

class actor

```python
class CriticNetwork(nn.Module):
    def __init__(self, beta, input_dims, fc1_dims, fc2_dims,
                 n_agents, n_actions, name, chkpt_dir):
        super(CriticNetwork, self).__init__()

        self.chkpt_file = os.path.join(chkpt_dir, name)

        self.fc1 = nn.Linear(input_dims+n_agents*n_actions, fc1_dims)
        self.fc2 = nn.Linear(fc1_dims, fc2_dims)
        self.q = nn.Linear(fc2_dims, 1)

        self.optimizer = optim.Adam(self.parameters(), lr=beta)
        self.device = T.device('cuda:0' if T.cuda.is_available() else 'cpu')

        self.to(self.device)
```

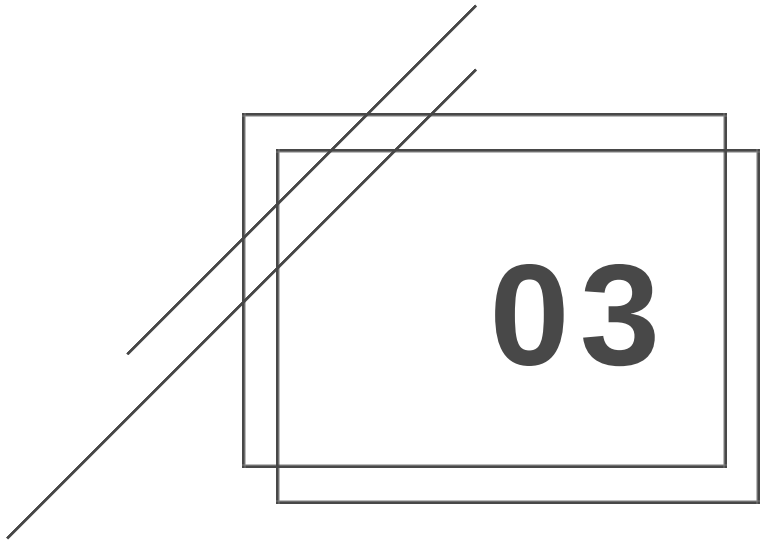
class critic

MADDPG.py

```
class MADDPG:
    def __init__(self, actor_dims, critic_dims, n_agents, n_actions,
                 scenario='simple',  alpha=0.01, beta=0.01, fc1=64,
                 fc2=64, gamma=0.99, tau=0.01, chkpt_dir='tmp/maddpg/'):
        self.agents = []
        self.n_agents = n_agents
        self.n_actions = n_actions
        chkpt_dir += scenario
        for agent_idx in range(self.n_agents):
            self.agents.append(Agent(actor_dims[agent_idx], critic_dims,
                            n_actions, n_agents, agent_idx, alpha=alpha, beta=beta,
                            chkpt_dir=chkpt_dir))
```

Main.py

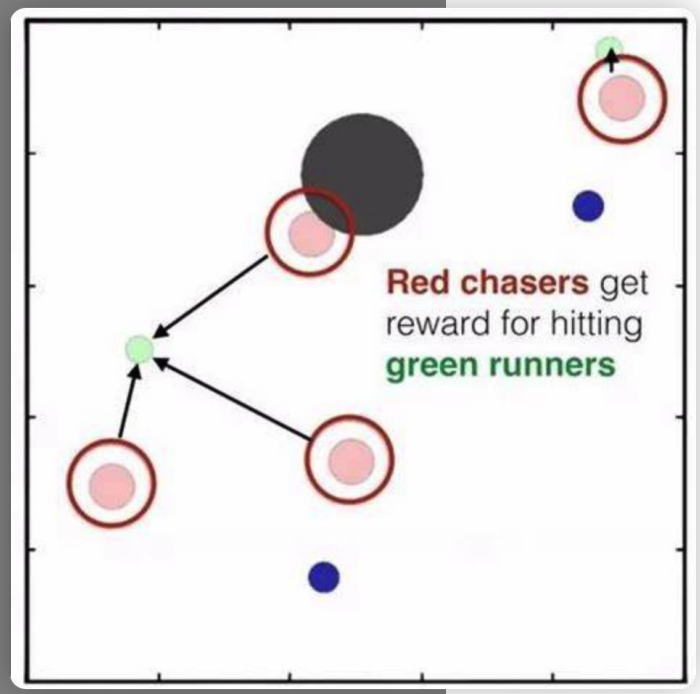
```
for i in range(n_agents):
    actor_dims.append(env.observation_space[i].shape[0])
critic_dims = sum(actor_dims)

# action space is a list of arrays, assume each agent has same action space
n_actions = env.action_space[0].n
maddpg_agents = MADDPG(actor_dims, critic_dims, n_agents, n_actions,
                       fc1=64, fc2=64,
                       alpha=0.01, beta=0.01, scenario=scenario,
                       chkpt_dir='tmp/maddpg/')

memory = MultiAgentReplayBuffer(1000000, critic_dims, actor_dims,
                    n_actions, n_agents, batch_size=1024)

PRINT_INTERVAL = 500
N_GAMES = 50000
MAX_STEPS = 25
```



03

Scenario Design

情景设计



多智能体粒子环境

这个环境是OpenAi 开源的多智能体学习环境，是为OpenAi提出的MADDPG算法的运行所构建的环境。

包含属性

- 智能体数量
- 完成任务类型及技能（竞争/合作/通讯）
- 状态动作空间
- 模拟真实性（真实物理条件）

Link：https://github.com/openai/multiagent-particle-envs

情景设计

追逐问题

- 逃亡者速度较快，希望避免被机器人群击中。无人机的速度较慢，追击智能的逃亡者。其中会有障碍物会挡住他们的路线。

（1）有食物，逃亡者在附近就会得到奖励。

（2）“掩体”，两方在里面时，不被外界看到。

（3）无人机集群在后面的训练中有一个 "领导"，可以随时看到无人机的位置，并可以与其他无人机进行沟通，协调追捕。

控制问题

- 这个场景由L个地标组成，包括一个目标地标，N个知道目标地标的合作智能体，根据他们与目标的距离给予奖励，M个无人机必须阻止前者到达目标。无人机通过将对方从地标上推开，占据地标来实现这一点。虽然无人机也是根据他们与目标地标的距离给予奖励，但是他们不知道正确的目标；这必须从对方智能体的行动中推断出来。

情景设计

追逐问题

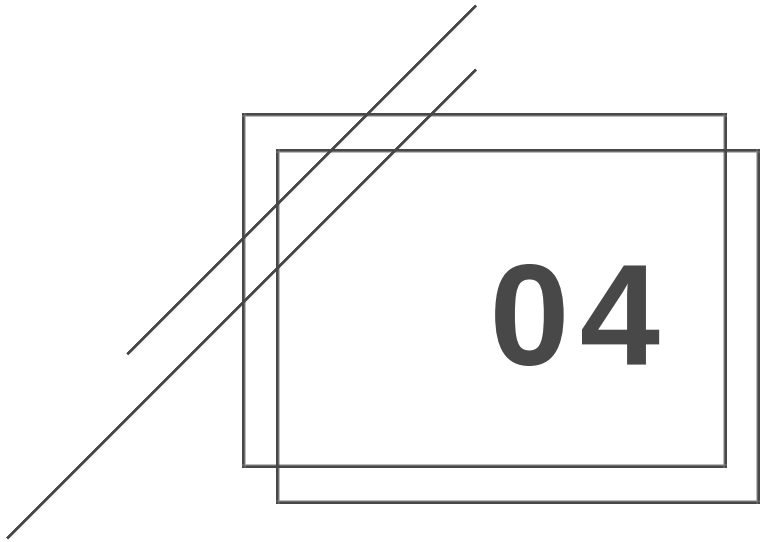
- - 一个drone搜寻固定点。
- - 一个drone 追逐一个逃亡者。
- - 多个drones 追逐一个逃亡者。相互之间平等的合作
- - 多个drones追逐多个逃跑者。（相互之间存在平等关系，以抓到最多的逃跑者为rewards）
- - 多个drones追逐多个逃跑者。其中有一个是leader drones，能够综合drones的位置信息来进行追捕（综合信息）
- - 两队drones追逐逃跑者。各自有leader

控制问题

- - 一个drones 对一个目标的追踪和控制。距离控制，keep away
- - 多个drones 对一个目标的控制。距离控制，进行一个控制和保护，可以保持其关系
- - 多个drones 对一个目标的控制. 其中有leader进行信息的统一


1v1
追逐问题


控制问题
1v1
-master> python main.py
-master> python main.py
-master> python main.py

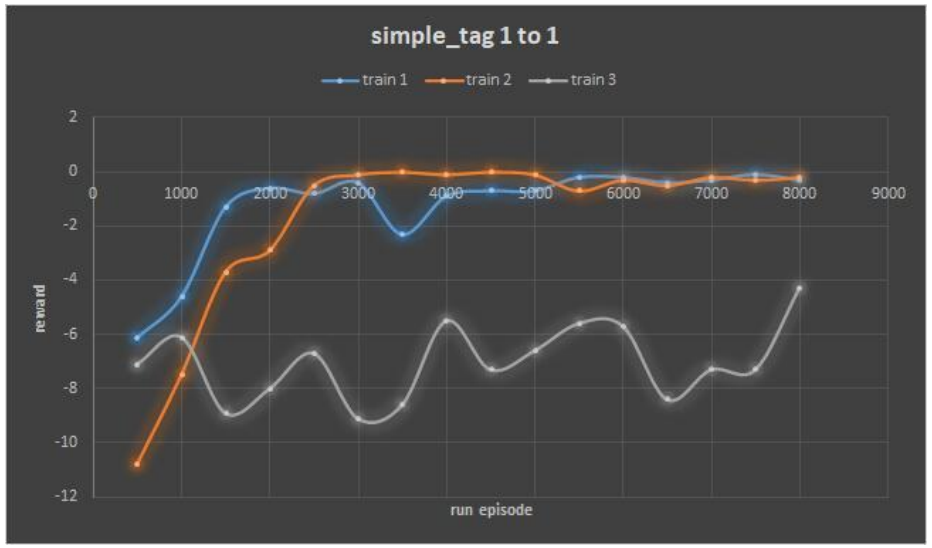


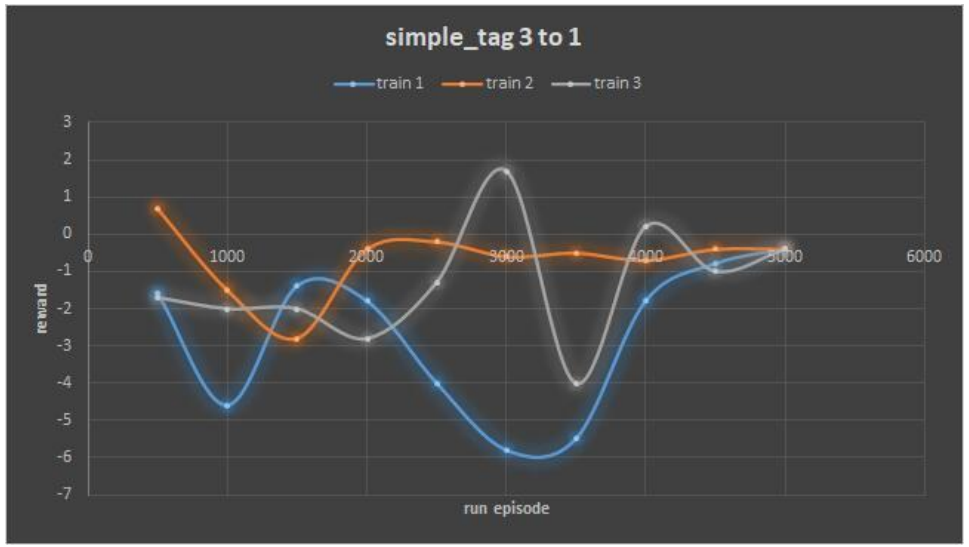
04

Analysis and conclusion

分析总结

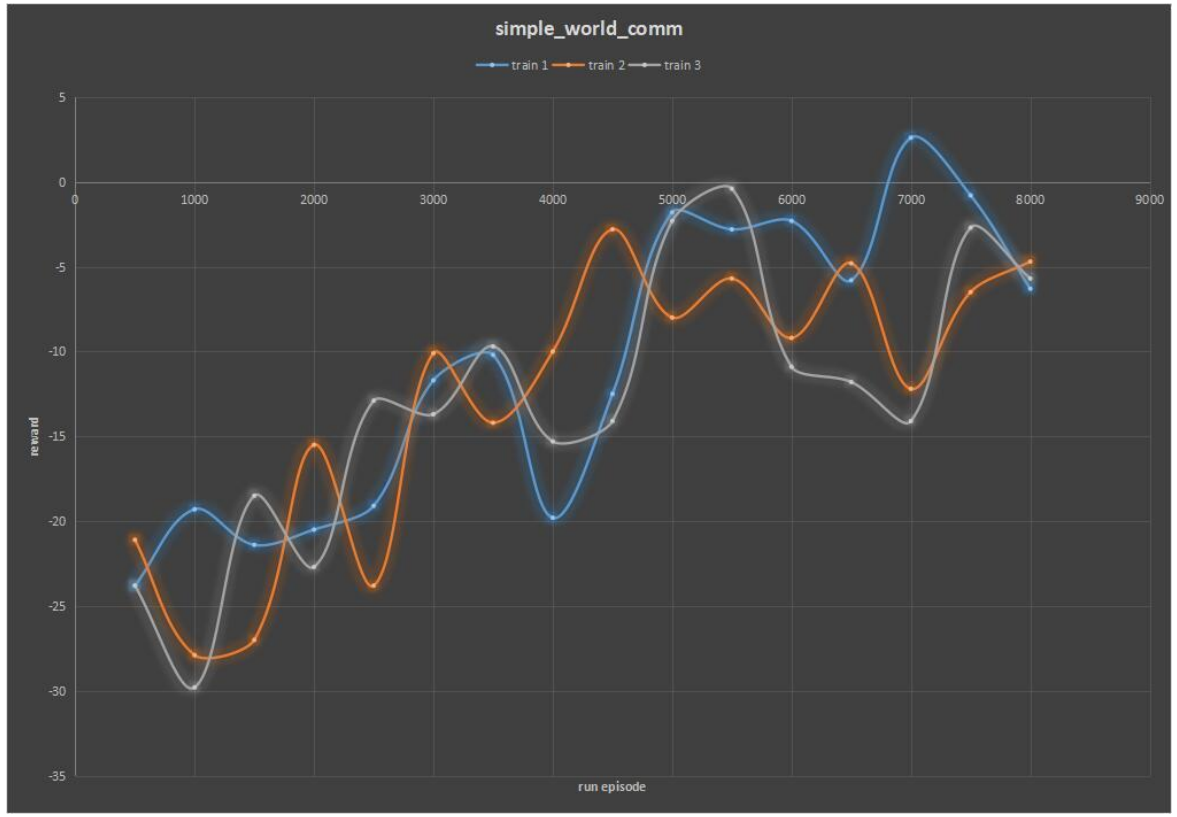
追逐问题（独立/合作）

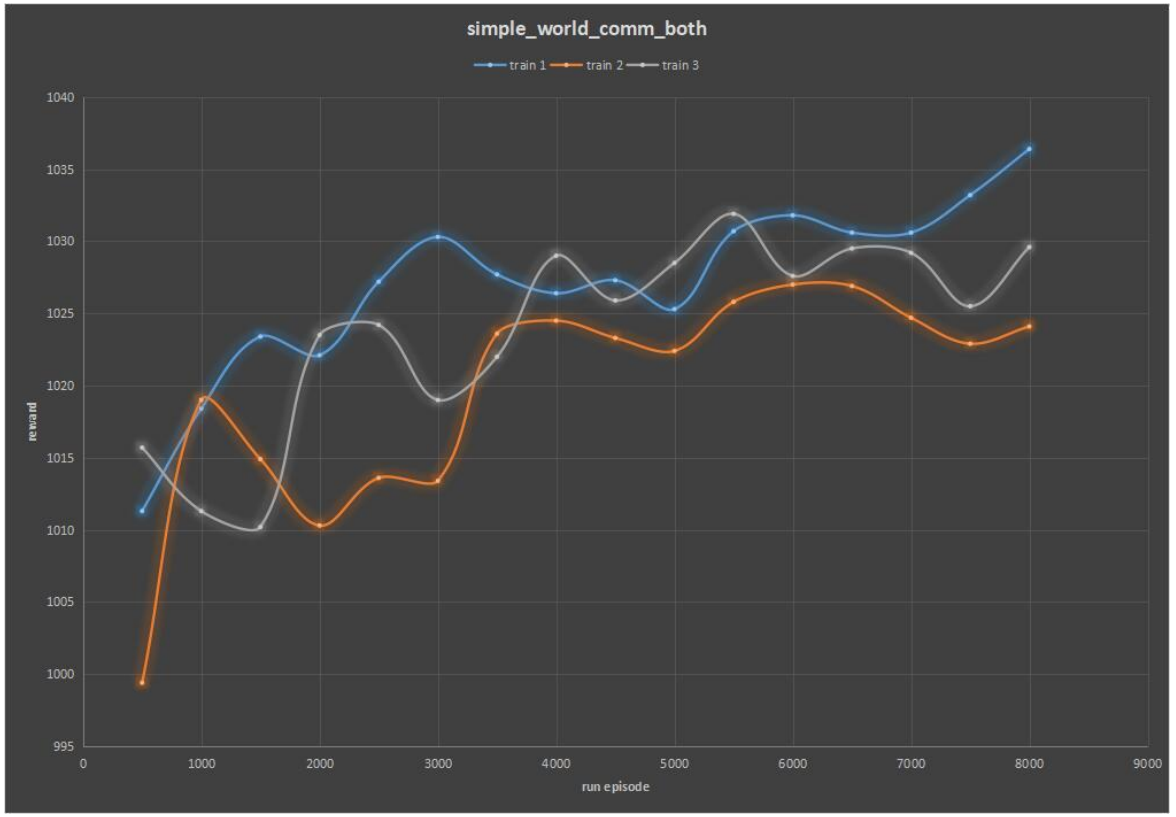

simple_tag 1 to 1
train 1
train 2
train 3
reward
run episode
2
0
-2
-4
-6
-8
-10
-12
0
1000
2000
3000
4000
5000
6000
7000
8000
9000


simple_tag 3 to 1
train 1
train 2
train 3
reward
run episode
3
2
1
0
-1
-2
-3
-4
-5
-6
-7
0
1000
2000
3000
4000
5000
6000

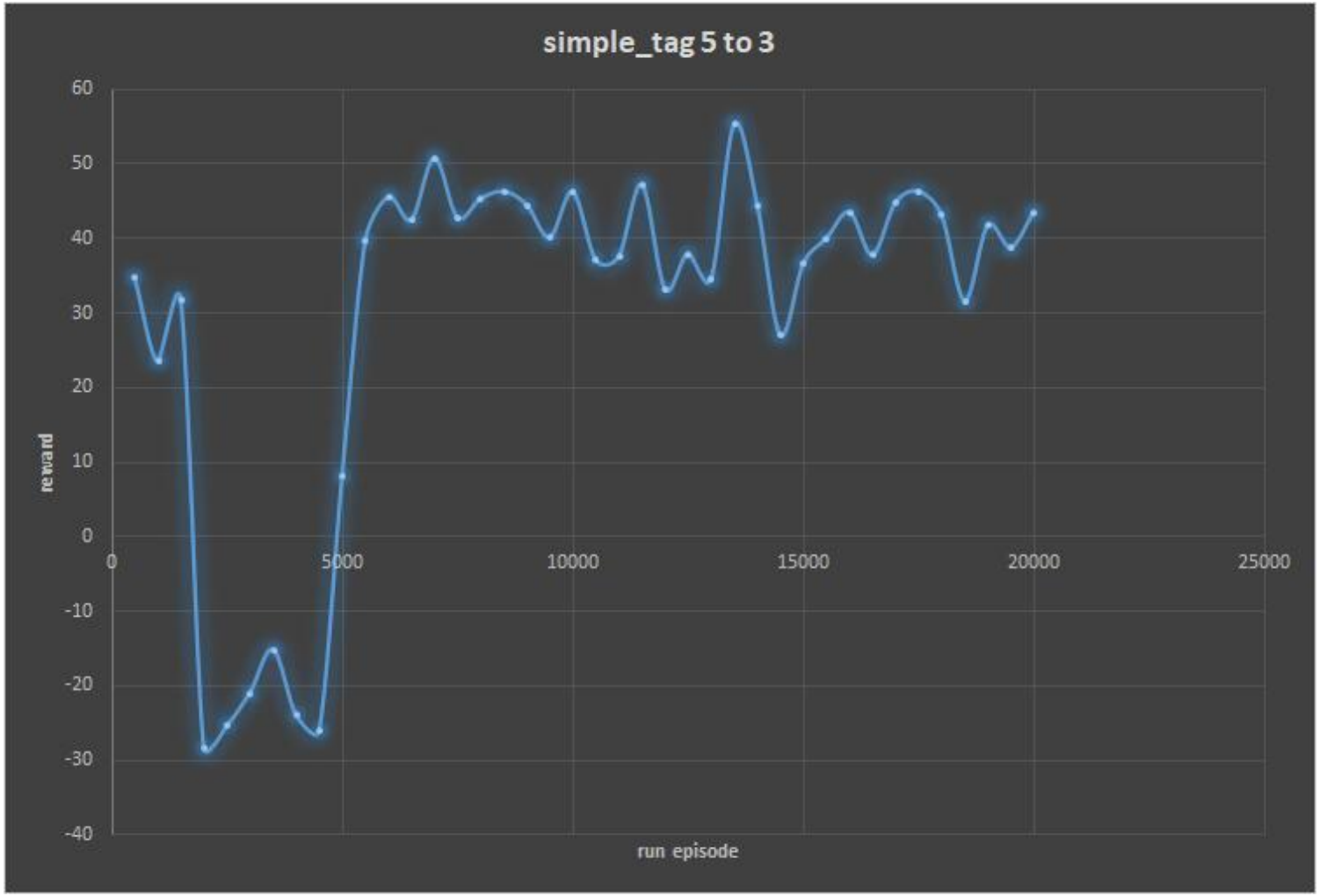

simple_tag 5 to 3
train 1
train 2
train 3
reward
run episode
80
60
40
20
0
-20
-40
-60
-80
-100
0
1000
2000
3000
4000
5000
6000

追逐问题（层级领导/竞争）


simple_world_comm
train 1
train 2
train 3
reward
run episode
5
0
-5
-10
-15
-20
-25
-30
-35
0
1000
2000
3000
4000
5000
6000
7000
8000
9000


simple_world_comm_both
train 1
train 2
train 3
reward
run episode
1040
1035
1030
1025
1020
1015
1010
1005
1000
995
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

追逐问题（多智能体收敛状态）

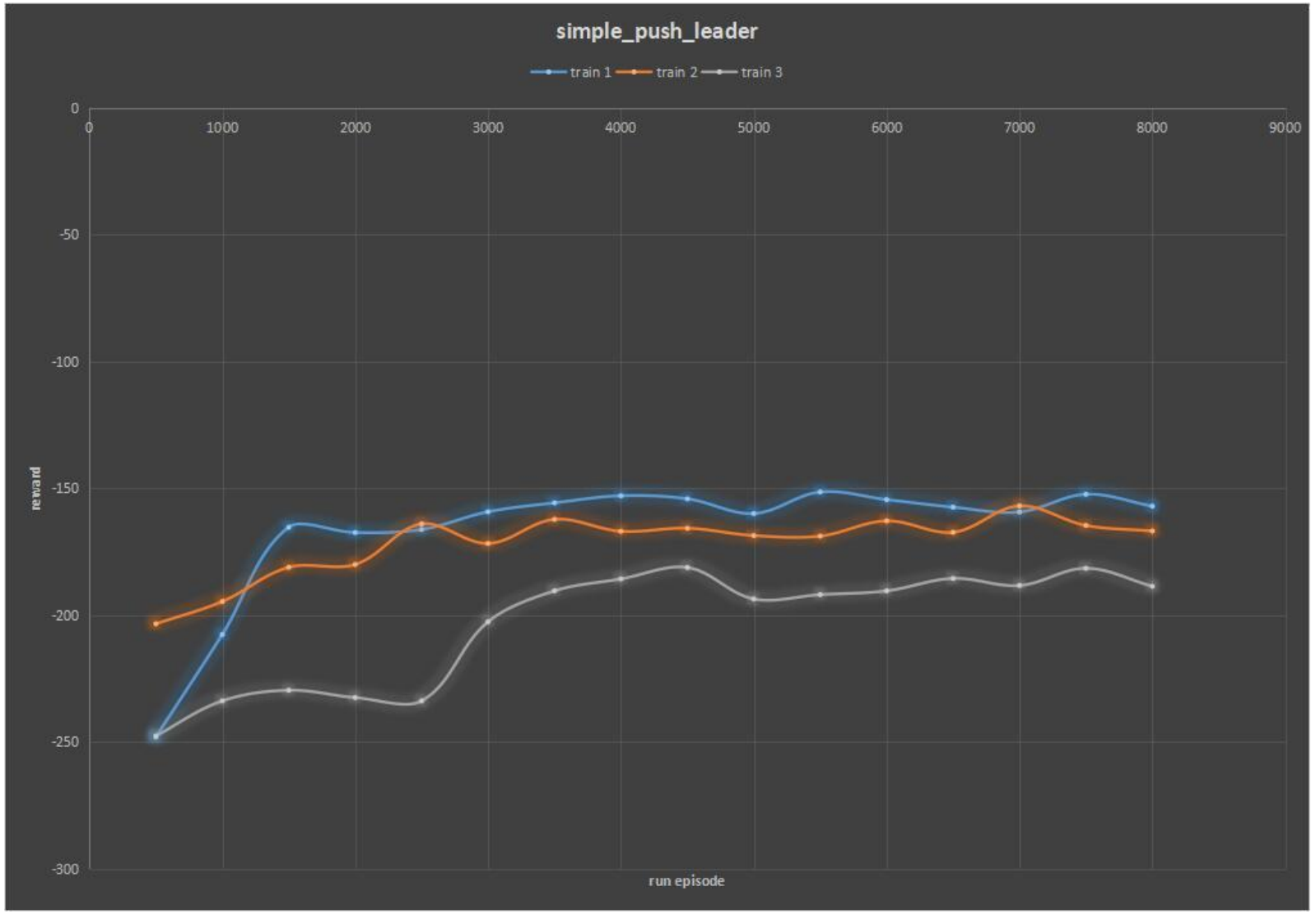


控制问题（独立/合作）

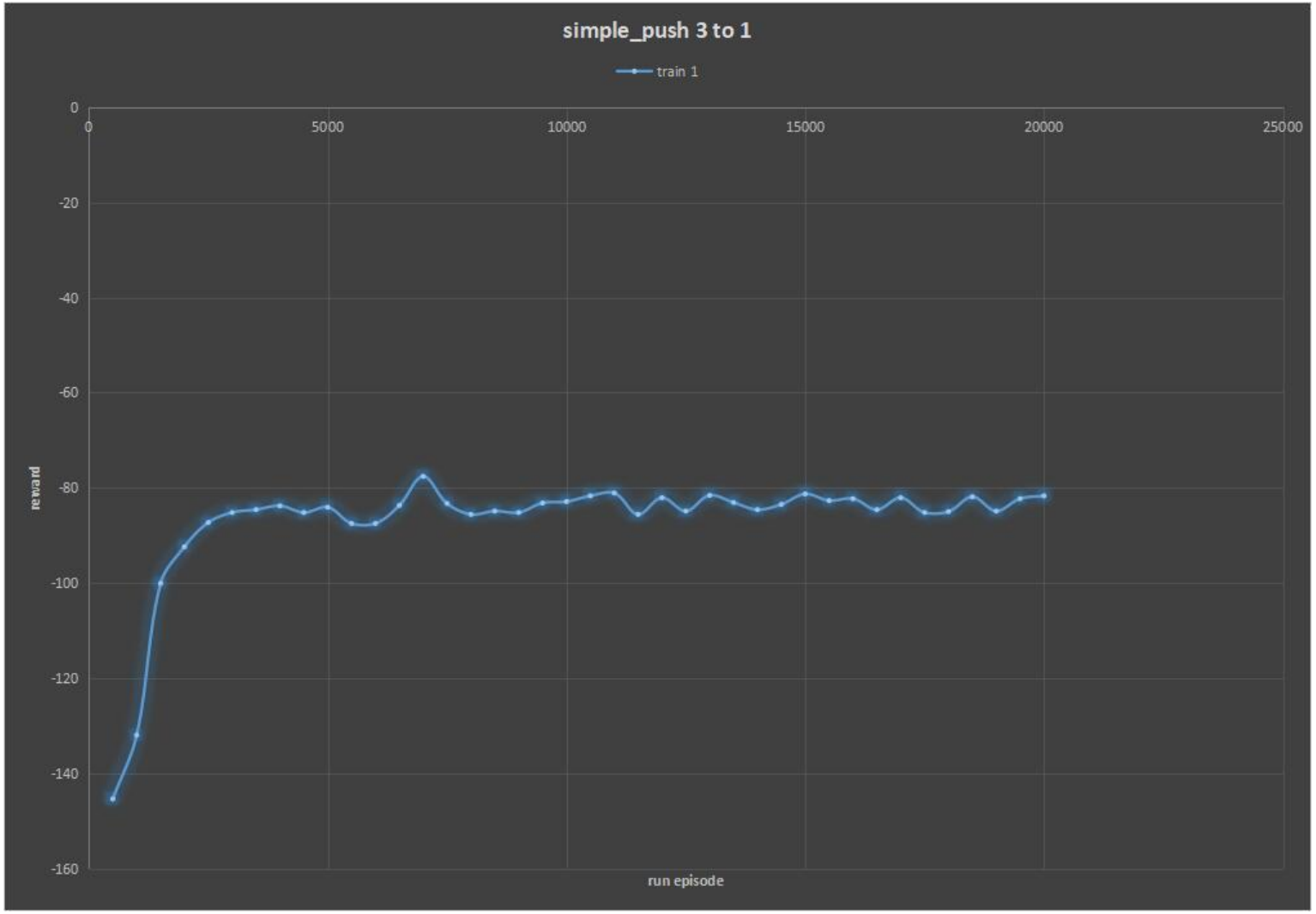

simple_push 1 to 1
train 1
train 2
train 3
0
-5
-10
-15
-20
-25
-30
0
1000
2000
3000
4000
5000
6000
reward
run episode


simple_push 3 to 1
train 1
train 2
train 3
80
60
40
20
0
-20
-40
-60
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
reward
run episode

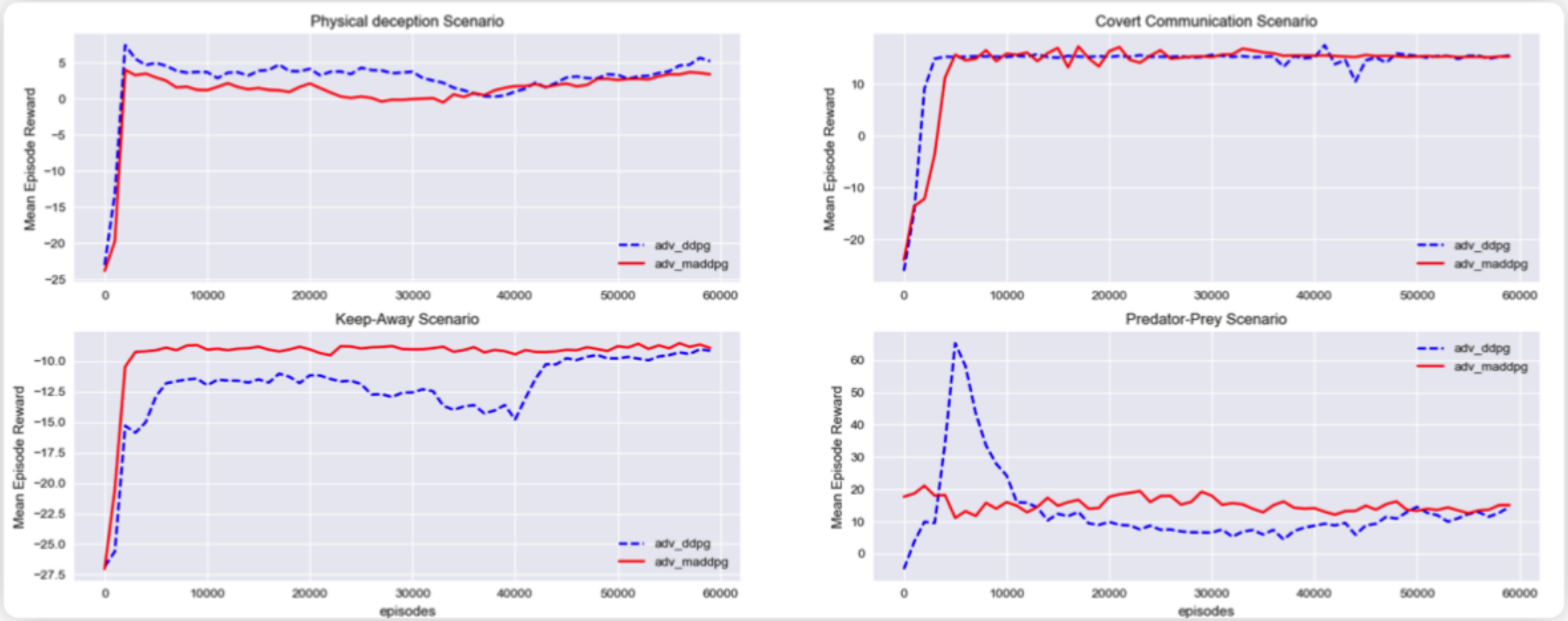
控制问题（层级领导）


simple_push_leader
train 1
train 2
train 3
0
-50
-100
-150
-200
-250
-300
reward
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
run episode

控制问题（好的优化趋势）


simple_push 3 to 1
train 1
0
5000
10000
15000
20000
25000
-20
-40
-60
-80
-100
-120
-140
-160
reward
run episode

OpenAi-reward



Conclusion

25%

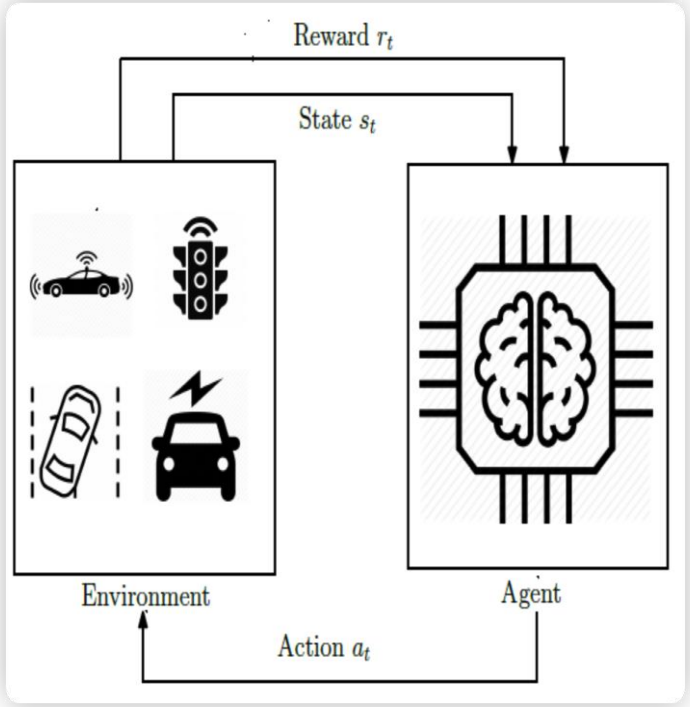
阶段一

随机探索，累计经验值

50%

阶段二

地图探索基本完成，开始针对性的学习对方的移动，并进行匹配，开始追逐靠近



100%

阶段四

逃跑者利用障碍和速度进行躲避（灵活走位）集群机器人协作进行搜捕和控制（提前占据目标/合作进行围捕）

75%

阶段三

已知对方目标，开始应对性的提前做出预判策略。

Conclusion

Outlook

在可以预见的将来，人类创造的机器和系统会变得越来越复杂，也会变得越来越能干，在很多任务上可以达到人类专家的水平，甚至超越专家的水平。

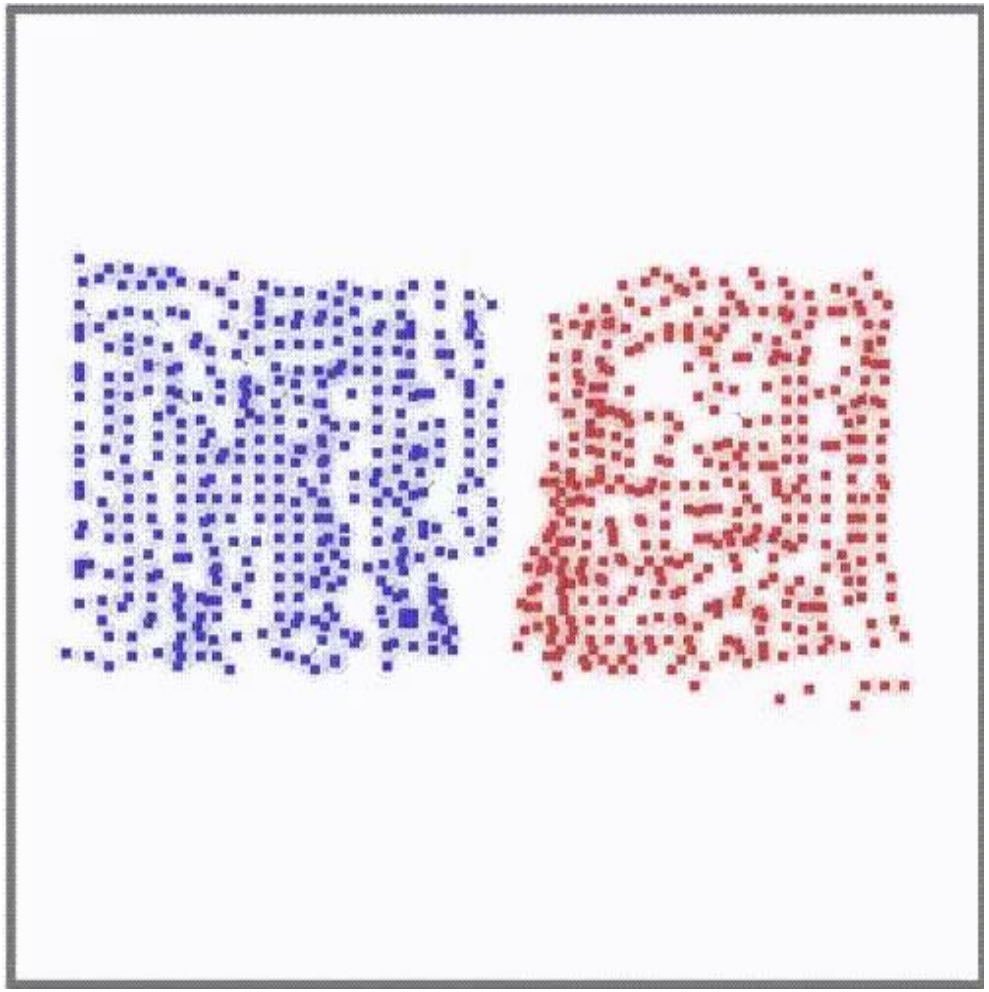
Thoughts about RL+DL

可解释性至关重要。可能的解决方案或许内置于集群的自组织机制中，以便使用户能够看到集群的当前状态和目标。

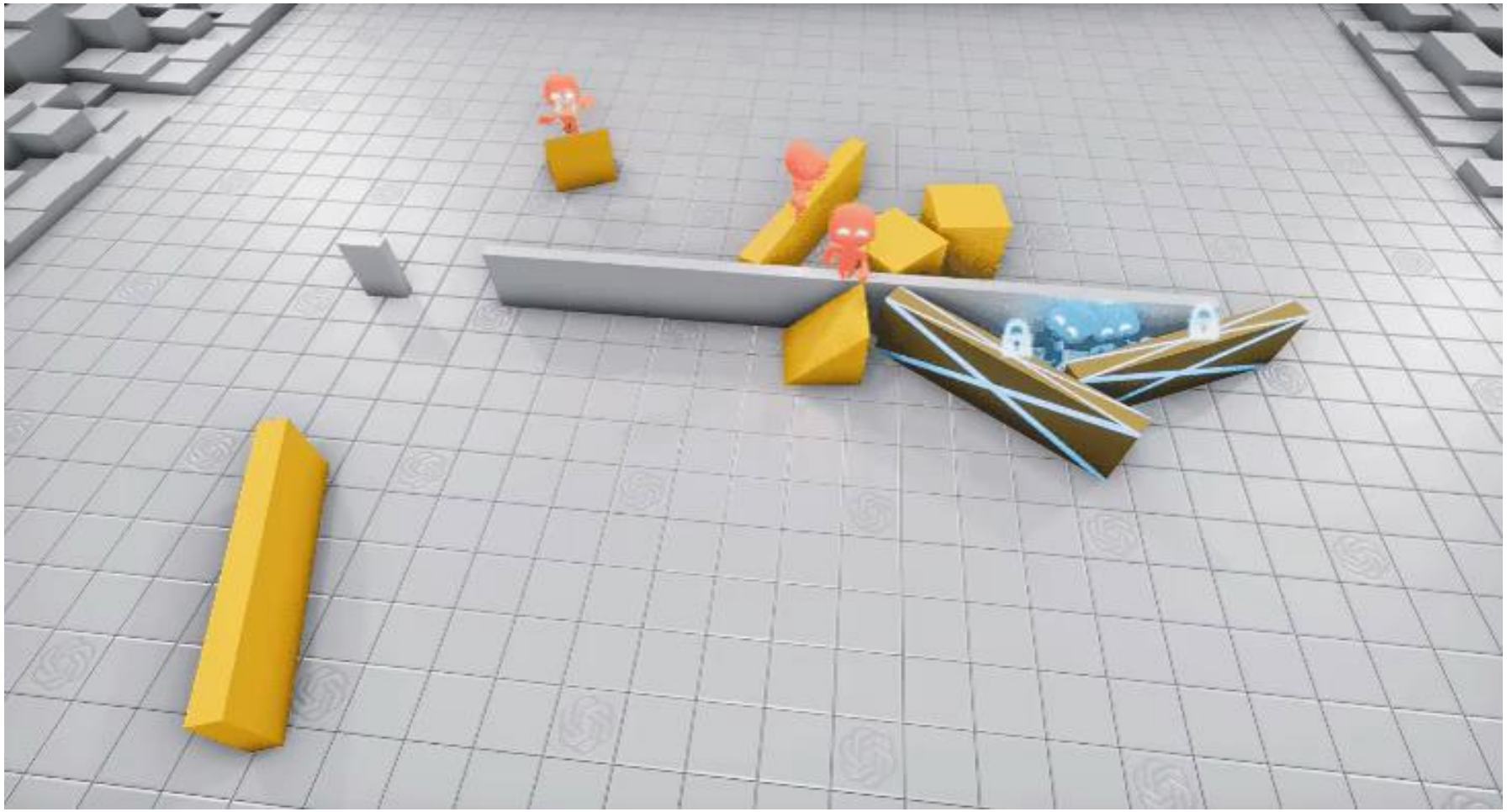
```
multiagent-particle-envs-master
├── bin
│   ├── interactive.py
│   └── __init__.py
├── multiagent
│   ├── scenarios
│   │   ├── __pycache__
│   │   ├── simple_world_comm.py
│   │   ├── simple_tag.py
│   │   ├── simple_reference.py
│   │   ├── simple_speaker_listener.py
│   │   ├── simple.py
│   │   ├── simple_push.py
│   │   ├── __init__.py
│   │   ├── simple_spread.py
│   │   ├── simple_crypto.py
│   │   └── simple_adversary.py
│   ├── environment.py
│   ├── core.py
│   ├── scenario.py
│   ├── __init__.py
│   ├── policy.py
│   ├── multi_discrete.py
│   └── rendering.py
├── LICENSE.txt
├── make_env.py
└── setup.py
```

- Drones：刚体，可移动，最大速度，加速度缩放系数，速度阻尼，其中领导者颜色较深一点，
- agents：刚体，可移动，最大速度，加速度缩放系数，速度阻尼，小绿色
- Food：刚体，不可移动，蓝色
- bunkers：非刚体，不可移动，大绿色
- landmarks：刚体，不可移动，黑色

Appendix（UCL汪军老师团队Mean Field ）



Appendix（Multiagent emergence 环境：mujoco）



Appendix（observation space and action space）

Drones

捕食者的观测状态为1*34的向量，具体为自身的速度(x和y两个方向，2位) + 自身的位置(x和y两个方向，2位) + 所有地标与自己的相对位置(地标位置-自身位置，10位) + 其他智能体与自己的相对位置(其他智能体位置-自身位置，10) + 被捕食者的速度(4) + 自身是否在树林里(2) + 交流信息(4)，数据格式float32，Box(34,)。非领导者的交流信息直接继承领导者的。

```
array([ 0.        ,  0.        , -0.81360341,  0.31675768,  0.25026168,
       -0.12137332,  1.26442749, -0.7671077 ,  0.90388104, -1.00294841,
        0.70155893, -0.62365125,  1.09197528, -0.92503425,  1.31906775,
        0.53801265,  1.30256252, -0.5290839 ,  1.3105693 , -0.16847554,
        1.34816312, -0.82404067,  0.61383961, -1.30914401,  0.        ,
        0.        ,  0.        ,  0.        , -1.        , -1.        ,
        0.        ,  0.        ,  0.        ,  0.        ])
```

agents

被捕食者的观测状态为1*28的向量，具体为自身的速度(x和y两个方向，2)+自身的位置(x和y两个方向，2)+所有地标与自己的相对位置(地标位置-自身位置，10)+其他智能体与自己的相对位置(其他智能体位置-自身位置，10)+其他被捕食者的速度(2)+自身是否在树林里(2)，数据格式float32，Box(28)。

```
array([ 0.        ,  0.        , -0.1997638 , -0.99238633, -0.36357793,
        1.18777069,  0.65058787,  0.54203631,  0.29004143,  0.3061956 ,
        0.08771932,  0.68549276,  0.47813566,  0.38410976, -0.61383961,
        1.30914401,  0.70522814,  1.84715666,  0.68872291,  0.78006011,
        0.69672969,  1.14066847,  0.        ,  0.        , -1.        ,
       -1.        ,  0.        ,  0.        ])
```

leader：MultiDiscrete2，1*9维的向量。第一位无操作，2-5位给定智能体x, y正负方向上的加速度，6-9位为交流信息。

```
np.array([0, 1, 0, 1, 0, 1, 1, 1, 1],dtype=np.float32)
```

none-leader： Discrete(5)，1*9维的向量。第一位无操作，2-5位给定智能体x,y正负方向上的加速度。后面位置空置

```
np.array([0, 1, 0, 1, 0], dtype=np.float32)
```

THANK YOU FOR LISTENING AND WATCHING !

Q&A