目录

0. 说明

不同算法分别建立单独的.py文件 (例如`MATD3_simple_spread.py`、`VDN_simple_spread.py`、`QMIX_simple_spread.py`等) 编写实现，算法参数均在代码中体现。

每个算法的训练数据见result文件下相应的`.log`文件和`.pkl`文件，例如`MATD3.log`、`MATD3.pkl`。学习曲线在`learning_curves/png`目录下。

所有随机种子均已固定，在本机上可以复现。

所有代码已设置为测试模式，即载入训练的模型进行测试，所有路径为相对路径可以直接运行。

1. MATD3 for simple spread

1.1 gumbel-softmax重参数技巧

MADDPG是确定性策略梯度算法，输出的是确定性动作，**针对离散型的动作**（例如MPE环境），DDPG本应用$\mathrm{u}_\theta(\mathrm{s}) = \underset{\mathrm{a}}{\operatorname{argmax}}\ \mathrm{A}(\mathrm{s},\mathrm{a})$作为其确定性策略来输出一个确定性的离散动作，但$\operatorname{argmax}$操作将使Actor网络的参数$\theta$梯度中断，进而导致后续无法用梯度上升的方法由$\mathrm{Q}(\mathrm{s},\mathrm{u}_\theta(\mathrm{s}))$来更新Actor网络。

解决方法：使用`gumbel-softmax`代替`argmax`，通过调低温度系数τ（代码中设置为0.5）使其逼近argmax。

1.2 熵正则化

通过阅读MADDPG源码，得知actor的更新使用了**熵正则化**技巧。区别与SAC的最大化熵学习，MADDPG的熵只是作为一个正则项加在loss上，而SAC显式地在优化目标中使用熵，并学习一个自动调节的温度系数α。

MADDPG源码部分如下，看到最终loss除了基本的梯度上升最大化Q值，还包括了熵正则项，系数为1e-3。实验中该部分的设置与MADDPG源码一致。

```
# wrap parameters in distribution
act_pd = act_pdtype_n[p_index].pdfromflat(p)

act_sample = act_pd.sample()
p_reg = tf.reduce_mean(tf.square(act_pd.flatparam()))

act_input_n = act_ph_n + []
act_input_n[p_index] = act_pd.sample()
q_input = tf.concat(obs_ph_n + act_input_n, 1)
if local_q_func:
    q_input = tf.concat([obs_ph_n[p_index], act_input_n[p_inde
q = q_func(q_input, 1, scope="q_func", reuse=True, num_units=n
pg_loss = -tf.reduce_mean(q)

loss = pg_loss + p_reg * 1e-3

```

使用熵正则化技巧可以让策略更鲁棒。对比实验结果具体见下图。

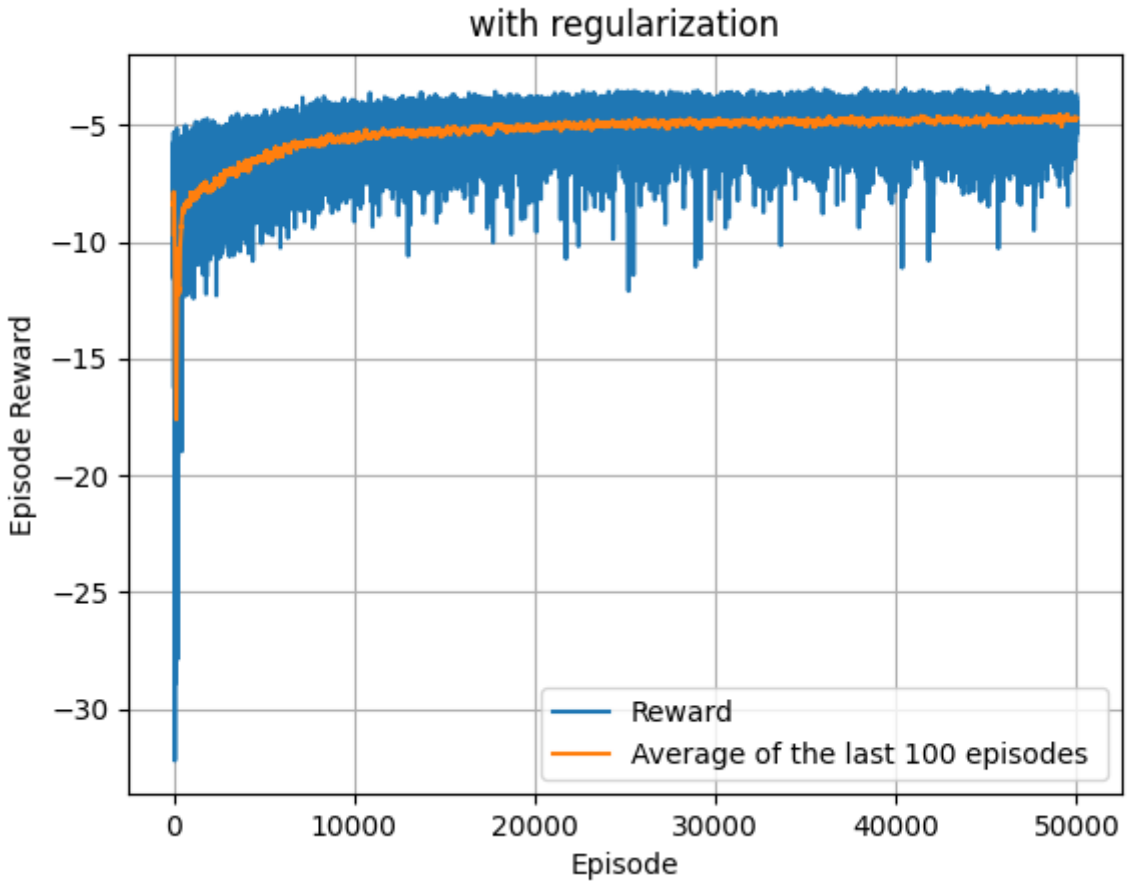
1.3 细节及展示

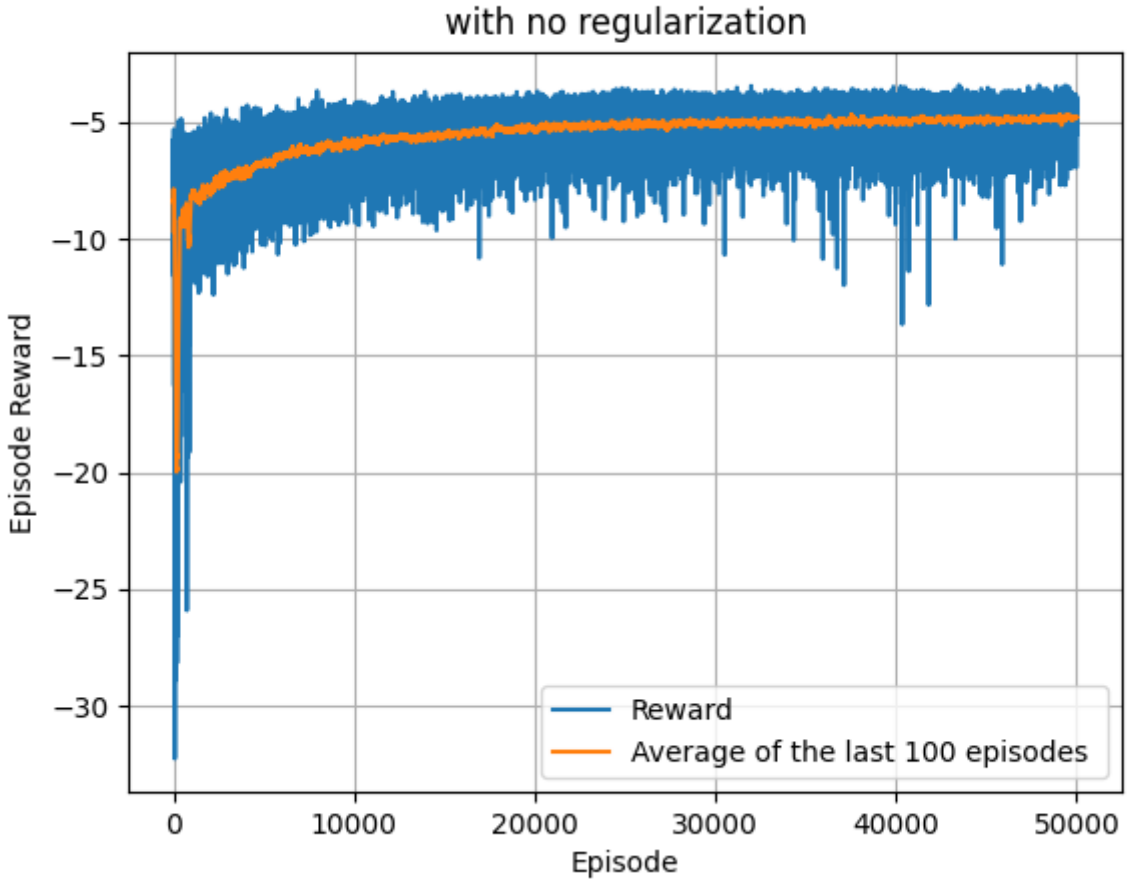
1. MPE的simple spread环境中，所有智能体的观测空间、动作空间以及功能完全一致，属于同构智能体，故所有智能体可以共享参数，进而减缓惰性智能体lazy agent的出现。

2. 与作业2一致，这里的MADDPG使用其改进版MATD3实现。即使用探索噪声和策略噪声、使用Twin网络、延迟更新Actor网络和目标网络。

3. 具体实现中设置一个episode后更新n次，而不是一个step更新一次，具体地设置n=1，即一个episode后更新一次网络参数。尽管n取2或者更大可能会带来更好的收敛效果，但这里没有进行繁琐的参数尝试和调试。

使用熵正则项和不使用熵正则项的**收敛值分别为-4.7、-4.8**，学习曲线如下，具体数值见相应的`MATD3.log`文件。
可以发现使用熵正则化后效果稍优，而且可以清晰地看到，在训练后期会更加稳定。





2. VDN for simple spread

2.1 VDN思想

VDN的思想是用各合作智能体自己的局部Q值之和，来近似团队的Q值，并基于团队Q值的TD error来更新参数。由于团队Q值$\mathrm{Q_{tot}}$来自各智能体的Q值$\mathrm{Q_a}$，故TD error反向传播会分散到各个智能体的Q网络，体现出值分解Value Decomposition。

2.2 细节及展示

1. 实现中使用DRQN代替DQN，希望使用循环神经网络（具体为GRU）结合过往的观测及动作信息 (具体实现为隐藏状态+上一时刻的动作)，缓解局部观测带来的局限性，进而使输出的Q值更加准确。
   由于每个timestep的动作不仅与当前的局部观测有关，还与上一时刻的动作以RNN的隐藏状态相关，因此就不能以timestep为单位从经验池随机抽取经验进行学习。所以这里以episode为单位存储经验，训练时从经验池一次随机采样多个episode进行学习。

2. 与Double DQN的思想类似，为减缓高估，计算下一时刻的Q值时，用$\max\limits_{\mathrm{a'}} \mathrm{Q_{target}(s', a')}$代替$\max\limits_{\mathrm{a'}} \mathrm{Q(s', a')}$，即单个智能体自己的局部Q值计算方式为

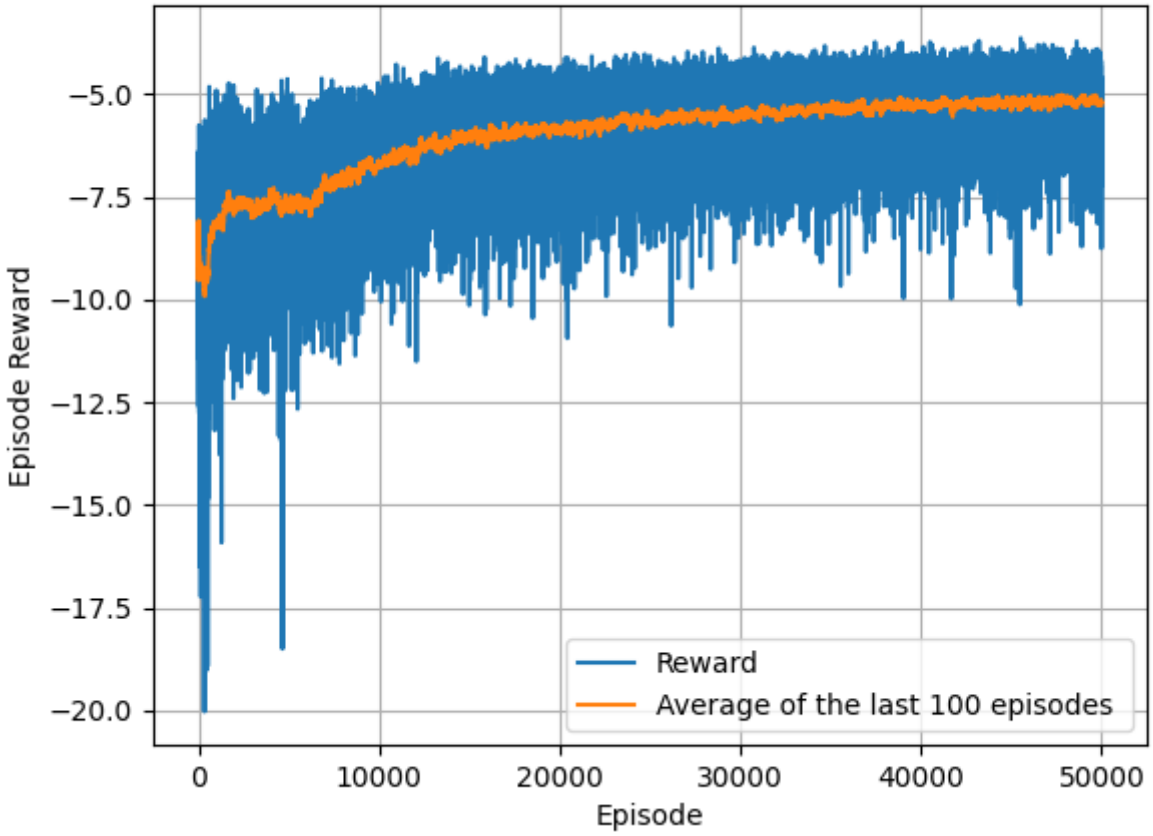
$$\mathrm{target\_q = r + \gamma * (1 - mask) * Q_{target}(s', \arg\max\limits_{a'} Q(s', a'))}$$

3. $\epsilon-\mathrm{greedy}$设值起始最大值为0.9，终止最小值为0.01，且在前$\mathrm{total_episodes}/5$ 回合内由起始最大值指数递减为终止最小值，然后保持最小值不变。

4. 同质智能体间使用共享参数，进而减缓惰性智能体lazy agent的出现。

5. 实现中设置一个episode后更新n次，而不是一个step更新一次，具体地设置n=1，即一个episode后更新一次网络参数。

学习曲线如下，**收敛在-5.2左右**，具体数值见`VDN.log`。



3. QMIX for simple spread

3.1 从VDN到QMIX

基于价值的MARL算法的基本假定是IGM (Individual Global Max)条件，即整体Q值$\mathrm{Q_{tot}}$最大时对应的联合动作为各智能体局部Q值$\mathrm{Q_a}$最大时对应的动作。
VDN为了满足IGM条件，直接简单地将$\mathrm{Q_{tot}}$分解为各$\mathrm{Q_a}$的加和形式，此时$\frac{\partial \mathrm{Q_{tot}}}{\mathrm{Q_a}} = 1$。
QMIX对VDN作出改进，针对如何分解$\mathrm{Q_{tot}}$这一点，提出了更一般化的单调性条件，即只要求$\frac{\partial \mathrm{Q_{tot}}}{\mathrm{Q_a}} \geq 0$即可，具体代码实现时将分解系数w设置为非负。

3.2 细节

使用的技巧与上述VDN中2.1节完全一致，即用DRQN代替DQN，Double DQN减缓Q值高估，$\epsilon - \mathrm{greedy}$的设置、参数共享等。它们在VDN部分已经介绍，这里不再展开。

3.3 全局状态分析

全局状态s的获取，通常有两种做法，使用环境接口提供的全局状态，或者拼接concate各个智能体的局部观测作为全局状态。

此外，MAPPO论文中还介绍了`Agent-Specific Global State (AS)`，即拼接环境提供的全局状态和各智能体的局部观测。显然AS会造成输入维度过高，因为存在重复冗余的信息。为此，MAPPO还提出了`Feature-Pruned Agent-Specific Global State (FP)`，即在AS基础上去除重复的特征。

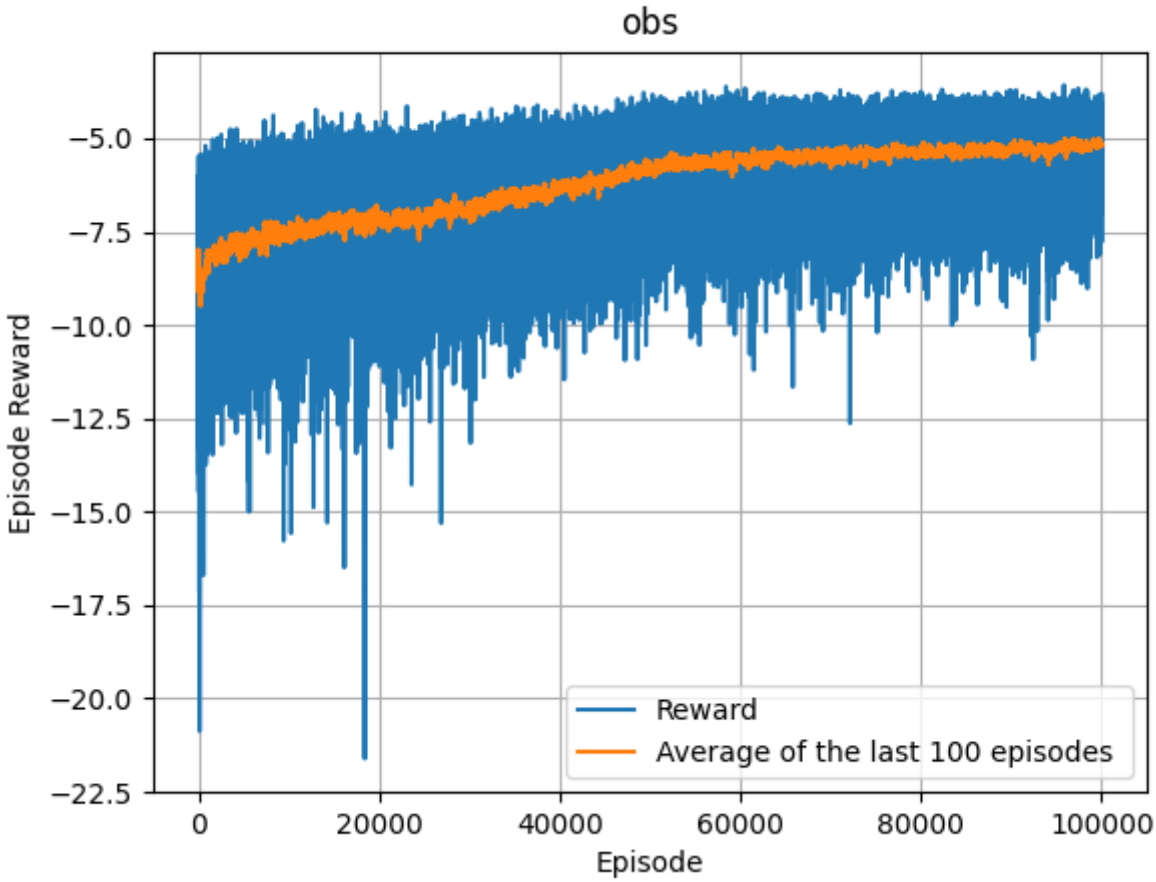
由于simple spread场景 (包括整个MPE环境) 没有提供全局状态，故MAPPO中的`AS`及`FP`方法没法使用。

关于全局状态，这里做了三种定义：

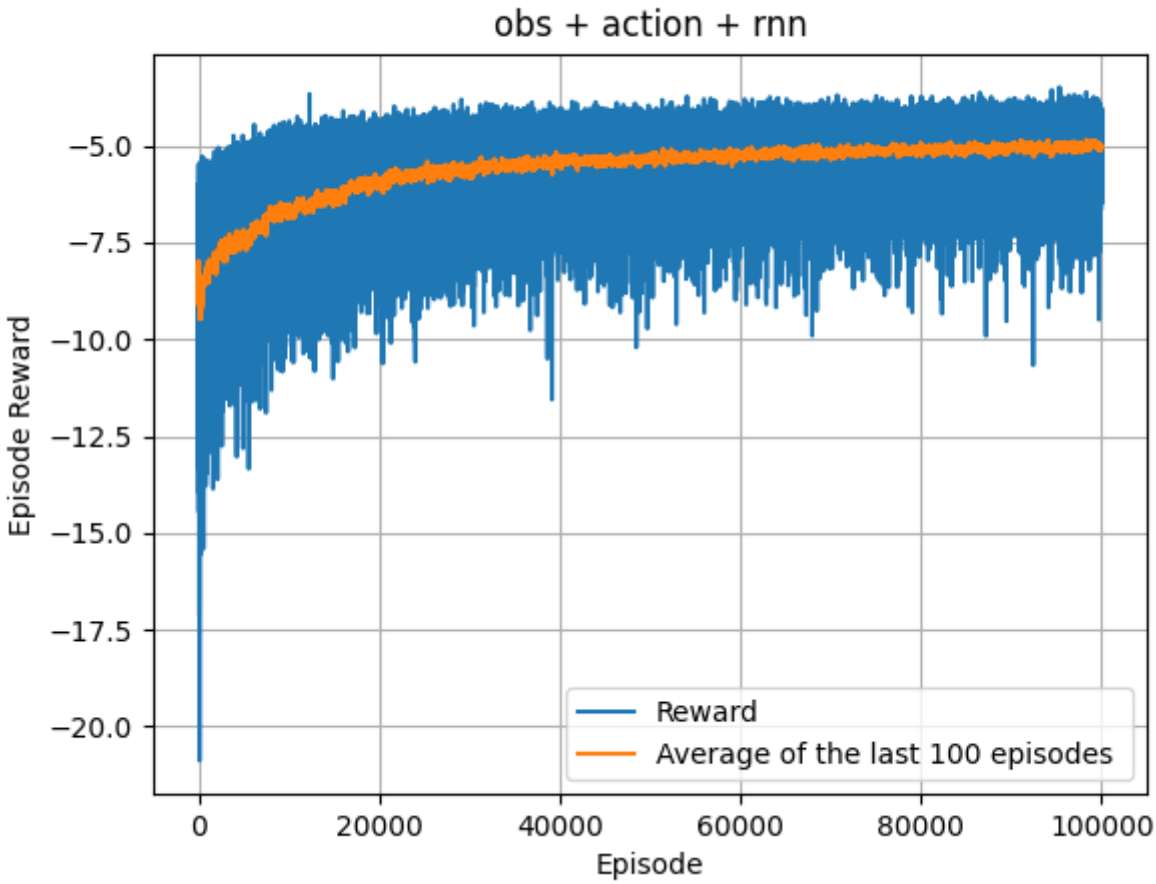
1. 拼接各智能体的**局部观测**作为全局状态，其
   $\text{dim\_state} = \text{dim\_obs} * \text{num\_agents};$
2. 拼接各智能体的**局部观测和上一时刻的动作**作为全局状态，其
   $\text{dim\_state} = (\text{dim\_obs} + \text{dim\_action}) * \text{num\_agents};$
3. 拼接各智能体的**局部观测、上一时刻的动作及RNN的隐藏状态**作为全局状态，其
   $\text{dim\_state} = (\text{dim\_obs} + \text{dim\_action} + \text{dim\_hidden\_size}) * \text{num\_agents}$。

由于作业要求QMIX算法对episodes总数不设限制，加之其本身网络结构比VDN更加复杂，这里又定义了三种全局状态用以对比实验，训练时间较久，调参实在不易，而且对算法思想的学习没有太大帮助，这里没有进行全面、细粒度的调参。

三类全局状态的对应的收敛曲线如下，最终的**收敛值分别为-5.2、-5.3、-5.0**，具体数值见相应的log文件。


obs
Episode Reward
−5.0
−7.5
−10.0
−12.5
−15.0
−17.5
−20.0
−22.5
0
20000
40000
60000
80000
100000
Episode
Reward
Average of the last 100 episodes


obs + action
Episode Reward
−5.0
−7.5
−10.0
−12.5
−15.0
−17.5
−20.0
0
20000
40000
60000
80000
100000
Episode
Reward
Average of the last 100 episodes



可以看到，拼接各智能体的**局部观测、上一时刻的动作及RNN的隐藏状态**作为全局状态，这种情况下，效果最优（收敛值在-5.0之上）。

4. 总结

MADDPG、VDN、QMIX都采用了集中式训练、分散式执行 (CTDE) 的框架，即训练时用到了全局信息，包括全局状态 (或所有智能体的局部观测) 和全局动作，每个智能体决策时仅基于自己的局部观测。它们的区别在于:

VDN、QMIX是基于值分解的算法，是单智能体DQN算法在多智能体上的扩展。由于基于团队奖励，它们只能处理合作型的任务或环境，并且和DQN一样，它们没法直接处理连续性动作，因为没法穷举所有动作来选取使Q最大的动作，即$\mathrm{argmax}_{\mathrm{a}}\ \mathrm{Q}(\mathrm{s},\mathrm{a})$

MADDPG算法，属于Actor-Critic算法，Actor网络使用策略梯度的梯度上升方法训练，Critic使用TD error训练。每个智能体都有其自己的中心式的Critic网络 (同质智能体可以采用参数共享)，专门对该智能体进行评价，而不是采用值分解的方式对整个团队评价，这一性质也使得MADDPG能够处理异质智能体间的竞争情形及混合情形，当然也能处理合作型的场景。