**MADDPG接入pysc2源代码说明文档**

修订记录

| 日期 | 修订版本 | 修改章节 | 修改描述 | 作者 |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

目录

[1. 原理介绍 4](#_Toc517182093)

[2. 总流程图 4](#_Toc517182094)

[3. 分模块介绍 5](#_Toc517182095)

[3.1 Maddpg算法: 5](#_Toc517182096)

[3.1.1 代码 5](#_Toc517182097)

[3.1.2 流程图 6](#_Toc517182098)

[3.1.3 详细说明 7](#_Toc517182099)

[3.2 动作空间 8](#_Toc517182100)

[3.2.1 代码 8](#_Toc517182101)

[3.2.2 功能描述 9](#_Toc517182102)

# 原理介绍

MADDPG主要是将深度强化学习应用于多智能体协同合作领域。传统的算法应用于多智能体环境下往往会遇到一定的困难： Q-learning会受到环境不稳定性的挑战，而policy gradient方法在智能体数目增多时，会有variance变大的问题。而MADDPG是一个actor-critic方法的变种，在考虑其他智能体的action policy的同时，能够成功地学到多智能体的协同policy。另外，MADDPG还引入了一种训练规则，它集成了各个智能体的policy，来得到一个更加鲁棒的多智能体policy。MADDPG相比现存方法在既合作又竞争的环境下具有明显的优势，在这种环境下，智能体群体能够发现各种物理和信息层面上的合作策略。

# 总流程图

图片包含 屏幕截图

已生成极高可信度的说明

具体步骤如下：

1. 初始化:

env设置方法见pysc2.env.sc2\_env.py的注释，在train\_maddpg.py文件中，通run\_thread函数初始化地图配置并开始游戏，FLAGS.agent指定agent路径，这里是

"maddpg.maddpg.agent.MADDPG"

MADDPG.selected\_units(obs)，获取当前obs里选中的单位数量和坐标

MADDPG.select\_unit(i，obs，env)，选择第i个控制组，即根据控制组编号选择单位

MADDPG.build\_group(obs，env)，建立控制组，每个红方单位为一组，即为单位编号

MADDPG.get\_obs(obs，env)，从obs获取各单位的相对位置信息

MADDPG.make\_one\_action(action\_id, spatial\_coordinates)，根据原始的524个动作编号和动作参数获得动作（目前已弃用这种方法，如果需要使用所有动作时可以考虑）

MADDPG.step(obs)，执行当前动作

在maddpg.sc2\_env.runner.run\_loop函数中，创建session

env.reset()后，选择红方所有单位，进行编组，即为智能体编号

为每个智能体初始化动作空间，目前为3个动作（等待，攻击，使用技能）

为每个智能体初始化训练器trainner

1. 开始训练

当胜率和loss数量大于1，保存胜率和loss

将相对位置作为状态输入训练器获取动作权重，选择权重最大的执行

用选择控制组的方式分别选择每一个智能体，执行动作

获取新的相对位置，reward

将旧的相对位置，动作，reward和新的相对位置添加到经验池

每一百个train\_step更新loss function

每一千个reward保存一次模型

# 分模块介绍

## Maddpg算法:

### 代码

|  |
| --- |
| 核心代码  def p\_train(make\_obs\_ph\_n, act\_space\_n, p\_index, p\_func, q\_func, optimizer, grad\_norm\_clipping=None, local\_q\_func=False, num\_units=64, scope="trainer", reuse=None):  with tf.variable\_scope(scope, reuse=reuse):  # create distribtuions  act\_pdtype\_n = [SoftCategoricalPdType(len(act\_space)) for act\_space in act\_space\_n]   # set up placeholders  obs\_ph\_n = make\_obs\_ph\_n  act\_ph\_n = [act\_pdtype\_n[i].sample\_placeholder([None], name="action"+str(i)) for i in range(len(act\_space\_n))]   p\_input = obs\_ph\_n[p\_index]   p = p\_func(p\_input, int(act\_pdtype\_n[p\_index].param\_shape()[0]), scope="p\_func", num\_units=num\_units)  p\_func\_vars = U.scope\_vars(U.absolute\_scope\_name("p\_func"))   # 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() #act\_pd.mode() #  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\_index]], 1)  q = q\_func(q\_input, 1, scope="q\_func", reuse=True, num\_units=num\_units)[:,0]  pg\_loss = -tf.reduce\_mean(q)   loss = pg\_loss + p\_reg \* 1e-3   optimize\_expr = U.minimize\_and\_clip(optimizer, loss, p\_func\_vars, grad\_norm\_clipping)   # Create callable functions  train = U.function(inputs=obs\_ph\_n + act\_ph\_n, outputs=loss, updates=[optimize\_expr])  act = U.function(inputs=[obs\_ph\_n[p\_index]], outputs=act\_sample)  p\_values = U.function([obs\_ph\_n[p\_index]], p)   # target network  target\_p = p\_func(p\_input, int(act\_pdtype\_n[p\_index].param\_shape()[0]), scope="target\_p\_func", num\_units=num\_units)  target\_p\_func\_vars = U.scope\_vars(U.absolute\_scope\_name("target\_p\_func"))  update\_target\_p = make\_update\_exp(p\_func\_vars, target\_p\_func\_vars)   target\_act\_sample = act\_pdtype\_n[p\_index].pdfromflat(target\_p).sample()  target\_act = U.function(inputs=[obs\_ph\_n[p\_index]], outputs=target\_act\_sample)   return act, train, update\_target\_p, {'p\_values': p\_values, 'target\_act': target\_act} |

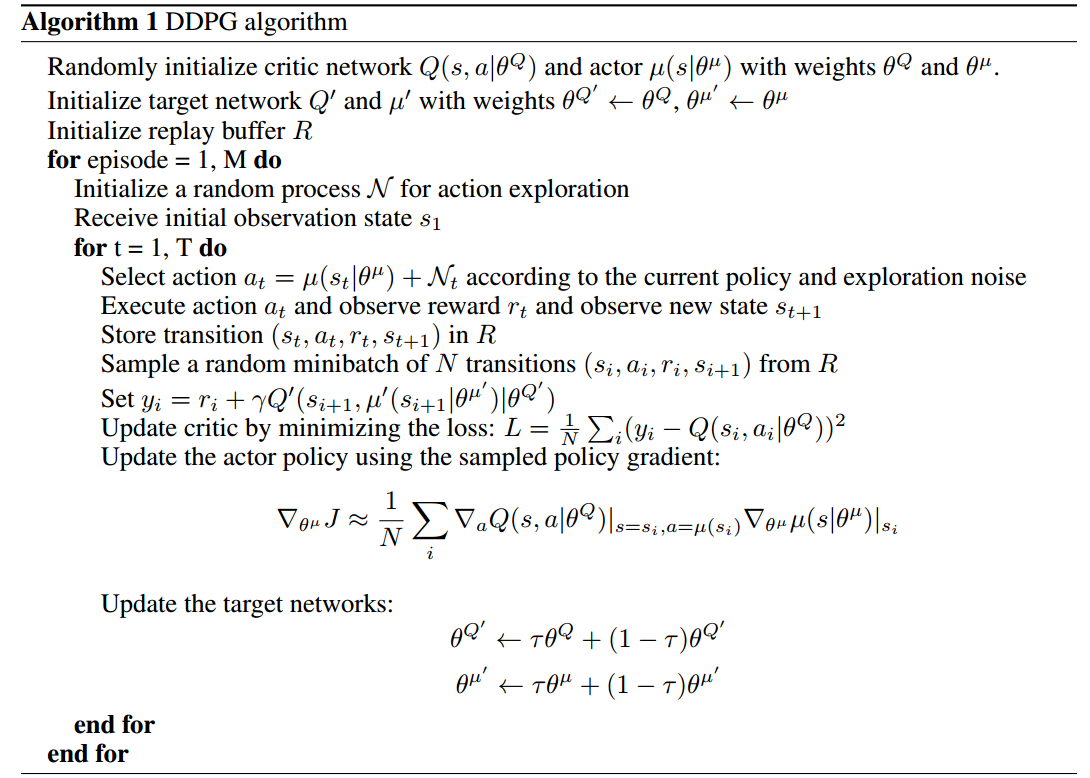
### 流程图

图片包含 屏幕截图

已生成极高可信度的说明

### 详细说明

q\_train和p\_train相似，都是返回可调用的函数，通过这种方式将tensorflow训练过程可视化，可以打印训练过程中每一步的数据内容，通过MADDPGAgentTrainer.update函数，每训练一百步更新一次网络。p相当于伪代码里的μ，算法流程可以参照伪代码理解。



## 动作空间

### 代码

|  |
| --- |
| 核心代码  import numpy as np from pysc2.lib import actions from pysc2.lib.features import SCREEN\_FEATURES  \_PLAYER\_RELATIVE = SCREEN\_FEATURES.player\_relative.index \_UNIT\_TYPE = SCREEN\_FEATURES.unit\_type.index \_SELECTED = SCREEN\_FEATURES.selected.index \_PLAYER\_FRIENDLY = 1 \_PLAYER\_NEUTRAL = 3 # beacon/minerals \_PLAYER\_HOSTILE = 4 \_NO\_OP = actions.FUNCTIONS.no\_op.id \_MOVE\_SCREEN = actions.FUNCTIONS.Move\_screen.id \_ATTACK\_SCREEN = actions.FUNCTIONS.Attack\_screen.id \_SELECT\_ARMY = actions.FUNCTIONS.select\_army.id \_SELECT\_UNIT = actions.FUNCTIONS.select\_unit.id \_KD8CHARGE = actions.FUNCTIONS.Effect\_KD8Charge\_screen.id \_SELECT\_CONTROL\_GROUP = actions.FUNCTIONS.select\_control\_group.id \_NOT\_QUEUED = [0] \_SELECT\_ALL = [0]  def wait(obs, env):  action = [actions.FunctionCall(\_NO\_OP, [])]  obs = env.step(action)[0]  return obs  def KD8Charge(obs, env):  player\_relative = obs.observation["screen"][\_PLAYER\_RELATIVE]  army\_y, army\_x = (player\_relative == \_PLAYER\_HOSTILE).nonzero()  if not army\_y.any():  action = [actions.FunctionCall(\_NO\_OP, [])]  obs = env.step(action)[0]  if obs.last():  return False  else:  return obs  elif \_KD8CHARGE in obs.observation["available\_actions"]:  index = np.argmax(army\_y)  target = [army\_x[index], army\_y[index]]  action = [actions.FunctionCall(\_KD8CHARGE, [\_NOT\_QUEUED, target])]  obs = env.step(action)[0]  if obs.last():  return False  return obs  def attack(obs, env):  player\_relative = obs.observation["screen"][\_PLAYER\_RELATIVE]  army\_y, army\_x = (player\_relative == \_PLAYER\_HOSTILE).nonzero()  if not army\_y.any():  action = [actions.FunctionCall(\_NO\_OP, [])]  obs = env.step(action)[0]  if obs.last():  return False  else:  return obs  elif \_ATTACK\_SCREEN in obs.observation["available\_actions"]:  index = np.argmax(army\_y)  target = [army\_x[index], army\_y[index]]  action = [actions.FunctionCall(\_ATTACK\_SCREEN, [\_NOT\_QUEUED, target])]  obs = env.step(action)[0]  if obs.last():  return False  return obs   def get\_action(action, obs, env):  combined\_action = [wait, KD8Charge, attack]  action = action.tolist()  action\_id = action.index(max(action))  obs = combined\_action[action\_id](obs, env)  return obs |

### 功能描述与修改说明

将动作权重输入get\_action函数，选择权重最大的动作执行，可执行动作包括[wait, KD8Charge, attack](等待，释放技能，攻击)，每一个技能都是基于原本的524个动作里的对应动作改写而成。

执行win\_pro.py绘制胜率曲线，load\_loss.py绘制loss曲线。

状态空间可以直接在MADDPG.get\_obs函数里修改，目前只获取相对位置。

Reward可在pysc2.env.sc2\_env.py文件中修改SC2Env.\_step函数中的reward。如图，reward设置为游戏本身的reward（杀死一个敌方单位+5，己方死亡-1）加上每局游戏胜负的reward（在图中outcome为胜负输出，胜1，负-1，平0，故乘以5表示每局游戏胜负的reward为5）

