# 基于模糊逻辑规则集控制器初始化的DDPG控制器算法

Mars 董林森

2017.7.2

版本号1.0

目录

[基于模糊逻辑规则集控制器初始化的DDPG控制器算法 1](#_Toc486882180)

[1. 算法模型： 2](#_Toc486882181)

[1.1 DDPG模型 2](#_Toc486882182)

[1.2 模糊逻辑规则模型 3](#_Toc486882183)

[2. 算法流程 4](#_Toc486882184)

[2.1输入输出的预处理 4](#_Toc486882185)

[2.2 DDPG的权值初始化 4](#_Toc486882186)

[2.3 DDPG 强化学习 5](#_Toc486882187)

[2.4 控制器使用 6](#_Toc486882188)

[3 代码文档 7](#_Toc486882189)

[3.1代码目录结构 7](#_Toc486882190)

[3.2 src 8](#_Toc486882191)

[3.2.1 Learner 8](#_Toc486882192)

[3.2.2 Fuzzy Logic Rule 15](#_Toc486882193)

[3.2.3 Environment 19](#_Toc486882194)

[3.3 test 20](#_Toc486882195)

[3.3.1 initializationTest: 20](#_Toc486882196)

# 算法模型：

算法模型实现了一种通用的控制器模型，该模型实现了两个功能，第一阶段为基于模糊逻辑规则初始化控制器，和第二阶段强化学习对控制器进行进一步优化。

该模型可以解决连续行为空间的控制问题，即控制的目标的行为空间存在连续型变量。

## 1.1 DDPG模型

模型基于强化学习中的DDPG（Deep Deterministic Policy Gradient）模型实现。在此基础上，通过使用模糊逻辑规则集对模型中的参数值进行初始化。

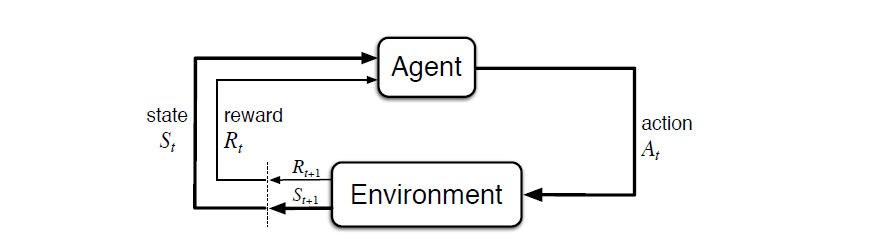
控制器的实现基于智能体（Agent）建模思想，将需要控制的实体抽象为Agent，Agent通过观测（Observation）和行为（Action）与外界环境交互，形成控制闭环。基于  
Agent的建模基本结构如下：  


图1

控制器的算法模型基于DDPG算法实现，主要包括两个神经网络，分别为Actor网络和Critic 网络。其结构如下图：

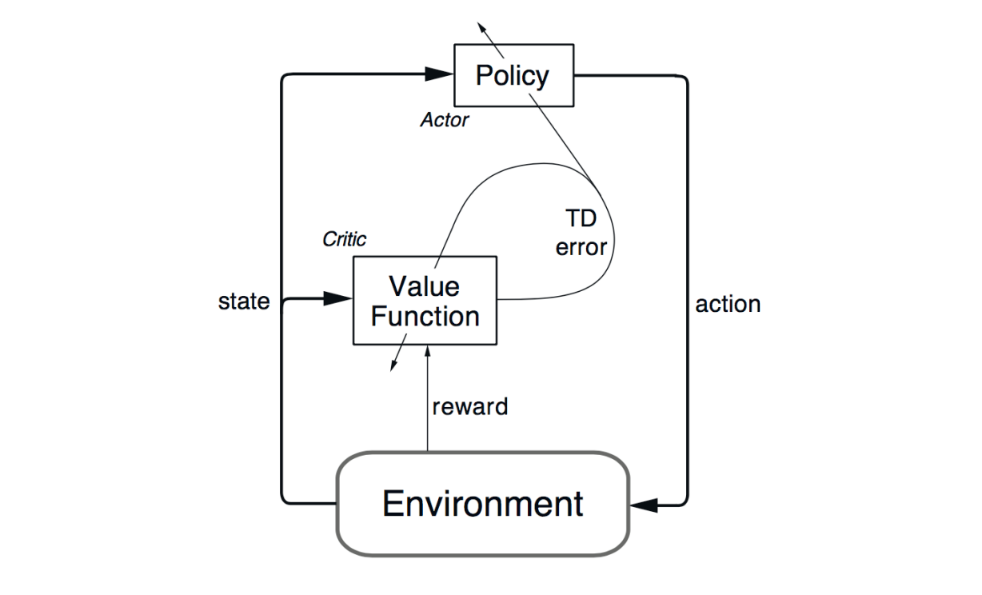


图2

图2中，粗线条表示DDPG前向的控制阶段，细线条表示完成一次控制后，根据环境的奖赏进行学习的反向的学习阶段。

Actor网络实现了一个强化学习中的策略函数（Policy Network），输入为某时刻的观测值，输出为在该观测值状态下的控制器的决策，即控制的行为，例如前进速度、方向等。

Critic网络实现了强化学习中状态-行为值函数（Q Network），即为对某个状态-行为组合好坏的评估。

## 1.2 模糊逻辑规则模型

同时在初始化阶段使用到基于模糊逻辑规则实现的控制器，表示了一种策略，即完成了从观测到行为的映射关系。

可以简单理解为如下形式：

其单条规则的基本规则的组成形式为

：称为后件，表示被推断出的值，即需要被决策的变量。

：表示前件，表示实际观测到的某些变量的值，如温度，高度，速度等。

：表示每个变量的模糊类别，或者称为Linguistic label。每个变量可包含多个类别，如： 表示温度，则对应的模糊类别的集合可以定义为 ，在任意一个的值下，通过Membership function 计算，可以得到变量在所以类别下的真值，每一个真值都为一个[0 1]的连续值，应尽量保持在任意某个输入值下，对应的所有类别的真值的和为1。通过Membership function的计算过程称为Fuzzy化的过程。

：连接所有前件变量的一个逻辑函数，它的输入为所有变量的在规则对应的模糊类别下的真值，输出为后件为类别的真值。

：定义类似，为输出变量的模糊类别。由函数可以得到在该类别下的真值，再通过对的Membership function进行逆运算过程，求出的实际值。逆运算的过程称为Defuzzy，主要有LMOM, TSK等。

# **算法**流程

这部分主要叙述基于模糊逻辑规则集初始化的DDPG控制器算法的流程。主要包括两步，第一通过模糊逻辑规则集对DDPG进行权值初始化的有监督学习过程；第二步为DDPG控制的强化学习过程。

## 2.1输入输出的预处理

根据环境和被控制的实体，进行输入输出的抽象。若将整个模型看做一个控制器，则每个时刻，系统的输入为实际观测到的值，如敌方船只数量，我方船只数量。输出为需要被控制变量值，如速度的大小，前进的方向。

但不应该直接将模型设计为原始输入到原始输出的映射，如控制一个物体的移动时，可以发起两个控制指令，以某一速度前进或停止。如果直接将输出映射为两个变量，分别表示前进和停止，则并不是一个合理的设计，因为两个指令存在一定的逻辑关系，所以应抽象为对物体移动速度进行控制，当决策得到的速度为0时，即表示停止。

所以最终控制器的实际输入和输出都通过了一定的人为设计的预处理。类似如下结构

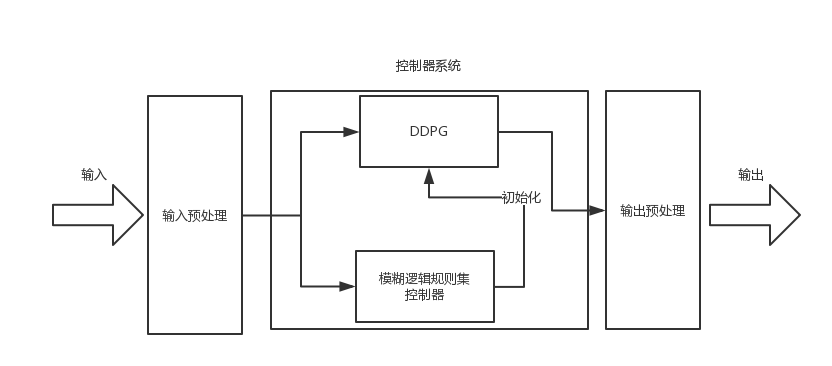


图3

## 2.2 DDPG的权值初始化

DDPG模型包括两个神经网络Actor和Critic，分别对其进行初始化，初始化的过程为可以看成有监督学习过程，过程不需要控制器和环境进行交互。

在确定模型的输入输出后，根据专家知识集等信息，分别产生两个模糊逻辑控制器，称为Fuzzy Logic Controller 和 Fuzzy Logic Valuer。前者对应Actor网络的初始化，后者对应Critic网络的初始化。Fuzzy Logic Controller对应Actor网络的初始化，其输入输出即对应图3控制器系统的输入输出，完成控制作用。Fuzzy Logic Valuer类似强化学习中的V函数，对应Critic网络的初始化，输入某个时刻的状态，输出为该状态下最终累计奖赏值。因此由于算法的需要，必须需要专家知识集提供能够生成这两种控制器的相关规则。

算法如下：

Algorithm: DDPG weight initialization

Define:

State Variable at time step is

Action Variable at time step is

Fuzzy logic controller is and its output action is

Fuzzy logic valuer is andi its output value is

Critic is ), and its output is

Actor is , and its output is

According to the DDPG algorithm, also create target network for critic and actor: ) and

for epoch =1, M do:

Randomly generate N state

Compute and according to and

Compute and according to

Compute in either ) or

Update critic by minimizing the loss :

Update actor by minimizing the loss:

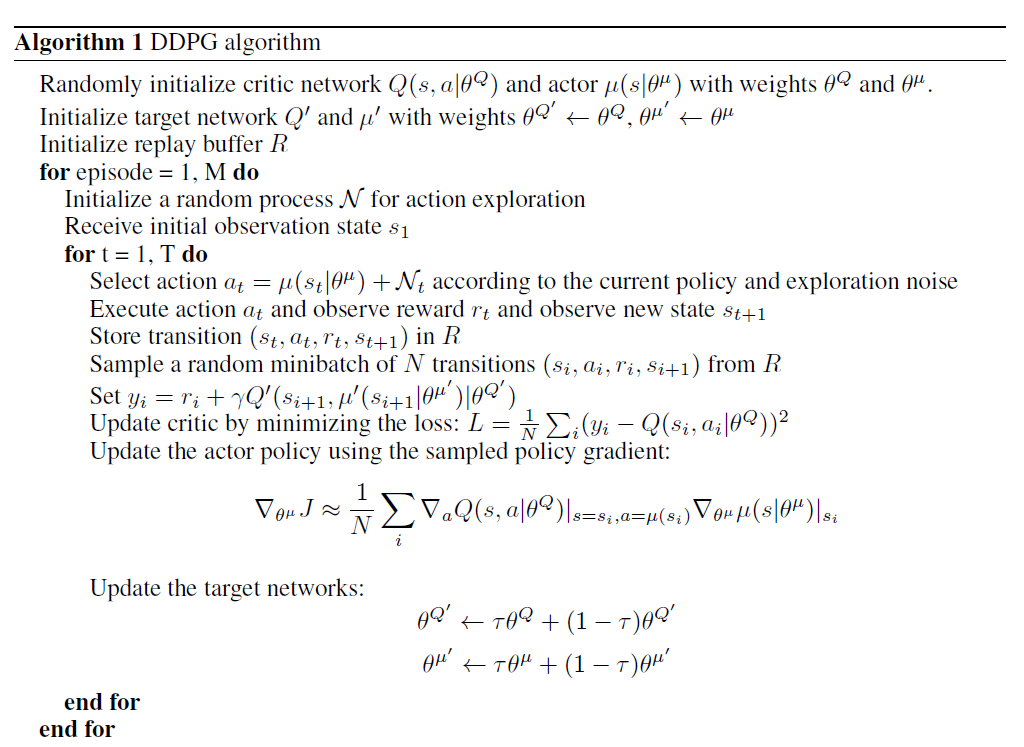
Update target networks according the DDPG algorithm

endfor

## 2.3 DDPG 强化学习

在完成初始化后，即可去掉模糊逻辑规则器部分，将DDPG放入环境中，进行强化学习。

DDPG算法的学习算法如下图：



在进行强化学习时，由于涉及到多智能体协作问题，可以考虑将DDPG模型的强化学习模型做相应更改，该版本的实现中，暂时没有考虑因协作问题对算法做特殊化的涉及。

在强化学习过程中，需要注意环境的Reward的设计，它对模型的学习过程起到了决定性的作用。在设计Reward的计算方法时，应避免将关于策略的先验知识加入其中，即设计者不能通过Reward告诉控制器以怎样的方式去最大化累计奖赏。

## 2.4 控制器使用

在完成一定的训练后，即可将当前的网络模型，使用到实际场景中，此时不在进行学习行为。

# 代码文档

## 3.1代码目录结构

GRAIC主目录下主要包括 \data, \doc, \log, \src, \test 五个部分。

3.1.1 data

初始设计的思想是为了做基本的数据处理， 暂时未用到。所以该部分可以保留或删除。

3.1.2 doc

文档存放目录

3.1.3 log

日志存放目录，init.py中获取了LOG\_PATH，为该目录的绝对路径，方便其他模块写入日志时获取地址。

初始化时产生的日志文件存在log\initialTrain 目录下，根据每次运行时间创建一个目录，如6-12-13-32-17 表示6月12日13点32分17秒。日志内容提供包括loss.txt存放了这次训练过程损失函数的值随训练次数的变化。Loss.txt为一个json格式文件，为一个list，每个元素为一个字典，结构如下：

{

"Critic Cost": "10.20656", // Critic网络的损失函数值

"epoch": 0, //第几轮训练

"Actor Cost": "10.24451" // Actor网路的损失函数值

},

log\initialTrain还可以用于存储网络训练过程中的模型文件，调用DDPGInitializer. save\_all\_model() 方法即可将训练时某一步的模型保存在对应的日志目录下。具体使用参考DDPGInitializer部分文档。

## 3.2 src

src\\_\_init\_\_.py中声明了 SRC\_PATH，表示src目录的绝对路径。

### 3.2.1 Learner

Learner部分实现了DDPG算法的神经网络和对对其初始化的方法。

3.2.1.1 Class Network(object)

表示网络模型的基类

Properties:

1. *sess*: 为该网络的TensorFlow session

Methods:

1. *\_\_init\_\_(self, sess)*:

使用一个tensorflow session初始化一个Network，

1. variable(self, shape, f):

返回一个形状为shape的tf.Variable，并以根据f的值均匀随机初始化权值

1. batch\_norm\_layer(self, x, training\_phase, scope\_bn, activation=None):

对x添加一个batch norm层，并根据是否在训练判断是否重用该层。

3.2.1.2 Class CriticNetwork(Network)

表示DDPG算法中Critic网络的类

Properties:

1. state\_input:

网络的state变量输入，类型为tf.placeholder, 数据类型为float

1. action\_input:

网络的action变量输入，类型为tf.placeholder, 数据类型为float

1. q\_value\_output:

网络的Q值输出，为一个浮点数

1. target\_state\_input:

属于Target网络，类比state\_input

1. target\_action\_input:

属于Target网络，类比action\_input

1. target\_q\_value\_output:

属于Target网络，类比q\_value\_output

1. target\_update：

对target网络的更新操作

1. target\_is\_training：

表示target网络是否正在训练，参考is\_training

1. net:

网络中所有的权值，为一个python list

1. is\_training:

表示是否正在训练的bool值，tf.placeholder，数据类型为tf.bool

1. y\_input:

表示DDPG算法中，用于计算loss中的y的值，为一个tf. placeholder

1. cost:

表示loss函数的计算

1. optimizer:

TensorFlow中的优化器，优化算法为AdamOptimizer

1. action\_gradients:

action输出的梯度，用于对actor网络的更新，类型为tf.gradients

1. time\_step:

用于记录训练次数的int值

Methods

1. \_\_init\_\_(self, sess, state\_dim, action\_dim):

Critic的构造函数，参数依次为tf.Session, 状态变量的形状，行为变量的形状

1. create\_training\_method(self):

创建训练模型，主要包括cost和 optimizer，无返回值

1. create\_q\_network(self, state\_dim, action\_dim):

创建Q网络，即Critic网络，返回值对应Properties中变量，依次为：

state\_input, action\_input, q\_value\_output, net, is\_training

1. create\_target\_q\_network(self, state\_dim, action\_dim, net):

创建Target Q网络，类比network，不同在于target网络维护了对于net网络的权值的滑动平均值。具体参考tf.train.ExponentialMovingAverage。返回值为state\_input, action\_input, q\_value\_output, target\_update, is\_training。target\_update表示对target网络的更新操作。

1. update\_target(self):

更新target网络

1. train(self, y\_batch, state\_batch, action\_batch):

训练Critic网络，参数依次为y，state，action。batch大小必须保持相同。返回这一次训练的cost。

1. gradients(self, state\_batch, action\_batch):

返回q值对action的梯度值

1. target\_q(self, state\_batch, action\_batch):

计算并返回target网络对应state和action输入的Q值

1. q\_value(self, state\_batch, action\_batch):

计算并返回网络对应state和action输入的Q值

3.2.1.3 Class ActorNetwork(Network)

表示DDPG算法中Actor网络的类

Properties:

1. state\_dim:

状态变量的大小

1. action\_dim

行为变量的大小

1. state\_input:

状态输入，类型为tf.placeholder，由self. create\_network返回

1. action\_output:

行为输入，类型为tf.placeholder，由self. create\_network返回

1. net:

所有网络权值的Python List变量

1. is\_training:

表示是否正在训练的bool值，tf.placeholder，数据类型为tf.bool

7． target\_state\_input:

target网络状态输入，类型为tf.placeholder，由self. create\_target\_network返回

1. target\_action\_output:

target网络状态输入，类比action\_output

1. target\_update:

target网络的更新操作

1. target\_is\_training:

类比is\_training

1. q\_gradient\_input:

通过Critic计算出的Q值函数对action的梯度值，为更新网络时需要的值。类型为tf.placeholder

1. parameters\_gradients:

由DDPG中对Actor的更新公式得出网络参数的梯度值

1. optimizer:

根据parameters\_gradients值，生成的优化器。

1. y\_input:

用于网络初始化算法中的y值输入，作为action的标签信息。类型为tf.placeholder，大小为action\_dim

1. cost:

初始化算法中的cost计算

1. initial\_optimizer:

初始化算法中的优化器

Methods:

1. \_\_init\_\_(self, sess, state\_dim, action\_dim):

构造函数

1. create\_training\_method(self):

构造网络训练方法，无返回值

1. create\_network(self, state\_dim, action\_dim):

创建网络，依次返回state\_input, action\_output, net, is\_training，对应property中的变量

1. create\_target\_network(self, state\_dim, action\_dim, net):

类似create\_network，依次返回state\_input, action\_output, target\_update, is\_training，对应target中的变量。

1. update\_target(self):

更新target网络

1. train(self, q\_gradient\_batch, state\_batch):

训练Actor网络

1. initial\_train(self, action\_label\_batch, state\_batch):

用于初始化算法中对Actor网络的训练。

1. actions(self, state\_batch):

用于训练过程中，输入state\_batch，返回action\_batch，is\_training设为True

1. action(self, state):

用于测试过程中，输入state，返回action，is\_training设为False

1. target\_actions(self, state\_batch):

返回Target网络的action\_batch，用于训练过程

3.2.1.4 Class NetworkCommon(object):

保存网络中的部分参数值，以及一些常用操作

Properties:

1. REPLAY\_BUFFER\_SIZE
2. REPLAY\_START\_SIZE
3. BATCH\_SIZE
4. GAMMA Reward的衰减系数
5. CRITIC\_LAYER1\_SIZE
6. CRITIC\_LAYER2\_SIZE
7. CRITIC\_LEARNING\_RATE
8. CRITIC\_TAU 维持Target网络的滑动平均值的参数
9. CRITIC\_L2 L2正则化系数
10. ACTOR\_LAYER1\_SIZE
11. ACTOR\_LAYER2\_SIZE
12. ACTOR\_LEARNING\_RATE
13. ACTOR\_TAU 维持Target网络的滑动平均值的参数
14. ACTOR\_BATCH\_SIZE
15. ACTOR\_L2

Methods:

* + - * 1. create\_variable\_summary(var, name\_scope):

用于保存变量的历史值，用于tensorboard的数据可视化使用

* + - * 1. return\_file\_writer(sess, log\_file\_dir):

返回一个目标文件为log\_file\_dir的tf.summary.FileWriter，用于tensorboard的数据可视化使用

3.2.1.5 Class DDPGController(object):

表示DDPG controller的类，主要包括了Critic和Actor的实例，与环境的交互，人为添加的噪声用于环境的探索等操作

Properties:

* + 1. action\_dim:

行为的大小

* + 1. state\_dim

状态的大小

* + 1. actor\_network:

一个class ActorNetwork的实例

* + 1. critic\_network:

一个class CriticNetwork的实例

* + 1. environment:

一个class Environment的实例

* + 1. exploration\_noise:

一个class OUNoise的实例，用于对action添加噪声，增加对环境的探索

* + 1. model\_saver:

用于对tensorflow模型进行保存的tf.train.Saver()

* + 1. name:

模型的名字，默认为 DDPG

* + 1. replay\_buffer:

一个class ReplayBuffer的实例，参考DDPG算法中的replay buffer的作用

* + 1. sess

TensorFlow的session变量

Methods:

1. \_\_init\_\_(self, env):

传入一个Environment的实例，构造一个DDPGController

1. train(self):

训练控制器，先对critic和actor网络训练，再更新对应的Target网络

1. noise\_action(self, state):

返回添加噪声后的actor网络的action值，用于训练阶段获取action

1. action(self, state):

返回不添加噪声的action值，用于测试阶段

1. perceive(self, state, action, reward, next\_state, done):

DDPG算法中，保存一次训练样本，并根据当前replay\_buffer状态决定是否进行一次训练

1. initial\_train(self, mini\_batch):

进行一次初始化算法中的训练。传入的mini\_batch为一个5\*batch\_size\*?大小的数组，依次存储了state\_batch，action\_batch，action\_label\_batch，value\_label\_batch，done\_batch

1. save\_model(self, path, check\_point):

保存tf.session中的网络模型保存在path路径中，参考tf.train.Saver.save()方法。

1. load\_model(self, path):

载入path路径下的模型，参考tf.train.Saver.restor()方法

Note：对于DDPG中输入的state，必须先经过归一化到[0, 1]，然后输入到模型中，输出的action也为归一化值，所以需要转换为实际的值后再输入到环境中。

3.2.1.6 Class DDPGInitializer(object):

提供初始化算法中训练数据的产生，模型保存，模型初始化训练等

Properties:

1. ddpgController:

需要被初始化的控制器，为Class DDPGController的实例

1. fuzzylogicController:

用于初始化DDPG的模糊逻辑控制器 FLC

1. fuzzyLogicValuer:

用于初始化DDPG的模糊逻辑Valuer FLV

1. log\_file\_dir:

用于保存日志文件的路径

1. model\_file\_dir:

用于保存模型文件的路径

1. mini\_batch\_size:

用于训练的数据的batch\_size大小

Methods:

1. \_\_init\_\_(self, ddpg\_controller,

fuzzy\_logic\_controller,

fuzzy\_logic\_valuer,

mini\_batch\_size):

构造函数，参数含义参考property部分

1. generate\_state\_done\_mini\_batch(self, mini\_batch\_size):

按照均匀分布随机产生数量为mini\_batch\_size大小的state数据，数据已经归一化到[0, 1]

1. generate\_training\_sample\_mini\_batch(self, mini\_batch\_size):

根据随机产生的state数据以及FLC和FLV产生训练数据，并返回

1. save\_all\_model(self, epoch):

保存DDPG模型

1. load\_model(self, path):

载入DDPG模型

1. train\_DDPG(self, epoch):

基于初始化算法训练DDPG模型，并将训练过程的日志文件保存

3.2.1.7 Class OUNoise (object):

用于产生随机白噪声的类

Properties:

1. action\_dimension:
2. mu:
3. sigma:
4. state:
5. theta:

Methods:

1. \_\_init\_\_(self, action\_dimension, mu=0, theta=0.15, sigma=0.2):
2. reset(self):

重置系统的状态

1. noise(self):

返回随机白噪声的值

3.2.1.8 Class ReplayBuffer (object):

用于保存DDPG算法中的训练样本的buffer

Properties:

1. buffer:

一个python deque，用于保存训练样本

1. buffer\_size:

buffer大小

1. num\_experiences:

当前样本数目

Methods:

1. \_\_init\_\_(self, buffer\_size):
2. get\_batch(self, batch\_size):

返回一个batch\_size大小的历史数据样本

1. size(self):

返回buffer\_size

1. add(self, state, action, reward, new\_state, done):

添加一个新的样本，超过buffer\_size，则自动溢出队头的数据

1. count(self):

返回当前历史数据样本数量

1. erase(self):

清空buffer

### 3.2.2 Fuzzy Logic Rule

3.2.2.1 Class Controller(object)

基于fuzzy logic rule实现的控制器，包括FLC和FLV

Properties:

1. input\_value\_dict:

存储控制器输入变量的字典，例如，输入变量有 ’input\_var\_1’, ’input\_var\_2’,则该字典类似为：

{

‘input\_var\_1’: 0.5,

‘input\_var\_2’:0.8

}

1. output\_value\_dict:

控制器输出变量的字典，类似Input\_value\_dic

1. rule\_section\_set:

存储rule section的list

1. name:

控制器的名称

Methods:

1. \_\_init\_\_(self, name):
2. add\_rule\_section(self, rule\_set):

添加一个rule section

1. \_assign\_input\_to\_section(self):

将输入变量字典中的值按照每个rule section各自的输入分别赋值，这么做的愿意在于每个rule section使用到的输入变量可能不同。

1. \_calc\_output(self):

计算模型的输出

Note：input\_var\_dict为一个python property类型，setter中如果被设置了新的值，则自动计算其对应的输出。

3.2.2.2 Class Defuzzifier (object)

用于对输出变量去模糊化的基类

Properties:

1. name

Methods:

1. \_\_init\_\_(self, name):
2. defuzzify(self, degree, mf):

参数为degree的值和一个membership function，然后计算其真实值

Note：该类作为基类不应该被直接使用，应该使用其子类。

Note:省略Defuzzifier的子类的文档，包括：LMOMDefuzzifier, TSKDefuzzifier。

具体计算方法的原理参见论文和资料

3.2.2.3 Class MembershipFunction (object)

成员函数类的基类

Properties:

1. name

Methods:

1. \_\_init\_\_(self, name):
2. calc(self, input\_value):

根据输入的值，计算其对应的逻辑值或者称为degree

1. inverse\_calc(self, input\_degree):

输入一个degree，进行函数的逆运算，对于反函数存在多个值的情况，求对应的多值的平均值

Note:省略MembershipFunction的子类的文档，包括LeftTriangleMF, RightTriangleMF, TriangleMF, GaussianMF

3.2.2.4 Class Variable (object)

用于表示模糊逻辑中的变量

Properties:

1. name
2. mf

成员函数的python list，因为一个变量存在多个模糊类别所以为一个list

1. value

存储变量实际的值

1. degree

计算变量在不同模糊类别下的逻辑值或称为degree，为一个字典，key为类别的名称，value为对应的逻辑值

1. linguistic\_label

该变量的模糊类别的名称的python list

1. upper\_range

变量value值的上界

1. lower\_range

变量value值的下界

Methods:

\_\_init\_\_(self, name, mf, range):

Note：value和degree都被设置为Python Property类型

3.2.2.5 Class InputVariable (Variable)

表示输入变量

Properties:

Methods:  
 1. \_\_init\_\_(self, name, range, mf, value=0.0):

1. calc\_degree(self):

分别计算该输入变量在各个模糊类别的下的真值

Note：输入变量的value的setter被重载，当value被设置时，会自动调用calc\_degree()方法

3.2.2.6 Class OutputVariable (Variable)

表示输出变量

Properties:

1. defuzzifier

一个class Defuzzifier的实例，用于对输出变量去模糊化

Methods:  
 1. \_init\_\_(self, name, mf, range, defuzzifier):

Note：degree的setter属性被重载，当通过控制器计算出新的degree时，自动调用self.defuzzifier.defuzzify()得到输出的真实值。

3.2.2.7 Class FuzzyRule(object)

表示一条模糊规则的类，包括存储这条规则的前件变量，后件变量

Properties:

1. rule\_str:

该规则的字符串表示，即”If……then……”。

1. true\_value:

这条规则在某个输入的下的真值

1. input\_var\_list:

输入变量的Python list，存储的为inputVariable类

1. input\_dict:

表示输入变量的值的字典，key为变量的名称，value为真实值

1. output\_var\_list:

类比input\_var\_list

1. output\_var\_value:

输出变量的真实值

1. \_min\_operation:

对前件逻辑值的最小化操作，有两种soft\_min 或 min

1. \_section:

该条规则所属的section的编号

Methods:

1. \_\_init\_\_(self,

input\_var\_list,

output\_var\_list,

rule\_str,

section,

min\_operation="softmin"

):

1. set\_input\_var\_value(self, new\_input\_var\_value\_dict):

设置新的输入值，传入一个字典，key为变量名，value为值

1. set\_input\_output\_dict(self, rule\_str=None):

分别设置新的输入输出的字典，即input\_dict和output\_dict，传入参数为一个规则的string

1. \_get\_true\_value(self):

计算该条规则的真值

1. set\_output\_var\_degree(self):

设置输出变量的degree为该条规则的真值，并通过去模糊化计算真实值后将其赋值给self.output\_var\_value

1. \_min\_op(self):

对输入真值的最小化操作，即取所有前件中最小的真值作为该条规则的真值

1. \_softmin\_op(self):

对输入真值的软小化操作，即对degree使用负指数加权求均值。公式如下：

其中 表示规则的真值， 表示输入变量的真值

3.2.2.8 Class FuzzyRuleSet (object)

该类为多条规则的集合，即成为一个Section

Properties:

1. input\_var\_value\_dict:

在该集合内的所有规则的前件的变量值的字典，key为变量的名称，value为真实值

1. name:
2. output\_var\_value\_dict:

类比input\_var\_value\_dict，规定一个FuzzyRuleSet中，所有的规则的输出有且仅有相同的一个，即一个FuzzyRuleSet控制系统的一个输出变量

1. rule\_list:

存储了该集合中所有规则的Python list

1. rule\_set\_output\_value:

表示对集合内所有规则计算后，得到的最终的该变量的输出值，计算方法为对集合内规则以该规则的真值为权值的加权平均值。

1. section:

表示section的编号

Methods:

1. \_\_init\_\_(self, name, section, rule\_list):

构造函数，rule\_list为包含了Class FuzzyRule 的列表

1. add\_fuzzy\_rule(self, fuzzy\_rule):

添加一条规则到实例中

1. get\_input\_var\_name(self):

将所有规则中的输入变量名称保存到一个列表中并返回

1. get\_output\_var\_name(self):

将所有规则中的输出变量名称保存到一个列表中并返回

1. assign\_input\_value(self):

将input\_var\_value\_dict中的值赋值给各条规则

1. \_calc\_value(self):

计算最终输出量的值，并返回

3.2.2.9 Class RuleParser (object)

一个用于分解规则“if ……then……”字符串的类

Methods:

parse\_if\_then\_rule(self, rule\_str):

### 3.2.3 Environment

3.2.3.1 Class Environment(object)

描述环境的类

Properties:

1. action\_dim:

行为的大小

1. action\_set:

行为的集合，存储的类型为Class Action

1. name:
2. reward:

一个Reward的实例

1. state\_dim:

状态的大小

1. state\_set:

状态的集合，存储的类型为Class State

1. time\_step\_limit:

时间步数的限制

Methods:

* + - 1. \_\_init\_\_(self, name, state\_set, action\_set):

构造一个环境实例

* + - 1. set\_action\_set(self, action\_set):

设置一个新的action\_set

* + - 1. set\_reward(self, reward):

设置一个新的reward

* + - 1. set\_state\_set(self, state\_set):

设置一个新的state\_set

* + - 1. reset(self):

重置环境

* + - 1. step(self, action):

传入一个action类，执行环境模拟。返回值为next\_state, reward, done依次表示下一个时刻状态，及时奖赏值，是否结束的标志

* + - 1. is\_done(self, state):

判断某个状态是否为一个合法的结束状态，返回一个bool

Note: 关于Environment中action，reward，state的文档，由于代码比较简单，不再单独说明，直接参照源码即可。

Note: Environment基类不应该直接被声明和使用，针对某个仿真环境或游戏环境，创建一个子类，并重载以上方法后使用。

Note:设计Environment类的想法在于，不管DDPG接入任何的仿真环境或模拟器中，其都应该提供以上方法，而这一过程通过实现一个Environment的子类进行包装和抽象。

## 3.3 test

test\ \_\_init\_\_.py中声明了 TEST\_PATH，表示test目录的绝对路径。

### 3.3.1 initializationTest:

initializationTest用于测试使用模糊逻辑规则集对DDPG控制器神经网络权值进行初始化的方法。

* + - 1. class DDPGInitializaionTest(object)

用于测试的类，用于产生一些随机的数据和控制器等。不应该被其他模块使用和继承等。

Properties:

1. controller:
2. input\_var\_list:
3. output\_var\_list:
4. section\_list:
5. section\_num:
6. value:
7. value\_rule:
8. valuer:

Methods:

1. \_\_init\_\_(self, state\_dim, action\_dim):
2. generate\_input\_var(self, input\_dim=100):

随机产生输入变量，返回的值为类型为Class InputVariable的列表

1. generate\_output\_var(self, output\_dim=10):

类比以上

1. generate\_value(self):

随机产生输入值

1. generate\_section\_list(self, rule\_per\_seciont\_num=10):

根据输入和输出变量构造section的列表

1. generate\_fuzzy\_logic\_controller(self, output\_dim=10):

产生FLC

1. generate\_fuzzy\_logic\_valuer(self):

产生FLV

* + - 1. 运行测试入口main：

使用如下方式测试，参数epoch表示训练轮数，mini\_batch 表示mini\_batch大小

Usage:

ddpginitializationTest.py epoch <epoch> mini\_batch <mini\_batch>

Options:

-h --help

main函数结构如下：

1. 依次声明
   1. 初始化测试实例DDPGInitializaionTest
   2. 环境实例 Environment
   3. 两个模糊逻辑控制器实例fuzzy\_controller和fuzzy\_valuer
   4. DDPG初始化器的实例ddpgInitializer
2. ddpgInitializer.train\_DDPG(epoch=epoch) 对模型进行初始化训练
3. ddpgInitializer.save\_all\_model(epoch=epoch) 对模型进行保存