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 0.5

目录

1.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：

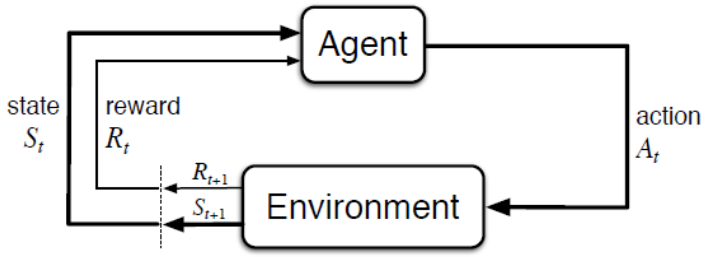

图 1

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：

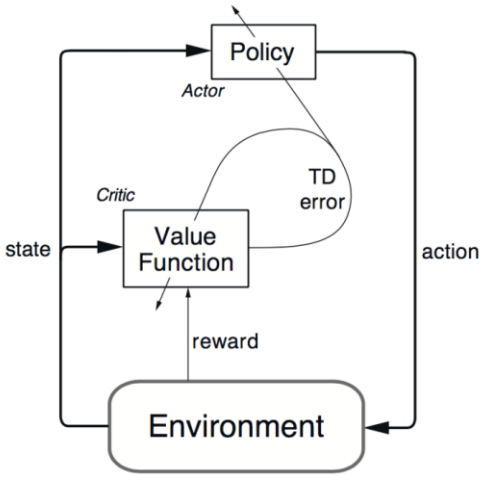

图 2

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

$$If\ \mathrm{f}(x_1\ \mathrm{is}\ A_1 \ldots x_k\ \mathrm{is}\ A_k)\ \ then\ \ y\ is\ B$$

y：称为后件，表示被推断出的值，即需要被决策的变量。

$x_1..x_1$：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。

$A_1..A_k$ ：表示每个变量的模糊类别，或者称为 Linguistic label。每个变量x可包含多个类别，如：x 表示温度，则x对应的模糊类别的集合可以定义为 $A=\{高，中，低\}$，在任意一个x的值下，通过 Membership function 计算，可以得到变量x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。

f ：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y为类别B的真值。

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

2.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。

1.1 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

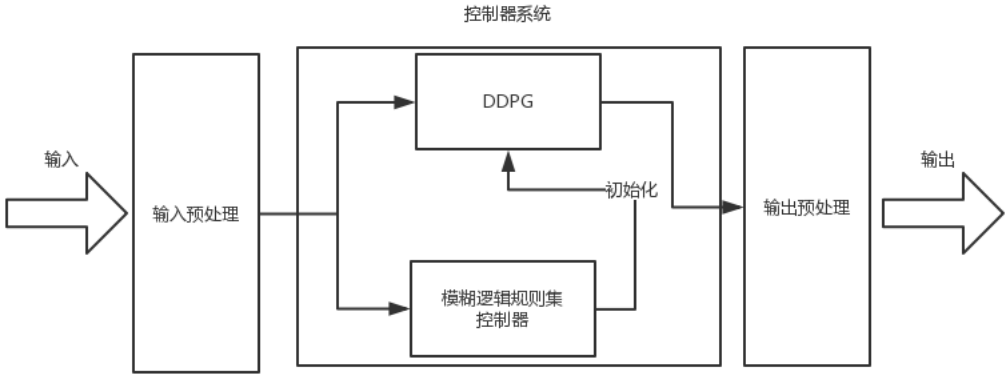

图 3

1.2 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 S_t

Action Variable at time step is A_t

Fuzzy logic controller is $FLC(S_t)$ and its output action is A_t^*

Fuzzy logic valuer is $FLV(S_t)$ andi its output value is V_t^*

Critic is $Q(S_t, A_t \mid \theta^Q)$, and its output is Q_t

Actor is $\mu(S_t \mid \theta^\mu)$, and its output is A_t

According to the DDPG algorithm, also create target network for critic and actor: $Q(S_t, A_t \mid \theta^{Q'})$ and $\mu(S_t \mid \theta^{\mu'})$

for epoch =1, M do:

Randomly generate N state S_t

Compute A_t^* and V_t^* according to $FLC(S_t)$ and $FLV(S_t)$

Compute A_t and according to $\mu(S_t)$

Compute Q_t in either $Q(S_t, A_t)$ or $Q(S_t, A_t^*)$

Update critic by minimizing the loss : $L = \frac{1}{N}\sum(V_t^* - Q_t)^2$

Update actor by minimizing the loss: $L = \frac{1}{N}\sum(A_t^* - A_t)^2$

Update target networks according the DDPG algorithm

$$\theta^{Q'} = \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} = \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

1.3 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：

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

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。

1.3 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。

3 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

Properties:

sess：一个 TensorFlow 的 session

Methods:

__init__(self, sess)：使用一个 tensorflow session 初始化一个 Network，

variable(self, shape, f):

batch_norm_layer(self, x, training_phase, scope_bn, activation=None):

3.2.1.2 Class CriticNetwork(Network)

Properties:

state_input:

action_input:

q_value_output:

target_state_input:

target_action_input:

target_q_value_output:

net:

is_training:

y_input:

cost:

optimizer:

action_gradients:

time_step:

Methods

__init__

create_training_method(self):

create_q_network(self, state_dim, action_dim):

create_target_q_network(self, state_dim, action_dim, net):

update_target(self):

train(self, y_batch, state_batch, action_batch):

gradients(self, state_batch, action_batch):

target_q(self, state_batch, action_batch):

q_value(self, state_batch, action_batch):

3.2.1.3 Class ActorNetwork(Network)

Properties:

state_dim:
action_dim
state_input:
action_input:
net:
is_training:
target_state_input:
target_action_output:
target_update:
target_is_training:

q_gradient_input:
parameters_gradients:
optimizer:

y_input:
cost:
initial_optimizer

Methods:

__init__(self, sess, state_dim, action_dim):
create_training_method(self):
create_network(self, state_dim, action_dim):
create_target_network(self, state_dim, action_dim, net):
update_target(self):
train(self, q_gradient_batch, state_batch):
initial_train(self, action_label_batch, state_batch):
actions(self, state_batch):
action(self, state):
target_actions(self, state_batch):

3.2.1.4 Class NetworkCommon(object):

Properties:

REPLAY_BUFFER_SIZE
REPLAY_START_SIZE
BATCH_SIZE
GAMMA

CRITIC_LAYER1_SIZE

CRITIC_LAYER2_SIZE
CRITIC_LEARNING_RATE
CRITIC_TAU
CRITIC_L2

ACTOR_LAYER1_SIZE
ACTOR_LAYER2_SIZE
ACTOR_LEARNING_RATE
ACTOR_TAU
ACTOR_BATCH_SIZE
ACTOR_L2

Methods:
create_variable_summary(var, name_scope):
return_file_writer(sess, log_file_dir):

3.2.1.5 Class DDPGController(object):
Properties:
action_dim:
actor_network:
critic_network:
environment:
exploration_noise:
model_saver:
name:
replay_buffer:
sess
state_dim

Methods:
__init__(self, env):
train(self):
noise_action(self, state):
action(self, state):
perceive(self, state, action, reward, next_state, done):
initial_train(self, mini_batch):
save_model(self, path, check_point):
load_model(self, path):

3.2.1.6 Class DDPGInitializer(object):
Properties:
ddpgController:

fuzzylogicController:
fuzzyLogicValuer:
log_file_dir:
model_file_dir:
mini_batch_size:

Methods:
__init__(self, ddpg_controller,
fuzzy_logic_controller,
fuzzy_logic_valuer,
mini_batch_size):
generate_state_done_mini_batch(self, mini_batch_size):
enerate_training_sample_mini_batch(self, mini_batch_size):
save_all_model(self, epoch):
load_model(self, path):
train_DDPG(self, epoch):

3.2.1.7 Class OUNoise (object):
Properties:
action_dimension:
mu:
sigma:
state:
theta:

Methods:
__init__(self, action_dimension, mu=0, theta=0.15, sigma=0.2):
reset(self):
noise(self):

3.2.1.8 Class ReplayBuffer (object):
Properties:
buffer:
buffer_size:
num_experiences:

Methods:
__init__(self, buffer_size):
get_batch(self, batch_size):
size(self):
add(self, state, action, reward, new_state, done):
count(self):

erase(self):

3.2.2 Fuzzy Logic Rule

3.2.2.1 Class Controller(object)

Properties:

input_value_dict:

output_value_dict:

rule_section_set:

name:

Methods:

__init__(self, name):

add_rule_section(self, rule_set):

_assign_input_to_section(self):

_calc_output(self):

3.2.2.2 Class Defuzzifier (object)

Properties:

name

Methods:

__init__(self, name):

defuzzify(self, degree, mf):

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

LMOMDefuzzifier 方法：

TSKDefuzzifier 方法：

3.2.2.3 Class MembershipFunction (object)

Properties:

name

Methods:

__init__(self, name):

calc(self, input_value):

inverse_calc(self, input_degree):

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

3.2.2.4 Class Variable (object)

Properties:

name

mf

value

degree

linguistic_label

upper_range

lower_range

Methods:

__init__(self, name, mf, range):

3.2.2.5 Class InputVariable (Variable)

Properties:

Methods:

__init__(self, name, range, mf, value=0.0):

calc_degree(self):

3.2.2.6 Class OutputVariable (Variable)

Properties:

defuzzifier

Methods:

_init__(self, name, mf, range, defuzzifier):

calc_degree(self):

3.2.2.7 Class FuzzyRule(object)

Properties:

rule_str:

true_value:

input_var_list:

input_dict:

output_var_list:

output_var_value:

_min_operation:

_section:

Methods:

__init__(self,
input_var_list,
output_var_list,
rule_str,
section,
min_operation="softmin"
):

set_input_var_value(self, new_input_var_value_dict):

set_input_output_dict(self, rule_str=None):

_get_true_value(self):

set_output_var_degree(self):

_min_op(self):

_softmin_op(self):

3.2.2.8 Class FuzzyRuleSet (object)

Properties:

input_var_value_dict:

name:

output_var_value_dict:

rule_list:

rule_set_output_value:

section:

Methods:

__init__(self, name, section, rule_list):

add_fuzzy_rule(self, fuzzy_rule):

get_input_var_name(self):

get_output_var_name(self):

assign_input_value(self):

_calc_value(self):

3.2.2.9 Class RuleParser (object)

Methods:

parse_if_then_rule(self, rule_str):

3.2.3 Environment

3.2.3.1 Class Environment(object)

Properties:

action_dim:

action_set:

name:

reward:

state_dim:

state_set:

time_step_limit:

Methods:

__init__(self, name, state_set, action_set):

set_action_set(self, action_set):

set_reward(self, reward):

set_state_set(self, state_set):

reset(self):

step(self, action):

is_done(self, state):

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。

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。

3.3.1.1 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:

controller:

input_var_list:

output_var_list:

section_list:

section_num:

value:

value_rule:

valuer:

Methods:

__init__(self, state_dim, action_dim):

generate_input_var(self, input_dim=100):

generate_output_var(self, output_dim=10):

generate_value(self):

generate_section_list(self, rule_per_seciont_num=10):

generate_fuzzy_logic_controller(self, output_dim=10):

generate_fuzzy_logic_valuer(self):

3.3.1.2 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
   a) 初始化测试实例 DDPGInitializaionTest
   b) 环境实例 Environment
   c) 两个模糊逻辑控制器实例 fuzzy_controller 和 fuzzy_valuer
   d) 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