
Cosmos Documentation

Release 1.0

Marcel van Gerven

May 13, 2017

CONTENTS

| | | |
|----------|---|-----------|
| 1 | reinforcement_learning package | 3 |
| 1.1 | Submodules | 3 |
| 1.2 | reinforcement_learning.agents module | 3 |
| 1.3 | reinforcement_learning.models module | 4 |
| 1.4 | reinforcement_learning.networks module | 4 |
| 1.5 | reinforcement_learning.tasks module | 5 |
| 1.6 | reinforcement_learning.unit_test module | 5 |
| 1.7 | reinforcement_learning.world module | 5 |
| 1.8 | Module contents | 6 |
| 2 | Indices and tables | 7 |
| | Python Module Index | 9 |
| | Index | 11 |

Contents:

REINFORCEMENT_LEARNING PACKAGE

1.1 Submodules

1.2 reinforcement_learning.agents module

class reinforcement_learning.agents.**AACAgent** (*model*, *optimizer=None*, *gamma=0.99*,
beta=0.01, *cutoff=None*)

Bases: *reinforcement_learning.agents.REINFORCEAgent*

Implements Advantage Actor-Critic algorithm

train (*observation*, *reward*, *done*)

Trains agent on cumulated reward (return)

Returns action (Variable)

class reinforcement_learning.agents.**NESAgent** (*model*, *nsteps=100*, *npop=50*, *sigma=0.1*, *al-*
pha=0.001)

Bases: object

Implements Natural Evolution Strategies algorithm

<https://blog.openai.com/evolution-strategies/> <https://gist.github.com/karpathy/77fbb6a8dac5395f1b73e7a89300318d>

hard to implement in same framework since the agent needs to run multiple versions of the task

train (*observation*, *reward*, *done*)

Trains agent on cumulated reward (return)

Returns action (Variable)

class reinforcement_learning.agents.**REINFORCEAgent** (*model*, *optimizer=None*,
gamma=0.99, *beta=0.01*, *cut-*
off=None)

Bases: object

Implements REINFORCE algorithm

entropy (*pi*)

Computes entropy of policy

Parameters policy –

reset_state ()

Resets persistent states

score_function (*action*, *policy*)

Computes score

Parameters

- **action** (*int*) –
- **policy** –

Returns score**test** (*observation, reward, done*)

Tests agent

Returns action (Variable)**train** (*observation, reward, done*)

Trains agent on cumulated reward (return)

Returns action (Variable)

1.3 reinforcement_learning.models module

class reinforcement_learning.models.**ActorCriticModel** (*net, gpu=-1*)Bases: *reinforcement_learning.models.Model*

An actor model computes the action and policy from a predictor

__call__ (*data*)**Parameters** **data** – observation**Returns** action and policy**class** reinforcement_learning.models.**ActorModel** (*net, gpu=-1*)Bases: *reinforcement_learning.models.Model*

An actor model computes the action and policy from a predictor

__call__ (*data*)**Parameters** **data** – observation**Returns** action and policy**class** reinforcement_learning.models.**Model** (*net, gpu=-1*)Bases: *chainer.link.Chain*

Model which wraps a network to generate predictions and compute policies

has_state

Checks if a network has persistent states

Returns bool**predict** (*data*)

Returns an action

reset_state ()

1.4 reinforcement_learning.networks module

class reinforcement_learning.networks.**MLP** (*n_input=None, n_output=1, n_hidden=10*)Bases: *chainer.link.Chain*

Multilayer perceptron

has_state

Checks if a network has persistent states

Returns bool

class `reinforcement_learning.networks.RNN` (*n_input=None, n_output=1, n_hidden=10*)

Bases: `chainer.link.Chain`

has_state

Checks if a network has persistent states

Returns bool

reset_state()

Resets persistent states

1.5 reinforcement_learning.tasks module

class `reinforcement_learning.tasks.EvidenceTask` (*n=2, p=0.8*)

Bases: `object`

Very simple task which only requires evaluating present evidence and does not require evidence integration. The actor gets a reward when it correctly decides on the ground truth. Ground truth 0/1 determines probabilistically the number of 0s or 1s as observations

reset()

Resets state and generates new observations

Returns observations, reward, done

step(action)

This task always produces a new state and observation after each decision

Parameters *action* – agent(s) action

Returns

1.6 reinforcement_learning.unit_test module

class `reinforcement_learning.unit_test.UnitTest` (*methodName='runTest'*)

Bases: `unittest.case.TestCase`

test_aac_stateful()

Test Advantage Actor-Critic on stateful network

test_reinforce_stateful()

Test REINFORCE on stateful network

test_reinforce_stateless()

Test REINFORCE on stateless network

1.7 reinforcement_learning.world module

class `reinforcement_learning.world.World` (*agents, out='result'*)

Bases: `object`

Wrapper object which takes care of training and testing on some data iterator for one or more agents

test (*task*, *n_steps*)

Parameters

- **task** – task to run agent(s) on
- **n_steps** (*int*) – number of steps to train on

Returns test loss and reward

train (*task*, *n_steps*, *snapshot=0*)

Parameters

- **task** – task to run agent(s) on
- **n_steps** (*int*) – number of steps to train on
- **snapshot** (*int*) – whether or not to save model after each epochs modulo snapshot

Returns rewards

1.8 Module contents

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

r

reinforcement_learning, 6
reinforcement_learning.agents, 3
reinforcement_learning.models, 4
reinforcement_learning.networks, 4
reinforcement_learning.tasks, 5
reinforcement_learning.unit_test, 5
reinforcement_learning.world, 5

Symbols

`__call__()` (reinforcement_learning.models.ActorCriticModel method), 4
`__call__()` (reinforcement_learning.models.ActorModel method), 4

A

AACAgent (class in reinforcement_learning.agents), 3
 ActorCriticModel (class in reinforcement_learning.models), 4
 ActorModel (class in reinforcement_learning.models), 4

E

entropy() (reinforcement_learning.agents.REINFORCEAgent method), 3
 EvidenceTask (class in reinforcement_learning.tasks), 5

H

has_state (reinforcement_learning.models.Model attribute), 4
 has_state (reinforcement_learning.networks.MLP attribute), 5
 has_state (reinforcement_learning.networks.RNN attribute), 5

M

MLP (class in reinforcement_learning.networks), 4
 Model (class in reinforcement_learning.models), 4

N

NESAgent (class in reinforcement_learning.agents), 3

P

predict() (reinforcement_learning.models.Model method), 4

R

REINFORCEAgent (class in reinforcement_learning.agents), 3
 reinforcement_learning (module), 6
 reinforcement_learning.agents (module), 3

reinforcement_learning.models (module), 4
 reinforcement_learning.networks (module), 4
 reinforcement_learning.tasks (module), 5
 reinforcement_learning.unit_test (module), 5
 reinforcement_learning.world (module), 5
 reset() (reinforcement_learning.tasks.EvidenceTask method), 5
 reset_state() (reinforcement_learning.agents.REINFORCEAgent method), 3
 reset_state() (reinforcement_learning.models.Model method), 4
 reset_state() (reinforcement_learning.networks.RNN method), 5
 RNN (class in reinforcement_learning.networks), 5

S

score_function() (reinforcement_learning.agents.REINFORCEAgent method), 3
 step() (reinforcement_learning.tasks.EvidenceTask method), 5

T

test() (reinforcement_learning.agents.REINFORCEAgent method), 4
 test() (reinforcement_learning.world.World method), 6
 test_aac_stateful() (reinforcement_learning.unit_test.UnitTest method), 5
 test_reinforce_stateful() (reinforcement_learning.unit_test.UnitTest method), 5
 test_reinforce_stateless() (reinforcement_learning.unit_test.UnitTest method), 5
 train() (reinforcement_learning.agents.AACAgent method), 3
 train() (reinforcement_learning.agents.NESAgent method), 3
 train() (reinforcement_learning.agents.REINFORCEAgent method), 4

`train()` (`reinforcement_learning.world.World` method), [6](#)

U

`UnitTest` (class in `reinforcement_learning.unit_test`), [5](#)

W

`World` (class in `reinforcement_learning.world`), [5](#)