
Cosmos Documentation

Release 0.1

Marcel van Gerven

April 24, 2017

CONTENTS

1	cosmos package	3
1.1	Subpackages	3
1.2	Module contents	8
2	cosmos.analysis package	9
2.1	Module contents	9
3	cosmos.reinforcement_learning package	11
3.1	Submodules	11
3.2	cosmos.reinforcement_learning.agents module	11
3.3	cosmos.reinforcement_learning.models module	11
3.4	cosmos.reinforcement_learning.networks module	12
3.5	cosmos.reinforcement_learning.tasks module	12
3.6	cosmos.reinforcement_learning.unit_test module	13
3.7	cosmos.reinforcement_learning.world module	13
3.8	Module contents	13
4	cosmos.supervised_learning package	15
4.1	Submodules	15
4.2	cosmos.supervised_learning.agents module	15
4.3	cosmos.supervised_learning.iterators module	15
4.4	cosmos.supervised_learning.models module	16
4.5	cosmos.supervised_learning.networks module	16
4.6	cosmos.supervised_learning.unit_test module	17
4.7	cosmos.supervised_learning.world module	17
4.8	Module contents	17
5	Welcome to cosmos's documentation!	19
5.1	cosmos package	19
6	Indices and tables	25
7	Indices and tables	27
Python Module Index		29
Index		31

Contents:

COSMOS PACKAGE

1.1 Subpackages

1.1.1 cosmos.analysis package

Module contents

1.1.2 cosmos.reinforcement_learning package

Submodules

cosmos.reinforcement_learning.agents module

```
class cosmos.reinforcement_learning.agents.REINFORCEAgent(model, optimizer=None,  
                                         gamma=0.99, cut-  
                                         off=None)
```

Bases: object

Implements REINFORCE algorithm

<https://webdocs.cs.ualberta.ca/%7Esutton/book/bookdraft2016sep.pdf> <https://github.com/dennybritz/reinforcement-learning/tree/master/PolicyGradient> <http://blog.shakirm.com/2015/11/machine-learning-trick-of-the-day-5-log-derivative-trick/> http://www.1-4-5.net/~dmm/ml/log_derivative_trick.pdf

reset_state()

Resets persistent states

score_function(action, policy)

Computes score

Parameters

- **action** (*int*) –
- **policy** –

Returns score

train(observation, reward, done)

Trains agent on cumulate reward (return)

Returns action (Variable)

cosmos.reinforcement_learning.models module

```
class cosmos.reinforcement_learning.models.ActorModel (net, gpu=-1)
    Bases: cosmos.reinforcement_learning.models.Model

    An actor model computes the action and policy from a predictor

    __call__ (data)

        Parameters data – observation

        Returns action and policy

    predict (data)

class cosmos.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)

    reset_state()
```

cosmos.reinforcement_learning.networks module

```
class cosmos.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 cosmos.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
```

cosmos.reinforcement_learning.tasks module

```
class cosmos.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

cosmos.reinforcement_learning.unit_test module

class `cosmos.reinforcement_learning.unit_test.UnitTest(methodName='runTest')`
Bases: `unittest.case.TestCase`

test_stateful_network()
Test training procedure for stateful network

test_stateless_network()
Test training procedure for stateless network

cosmos.reinforcement_learning.world module

class `cosmos.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

Module contents

1.1.3 `cosmos.supervised_learning` package

Submodules

`cosmos.supervised_learning.agents` module

```
class cosmos.supervised_learning.agents.SupervisedAgent (model, optimizer, cut-off=None)
```

Bases: object

Agent which trains on labelled data

```
__call__(data)
```

Runs networks in forward mode and applies optional output function

Parameters `data` –

Returns post-processed output

```
reset_state()
```

Resets persistent states

```
test(data)
```

Returns the loss for one batch

Parameters `data` –

Returns loss

```
train(data)
```

Train agent on one batch :param data: :return: loss

`cosmos.supervised_learning.iterators` module

```
class cosmos.supervised_learning.iterators.RandomIterator (data, batch_size=None)
```

Bases: object

Generates random subsets of data

```
next()
```

```
class cosmos.supervised_learning.iterators.SequentialIterator (data,
```

`batch_size=None`)

Bases: object

Generates subsets of data such that each batch contains data for the next time point

```
next()
```

`cosmos.supervised_learning.models` module

```
class cosmos.supervised_learning.models.Classifier (net, gpu=-1)
```

Bases: `cosmos.supervised_learning.models.Model`

Wrapper for classification problems

```
class cosmos.supervised_learning.models.Model (net,           loss_function,
                                              put_function=<function <lambda>>,
                                              gpu=-1)
Bases: chainer.link.Chain

Model which wraps a network to compute loss and generate actual predictions

__call__(data)
Compute loss for minibatch of data

    Parameters data – list of minibatches (e.g. inputs and targets for supervised learning)

    Returns loss

has_state
Checks if a network has persistent states

    Returns bool

predict(data)
Returns prediction, which can be different than raw output (e.g. for softmax function)

    Parameters data – minibatch or list of minibatches representing input to the model

    Returns prediction

reset_state()

class cosmos.supervised_learning.models.Regressor (net, gpu=-1)
Bases: cosmos.supervised_learning.models.Model

Wrapper for regression problems
```

cosmos.supervised_learning.networks module

```
class cosmos.supervised_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 cosmos.supervised_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
```

cosmos.supervised_learning.unit_test module

```
class cosmos.supervised_learning.unit_test.UnitTest (methodName='runTest')
Bases: unittest.case.TestCase

test_gpu
```

```
test_stateful_network()
    Test training procedure for stateful network

test_stateless_network()
    Test training procedure for stateless network
```

cosmos.supervised_learning.world module

```
class cosmos.supervised_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(test_iter)
```

Parameters `test_iter` – iterator over the test data

Returns test loss

```
train(train_iter, n_epochs, test_iter=None, snapshot=0)
```

Parameters

- `train_iter` – iterator over the training data
- `n_epochs (int)` – number of epochs to train on
- `test_iter` – optional iterator over the test data (returns optimal model)
- `snapshot (int)` – whether or not to save model after each epochs modulo snapshot

Returns train loss and optional test loss

Module contents

1.2 Module contents

CHAPTER
TWO

COSMOS.ANALYSIS PACKAGE

2.1 Module contents

COSMOS.REINFORCEMENT_LEARNING PACKAGE

3.1 Submodules

3.2 cosmos.reinforcement_learning.agents module

```
class cosmos.reinforcement_learning.agents.REINFORCEAgent (model, optimizer=None,  
                                              gamma=0.99, cut-  
                                              off=None)
```

Bases: object

Implements REINFORCE algorithm

<https://webdocs.cs.ualberta.ca/%7Esutton/book/bookdraft2016sep.pdf> <https://github.com/dennybritz/reinforcement-learning/tree/master/PolicyGradient> <http://blog.shakirm.com/2015/11/machine-learning-trick-of-the-day-5-log-derivative-trick/> http://www.1-4-5.net/~dmm/ml/log_derivative_trick.pdf

reset_state()

Resets persistent states

score_function(action, policy)

Computes score

Parameters

- **action** (*int*) –
- **policy** –

Returns score

train(observation, reward, done)

Trains agent on cumulate reward (return)

Returns action (Variable)

3.3 cosmos.reinforcement_learning.models module

```
class cosmos.reinforcement_learning.models.ActorModel (net, gpu=-1)
```

Bases: *cosmos.reinforcement_learning.models.Model*

An actor model computes the action and policy from a predictor

__call__(data)

Parameters **data** – observation

Returns action and policy

predict (*data*)

class `cosmos.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*)

reset_state ()

3.4 cosmos.reinforcement_learning.networks module

class `cosmos.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 `cosmos.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

3.5 cosmos.reinforcement_learning.tasks module

class `cosmos.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

3.6 cosmos.reinforcement_learning.unit_test module

```
class cosmos.reinforcement_learning.unit_test.UnitTest(methodName='runTest')
Bases: unittest.case.TestCase

test_stateful_network()
    Test training procedure for stateful network

test_stateless_network()
    Test training procedure for stateless network
```

3.7 cosmos.reinforcement_learning.world module

```
class cosmos.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
```

3.8 Module contents

COSMOS.SUPERVISED_LEARNING PACKAGE

4.1 Submodules

4.2 cosmos.supervised_learning.agents module

```
class cosmos.supervised_learning.agents.SupervisedAgent (model, optimizer, cut-off=None)
Bases: object
Agent which trains on labelled data

__call__ (data)
    Runs networks in forward mode and applies optional output function

    Parameters data -
    Returns post-processed output

reset_state ()
    Resets persistent states

test (data)
    Returns the loss for one batch

    Parameters data -
    Returns loss

train (data)
    Train agent on one batch :param data: :return: loss
```

4.3 cosmos.supervised_learning.iterators module

```
class cosmos.supervised_learning.iterators.RandomIterator (data, batch_size=None)
Bases: object
Generates random subsets of data

next ()

class cosmos.supervised_learning.iterators.SequentialIterator (data,
                                                               batch_size=None)
Bases: object
Generates subsets of data such that each batch contains data for the next time point
```

`next()`

4.4 cosmos.supervised_learning.models module

```
class cosmos.supervised_learning.models.Classifier(net, gpu=-1)
    Bases: cosmos.supervised_learning.models.Model
        Wrapper for classification problems

class cosmos.supervised_learning.models.Model(net, loss_function, out-
                                              put_function=<function <lambda>>, gpu=-1)
    Bases: chainer.link.Chain
        Model which wraps a network to compute loss and generate actual predictions

    __call__(data)
        Compute loss for minibatch of data
            Parameters data – list of minibatches (e.g. inputs and targets for supervised learning)
            Returns loss

    has_state
        Checks if a network has persistent states
            Returns bool

    predict(data)
        Returns prediction, which can be different than raw output (e.g. for softmax function)
            Parameters data – minibatch or list of minibatches representing input to the model
            Returns prediction

    reset_state()

```

class cosmos.supervised_learning.models.Regressor(net, gpu=-1)
Bases: *cosmos.supervised_learning.models.Model*

Wrapper for regression problems

4.5 cosmos.supervised_learning.networks module

```
class cosmos.supervised_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 cosmos.supervised_learning.networks.RNN(n_input=None, n_output=1, n_hidden=10)
    Bases: chainer.link.Chain
        RNN

    has_state
        Checks if a network has persistent states
            Returns bool
```

```
reset_state()
    Resets persistent states
```

4.6 cosmos.supervised_learning.unit_test module

```
class cosmos.supervised_learning.unit_test.UnitTest (methodName='runTest')
Bases: unittest.case.TestCase

test_gpu()
    Test training procedure for stateless network on GPU

test_stateful_network()
    Test training procedure for stateful network

test_stateless_network()
    Test training procedure for stateless network
```

4.7 cosmos.supervised_learning.world module

```
class cosmos.supervised_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 (test_iter)
    Parameters test_iter – iterator over the test data
    Returns test loss

train (train_iter, n_epochs, test_iter=None, snapshot=0)
    Parameters
        • train_iter – iterator over the training data
        • n_epochs (int) – number of epochs to train on
        • test_iter – optional iterator over the test data (returns optimal model)
        • snapshot (int) – whether or not to save model after each epochs modulo snapshot
    Returns train loss and optional test loss
```

4.8 Module contents

WELCOME TO COSMOS'S DOCUMENTATION!

Contents:

5.1 cosmos package

5.1.1 Subpackages

`cosmos.analysis` package

Module contents

`cosmos.reinforcement_learning` package

Submodules

`cosmos.reinforcement_learning.agents` module

```
class cosmos.reinforcement_learning.agents.REINFORCEAgent(model, optimizer=None,  
                                         gamma=0.99, cut-off=None)
```

Bases: `object`

Implements REINFORCE algorithm

<https://webdocs.cs.ualberta.ca/%7Esutton/book/bookdraft2016sep.pdf> <https://github.com/dennybritz/reinforcement-learning/tree/master/PolicyGradient> <http://blog.shakirm.com/2015/11/machine-learning-trick-of-the-day-5-log-derivative-trick/> http://www.1-4-5.net/~dmm/ml/log_derivative_trick.pdf

`reset_state()`

Resets persistent states

`score_function(action, policy)`

Computes score

Parameters

- `action (int)` –
- `policy` –

Returns score

`train(observation, reward, done)`

Trains agent on cumulate reward (return)

Returns action (Variable)

`cosmos.reinforcement_learning.models module`

class `cosmos.reinforcement_learning.models.ActorModel (net, gpu=-1)`

Bases: `cosmos.reinforcement_learning.models.Model`

An actor model computes the action and policy from a predictor

__call__ (data)

Parameters `data` – observation

Returns action and policy

predict (data)

class `cosmos.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)

reset_state ()

`cosmos.reinforcement_learning.networks module`

class `cosmos.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 `cosmos.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

`cosmos.reinforcement_learning.tasks module`

class `cosmos.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

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

Bases: unittest.case.TestCase

test_stateful_network()

Test training procedure for stateful network

test_stateless_network()

Test training procedure for stateless network

cosmos.reinforcement_learning.world module**class cosmos.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

Module contents

`cosmos.supervised_learning` package

Submodules

`cosmos.supervised_learning.agents` module

class `cosmos.supervised_learning.agents.SupervisedAgent` (*model*, *optimizer*, *cut-off=None*)
Bases: `object`

Agent which trains on labelled data

__call__ (*data*)

Runs networks in forward mode and applies optional output function

Parameters `data` –

Returns post-processed output

reset_state ()

Resets persistent states

test (*data*)

Returns the loss for one batch

Parameters `data` –

Returns loss

train (*data*)

Train agent on one batch :param data: :return: loss

`cosmos.supervised_learning.iterators` module

class `cosmos.supervised_learning.iterators.RandomIterator` (*data*, *batch_size=None*)
Bases: `object`

Generates random subsets of data

next ()

class `cosmos.supervised_learning.iterators.SequentialIterator` (*data*,
batch_size=None)
Bases: `object`

Generates subsets of data such that each batch contains data for the next time point

next ()

`cosmos.supervised_learning.models` module

class `cosmos.supervised_learning.models.Classifier` (*net*, *gpu=-1*)
Bases: `cosmos.supervised_learning.models.Model`

Wrapper for classification problems

```
class cosmos.supervised_learning.models.Model (net, loss_function, out-  

put_function=<function <lambda>>,  

gpu=-1)
```

Bases: chainer.link.Chain

Model which wraps a network to compute loss and generate actual predictions

```
__call__ (data)
```

Compute loss for minibatch of data

Parameters **data** – list of minibatches (e.g. inputs and targets for supervised learning)

Returns loss

```
has_state
```

Checks if a network has persistent states

Returns bool

```
predict (data)
```

Returns prediction, which can be different than raw output (e.g. for softmax function)

Parameters **data** – minibatch or list of minibatches representing input to the model

Returns prediction

```
reset_state ()
```

```
class cosmos.supervised_learning.models.Regressor (net, gpu=-1)
```

Bases: *cosmos.supervised_learning.models.Model*

Wrapper for regression problems

cosmos.supervised_learning.networks module

```
class cosmos.supervised_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 cosmos.supervised_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

cosmos.supervised_learning.unit_test module

```
class cosmos.supervised_learning.unit_test.UnitTest (methodName='runTest')
```

Bases: unittest.case.TestCase

```
test_gpu()
    Test training procedure for stateless network on GPU

test_stateful_network()
    Test training procedure for stateful network

test_stateless_network()
    Test training procedure for stateless network
```

cosmos.supervised_learning.world module

```
class cosmos.supervised_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(test_iter)
        Parameters test_iter – iterator over the test data
        Returns test loss

    train(train_iter, n_epochs, test_iter=None, snapshot=0)
        Parameters
            • train_iter – iterator over the training data
            • n_epochs (int) – number of epochs to train on
            • test_iter – optional iterator over the test data (returns optimal model)
            • snapshot (int) – whether or not to save model after each epochs modulo snapshot
        Returns train loss and optional test loss
```

Module contents

5.1.2 Module contents

**CHAPTER
SIX**

INDICES AND TABLES

- genindex
- modindex
- search

**CHAPTER
SEVEN**

INDICES AND TABLES

- genindex
- modindex
- search

C

cosmos, 24
cosmos.analysis, 19
cosmos.reinforcement_learning, 22
cosmos.reinforcement_learning.agents,
 19
cosmos.reinforcement_learning.models,
 20
cosmos.reinforcement_learning.networks,
 20
cosmos.reinforcement_learning.tasks, 20
cosmos.reinforcement_learning.unit_test,
 21
cosmos.reinforcement_learning.world, 21
cosmos.supervised_learning, 24
cosmos.supervised_learning.agents, 22
cosmos.supervised_learning.iterators,
 22
cosmos.supervised_learning.models, 22
cosmos.supervised_learning.networks, 23
cosmos.supervised_learning.unit_test,
 23
cosmos.supervised_learning.world, 24

Symbols

`__call__()` (`cosmos.reinforcement_learning.models.ActorModel` method), 4, 11, 20
`__call__()` (`cosmos.supervised_learning.agents.SupervisedAgent` method), 6, 15, 22
`__call__()` (`cosmos.supervised_learning.models.Model` method), 7, 16, 23

A

`ActorModel` (class in `cosmos.reinforcement_learning.models`), 4, 11, 20

C

`Classifier` (class in `cosmos.supervised_learning.models`), 6, 16, 22
`cosmos` (module), 8, 24
`cosmos.analysis` (module), 3, 9, 19
`cosmos.reinforcement_learning` (module), 6, 13, 22
`cosmos.reinforcement_learning.agents` (module), 3, 11, 19
`cosmos.reinforcement_learning.models` (module), 4, 11, 20
`cosmos.reinforcement_learning.networks` (module), 4, 12, 20
`cosmos.reinforcement_learning.tasks` (module), 4, 12, 20
`cosmos.reinforcement_learning.unit_test` (module), 5, 13, 21
`cosmos.reinforcement_learning.world` (module), 5, 13, 21
`cosmos.supervised_learning` (module), 8, 17, 24
`cosmos.supervised_learning.agents` (module), 6, 15, 22
`cosmos.supervised_learning.iterators` (module), 6, 15, 22
`cosmos.supervised_learning.models` (module), 6, 16, 22
`cosmos.supervised_learning.networks` (module), 7, 16, 23
`cosmos.supervised_learning.unit_test` (module), 7, 17, 23
`cosmos.supervised_learning.world` (module), 8, 17, 24

E

`EvidenceTask` (class in `cosmos.reinforcement_learning.tasks`), 4, 12, 20

H

`has_state` (`cosmos.reinforcement_learning.models.Model` attribute), 4, 12, 20
`has_state` (`cosmos.reinforcement_learning.networks.MLP` attribute), 4, 12, 20
`has_state` (`cosmos.supervised_learning.networks.RNN` attribute), 4, 12, 20
`has_state` (`cosmos.supervised_learning.models.Model` attribute), 7, 16, 23
`has_state` (`cosmos.supervised_learning.networks.MLP` attribute), 7, 16, 23
`has_state` (`cosmos.supervised_learning.networks.RNN` attribute), 7, 16, 23

M

`MLP` (class in `cosmos.reinforcement_learning.networks`), 4, 12, 20
`MLP` (class in `cosmos.supervised_learning.networks`), 7, 16, 23
`Model` (class in `cosmos.reinforcement_learning.models`), 4, 12, 20
`Model` (class in `cosmos.supervised_learning.models`), 6, 16, 22

N

`next()` (`cosmos.supervised_learning.iterators.RandomIterator` method), 6, 15, 22
`next()` (`cosmos.supervised_learning.iterators.SequentialIterator` method), 6, 15, 22

P

`predict()` (`cosmos.reinforcement_learning.models.ActorModel` method), 4, 12, 20
`predict()` (`cosmos.reinforcement_learning.models.Model` method), 4, 12, 20
`predict()` (`cosmos.supervised_learning.models.Model` method), 7, 16, 23

R

`RandomIterator` (class in `cosmos.supervised_learning.iterators`), 6, 15, 22

Regressor (class in `cosmos.supervised_learning.models`), `test_stateless_network()` (cosmos.`supervised_learning.unit_test`.`UnitTest` method), 8, 17, 24
REINFORCEAgent (class in `cosmos.reinforcement_learning.agents`), 3, 11, 19, `train()` (cosmos.`reinforcement_learning.agents`.`REINFORCEAgent` method), 3, 11, 19
`reset()` (cosmos.`reinforcement_learning.tasks`.`EvidenceTask` train() method), 4, 12, 21
`reset_state()` (cosmos.`reinforcement_learning.agents`.`REINFORCEAgent`) (cosmos.`supervised_learning.agents`.`SupervisedAgent` method), 6, 15, 22
`reset_state()` (cosmos.`reinforcement_learning.models`.`Model`) (cosmos.`supervised_learning.world`.`World` method), 8, 17, 24
`reset_state()` (cosmos.`reinforcement_learning.networks`.`RNN`) (cosmos.`supervised_learning.world`.`World` method), 8, 17, 24
`reset_state()` (cosmos.`supervised_learning.agents`.`SupervisedAgent`) (cosmos.`reinforcement_learning.unit_test`), 5, 13, 21
`reset_state()` (cosmos.`supervised_learning.models`.`Model`) (cosmos.`supervised_learning.world`.`World` method), 8, 17, 24
`reset_state()` (cosmos.`supervised_learning.networks`.`RNN`) (cosmos.`supervised_learning.world`.`World` method), 8, 17, 24
RNN (class in `cosmos.reinforcement_learning.networks`), 4, 12, 20
RNN (class in `cosmos.supervised_learning.networks`), 7, 16, 23

U

`UnitTest` (class in `cosmos.supervised_learning.unit_test`), 7, 17, 23

W

World (class in `cosmos.reinforcement_learning.world`), 5, 13, 21
World (class in `cosmos.supervised_learning.world`), 8, 17, 24

S

`score_function()` (cosmos.`reinforcement_learning.agents`.`REINFORCEAgent` method), 3, 11, 19
SequentialIterator (class in `cosmos.supervised_learning.iterators`), 6, 15, 22
`step()` (cosmos.`reinforcement_learning.tasks`.`EvidenceTask` method), 5, 12, 21
SupervisedAgent (class in `cosmos.supervised_learning.agents`), 6, 15, 22

T

`test()` (cosmos.`reinforcement_learning.world`.`World` method), 5, 13, 21
`test()` (cosmos.`supervised_learning.agents`.`SupervisedAgent` method), 6, 15, 22
`test()` (cosmos.`supervised_learning.world`.`World` method), 8, 17, 24
`test_gpu()` (cosmos.`supervised_learning.unit_test`.`UnitTest` method), 7, 17, 23
`test_stateful_network()` (cosmos.`reinforcement_learning.unit_test`.`UnitTest` method), 5, 13, 21
`test_stateful_network()` (cosmos.`supervised_learning.unit_test`.`UnitTest` method), 7, 17, 24
`test_stateless_network()` (cosmos.`reinforcement_learning.unit_test`.`UnitTest` method), 5, 13, 21