
Cosmos Documentation

Release 0.1

Marcel van Gerven

April 22, 2017

CONTENTS

1	cosmos package	3
1.1	Subpackages	3
1.2	Module contents	7
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.iterators module	11
3.4	cosmos.reinforcement_learning.networks module	12
3.5	cosmos.reinforcement_learning.unit_test module	12
3.6	cosmos.reinforcement_learning.world module	12
3.7	Module contents	12
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!	21
5.1	cosmos package	21
6	Indices and tables	27
7	cosmos	29
7.1	cosmos package	29
8	Indices and tables	35
	Python Module Index	37
	Index	39

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.ActorCriticAgent (net, optimizer, gpu=-1, cutoff=None, gamma=0.99, beta=0.01, aac=True)
```

Bases: object

Implements advantage actor critic and REINFORCE (which does not use a value baseline)

Note that REINFORCE is a policy gradient method which does not use a critic. Instead the return is computed as a running estimate

<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

entropy (*pi*)

run (*data*, *train=True*, *idx=None*, *final=False*)

Parameters

- **data** – a new observation and the reward associated with the previous observation and action
- **train** –
- **idx** –
- **final** –

Returns

score_function (*action*, *pi*)

cosmos.reinforcement_learning.iterators module

```
class cosmos.reinforcement_learning.iterators.FooTask (n=2, p=0.8, batch_size=1,  
                                                    n_batches=inf)
```

Bases: object

Very simple environment for testing fully observed models. 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

```
get_observation ()
```

Returns observation given the state

```
get_state ()
```

Returns new state

```
next ()
```

```
process (agent)
```

Process agent action, compute reward and generate new state and observation

Parameters *agent* –

Returns

cosmos.reinforcement_learning.networks module

```
class cosmos.reinforcement_learning.networks.MLP (n_input=None, n_output=1,  
                                                    n_hidden=10)
```

Bases: chainer.link.Chain

```
reset ()
```

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

Bases: chainer.link.Chain

```
reset ()
```

cosmos.reinforcement_learning.unit_test module**cosmos.reinforcement_learning.world module**

```
class cosmos.reinforcement_learning.world.World (agents)
```

Bases: object

```
test (test_iter)
```

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


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,          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)
- **train** – call predictor in train or test mode

Returns loss

`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

`reset_state`()

Resets persistent states

```
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

1.2 Module contents

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.ActorCriticAgent (net, optimizer, gpu=-1, cutoff=None, gamma=0.99, beta=0.01, aac=True)
```

Bases: object

Implements advantage actor critic and REINFORCE (which does not use a value baseline)

Note that REINFORCE is a policy gradient method which does not use a critic. Instead the return is computed as a running estimate

<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

entropy (*pi*)

run (*data*, *train=True*, *idx=None*, *final=False*)

Parameters

- **data** – a new observation and the reward associated with the previous observation and action
- **train** –
- **idx** –
- **final** –

Returns

score_function (*action*, *pi*)

3.3 cosmos.reinforcement_learning.iterators module

```
class cosmos.reinforcement_learning.iterators.FooTask (n=2, p=0.8, batch_size=1, n_batches=inf)
```

Bases: object

Very simple environment for testing fully observed models. 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

get_observation()

Returns observation given the state

get_state()

Returns new state

next()

process(agent)

Process agent action, compute reward and generate new state and observation

Parameters agent –

Returns

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
    reset()
```

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

3.5 cosmos.reinforcement_learning.unit_test module

3.6 cosmos.reinforcement_learning.world module

```
class cosmos.reinforcement_learning.world.World(agents)
    Bases: object
    test(test_iter)
    train(train_iter, n_epochs, test_iter=None)
```

3.7 Module contents

```
class cosmos.reinforcement_learning.agents.ActorCriticAgent(net, optimizer, gpu=-
                                                            1, cutoff=None,
                                                            gamma=0.99,
                                                            beta=0.01, aac=True)
    Bases: object
    Implements advantage actor critic and REINFORCE (which does not use a value baseline)
```


Note that REINFORCE is a policy gradient method which does not use a critic. Instead the return is computed as a running estimate

<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

entropy (*pi*)

run (*data*, *train=True*, *idx=None*, *final=False*)

Parameters

- **data** – a new observation and the reward associated with the previous observation and action
- **train** –
- **idx** –
- **final** –

Returns

score_function (*action*, *pi*)

class `cosmos.reinforcement_learning.iterators.FooTask` (*n=2*, *p=0.8*, *batch_size=1*,
n_batches=inf)

Bases: `object`

Very simple environment for testing fully observed models. 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

get_observation ()

Returns observation given the state

get_state ()

Returns new state

next ()

process (*agent*)

Process agent action, compute reward and generate new state and observation

Parameters *agent* –

Returns

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

Bases: `chainer.link.Chain`

reset ()

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

Bases: `chainer.link.Chain`

reset ()

class `cosmos.reinforcement_learning.world.World` (*agents*)

Bases: `object`

test (*test_iter*)

train (*train_iter*, *n_epochs*, *test_iter=None*)

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)
- **train** – call predictor in train or test mode

Returns loss

```
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

```
reset_state()
```

Resets persistent states

```
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

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

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

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 ()

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*, *output_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)
- **train** – call predictor in train or test mode

Returns loss

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

```
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

    reset_state ()
        Resets persistent states

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

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

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
```


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.ActorCriticAgent (net, optimizer, gpu=-1, cutoff=None, gamma=0.99, beta=0.01, aac=True)
```

Bases: object

Implements advantage actor critic and REINFORCE (which does not use a value baseline)

Note that REINFORCE is a policy gradient method which does not use a critic. Instead the return is computed as a running estimate

<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

entropy (*pi*)

run (*data, train=True, idx=None, final=False*)

Parameters

- **data** – a new observation and the reward associated with the previous observation and action
- **train** –
- **idx** –

- **final** –

Returns

score_function (*action, pi*)

cosmos.reinforcement_learning.iterators module

class cosmos.reinforcement_learning.iterators.**FooTask** (*n=2, p=0.8, batch_size=1, n_batches=inf*)

Bases: object

Very simple environment for testing fully observed models. 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

get_observation ()

Returns observation given the state

get_state ()

Returns new state

next ()

process (*agent*)

Process agent action, compute reward and generate new state and observation

Parameters agent –

Returns

cosmos.reinforcement_learning.networks module

class cosmos.reinforcement_learning.networks.**MLP** (*n_input=None, n_output=1, n_hidden=10*)

Bases: chainer.link.Chain

reset ()

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

Bases: chainer.link.Chain

reset ()

cosmos.reinforcement_learning.unit_test module

cosmos.reinforcement_learning.world module

class cosmos.reinforcement_learning.world.**World** (*agents*)

Bases: object

test (*test_iter*)

train (*train_iter, n_epochs, test_iter=None*)

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)
- **train** – call predictor in train or test mode

Returns loss

`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

`reset_state()`

Resets persistent states

```
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

INDICES AND TABLES

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

COSMOS

7.1 cosmos package

7.1.1 Subpackages

cosmos.analysis package

Module contents

cosmos.reinforcement_learning package

Submodules

cosmos.reinforcement_learning.agents module

```
class cosmos.reinforcement_learning.agents.ActorCriticAgent (net, optimizer, gpu=-1, cutoff=None, gamma=0.99, beta=0.01, aac=True)
```

Bases: object

Implements advantage actor critic and REINFORCE (which does not use a value baseline)

Note that REINFORCE is a policy gradient method which does not use a critic. Instead the return is computed as a running estimate

<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

entropy (*pi*)

run (*data, train=True, idx=None, final=False*)

Parameters

- **data** – a new observation and the reward associated with the previous observation and action
- **train** –
- **idx** –
- **final** –

Returns**score_function** (*action*, *pi*)**cosmos.reinforcement_learning.iterators module****class** cosmos.reinforcement_learning.iterators.**FooTask** (*n=2*, *p=0.8*, *batch_size=1*,
n_batches=inf)

Bases: object

Very simple environment for testing fully observed models. 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

get_observation ()**Returns** observation given the state**get_state** ()**Returns** new state**next** ()**process** (*agent*)

Process agent action, compute reward and generate new state and observation

Parameters *agent* –**Returns****cosmos.reinforcement_learning.networks module****class** cosmos.reinforcement_learning.networks.**MLP** (*n_input=None*, *n_output=1*,
n_hidden=10)

Bases: chainer.link.Chain

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

Bases: chainer.link.Chain

reset ()**cosmos.reinforcement_learning.unit_test module****cosmos.reinforcement_learning.world module****class** cosmos.reinforcement_learning.world.**World** (*agents*)

Bases: object

test (*test_iter*)**train** (*train_iter*, *n_epochs*, *test_iter=None*)

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)
- **train** – call predictor in train or test mode

Returns loss

`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

`reset_state()`

Resets persistent states

```
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

7.1.2 Module contents

INDICES AND TABLES

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

C

- cosmos, 33
- cosmos.analysis, 29
- cosmos.reinforcement_learning, 31
- cosmos.reinforcement_learning.agents, 29
- cosmos.reinforcement_learning.iterators, 30
- cosmos.reinforcement_learning.networks, 30
- cosmos.reinforcement_learning.world, 30
- cosmos.supervised_learning, 33
- cosmos.supervised_learning.agents, 31
- cosmos.supervised_learning.iterators, 31
- cosmos.supervised_learning.models, 31
- cosmos.supervised_learning.networks, 32
- cosmos.supervised_learning.unit_test, 32
- cosmos.supervised_learning.world, 19

Symbols

`__call__()` (cosmos.supervised_learning.agents.SupervisedAgent method), 5, 15, 17, 23, 31
`__call__()` (cosmos.supervised_learning.models.Model method), 6, 16, 18, 24, 32

A

`ActorCriticAgent` (class in cosmos.reinforcement_learning.agents), 3, 11, 12, 21, 29

C

`Classifier` (class in cosmos.supervised_learning.models), 5, 16, 18, 23, 31
`cosmos` (module), 7, 25, 33
`cosmos.analysis` (module), 3, 9, 21, 29
`cosmos.reinforcement_learning` (module), 5, 12, 23, 31
`cosmos.reinforcement_learning.agents` (module), 3, 11, 12, 21, 29
`cosmos.reinforcement_learning.iterators` (module), 4, 11, 13, 22, 30
`cosmos.reinforcement_learning.networks` (module), 4, 12, 13, 22, 30
`cosmos.reinforcement_learning.world` (module), 4, 12, 13, 22, 30
`cosmos.supervised_learning` (module), 7, 17, 25, 33
`cosmos.supervised_learning.agents` (module), 5, 15, 17, 23, 31
`cosmos.supervised_learning.iterators` (module), 5, 15, 18, 23, 31
`cosmos.supervised_learning.models` (module), 5, 16, 18, 23, 31
`cosmos.supervised_learning.networks` (module), 6, 16, 18, 24, 32
`cosmos.supervised_learning.unit_test` (module), 6, 17, 19, 24, 32
`cosmos.supervised_learning.world` (module), 7, 17, 19, 25, 33

E

`entropy()` (cosmos.reinforcement_learning.agents.ActorCriticAgent method), 3, 11, 13, 21, 29

F

`FooTask` (class in cosmos.reinforcement_learning.iterators), 4, 11, 13, 22, 30

G

`get_observation()` (cosmos.reinforcement_learning.iterators.FooTask method), 4, 12, 13, 22, 30
`get_state()` (cosmos.reinforcement_learning.iterators.FooTask method), 4, 12, 13, 22, 30

H

`has_state()` (cosmos.supervised_learning.networks.MLP method), 6, 16, 19, 24, 32
`has_state()` (cosmos.supervised_learning.networks.RNN method), 6, 16, 19, 24, 32

M

`MLP` (class in cosmos.reinforcement_learning.networks), 4, 12, 13, 22, 30
`MLP` (class in cosmos.supervised_learning.networks), 6, 16, 18, 24, 32
`Model` (class in cosmos.supervised_learning.models), 5, 16, 18, 23, 31

N

`next()` (cosmos.reinforcement_learning.iterators.FooTask method), 4, 12, 13, 22, 30
`next()` (cosmos.supervised_learning.iterators.RandomIterator method), 5, 15, 18, 23, 31
`next()` (cosmos.supervised_learning.iterators.SequentialIterator method), 5, 15, 18, 23, 31

P

`predict()` (cosmos.supervised_learning.models.Model method), 6, 16, 18, 24, 32
`process()` (cosmos.reinforcement_learning.iterators.FooTask method), 4, 12, 13, 22, 30

R

`RandomIterator` (class in cosmos.supervised_learning.iterators), 5, 15, 18, 23, 31

Regressor (class in cosmos.supervised_learning.models),
6, 16, 18, 24, 32

reset() (cosmos.reinforcement_learning.networks.MLP
method), 4, 12, 13, 22, 30

reset() (cosmos.reinforcement_learning.networks.RNN
method), 4, 12, 13, 22, 30

reset_state() (cosmos.supervised_learning.agents.SupervisedAgent
method), 5, 15, 18, 23, 31

reset_state() (cosmos.supervised_learning.models.Model
method), 6, 16, 18, 24, 32

reset_state() (cosmos.supervised_learning.networks.MLP
method), 6, 16, 19, 24, 32

reset_state() (cosmos.supervised_learning.networks.RNN
method), 6, 17, 19, 24, 32

RNN (class in cosmos.reinforcement_learning.networks),
4, 12, 13, 22, 30

RNN (class in cosmos.supervised_learning.networks), 6,
16, 19, 24, 32

run() (cosmos.reinforcement_learning.agents.ActorCriticAgent
method), 3, 11, 13, 21, 29

S

score_function() (cosmos.reinforcement_learning.agents.ActorCriticAgent
method), 3, 11, 13, 22, 30

SequentialIterator (class in cos-
mos.supervised_learning.iterators), 5, 15,
18, 23, 31

SupervisedAgent (class in cos-
mos.supervised_learning.agents), 5, 15,
17, 23, 31

T

test() (cosmos.reinforcement_learning.world.World
method), 4, 12, 13, 22, 30

test() (cosmos.supervised_learning.agents.SupervisedAgent
method), 5, 15, 18, 23, 31

test() (cosmos.supervised_learning.world.World method),
7, 17, 19, 25, 33

test_gpu() (cosmos.supervised_learning.unit_test.UnitTest
method), 6, 17, 19, 24, 32

test_stateful_network() (cos-
mos.supervised_learning.unit_test.UnitTest
method), 7, 17, 19, 25, 33

test_stateless_network() (cos-
mos.supervised_learning.unit_test.UnitTest
method), 7, 17, 19, 25, 33

train() (cosmos.reinforcement_learning.world.World
method), 4, 12, 13, 22, 30

train() (cosmos.supervised_learning.agents.SupervisedAgent
method), 5, 15, 18, 23, 31

train() (cosmos.supervised_learning.world.World
method), 7, 17, 19, 25, 33

U

UnitTest (class in cosmos.supervised_learning.unit_test),
6, 17, 19, 24, 32

W

World (class in cosmos.reinforcement_learning.world), 4,
12, 13, 22, 30

World (class in cosmos.supervised_learning.world), 7, 17,
19, 25, 33