Cosmos Documentation

Release 0.1

Marcel van Gerven

April 22, 2017

CONTENTS

Contents:

CHAPTER

ONE

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*, *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

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

CHAPTER

TWO

COSMOS.ANALYSIS PACKAGE

2.1 Module contents

CHAPTER

THREE

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*)

CHAPTER

FOUR

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*, *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

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

CHAPTER

FIVE

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*, *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

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

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

CHAPTER

SEVEN

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*, *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

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

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

C

Symbols

A

C

E

F

G

H

M

N

P

R

S

T

U

W