TorchRL Documentation

Release 0.0.1

Lucas Vazquez

Aug 31, 2018

Package Reference

CHAPTER 1

Agents

The agent is the bridge between the model and the environment.
It implements high level functions ready to be used by the user.

1.1 BaseAgent

class torchrl.agents.**BaseAgent**(*batcher*, *optimizer*, ***, *gamma=0.99*, *log_dir='runs'*)

Bases: abc.ABC

Basic TorchRL agent. Encapsulate an environment and a model.

Parameters

- **env** (*torchrl.envs*) – A torchrl environment.
- **gamma** (*float*) – Discount factor on future rewards (Default is 0.99).
- **log_dir** (*string*) – Directory where logs will be written (Default is *runs*).

step()

This method is called at each interaction of the training loop, and defines the training procedure.

_check_termination()

Check if the training loop reached the end.

Returns

- *bool*
- *True if done, False otherwise.*

_register_model(*name*, *model*)

Save a torchrl model to the internal memory.

Parameters

- **name** (*str*) – Desired name for the model.
- **model** (*torchrl.models*) – The model to register.

train(*, *max_iters=-1*, *max_episodes=-1*, *max_steps=-1*, *log_freq=1*, *eval_env=None*, *eval_freq=None*)

Defines the training loop of the algorithm, calling *step()* at every iteration.

Parameters

- **max_updates** (*int*) – Maximum number of gradient updates (Default is -1, meaning it doesn't matter).
- **max_episodes** (*int*) – Maximum number of episodes (Default is -1, meaning it doesn't matter).
- **max_steps** (*int*) – Maximum number of steps (Default is -1, meaning it doesn't matter).

select_action(*state*, *step*)

Receive a state and use the model to select an action.

Parameters **state** (*numpy.ndarray*) – The environment state.

Returns **action** – The selected action.

Return type int or numpy.ndarray

write_logs()

Use the logger to write general information about the training process.

1.2 PGAgent

class torchrl.agents.**PGAgent**(*batcher*, *, *policy_model*, *value_model=None*, *normalize_advantages=True*, *advantage=<torchrl.utils.estimators.advantage.estimators.GAE object>*, *vtarget=<torchrl.utils.estimators.value.estimators.FromAdvantage object>*, ***kwargs*)

Bases: torchrl.agents.base_agent.BaseAgent

Policy Gradient Agent, compatible with all PG models.

This agent encapsulates a policy_model and optionally a value_model, it defines the steps needed for the training loop (see *step()*), and calculates all the necessary values to train the model(s).

Parameters

- **env** (*torchrl.envs*) – A torchrl environment.
- **policy_model** (*torchrl.models*) – Should be a subclass of torchrl.models.BasePGModel
- **value_model** (*torchrl.models*) – Should be an instance of torchrl.models.ValueModel (Default is None)
- **normalize_advantages** (*bool*) – If True, normalize the advantages per batch.
- **advantage** (*torchrl.utils.estimators.advantage*) – Class used for calculating the advantages.
- **vtarget** (*torchrl.utils.estimators.value*) – Class used for calculating the states target values.

step()
This method is called at each interaction of the training loop, and defines the training procedure.

CHAPTER 2

Models

A model encapsulate two PyTorch networks (body and head).
It defines how actions are sampled from the network and a training procedure.

2.1 BaseModel

class torchrl.models.**BaseModel**(*model*, *batcher*, *, *cuda_default=True*)

Bases: torchrl.nn.container.ModuleExtended, abc.ABC

Basic TorchRL model. Takes two Config objects that identify the body(ies) and head(s) of the model.

Parameters

- **model** (*nn.Module*) – A pytorch model.
- **batcher** (*torchrl.batcher*) – A torchrl batcher.
- **num_epochs** (*int*) – How many times to train over the entire dataset (Default is 1).
- **num_mini_batches** (*int*) – How many mini-batches to subset the batch (Default is 1, so all the batch is used at once).
- **opt_fn** (*torch.optim*) – The optimizer reference function (the constructor, not the instance) (Default is Adam).
- **opt_params** (*dict*) – Parameters for the optimizer (Default is empty dict).
- **clip_grad_norm** (*float*) – Max norm of the gradients, if float('inf') no clipping is done (Default is float('inf')).
- **loss_coef** (*float*) – Used when sharing networks, should balance the contribution of the grads of each model.
- **cuda_default** (*bool*) – If True and cuda is supported, use it (Default is True).

batch_keys
The batch keys needed for computing all losses. This is done to reduce overhead when sampling a dataloader, it makes sure only the requested keys are being sampled.

register_losses
Append losses to `self.losses`, the losses are used at `optimizer_step()` for calculating the gradients.

Parameters **batch** (*dict*) – The batch should contain all the information necessary to compute the gradients.

static output_layer(*input_shape*, *action_info*)
The final layer of the model, will be appended to the model head.

Parameters

- **input_shape** (*int or tuple*) – The shape of the input to this layer.
- **action_info** (*dict*) – Dictionary containing information about the action space.

Examples

The output of most PG models have the same dimension as the action, but the output of the Value models is rank 1. This is where this is defined.

forward(*x*)
Defines the computation performed at every call.

Parameters **x** (*numpy.ndarray*) – The environment state.

attach_logger(*logger*)
Register a logger to this model.

Parameters **logger** (*torchrl.utils.logger*) –

write_logs(*batch*)
Write logs to the terminal and to a tf log file.

Parameters **batch** (*Batch*) – Some logs might need the batch for calculation.

classmethod from_config(*config*, *batcher=None*, *body=None*, *head=None*, ***kwargs*)
Creates a model from a configuration file.

Parameters

- **config** (Config) – Should contatin at least a network definition (`nn_config` section).
- **env** (*torchrl.envs*) – A torchrl environment (Default is None and must be present in the config).
- **kwargs** (*key-word arguments*) – Extra arguments that will be passed to the class constructor.

Returns A TorchRL model.

Return type torchrl.models

2.2 ValueModel

class torchrl.models.**ValueModel**(*model*, *batcher*, ***kwargs*)
Bases: `torchrl.models.base_model.BaseModel`

A standard regression model, can be used to estimate the value of states or Q values.

Parameters **clip_range** (*float*) – Similar to PPOClip, limits the change between the new and old value function.

batch_keys
The batch keys needed for computing all losses. This is done to reduce overhead when sampling a dataloader, it makes sure only the requested keys are being sampled.

register_losses()
Append losses to `self.losses`, the losses are used at `optimizer_step()` for calculating the gradients.

Parameters **batch** (*dict*) – The batch should contain all the information necessary to compute the gradients.

write_logs(*batch*)
Write logs to the terminal and to a tf log file.

Parameters **batch** (*Batch*) – Some logs might need the batch for calculation.

static output_layer(*input_shape*, *action_info*)
The final layer of the model, will be appended to the model head.

Parameters

- **input_shape** (*int or tuple*) – The shape of the input to this layer.
- **action_info** (*dict*) – Dictionary containing information about the action space.

Examples

The output of most PG models have the same dimension as the action, but the output of the Value models is rank 1. This is where this is defined.

2.3 BasePGModel

class torchrl.models.**BasePGModel**(*model*, *batcher*, *, *entropy_coef=0*, ***kwargs*)
Bases: torchrl.models.base_model.BaseModel

Base class for all Policy Gradient Models.

entropy_loss(*batch*)
Adds a entropy cost to the loss function, with the intent of encouraging exploration.

Parameters **batch** (*Batch*) – The batch should contain all the information necessary to compute the gradients.

create_dist(*parameters*)
Specify how the policy distributions should be created. The type of the distribution depends on the environment.

Parameters

- **parameters** (*np.array*) –
- **parameters are used to create a distribution** (*The*) –
- **or discrete depending on the type of the environment)** ((*continuous*) –

write_logs(*batch*)
Write logs to the terminal and to a tf log file.

Parameters **batch** (*Batch*) – Some logs might need the batch for calculation.

static output_layer(*input_shape*, *action_info*)
The final layer of the model, will be appended to the model head.

Parameters

- **input_shape** (*int or tuple*) – The shape of the input to this layer.
- **action_info** (*dict*) – Dictionary containing information about the action space.

Examples

The output of most PG models have the same dimension as the action, but the output of the Value models is rank 1. This is where this is defined.

static select_action(*model*, *state*, *step*)
Define how the actions are selected, in this case the actions are sampled from a distribution which values are given be a NN.

Parameters **state** (*np.array*) – The state of the environment (can be a batch of states).

2.4 VanillaPGModel

class torchrl.models.**VanillaPGModel**(*model*, *batcher*, *, *entropy_coef=0*, ***kwargs*)
Bases: torchrl.models.base_pg_model.BasePGModel

The classical Policy Gradient algorithm.

batch_keys
The batch keys needed for computing all losses. This is done to reduce overhead when sampling a dataloader, it makes sure only the requested keys are being sampled.

register_losses()
Append losses to self.losses, the losses are used at optimizer_step() for calculating the gradients.

Parameters **batch** (*dict*) – The batch should contain all the information necessary to compute the gradients.

pg_loss(*batch*)
Compute loss based on the policy gradient theorem.

Parameters **batch** (*Batch*) – The batch should contain all the information necessary to compute the gradients.

2.5 A2CModel

class torchrl.models.**A2CModel**(*model*, *batcher*, *, *entropy_coef=0*, ***kwargs*)
Bases: torchrl.models.vanilla_pg_model.VanillaPGModel

A2C is just a parallel implementation of the actor-critic algorithm.

So just be sure to create a list of envs and pass to torchrl.envs.ParallelEnv to reproduce A2C.

2.6 SurrogatePGModel

class torchrl.models.**SurrogatePGModel**(*model*, *batcher*, *, *entropy_coef=0*, ***kwargs*)
Bases: torchrl.models.base_pg_model.BasePGModel

The Surrogate Policy Gradient algorithm instead maximizes a "surrogate" objective, given by:

$$L^{CPI}(\theta) = \hat{E}_t \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} \hat{A} \right]$$

batch_keys
The batch keys needed for computing all losses. This is done to reduce overhead when sampling a dataloader, it makes sure only the requested keys are being sampled.

register_losses()
Append losses to self.losses, the losses are used at optimizer_step() for calculating the gradients.

Parameters **batch** (*dict*) – The batch should contain all the information necessary to compute the gradients.

surrogate_pg_loss(*batch*)
The surrogate pg loss, as described before.

Parameters **batch** (*Batch*) –

calculate_prob_ratio(*new_log_probs*, *old_log_probs*)
Calculates the probability ratio between two policies.

Parameters

- **new_log_probs** (*torch.Tensor*) –
- **old_log_probs** (*torch.Tensor*) –

write_logs(*batch*)
Write logs to the terminal and to a tf log file.

Parameters **batch** (*Batch*) – Some logs might need the batch for calculation.

2.7 PPOClipModel

class torchrl.models.**PPOClipModel**(*model*, *batcher*, *ppo_clip_range=0.2*, ***kwargs*)
Bases: torchrl.models.surrogate_pg_model.SurrogatePGModel

Proximal Policy Optimization as described in https://arxiv.org/pdf/1707.06347.pdf.

Parameters

- **ppo_clip_range** (*float*) – Clipping value for the probability ratio (Default is 0.2).
- **num_epochs** (*int*) – How many times to train over the entire dataset (Default is 10).

register_losses()
Append losses to self.losses, the losses are used at optimizer_step() for calculating the gradients.

Parameters **batch** (*dict*) – The batch should contain all the information necessary to compute the gradients.

ppo_clip_loss(*batch*)
Calculate the PPO Clip loss as described in the paper.

Parameters **batch** (*Batch*) –

write_logs(*batch*)
Write logs to the terminal and to a tf log file.

Parameters **batch** (*Batch*) – Some logs might need the batch for calculation.

2.8 PPOAdaptiveModel

class torchrl.models.**PPOAdaptiveModel**(*model*, *batcher*, ***, *kl_target=0.01*, *kl_penalty=1.0*, ***kwargs*)
Bases: torchrl.models.surrogate_pg_model.SurrogatePGModel

Proximal Policy Optimization as described in https://arxiv.org/pdf/1707.06347.pdf.

Parameters **num_epochs** (*int*) – How many times to train over the entire dataset (Default is 10).

register_losses()
Append losses to `self.losses`, the losses are used at `optimizer_step()` for calculating the gradients.

Parameters **batch** (*dict*) – The batch should contain all the information necessary to compute the gradients.

write_logs(*batch*)
Write logs to the terminal and to a tf log file.

Parameters **batch** (*Batch*) – Some logs might need the batch for calculation.

CHAPTER 3

torchrl.envs

The environment is the world that the agent interacts with, it could be a game, a physics engine or anything you would like. It should receive and execute an action and return to the agent the next observation and a reward.

3.1 BaseEnv

class torchrl.envs.**BaseEnv**(*env_name*)

Bases: abc.ABC

Abstract base class used for implementing new environments.

Includes some basic functionalities, like the option to use a running mean and standard deviation for normalizing states.

Parameters

- **env_name** (*str*) – The environment name.
- **fixed_normalize_states** (*bool*) – If True, use the state min and max value to normalize the states (Default is False).
- **running_normalize_states** (*bool*) – If True, use the running mean and std to normalize the states (Default is False).
- **scale_reward** (*bool*) – If True, use the running std to scale the rewards (Default is False).

get_state_info()

Returns a dict containing information about the state space.

The dict should contain two keys: `shape` indicating the state shape, and `dtype` indicating the state type.

Example

State space containing 4 continuous actions:

```
return dict(shape=(4,), dtype='continuous')
```

get_action_info()
Returns a dict containing information about the action space.

The dict should contain two keys: `shape` indicating the action shape, and `dtype` indicating the action type.

If dtype is `int` it will be assumed a discrete action space.

Example

Action space containing 4 float numbers:

```
return dict(shape=(4,), dtype='float')
```

simulator
Returns the name of the simulator being used as a string.

_create_env()
Creates ans returns an environment.

Returns

Return type Environment object.

reset()
Resets the environment to an initial state.

Returns A numpy array with the state information.

Return type numpy.ndarray

step(*action*)
Receives an action and execute it on the environment.

Parameters **action** (*int or float or numpy.ndarray*) – The action to be executed in the environment, it should be an `int` for discrete enviroments and `float` for continuous. There's also the possibility of executing multiple actions (if the environment supports so), in this case it should be a `numpy.ndarray`.

Returns

- **next_state** (*numpy.ndarray*) – A numpy array with the state information.
- **reward** (*float*) – The reward.
- **done** (*bool*) – Flag indicating the termination of the episode.
- **info** (*dict*) – Dict containing additional information about the state.

update_config(*config*)
Updates a Config object to include information about the environment.

Parameters **config** (Config) – Object used for storing configuration.

3.2 GymEnv

class torchrl.envs.**GymEnv**(*env_name*, ***kwargs*)
Bases: torchrl.envs.base_env.BaseEnv

Creates and wraps a gym environment.

Parameters

- **env_name** (*str*) – The Gym ID of the env. For a list of available envs check this page.
- **wrappers** (*list*) – List of wrappers to be applied on the env. Each wrapper should be a function that receives and returns the env.

simulator
Returns the name of the simulator being used as a string.

reset()
Calls the reset method on the gym environment.

Returns **state** – A numpy array with the state information.

Return type numpy.ndarray

step(*action*)
Calls the step method on the gym environment.

Parameters **action** (*int or float or numpy.ndarray*) – The action to be executed in the environment, it should be an int for discrete enviroments and float for continuous. There's also the possibility of executing multiple actions (if the environment supports so), in this case it should be a numpy.ndarray.

Returns

- **next_state** (*numpy.ndarray*) – A numpy array with the state information.
- **reward** (*float*) – The reward.
- **done** (*bool*) – Flag indicating the termination of the episode.

get_state_info()
Dictionary containing the shape and type of the state space. If it is continuous, also contains the minimum and maximum value.

get_action_info()
Dictionary containing the shape and type of the action space. If it is continuous, also contains the minimum and maximum value.

update_config(*config*)
Updates a Config object to include information about the environment.

Parameters **config** (Config) – Object used for storing configuration.

static get_space_info(*space*)
Gets the shape of the possible types of states in gym.

Parameters **space** (*gym.spaces*) – Space object that describes the valid actions and observations

Returns Dictionary containing the space shape and type

Return type dict

3.3 RoboschoolEnv

class torchrl.envs.**RoboschoolEnv**(**args*, ***kwargs*)
Bases: torchrl.envs.gym_env.GymEnv

Support for gym Roboschool.

get_action_info()
Dictionary containing the shape and type of the action space. If it is continuous, also contains the minimum and maximum value.

static get_space_info(*space*)
Gets the shape of the possible types of states in gym.

Parameters **space** (*gym.spaces*) – Space object that describes the valid actions and observations

Returns Dictionary containing the space shape and type

Return type dict

get_state_info()
Dictionary containing the shape and type of the state space. If it is continuous, also contains the minimum and maximum value.

reset()
Calls the reset method on the gym environment.

Returns **state** – A numpy array with the state information.

Return type numpy.ndarray

simulator
Returns the name of the simulator being used as a string.

step(*action*)
Calls the step method on the gym environment.

Parameters **action** (*int or float or numpy.ndarray*) – The action to be executed in the environment, it should be an int for discrete enviroments and float for continuous. There's also the possibility of executing multiple actions (if the environment supports so), in this case it should be a numpy.ndarray.

Returns

- **next_state** (*numpy.ndarray*) – A numpy array with the state information.
- **reward** (*float*) – The reward.
- **done** (*bool*) – Flag indicating the termination of the episode.

update_config(*config*)
Updates a Config object to include information about the environment.

Parameters **config** (Config) – Object used for storing configuration.

CHAPTER 4

Containers

4.1 ModuleExtended

class torchrl.nn.**ModuleExtended**
Bases: torch.nn.modules.module.Module

A torch module with added functionalities.

4.2 SequentialExtended

class torchrl.nn.**SequentialExtended**(**args*, ***kwargs*)
Bases: torchrl.nn.container.ModuleExtended

A torch sequential module with added functionalities.

4.3 FlattenLinear

class torchrl.nn.**FlattenLinear**(*in_features*, *out_features*, ***kwargs*)
Bases: torch.nn.modules.linear.Linear

Flatten the input and apply a linear layer.

Parameters

- **in_features** (*list*) – Size of each input sample.
- **out_features** (*list*) – Size of each output sample.

4.4 ActionLinear

class torchrl.nn.**ActionLinear**(*in_features*, *action_info*, ***kwargs*)

Bases: torch.nn.modules.module.Module

A linear layer that automatically calculates the output shape based on the action_info.

Parameters

- **in_features** (*list*) – Size of each input sample
- **action_info** (*dict*) – Dict containing information about the environment actions (e.g. shape).

CHAPTER 5

torchrl.utils

5.1 Config

Configuration object used by other modules. Can be saved and imported as a YAML file

class torchrl.utils.config.**Config**(**args*, ***kwargs*)
Bases: object

Configuration object used for initializing an Agent. It maintains the order from which the attributes have been set.

Parameters **configs** (*Keyword arguments*) – Additional parameters that will be stored.

Returns An object containing all configuration details (with possibly nested Config).

Return type Config object

as_dict()
Returns all object attributes as a nested OrderedDict.

Returns Nested OrderedDict containing all object attributes.

Return type dict

new_section(*name*, ***configs*)
Creates a new Config object and add as an attribute of this instance.

Parameters

- **name** (*str*) – Name of the new section.
- **configs** (*Keyword arguments*) – Parameters that will be stored in this section, accepts nested parameters.

Examples

Simple use case:

```
config.new_section('new_section_name', attr1=value1, attr2=value2, ...)
```

Nested parameters:

```
config.new_section('new_section_name', attr1=Config(attr1=value1,␣
↪attr2=value2))
```

It's possible to access the variable like so:

```
config.new_section_name.attr1
```

save (*file_path*)
Saves current configuration to a JSON file. The configuration is stored as a nested dictionary (maintaining the order).

Parameters **file_path** (*str*) – Path to write the file

classmethod from_default (*name*)
Loads configuration from a default agent.

Parameters **name** (*str*) – Name of the desired config file ('VanillaPG', add_more)

Returns A configuration object loaded from a JSON file

Return type *Config*

static load (*file_path*)
Loads configuration from a JSON file.

Parameters **file_path** (*str*) – Path of the file to be loaded.

Returns A configuration object loaded from a JSON file

Return type *Config*

5.2 Memories

5.2.1 SimpleMemory

class torchrl.utils.memories.**SimpleMemory** (**args*, *initial_keys=None*, ***kwargs*)
Bases: dict

A dict whose keys can be accessed as attributes.

Parameters **initial_keys** (*list of strings*) – Each key will be initialized as an empty list.

5.2.2 DefaultMemory

class torchrl.utils.memories.**DefaultMemory** (**args*, ***kwargs*)
Bases: collections.defaultdict

A defaultdict whose keys can be accessed as attributes.

5.3 Logger

class torchrl.utils.logger.**Logger**(*log_dir=None*, ***, *debug=False*, *log_freq=1*)
Common logger used by all agents, aggregates values and print a nice table.

Parameters **log_dir** (*str*) – Path to write logs file.

add_log(*name*, *value*, *precision=2*)
Register a value to a name, this function can be called multiple times and the values will be averaged when logging.

Parameters

- **name** (*str*) – Name displayed when printing the table.
- **value** (*float*) – Value to log.
- **precision** (*int*) – Decimal points displayed for the value (Default is 2).

add_tf_only_log(*name*, *value*, *precision=2*)
Register a value to a name, this function can be called multiple times and the values will be averaged when logging. Will not display the logs on the console but just write on the file.

Parameters

- **name** (*str*) – Name displayed when printing the table.
- **value** (*float*) – Value to log.
- **precision** (*int*) – Decimal points displayed for the value (Default is 2).

add_histogram(*name*, *values*)
Register a histogram that can be seen at tensorboard.

Parameters

- **name** (*str*) – Name displayed when printing the table.
- **value** (*torch.Tensor*) – Value to log.

log(*header=None*)
Use the aggregated values to print a table and write to the log file.

Parameters **header** (*str*) – Optional header to include at the top of the table (Default is None).

timeit(*i_step*, *max_steps=-1*)
Estimates steps per second by counting how many steps passed between each call of this function.

Parameters

- **i_step** (*int*) – The current time step.
- **max_steps** (*int*) – The maximum number of steps of the training loop (Default is -1).

5.4 Net Builder

5.4.1 auto_input_shape

torchrl.utils.net_builder.**auto_input_shape**(*obj_config*, *input_shape*)
Create the right input parameter for the type of layer

Parameters

- **obj_config** (*dict*) – A dict containing the function and the parameters for creating the object.
- **input_shape** (*list*) – The input dimensions.

5.4.2 get_module_list

torchrl.utils.net_builder.**get_module_list**(*config*, *input_shape*, *action_info*)

Receives a config object and creates a list of layers.

Parameters

- **config** (Config) – The configuration object that should contain the basic network structure.
- **input_shape** (*list*) – The input dimensions.
- **action_info** (*dict*) – Dict containing information about the environment actions (e.g. shape).

Returns A list containing all the instantiated layers.

Return type list of layers

5.4.3 nn_from_config

torchrl.utils.net_builder.**nn_from_config**(*config*, *state_info*, *action_info*, *body=None*, *head=None*)

Creates a pytorch model following the instructions of config.

Parameters

- **config** (Config) – The configuration object that should contain the basic network structure.
- **state_info** (*dict*) – Dict containing information about the environment states (e.g. shape).
- **action_info** (*dict*) – Dict containing information about the environment actions (e.g. shape).
- **body** (*Module*) – If given use it instead of creating (Default is None).
- **head** (*Module*) – If given use it instead of creating (Default is None).

Returns A torchrl NN (basically a pytorch NN with extended functionalities).

Return type torchrl.SequentialExtended

5.5 Utils

5.5.1 get_obj

torchrl.utils.utils.**get_obj**(*config*)

Creates an object based on the given config.

Parameters **config** (*dict*) – A dict containing the function and the parameters for creating the object.

Returns The created object.

Return type obj

5.5.2 env_from_config

torchrl.utils.utils.**env_from_config**(*config*)

Tries to create an environment from a configuration obj.

Parameters **config** (Config) – Configuration file containing the environment function.

Returns **env** – A torchrl environment.

Return type torchrl.envs

Raises AttributeError – If no env is defined in the config obj.

5.5.3 join_transitions

5.5.4 to_np

torchrl.utils.utils.**to_np**(*value*)

5.5.5 explained_var

torchrl.utils.utils.**explained_var**(*target*, *preds*)

Calculates the explained variance between two datasets. Useful for estimating the quality of the value function

Parameters

- **target** (*np.array*) – Target dataset.
- **preds** (*np.array*) – Predictions array.

Returns The explained variance.

Return type float

5.5.6 normalize

torchrl.utils.utils.**normalize**(*array*)

Normalize an array by subtracting the mean and diving by the std dev.

CHAPTER 6

Indices and tables

- genindex
- modindex

Python Module Index

t

Index

Symbols

A

B

C

D

E

F

G

L

M

N