DeepSphere

Release 0.2.1

Arcanite Solutions

Apr 04, 2022

CONTENT:

CHAPTER

ONE

DEEPSPHERE PACKAGE

1.1 Subpackages

1.1.1 deepsphere.data package

Subpackages

deepsphere.data.datasets package

Submodules

deepsphere.data.datasets.dataset module

Datasets for reduced atmospheric river and tropical cyclone detection dataset.

class deepsphere.data.datasets.dataset.**ARTCDataset**(*path*, *indices=None*, *transform_data=None*, *transform_labels=None*, *download=False*)

Bases: torch.utils.data.Dataset

Dataset for reduced atmospheric river and tropical cyclone dataset.

check_exists()
Check if dataset already exists.

download()
Download the dataset if it doesn't already exist.

get_runs(*runs*)
Get datapoints corresponding to specific runs.

Parameters **runs** (*list*) – List of desired runs.

Returns List of strings, which represents the files in the dataset, which belong to one of the desired runs.

Return type list

property indices
Get files.

Returns List of strings, which represent the files contained in the dataset.

Return type list

resource = 'http://island.me.berkeley.edu/ugscnn/data/climate_sphere_l5.zip'

class deepsphere.data.datasets.dataset.**ARTCTemporaldataset**(*path*, *sequence_length*, *prediction_shift=0*, *indices=None*, *transform_image=None*, *transform_labels=None*, *transform_sample=None*, *download=False*)

Bases: *deepsphere.data.datasets.dataset.ARTCDataset*

Dataset for reduced ARTC dataset with temporality functionality.

Module contents

deepsphere.data.transforms package

Submodules

deepsphere.data.transforms.transforms module

Transformations for samples of atmospheric rivers and tropical cyclones dataset.

class deepsphere.data.transforms.transforms.**Normalize**(*mean*, *std*)

Bases: object

Normalize using mean and std.

class deepsphere.data.transforms.transforms.**Permute**

Bases: object

Permute first and second dimension.

class deepsphere.data.transforms.transforms.**Stack**(*dimension=0*)

Bases: object

Stack images in torch tensor.

class deepsphere.data.transforms.transforms.**ToTensor**

Bases: object

Convert raw data and labels to PyTorch tensor.

Module contents

Module contents

1.1.2 deepsphere.layers package

Subpackages

deepsphere.layers.samplings package

Submodules

deepsphere.layers.samplings.equiangular_pool_unpool module

EquiAngular Sampling's Pooling and Unpooling. The pooling goes down two bandwidths at a time. This represents (in the term of classic pooling kernel sizes) a division (pooling) or multiplication (unpooling) of the number of pixels by 4. The kernel size for all modules is henced fixed.

Equiangular sampling theory from: *FFTs for the 2-Sphere:Improvements and Variations* by Healy (doi=10.1.1.51.5335)

Bandwidth : int or list or tuple. Hence we have a symetric or asymetric sampling. It corresponds to the resolution of the sampling scheme. $pixels = (2 * bw)^2$ Allowed number of pixels:

- (bw=1) 4 pixels,
- (bw=2) 16 pixels,
- (bw=3) 36 pixels,
- (bw=4) 64 pixels,
- (bw=5) 100 pixels.

If latitude bandwidth is different from longitude bandwidth then we have: $pixels = ((2 * bw_{latitude}) * *2) * ((2 * bw_{longitude}) * *2)$

class deepsphere.layers.samplings.equiangular_pool_unpool.**Equiangular**(*ratio=1*, *mode='average'*)

Bases: object

Equiangular class, which groups together the corresponding pooling and unpooling.

property pooling
Getter for the pooling class

property unpooling
Getter for the unpooling class

class deepsphere.layers.samplings.equiangular_pool_unpool.**EquiangularAvgPool**(*ratio*)

Bases: torch.nn.AvgPool1d

EquiAngular Average Pooling using Average Pooling 1d from pytorch

forward(*x*)
calls Avgpool1d

Parameters **x** (torch.tensor) – batch x pixels x features

Returns torch.tensor – batch x pooled pixels x features

class deepsphere.layers.samplings.equiangular_pool_unpool.**EquiangularAvgUnpool**(*ratio*)

Bases: torch.nn.Module

EquiAngular Average Unpooling version 1 using the interpolate function when unpooling

forward(*x*)
calls pytorch's interpolate function to create the values while unpooling based on the nearby values :param x: batch x pixels x features :type x: torch.tensor

Returns batch x unpooled pixels x features

Return type torch.tensor

class deepsphere.layers.samplings.equiangular_pool_unpool.**EquiangularMaxPool**(*ratio*, *return_indices=False*)

Bases: torch.nn.MaxPool1d

EquiAngular Maxpooling module using MaxPool 1d from torch

forward(*x*)

calls Maxpool1d and if desired, keeps indices of the pixels pooled to unpool them

Parameters **input** (torch.tensor) – batch x pixels x features

Returns batch x pooled pixels x features and the indices of the pixels pooled

Return type tuple(torch.tensor, list(int))

class deepsphere.layers.samplings.equiangular_pool_unpool.**EquiangularMaxUnpool**(*ratio*)

Bases: torch.nn.MaxUnpool1d

Equiangular Maxunpooling using the MaxUnpool1d of pytorch

forward(*x*, *indices*)

calls MaxUnpool1d using the indices returned previously by EquiAngMaxPool

Parameters

- **x** (torch.tensor) – batch x pixels x features
- **indices** (*int*) – indices of pixels equiangular maxpooled previously

Returns batch x unpooled pixels x features

Return type torch.tensor

deepsphere.layers.samplings.equiangular_pool_unpool.**reformat**(*x*)

Reformat the input from a 4D tensor to a 3D tensor

Parameters **x** (torch.tensor) – a 4D tensor

Returns a 3D tensor

Return type torch.tensor

deepsphere.layers.samplings.healpix_pool_unpool module

Healpix Sampling's Pooling and Unpooling The pooling divides the number of nsides by 2 each time. This represents (in the term of classic pooling kernel sizes) a division (pooling) or multiplication (unpooling) of the number of pixels by 4. The kernel size for all modules is hence fixed.

Sampling theory from: *HEALPix — a Framework for High Resolution Discretization, and Fast Analysis of Data Distributed on the Sphere* by Gorski (doi: 10.1086/427976)

Figure 1 for relation number of sides and number of pixels and for unpooling using tile. The area of the pixels are the same hence latitude and longitude of the resolution are the same.

The lowest resolution possible with the HEALPix base partitioning of the sphere surface into 12 equal sized pixels See: https://healpix.jpl.nasa.gov/

$N_{pixels} = 12 * N_{sides}^2$ Nsides is the number of divisions from the baseline of 12 equal sized pixels

class deepsphere.layers.samplings.healpix_pool_unpool.**Healpix**(*mode='average'*)

Bases: object

Healpix class, which groups together the corresponding pooling and unpooling.

property pooling
Get pooling

property unpooling
Get unpooling

class deepsphere.layers.samplings.healpix_pool_unpool.**HealpixAvgPool**
Bases: torch.nn.AvgPool1d

Healpix Average pooling module

forward(*x*)
forward call the 1d Averagepooling of pytorch

Parameters **x** (torch.tensor) – [batch x pixels x features]

Returns [batch x pooled pixels x features]

Return type [torch.tensor]

class deepsphere.layers.samplings.healpix_pool_unpool.**HealpixAvgUnpool**
Bases: torch.nn.Module

Healpix Average Unpooling module

forward(*x*)
forward repeats (here more like a numpy tile for the moment) the incoming tensor

Parameters **x** (torch.tensor) – [batch x pixels x features]

Returns [batch x unpooled pixels x features]

Return type [torch.tensor]

class deepsphere.layers.samplings.healpix_pool_unpool.**HealpixMaxPool**(*return_indices=False*)
Bases: torch.nn.MaxPool1d

Healpix Maxpooling module

forward(*x*)
Forward call the 1d Maxpooling of pytorch

Parameters **x** (torch.tensor) – [batch x pixels x features]

Returns [batch x pooled pixels x features] and indices of pooled pixels

Return type tuple((torch.tensor), indices (int))

class deepsphere.layers.samplings.healpix_pool_unpool.**HealpixMaxUnpool**
Bases: torch.nn.MaxUnpool1d

Healpix Maxunpooling using the MaxUnpool1d of pytorch

forward(*x*, *indices*)
calls MaxUnpool1d using the indices returned previously by HealpixMaxPool

Parameters

- **tuple** (x (torch.tensor) – [batch x pixels x features]
- **indices** (*int)*) – indices of pixels equiangular maxpooled previously

Returns [torch.tensor] – [batch x unpooled pixels x features]

deepsphere.layers.samplings.icosahedron_pool_unpool module

Icosahedron Sampling's Pooling and Unpooling. Each pooling takes down an order in the icosahedron. Each unpooling adds the number of pixels corresponding to the next order.

Icosahedron is a polyhedron with 12 vertices and, 20 faces, where a regular icosahedron is a Platonic solid. All faces are regular (equilateral) triangles. This default Icosahedron can be considered at level 0, meaning that no further subdivision has occurred from the platonic solid. See: https://github.com/maxjiang93/ugscnn/blob/master/meshcnn/mesh.py from Max Jiang

class deepsphere.layers.samplings.icosahedron_pool_unpool.**Icosahedron**

Bases: object

Icosahedron class, which simply groups together the corresponding pooling and unpooling.

property pooling

Get pooling.

property unpooling

Get unpooling.

class deepsphere.layers.samplings.icosahedron_pool_unpool.**IcosahedronPool**(**args*, ***kwargs*)

Bases: torch.nn.Module

Isocahedron Pooling, consists in keeping only a subset of the original pixels (considering the ordering of an isocahedron sampling method).

forward(*x*)

Forward function calculates the subset of pixels to keep based on input size and the kernel_size.

Parameters **x** (torch.tensor) – [batch x pixels x features]

Returns [batch x pixels pooled x features]

Return type [torch.tensor]

class deepsphere.layers.samplings.icosahedron_pool_unpool.**IcosahedronUnpool**(**args*, ***kwargs*)

Bases: torch.nn.Module

Isocahedron Unpooling, consists in adding 1 values to match the desired un pooling size

forward(*x*)

Forward calculates the subset of pixels that will result from the unpooling kernel_size and then adds 1 valued pixels to match this size

Parameters **x** (torch.tensor) – [batch x pixels x features]

Returns [batch x pixels unpooled x features]

Return type [torch.tensor]

Module contents

DeepSphere Base Documentation doc

Submodules

deepsphere.layers.chebyshev module

Chebyshev convolution layer. For the moment taking as-is from Michaël Defferrard's implementation. For v0.15 we will rewrite parts of this layer.

class deepsphere.layers.chebyshev.**ChebConv**(*in_channels*, *out_channels*, *kernel_size*, *bias=True*, *conv=<function cheb_conv>*)

Bases: torch.nn.Module

Graph convolutional layer.

forward(*laplacian*, *inputs*)
Forward graph convolution.

Parameters

- **laplacian** (torch.sparse.Tensor) – The laplacian corresponding to the current sampling of the sphere.
- **inputs** (torch.Tensor) – The current input data being forwarded.

Returns The convoluted inputs.

Return type torch.Tensor

kaiming_initialization()
Initialize weights and bias.

class deepsphere.layers.chebyshev.**SphericalChebConv**(*in_channels*, *out_channels*, *lap*, *kernel_size*)

Bases: torch.nn.Module

Building Block with a Chebyshev Convolution.

forward(*x*)
Forward pass.

Parameters **x** (torch.tensor) – input [batch x vertices x channels/features]

Returns output [batch x vertices x channels/features]

Return type torch.tensor

state_dict(**args*, ***kwargs*)
! WARNING !

This function overrides the state dict in order to be able to save the model. This can be removed as soon as saving sparse matrices has been added to Pytorch.

deepsphere.layers.chebyshev.**cheb_conv**(*laplacian*, *inputs*, *weight*)
Chebyshev convolution.

Parameters

- **laplacian** (torch.sparse.Tensor) – The laplacian corresponding to the current sampling of the sphere.
- **inputs** (torch.Tensor) – The current input data being forwarded.
- **weight** (torch.Tensor) – The weights of the current layer.

Returns Inputs after applying Chebyshev convolution.

Return type torch.Tensor

Module contents

1.1.3 deepsphere.models package

Subpackages

deepsphere.models.spherical_unet package

Submodules

deepsphere.models.spherical_unet.decoder module

Decoder for Spherical UNet.

class deepsphere.models.spherical_unet.decoder.**Decoder**(*unpooling*, *laps*, *kernel_size*)

Bases: torch.nn.Module

The decoder of the Spherical UNet.

forward(*x_enc0*, *x_enc1*, *x_enc2*, *x_enc3*, *x_enc4*)

Forward Pass.

Parameters **x_enc*** (torch.Tensor) – input tensors.

Returns output after forward pass.

Return type torch.Tensor

class deepsphere.models.spherical_unet.decoder.**SphericalChebBNPoolCheb**(*in_channels*, *middle_channels*, *out_channels*, *lap*, *pooling*, *kernel_size*)

Bases: torch.nn.Module

Building Block calling a SphericalChebBNPool block then a SphericalCheb.

forward(*x*)

Forward Pass.

Parameters **x** (torch.Tensor) – input [batch x vertices x channels/features]

Returns output [batch x vertices x channels/features]

Return type torch.Tensor

class deepsphere.models.spherical_unet.decoder.**SphericalChebBNPoolConcat**(*in_channels*, *out_channels*, *lap*, *pooling*, *kernel_size*)

Bases: torch.nn.Module

Building Block calling a SphericalChebBNPool Block then concatenating the output with another tensor and calling a SphericalChebBN block.

forward(*x*, *concat_data*)
Forward Pass.

Parameters

- **x** (torch.Tensor) – input [batch x vertices x channels/features]
- **concat_data** (torch.Tensor) – encoder layer output [batch x vertices x channels/features]

Returns output [batch x vertices x channels/features]

Return type torch.Tensor

deepsphere.models.spherical_unet.encoder module

Encoder for Spherical UNet.

class deepsphere.models.spherical_unet.encoder.**Encoder**(*pooling*, *laps*, *kernel_size*)
Bases: torch.nn.Module

Encoder for the Spherical UNet.

forward(*x*)
Forward Pass.

Parameters **x** (torch.Tensor) – input [batch x vertices x channels/features]

Returns obj: *torch.Tensor*: output [batch x vertices x channels/features]

Return type x_enc*

class deepsphere.models.spherical_unet.encoder.**EncoderTemporalConv**(*pooling*, *laps*, *sequence_length*, *kernel_size*)
Bases: *deepsphere.models.spherical_unet.encoder.Encoder*

Encoder for the Spherical UNet temporality with convolution.

class deepsphere.models.spherical_unet.encoder.**SphericalChebBN2**(*in_channels*, *middle_channels*, *out_channels*, *lap*, *kernel_size*)
Bases: torch.nn.Module

Building Block made of 2 Building Blocks (convolution, batchnorm, activation).

forward(*x*)
Forward Pass.

Parameters **x** (torch.Tensor) – input [batch x vertices x channels/features]

Returns output [batch x vertices x channels/features]

Return type torch.Tensor

class deepsphere.models.spherical_unet.encoder.**SphericalChebPool**(*in_channels*, *out_channels*, *lap*, *pooling*, *kernel_size*)

Bases: torch.nn.Module

Building Block with a pooling/unpooling and a Chebyshev Convolution.

forward(*x*)

Forward Pass.

Parameters **x** (torch.Tensor) – input [batch x vertices x channels/features]

Returns output [batch x vertices x channels/features]

Return type torch.Tensor

deepsphere.models.spherical_unet.unet_model module

Spherical Graph Convolutional Neural Network with UNet autoencoder architecture.

class deepsphere.models.spherical_unet.unet_model.**SphericalUNet**(*pooling_class*, *N*, *depth*, *laplacian_type*, *kernel_size*, *ratio=1*)

Bases: torch.nn.Module

Spherical GCNN Autoencoder.

forward(*x*)

Forward Pass.

Parameters **x** (torch.Tensor) – input to be forwarded.

Returns output

Return type torch.Tensor

class deepsphere.models.spherical_unet.unet_model.**SphericalUNetTemporalConv**(*pooling_class*, *N*, *depth*, *laplacian_type*, *sequence_length*, *kernel_size*, *ratio=1*)

Bases: *deepsphere.models.spherical_unet.unet_model.SphericalUNet*

Spherical GCNN Autoencoder with temporality by means of convolution over time.

forward(*x*)

Forward Pass.

Parameters **x** (torch.Tensor) – input to be forwarded.

Returns output

Return type torch.Tensor

class deepsphere.models.spherical_unet.unet_model.**SphericalUNetTemporalLSTM**(*pooling_class*, *N*, *depth*, *laplacian_type*, *sequence_length*, *kernel_size*, *ratio=1*)

Bases: *deepsphere.models.spherical_unet.unet_model.SphericalUNet*

Sphericall GCNN Autoencoder with LSTM.

forward(*x*)

Forward Pass.

Parameters **x** (torch.Tensor) – input to be forwarded.

Returns output

Return type torch.Tensor

deepsphere.models.spherical_unet.utils module

Layers used in both Encoder and Decoder.

class deepsphere.models.spherical_unet.utils.**SphericalChebBN**(*in_channels*, *out_channels*, *lap*, *kernel_size*)

Bases: torch.nn.Module

Building Block with a Chebyshev Convolution, Batchnormalization, and ReLu activation.

forward(*x*)

Forward Pass.

Parameters **x** (torch.tensor) – input [batch x vertices x channels/features]

Returns output [batch x vertices x channels/features]

Return type torch.tensor

class deepsphere.models.spherical_unet.utils.**SphericalChebBNPool**(*in_channels*, *out_channels*, *lap*, *pooling*, *kernel_size*)

Bases: torch.nn.Module

Building Block with a pooling/unpooling, a calling the SphericalChebBN block.

forward(*x*)

Forward Pass.

Parameters **x** (torch.tensor) – input [batch x vertices x channels/features]

Returns output [batch x vertices x channels/features]

Return type torch.tensor

Module contents

Module contents

1.1.4 deepsphere.tests package

Submodules

deepsphere.tests.test_foo module

Fake file to test the doc

class deepsphere.tests.test_foo.**TestFoo**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Fake test class in order to setup the tests module

test_foo()
Fake test method in order to setup the test module

Module contents

The tests module contains different directory and files that have the goal to test different parts of the code

Class

You can see in this module the TestFoo that contain the different method:

TestFoo.test_foo()	Fake test method in order to setup the test module

More Doc / Example

You can add then more doc and even examples

1.1.5 deepsphere.utils package

Submodules

deepsphere.utils.initialization module

Initializing device

deepsphere.utils.initialization.**init_dataset_temp**(*parser*, *indices*, *transform_image*, *transform_labels*)

Initialize the dataset

Parameters

- **parser** (*dict*) – parser arguments
- **indices** (*list*) – The list of indices we want included in the dataset

- **transform_image** (*list*) – The list of torchvision transforms we want to apply to the images
- **transform_labels** (*list*) – The list of torchvision transforms we want to apply to the labels

Returns the dataset

Return type dataset

deepsphere.utils.initialization.**init_device**(*device*, *unet*)
Initialize device based on cpu/gpu and number of gpu

Parameters

- **device** (*str*) – cpu or gpu
- **ids** (*list of int or str*) – list of gpus that should be used
- **unet** (*torch.Module*) – the model to place on the device(s)

Raises **Exception** – There is an error in configuring the cpu or gpu

Returns the model placed on device, the device

Return type torch.Module, torch.device

deepsphere.utils.initialization.**init_unet_temp**(*parser*)
Initialize UNet

Parameters **parser** (*dict*) – parser arguments

Returns the model

Return type unet

deepsphere.utils.laplacian_funcs module

Functions related to getting the laplacian and the right number of pixels after pooling/unpooling.

deepsphere.utils.laplacian_funcs.**get_equiangular_laplacians**(*nodes*, *depth*, *ratio*, *laplacian_type*)
Get the equiangular laplacian list for a certain depth. :param nodes: initial number of nodes. :type nodes: int :param depth: the depth of the UNet. :type depth: int :param laplacian_type ["combinatorial", "normalized"]: the type of the laplacian.

Returns increasing list of laplacians

Return type laps (list)

deepsphere.utils.laplacian_funcs.**get_healpix_laplacians**(*nodes*, *depth*, *laplacian_type*)
Get the healpix laplacian list for a certain depth. :param nodes: initial number of nodes. :type nodes: int :param depth: the depth of the UNet. :type depth: int :param laplacian_type ["combinatorial", "normalized"]: the type of the laplacian.

Returns increasing list of laplacians.

Return type laps (list)

deepsphere.utils.laplacian_funcs.**get_icosahedron_laplacians**(*nodes*, *depth*, *laplacian_type*)
Get the icosahedron laplacian list for a certain depth. :param nodes: initial number of nodes. :type nodes: int :param depth: the depth of the UNet. :type depth: int :param laplacian_type ["combinatorial", "normalized"]: the type of the laplacian.

Returns increasing list of laplacians.

Return type laps (list)

deepsphere.utils.laplacian_funcs.**prepare_laplacian**(*laplacian*)
Prepare a graph Laplacian to be fed to a graph convolutional layer.

deepsphere.utils.laplacian_funcs.**scipy_csr_to_sparse_tensor**(*csr_mat*)
Convert scipy csr to sparse pytorch tensor.

Parameters **csr_mat** (*csr_matrix*) – The sparse scipy matrix.

Returns The sparse torch matrix.

Return type sparse_tensor `torch.sparse.FloatTensor`

deepsphere.utils.parser module

Command Line Parser realated functions. One function creates the parser. Another function allows hybird usage of: - a yaml file with predefined parameters and - user inputted parameters through the command line.

deepsphere.utils.parser.**create_parser**()
Creates a parser with all the variables that can be edited by the user.

Returns a parser for the command line

Return type parser

deepsphere.utils.parser.**parse_config**(*parser*)
Takes the yaml file given through the command line Adds all the yaml file parameters, unless they have already been defined in the command line. Checks all values have been set else raises a Value error. :param parser: parser to be updated by the yaml file parameters :type parser: argparse.parser

Raises **ValueError** – All fields must be set in the yaml config file or in the command line. Raises error if value is None (was not set).

Returns parsed args of the parser

Return type dict

deepsphere.utils.samplings module

Different samplings require various calculations. The calculations present here are for equiangular, healpix, icosahedron samplings.

deepsphere.utils.samplings.**equiangular_bandwidth**(*nodes*)
Calculate the equiangular bandwidth based on input nodes

Parameters **nodes** (*int*) – the number of nodes should be a power of 4

Returns the corresponding bandwidth

Return type int

deepsphere.utils.samplings.**equiangular_calculator**(*tensor*, *ratio*)
From a 3D input tensor and a known ratio between the latitude dimension and longitude dimension of the data, reformat the 3D input into a 4D output while also obtaining the bandwidth.

Parameters

- **tensor** (`torch.tensor`) – 3D input tensor
- **ratio** (*float*) – the ratio between the latitude and longitude dimension of the data

Returns 4D tensor, the bandwidths for lat. and long.

Return type `torch.tensor`, int, int

`deepsphere.utils.samplings.`**`equiangular_dimension_unpack`**(*nodes*, *ratio*)
Calculate the two underlying dimensions from the total number of nodes

Parameters

- **`nodes`** (*int*) – combined dimensions
- **`ratio`** (*float*) – ratio between the two dimensions

Returns separated dimensions

Return type int, int

`deepsphere.utils.samplings.`**`healpix_resolution_calculator`**(*nodes*)
Calculate the resolution of a healpix graph for a given number of nodes.

Parameters **`nodes`** (*int*) – number of nodes in healpix sampling

Returns resolution for the matching healpix graph

Return type int

`deepsphere.utils.samplings.`**`icosahedron_nodes_calculator`**(*order*)
Calculate the number of nodes corresponding to the order of an icosahedron graph

Parameters **`order`** (*int*) – order of an icosahedron graph

Returns number of nodes in icosahedron sampling for that order

Return type int

`deepsphere.utils.samplings.`**`icosahedron_order_calculator`**(*nodes*)
Calculate the order of a icosahedron graph for a given number of nodes.

Parameters **`nodes`** (*int*) – number of nodes in icosahedron sampling

Returns order for the matching icosahedron graph

Return type int

deepsphere.utils.stats_extractor module

Get Means and Standard deviations for all features of a dataset.

`deepsphere.utils.stats_extractor.`**`stats_extractor`**(*dataset*)
Iterates over a dataset object It is iterated over so as to calculate the mean and standard deviation.

Parameters **`dataset`** (`torch.utils.data.dataloader`) – dataset object to iterate over

Returns obj:numpy.array, :obj:numpy.array : computed means and standard deviation

Module contents

1.2 Module contents

DeepSphere Base Documentation doc

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

INDEX

F

G

H

I

K