EchoTorch Documentation

Release 0.1

Nils Schaetti

Feb 22, 2018

Notes

EchoTorch is an pyTorch-based library for Reservoir Computing and Echo State Network using GPUs and CPUs.

CHAPTER 1

Echo State Network learning mechanics

This note will present an overview of how Echo State Networks works works and its learning mechanics. It's not mandatory to understand the complete learning phase, but we recommend understanding the differnce between classical ESN learning and gradient descent, it will help you to choose which one to use according to cases.

1.1 The Echo State Network model

1.1.1 `esn_learning`

CHAPTER 2

echotorch package

2.1 Subpackages

2.1.1 echotorch.datasets package

Submodules

echotorch.datasets.MackeyGlassDataset module

class echotorch.datasets.MackeyGlassDataset.**MackeyGlassDataset**(*sample_len*, *n_samples*, *tau=17*, *seed=None*)

Bases: torch.utils.data.dataset.Dataset

Mackey Glass dataset

echotorch.datasets.MemTestDataset module

class echotorch.datasets.MemTestDataset.**MemTestDataset**(*sample_len*, *n_samples*, *n_delays=10*, *seed=None*)

Bases: torch.utils.data.dataset.Dataset

Generates a series of input timeseries and delayed versions as outputs. Delay is given in number of timesteps. Can be used to empirically measure the memory capacity of a system.

echotorch.datasets.NARMADataset module

class echotorch.datasets.NARMADataset.**NARMADataset**(*sample_len*, *n_samples*, *system_order=10*, *seed=None*)

Bases: torch.utils.data.dataset.Dataset

xth order NARMA task WARNING: this is an unstable dataset. There is a small chance the system becomes unstable, leading to an unusable dataset. It is better to use NARMA30 which where this problem happens less often.

Module contents

class echotorch.datasets.**MackeyGlassDataset**(*sample_len*, *n_samples*, *tau=17*, *seed=None*)

Bases: torch.utils.data.dataset.Dataset

Mackey Glass dataset

class echotorch.datasets.**MemTestDataset**(*sample_len*, *n_samples*, *n_delays=10*, *seed=None*)

Bases: torch.utils.data.dataset.Dataset

Generates a series of input timeseries and delayed versions as outputs. Delay is given in number of timesteps. Can be used to empirically measure the memory capacity of a system.

class echotorch.datasets.**NARMADataset**(*sample_len*, *n_samples*, *system_order=10*, *seed=None*)

Bases: torch.utils.data.dataset.Dataset

xth order NARMA task WARNING: this is an unstable dataset. There is a small chance the system becomes unstable, leading to an unusable dataset. It is better to use NARMA30 which where this problem happens less often.

class echotorch.datasets.**ReutersC50Dataset**(*root='./data'*, *download=False*, *n_authors=50*, *dataset_size=100*, *dataset_start=0*, *authors=None*, *transform=None*, *train=True*, *k=10*, *retain_transform=False*, *load_transform=False*)

Bases: torch.utils.data.dataset.Dataset

Reuters C50 dataset

set_fold(*fold*)
Set fold :param fold: :return:

set_start(*start*)
Set start :param start: :return:

set_train(*mode*)
Set train (true, false) :param mode: :return:

class echotorch.datasets.**SFGramDataset**(*tokenizer*, *root='./data'*, *download=False*, *transform=None*, *train=True*, *k=10*, *dataset_size=91*)

Bases: torch.utils.data.dataset.Dataset

SFGram dataset

set_fold(*fold*)
Set fold :param fold: :return:

set_train(*mode*)
Set train (true, false) :param mode: :return:

tag_text(*text_content*)
Tag text :param text_content: :return:

2.1.2 echotorch.nn

Echo State Layers

ESNCell

class nn.**ESNCell**(*input_dim, output_dim, spectral_radius=0.9, bias_scaling=0, input_scaling=1.0, w=None, w_in=None, w_bias=None, w_fdb=None, sparsity=None, input_set=[1.0, -1.0], w_sparsity=None, nonlin_func=<built-in function tanh>, feedbacks=False*)

Echo State Network layer

forward(*u*, *y=None*, *w_out=None*)

Forward :param u: Input signal :param y: Target output signal for teacher forcing :param w_out: Output weights for teacher forcing :return: Resulting hidden states

static generate_w(*output_dim*, *w_sparsity=None*)

Generate W matrix :param output_dim: :param w_sparsity: :return:

get_spectral_radius()

Get W's spectral radius :return: W's spectral radius

init_hidden()

Init hidden layer :return: Initiated hidden layer

reset_hidden()

Reset hidden layer :return:

static to_sparse(*m*)

To sparse matrix :param m: :return:

ESN

class nn.**ESN**(*input_dim, hidden_dim, output_dim, spectral_radius=0.9, bias_scaling=0, input_scaling=1.0, w=None, w_in=None, w_bias=None, w_fdb=None, sparsity=None, input_set=[1.0, -1.0], w_sparsity=None, nonlin_func=<built-in function tanh>, learning_algo='inv', ridge_param=0.0, create_cell=True, feedbacks=False, with_bias=True*)

Echo State Network module

finalize()

Finalize training with LU factorization

forward(*u*, *y=None*)

Forward :param u: Input signal. :param y: Target outputs :return: Output or hidden states

get_spectral_radius()

Get W's spectral radius :return: W's spectral radius

get_w_out()

Output matrix :return:

hidden

Hidden layer :return:

reset()

Reset learning :return:

reset_hidden()

Reset hidden layer :return:

set_w(*w*)
Set W :param w: :return:

w
Hidden weight matrix :return:

w_in
Input matrix :return:

LiESNCell

class nn.**LiESNCell**(*leaky_rate=1.0*, *train_leaky_rate=False*, **args*, ***kwargs*)
Leaky-Integrated Echo State Network layer

forward(*u*, *y=None*, *w_out=None*)
Forward :param u: Input signal. :return: Resulting hidden states.

LiESN

class nn.**LiESN**(*input_dim, hidden_dim, output_dim, spectral_radius=0.9, bias_scaling=0, input_scaling=1.0, w=None, w_in=None, w_bias=None, sparsity=None, input_set=[1.0, -1.0], w_sparsity=None, nonlin_func=<built-in function tanh>, learning_algo='inv', ridge_param=0.0, leaky_rate=1.0, train_leaky_rate=False, feedbacks=False*)
Leaky-Integrated Echo State Network module

2.1.3 echotorch.tools package

Submodules

echotorch.utils.error_measures module

echotorch.utils.error_measures.**mse**(*outputs*, *targets*)
Mean square error :param outputs: Module's outputs :param targets: Target signal to be learned :return: Mean square deviation

echotorch.utils.error_measures.**nmse**(*outputs*, *targets*)
Normalized mean square error :param outputs: Module's output :param targets: Target signal to be learned :return: Normalized mean square deviation

echotorch.utils.error_measures.**nrmse**(*outputs*, *targets*)
Normalized root-mean square error :param outputs: Module's outputs :param targets: Target signal to be learned :return: Normalized root-mean square deviation

echotorch.utils.error_measures.**rmse**(*outputs*, *targets*)
Root-mean square error :param outputs: Module's outputs :param targets: Target signal to be learned :return: Root-mean square deviation

echotorch.utils.utility_functions module

echotorch.utils.utility_functions.**average_prob**(*tensor*, *dim=0*)
Average probabilities through time :param tensor: :param dim: :return:

echotorch.utils.utility_functions.**max_average_through_time**(*tensor*, *dim=0*)
Max average through time :param tensor: :param dim: Time dimension :return:

echotorch.utils.utility_functions.**normalize**(*tensor*, *dim=1*)
Normalize a tensor on a single dimension :param t: :return:

echotorch.utils.utility_functions.**spectral_radius**(*m*)
Compute spectral radius of a square 2-D tensor :param m: squared 2D tensor :return:

Module contents

echotorch.utils.**nrmse**(*outputs*, *targets*)
Normalized root-mean square error :param outputs: Module's outputs :param targets: Target signal to be learned :return: Normalized root-mean square deviation

echotorch.utils.**nmse**(*outputs*, *targets*)
Normalized mean square error :param outputs: Module's output :param targets: Target signal to be learned :return: Normalized mean square deviation

echotorch.utils.**rmse**(*outputs*, *targets*)
Root-mean square error :param outputs: Module's outputs :param targets: Target signal to be learned :return: Root-mean square deviation

echotorch.utils.**mse**(*outputs*, *targets*)
Mean square error :param outputs: Module's outputs :param targets: Target signal to be learned :return: Mean square deviation

echotorch.utils.**spectral_radius**(*m*)
Compute spectral radius of a square 2-D tensor :param m: squared 2D tensor :return:

echotorch.utils.**normalize**(*tensor*, *dim=1*)
Normalize a tensor on a single dimension :param t: :return:

echotorch.utils.**average_prob**(*tensor*, *dim=0*)
Average probabilities through time :param tensor: :param dim: :return:

echotorch.utils.**max_average_through_time**(*tensor*, *dim=0*)
Max average through time :param tensor: :param dim: Time dimension :return:

2.2 Module contents

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

e

t

Index

T

W