EchoTorch Documentation

Release 0.1

Nils Schaetti

Jan 22, 2021

Notes

1 Echo State Network learning mechanics 3

2 echotorch package 5

3 Indices and tables 13

Python Module Index 15

Index 17

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.**DatasetComposer**(*datasets*, **args*, ***kwargs*)
Bases: torch.utils.data.dataset.Dataset

Compose dataset

class echotorch.datasets.**HenonAttractor**(*sample_len*, *n_samples*, *xy*, *a*, *b*, *washout=0*, *normalize=False*, *seed=None*)
Bases: torch.utils.data.dataset.Dataset

The Rössler attractor is the attractor for the Rössler system, a system of three non-linear ordinary differential equations originally studied by Otto Rössler. These differential equations define a continuous-time dynamical system that exhibits chaotic dynamics associated with the fractal properties of the attractor.

regenerate()
Regenerate :return:

class echotorch.datasets.**LambdaDataset**(*sample_len*, *n_samples*, *func*, *start=0*, *dtype=torch.float32*)
Bases: torch.utils.data.dataset.Dataset

Create simple periodic signal timeseries

class echotorch.datasets.**LogisticMapDataset**(*sample_len*, *n_samples*, *alpha=5*, *beta=11*, *gamma=13*, *c=3.6*, *b=0.13*, *seed=None*)
Bases: torch.utils.data.dataset.Dataset

Logistic Map dataset

class echotorch.datasets.**LorenzAttractor**(*sample_len*, *n_samples*, *xyz*, *sigma*, *b*, *r*, *dt=0.01*, *washout=0*, *normalize=False*, *seed=None*)
Bases: torch.utils.data.dataset.Dataset

The Rössler attractor is the attractor for the Rössler system, a system of three non-linear ordinary differential equations originally studied by Otto Rössler. These differential equations define a continuous-time dynamical system that exhibits chaotic dynamics associated with the fractal properties of the attractor.

regenerate()
Regenerate :return:

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.**RosslerAttractor**(*sample_len*, *n_samples*, *xyz*, *a*, *b*, *c*, *dt=0.01*, *washout=0*, *normalize=False*, *seed=None*)

Bases: torch.utils.data.dataset.Dataset

The Rössler attractor is the attractor for the Rössler system, a system of three non-linear ordinary differential equations originally studied by Otto Rössler. These differential equations define a continuous-time dynamical system that exhibits chaotic dynamics associated with the fractal properties of the attractor.

regenerate()
Regenerate :return:

class echotorch.datasets.**SinusoidalTimeseries**(*sample_len*, *n_samples*, *period*, *a=1.0*, *m=0.0*, *start=1*, *dtype=torch.float64*)

Bases: torch.utils.data.dataset.Dataset

Sinusoidal timeseries

random_initial_points()
Random initial points :return:

regenerate()
Regenerate :return:

class echotorch.datasets.**PeriodicSignalDataset**(*sample_len*, *n_samples*, *period*, *start=1*, *dtype=torch.float64*)

Bases: torch.utils.data.dataset.Dataset

Create simple periodic signal timeseries

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, feedbacks_dim=None, wfdb_sparsity=None, normalize_feedbacks=False, seed=None, w_distrib='uniform', win_distrib='uniform', wbias_distrib='uniform', win_normal=(0.0, 1.0), w_normal=(0.0, 1.0), wbias_normal=(0.0, 1.0), dtype=torch.float32*)

Echo State Network layer

forward(*u*, *y=None*, *w_out=None*, *reset_state=True*)
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_gaussian_matrix(*size*, *sparsity*, *mean=0.0*, *std=1.0*, *dtype=torch.float32*)
Generate gaussian Win matrix :return:

static generate_uniform_matrix(*size*, *sparsity*, *input_set*)
Generate uniform Win matrix :param w_in: :param seed: :return:

static generate_w(*output_dim*, *w_distrib='uniform'*, *w_sparsity=None*, *mean=0.0*, *std=1.0*, *seed=None*, *dtype=torch.float32*)
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:

set_hidden(*x*)
Set hidden layer :param x: :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, wfdb_sparsity=None, normalize_feedbacks=False, softmax_output=False, seed=None, washout=0, w_distrib='uniform', win_distrib='uniform', wbias_distrib='uniform', win_normal=(0.0, 1.0), w_normal=(0.0, 1.0), wbias_normal=(0.0, 1.0), dtype=torch.float32*)
Echo State Network module

finalize()
Finalize training with LU factorization

forward(*u*, *y=None*, *reset_state=True*)
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*, *reset_state=True*)
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, wfdb_sparsity=None, normalize_feedbacks=False, softmax_output=False, seed=None, washout=0, w_distrib='uniform', win_distrib='uniform', wbias_distrib='uniform', win_normal=(0.0, 1.0), w_normal=(0.0, 1.0), wbias_normal=(0.0, 1.0), dtype=torch.float32*)
Leaky-Integrated Echo State Network module

2.1.3 echotorch.tools package

Submodules

echotorch.utils.error_measures module

echotorch.utils.error_measures.**cumperplexity**(*output_probs*, *targets*, *log=False*)
Cumulative perplexity :param output_probs: :param targets: :param log: :return:

echotorch.utils.error_measures.**generalized_squared_cosine**(*Sa*, *Ua*, *Sb*, *Ub*)
Generalized square cosine :param Sa: :param Ua: :param Sb: :param Ub: :return:

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.**perplexity**(*output_probs*, *targets*, *log=False*)
Perplexity :param output_probs: Output probabilities for each word/tokens (length x n_tokens) :param targets: Real word index :return: Perplexity

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.**align_pattern**(*interpolation_rate*, *truth_pattern*, *generated_pattern*)
Align pattern :param interpolation_rate: :param truth_pattern: :param generated_pattern: :return:

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

echotorch.utils.utility_functions.**compute_correlation_matrix**(*states*)
Compute correlation matrix :param states: :return:

echotorch.utils.utility_functions.**compute_similarity_matrix**(*svd_list*)
Compute similarity matrix :param svd_list: :return:

echotorch.utils.utility_functions.**compute_singular_values**(*stats*)
Compute singular values :param states: :return:

echotorch.utils.utility_functions.**deep_spectral_radius**(*m*, *leaky_rate*)
Compute spectral radius of a square 2-D tensor for stacked-ESN :param m: squared 2D tensor :param leaky_rate: Layer's leaky rate :return:

echotorch.utils.utility_functions.**find_phase_shift**(*p*, *y*, *interpolation_rate*, *error_measure=<function nrmse>*)
Find phase shift :param s1: :param s2: :param window_size: :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.**align_pattern**(*interpolation_rate*, *truth_pattern*, *generated_pattern*)
Align pattern :param interpolation_rate: :param truth_pattern: :param generated_pattern: :return:

echotorch.utils.**compute_correlation_matrix**(*states*)
Compute correlation matrix :param states: :return:

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.**perplexity**(*output_probs*, *targets*, *log=False*)
Perplexity :param output_probs: Output probabilities for each word/tokens (length x n_tokens) :param targets: Real word index :return: Perplexity

echotorch.utils.**cumperplexity**(*output_probs*, *targets*, *log=False*)
Cumulative perplexity :param output_probs: :param targets: :param log: :return:

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

echotorch.utils.**deep_spectral_radius**(*m*, *leaky_rate*)
Compute spectral radius of a square 2-D tensor for stacked-ESN :param m: squared 2D tensor :param leaky_rate: Layer's leaky rate :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:

echotorch.utils.**show_3d_timeseries**(*ts*, *title*)
Show 3D timeseries :param axis: :param title: :return:

echotorch.utils.**show_2d_timeseries**(*ts*, *title*)
Show 2D timeseries :param ts: :param title: :return:

echotorch.utils.**show_1d_timeseries**(*ts*, *title*, *xmin*, *xmax*, *ymin*, *ymax*, *start=0*, *timesteps=-1*)
Show 1D time series :param ts: :param title: :return:

echotorch.utils.**neurons_activities_1d**(*stats*, *neurons*, *title*, *colors*, *xmin*, *xmax*, *ymin*, *ymax*, *timesteps=-1*, *start=0*)
Display neurons activities :param stats: :param neurons: :return:

echotorch.utils.**neurons_activities_2d**(*stats*, *neurons*, *title*, *colors*, *timesteps=-1*, *start=0*)
Display neurons activities on a 2D plot :param stats: :param neurons: :param title: :param timesteps: :param start: :return:

echotorch.utils.**neurons_activities_3d**(*stats*, *neurons*, *title*, *timesteps=-1*, *start=0*)
Display neurons activities on a 3D plot :param stats: :param neurons: :param title: :param timesteps: :param start: :return:

echotorch.utils.**plot_singular_values**(*stats*, *title*, *xmin*, *xmax*, *ymin*, *ymax*, *log=False*)
Plot singular values :param stats: :param title: :param timestep: :param start: :return:

echotorch.utils.**compute_singular_values**(*stats*)
Compute singular values :param states: :return:

echotorch.utils.**generalized_squared_cosine**(*Sa*, *Ua*, *Sb*, *Ub*)
Generalized square cosine :param Sa: :param Ua: :param Sb: :param Ub: :return:

echotorch.utils.**compute_similarity_matrix**(*svd_list*)
Compute similarity matrix :param svd_list: :return:

echotorch.utils.**show_similarity_matrix**(*sim_matrix*, *title*, *column_labels=None*, *row_labels=None*)
Show similarity matrix :param sim_matrix: :return:

echotorch.utils.**show_conceptors_similarity_matrix**(*conceptors*, *title*)
Show conceptors similarity matrix :param conceptors: :param title: :return:

echotorch.utils.**show_sv_for_increasing_aperture**(*conceptor*, *factor*, *title*)
Show singular values for increasing aperture :param conceptors: :param factor: :param title: :return:

echotorch.utils.**find_phase_shift**(*p*, *y*, *interpolation_rate*, *error_measure=<function nrmse>*)
Find phase shift :param s1: :param s2: :param window_size: :return:

2.2 Module contents

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

e

echotorch, 12
echotorch.datasets, 6
echotorch.datasets.MackeyGlassDataset, 5
echotorch.datasets.MemTestDataset, 5
echotorch.datasets.NARMADataset, 5
echotorch.utils, 10
echotorch.utils.error_measures, 9
echotorch.utils.utility_functions, 10

t

torch.nn, 7

Index

A

align_pattern() (*in module echotorch.utils*), 10
align_pattern() (*in module echotorch.utils.utility_functions*), 10
average_prob() (*in module echotorch.utils*), 11
average_prob() (*in module echotorch.utils.utility_functions*), 10

C

compute_correlation_matrix() (*in module echotorch.utils*), 10
compute_correlation_matrix() (*in module echotorch.utils.utility_functions*), 10
compute_similarity_matrix() (*in module echotorch.utils*), 11
compute_similarity_matrix() (*in module echotorch.utils.utility_functions*), 10
compute_singular_values() (*in module echotorch.utils*), 11
compute_singular_values() (*in module echotorch.utils.utility_functions*), 10
cumperplexity() (*in module echotorch.utils*), 11
cumperplexity() (*in module echotorch.utils.error_measures*), 9

D

DatasetComposer (*class in echotorch.datasets*), 6
deep_spectral_radius() (*in module echotorch.utils*), 11
deep_spectral_radius() (*in module echotorch.utils.utility_functions*), 10

E

echotorch (*module*), 12
echotorch.datasets (*module*), 6
echotorch.datasets.MackeyGlassDataset (*module*), 5
echotorch.datasets.MemTestDataset (*module*), 5
echotorch.datasets.NARMADataset (*module*), 5
echotorch.utils (*module*), 10
echotorch.utils.error_measures (*module*), 9
echotorch.utils.utility_functions (*module*), 10
ESN (*class in nn*), 8
ESNCell (*class in nn*), 7

F

finalize() (*nn.ESN method*), 8
find_phase_shift() (*in module echotorch.utils*), 12
find_phase_shift() (*in module echotorch.utils.utility_functions*), 10
forward() (*nn.ESN method*), 8
forward() (*nn.ESNCell method*), 7
forward() (*nn.LiESNCell method*), 9

G

generalized_squared_cosine() (*in module echotorch.utils*), 11
generalized_squared_cosine() (*in module echotorch.utils.error_measures*), 9
generate_gaussian_matrix() (*nn.ESNCell static method*), 7
generate_uniform_matrix() (*nn.ESNCell static method*), 7
generate_w() (*nn.ESNCell static method*), 7
get_spectral_radius() (*nn.ESN method*), 8
get_spectral_radius() (*nn.ESNCell method*), 8
get_w_out() (*nn.ESN method*), 8

H

HenonAttractor (*class in echotorch.datasets*), 6
hidden (*nn.ESN attribute*), 8

I

init_hidden() (*nn.ESNCell method*), 8

L

LambdaDataset (*class in echotorch.datasets*), 6
LiESN (*class in nn*), 9
LiESNCell (*class in nn*), 9
LogisticMapDataset (*class in echotorch.datasets*), 6
LorenzAttractor (*class in echotorch.datasets*), 6

M

MackeyGlassDataset (*class in echotorch.datasets*), 6
MackeyGlassDataset (*class in echotorch.datasets.MackeyGlassDataset*), 5
max_average_through_time() (*in module echotorch.utils*), 11
max_average_through_time() (*in module echotorch.utils.utility_functions*), 10
MemTestDataset (*class in echotorch.datasets*), 6
MemTestDataset (*class in echotorch.datasets.MemTestDataset*), 5
mse() (*in module echotorch.utils*), 10
mse() (*in module echotorch.utils.error_measures*), 9

N

NARMADataset (*class in echotorch.datasets*), 6
NARMADataset (*class in echotorch.datasets.NARMADataset*), 5
neurons_activities_1d() (*in module echotorch.utils*), 11
neurons_activities_2d() (*in module echotorch.utils*), 11
neurons_activities_3d() (*in module echotorch.utils*), 11
nmse() (*in module echotorch.utils*), 10
nmse() (*in module echotorch.utils.error_measures*), 9
normalize() (*in module echotorch.utils*), 11
normalize() (*in module echotorch.utils.utility_functions*), 10
nrmse() (*in module echotorch.utils*), 10
nrmse() (*in module echotorch.utils.error_measures*), 9

P

PeriodicSignalDataset (*class in echotorch.datasets*), 7
perplexity() (*in module echotorch.utils*), 10
perplexity() (*in module echotorch.utils.error_measures*), 9
plot_singular_values() (*in module echotorch.utils*), 11

R

random_initial_points() (*echotorch.datasets.SinusoidalTimeseries method*), 7
regenerate() (*echotorch.datasets.HenonAttractor method*), 6
regenerate() (*echotorch.datasets.LorenzAttractor method*), 6
regenerate() (*echotorch.datasets.RosslerAttractor method*), 7
regenerate() (*echotorch.datasets.SinusoidalTimeseries method*), 7
reset() (*nn.ESN method*), 8
reset_hidden() (*nn.ESN method*), 8
reset_hidden() (*nn.ESNCell method*), 8
rmse() (*in module echotorch.utils*), 10
rmse() (*in module echotorch.utils.error_measures*), 9
RosslerAttractor (*class in echotorch.datasets*), 7

S

set_hidden() (*nn.ESNCell method*), 8
set_w() (*nn.ESN method*), 8
show_1d_timeseries() (*in module echotorch.utils*), 11
show_2d_timeseries() (*in module echotorch.utils*), 11
show_3d_timeseries() (*in module echotorch.utils*), 11
show_conceptors_similarity_matrix() (*in module echotorch.utils*), 11
show_similarity_matrix() (*in module echotorch.utils*), 11
show_sv_for_increasing_aperture() (*in module echotorch.utils*), 11
SinusoidalTimeseries (*class in echotorch.datasets*), 7
spectral_radius() (*in module echotorch.utils*), 11
spectral_radius() (*in module echotorch.utils.utility_functions*), 10

T

to_sparse() (*nn.ESNCell static method*), 8
torch.nn (*module*), 7

W

w (*nn.ESN attribute*), 8
w_in (*nn.ESN attribute*), 8