deepSIP

Release 0.1.dev0

Jun 11, 2020

Contents

deepSIP (deep learning of Supernova Ia Parameters) is an open-source toolkit for measuring the phase and light-curve shape (parameterized by SNooPy's Δm_{15}) of a Type Ia Supernova (SN Ia) from an optical spectrum. The primary contents of the package are a set of three trained Convolutional Neural Networks (CNNs) for the aforementioned purposes, but tools for preprocessing spectra, modifying the neural architecture, training models, and sweeping through hyperparameters are also included.

The entire code base is available on GitHub: https://github.com/benstahl92/deepSIP

If you use deepSIP in your research, please cite the following paper:

CHAPTER 1

Installation

First, you'll need to clone deepSIP and enter its directory.

```
git clone https://github.com/benstahl92/deepSIP.git
cd deepSIP
```

(optional) It is recommended that you use a virtual environment for deepSIP and its dependencies.

```
python -m venv dsenv # create virtual environment (one time only)
source dsenv/bin/active # each time you need to activate the environment
deactivate # if/when you need to leave the environment
```

Install dependencies and deepSIP.

```
pip install -r requirements.txt
pip install .
```

CHAPTER 2

Standard Usage

```
from deepSIP import deepSIP
ds = deepSIP()
# spectra is a pd.DataFrame with columns including ['SN', 'filename', 'z']
predictions = ds.predict(spectra, threshold = 0.5, status = True)
```

Placeholder for example section on GitHub

CHAPTER 3

Full Documentation

Beyond standard usage, there may be occasions to use the underlying toolkit provided by deepSIP. We therefore provide full documentation of its capabilities below.

3.1 deepSIP

class deepSIP.model.**deepSIP**(*spec_len=1024*, *seed=100*, *drop_rate=0.02*)

Bases: object

class for deploying trained deepSIP models

Parameters

spec_len [int, optional] number of wavelength bins for pre-processed spectra (must match what was used in training models)

seed [int, optional] seed for random number generator

Attributes

models [pd.DataFrame] models, along with metadata and utilities, indexed by purpose

device [torch.device] device type being used (GPU if available, else CPU)

Methods

predict(self, spectra[, threshold, mcnum, ...])	make predictions with trained models

predict(*self*, *spectra*, *threshold=0.9*, *mcnum=30*, *status=False*)

make predictions with trained models

Parameters

spectra [np.ndarray or pd.DataFrame] pre-preocessed spectra if np.ndarray else

pd.DataFrame with columns of [SN, filename, z] and optionally obsframe as bool

threshold [float, optional] minimum threshold for 'in' classification by Domain model

mcnum [int, optional] number of stochastic forward passes to perform

status [bool, optional] show status bars

Returns

pd.DataFrame predictions generated by each model

3.2 deepSIP.preprocessing

class deepSIP.preprocessing.**Spectrum**(*filename*, *z*, *obsframe=True*)

Bases: object

container for individual spectra

Parameters

filename [str] name of text file to read spectrum from

z [float] redshift of SN

obsframe [bool, optional] indicates spectrum is in observer frame (and thus needs to be de-redshifted)

Attributes

signal_window_angstroms [int] window size in angstroms for signal smoothing

signal_smoothing_order [int] polynomial order for signal smoothing

continuum_window_angstroms [int] window size in angstroms for continuum smoothing

continuum_smoothing_order [int] polynomial order for continuum smoothing

lwave_bounds [tuple, list, or other iterable of length 2] lower and upper limit of logarithmic wavelength array

lwave_n_bins [int] number of bins in logarithmic wavelength grid

lwave [np.array] logarithmic wavelength grid

apodize_end_pct [float] percentage from each end of flux array to apodize

aug_drop_frac [float] maximum percentage of each of flux array to drop during augmentation

wave [np.array] rest frame wavelength grid

flux [np.array] fluxes on wavelength grid

Methods

SNID(self, **kwargs)	run SNID on spectrum using pySNID (if available)
augprocess(self, augz, sig_wA)	process spectrum with augmentation
plot(self[, show])	plot original or processed spectrum for quick inspection
process(self)	process spectrum with no augmentation

process(*self*)
process spectrum with no augmentation

complete pre-processing: smoothes spectrum, identifies and subtracts pseudo-continuum, does log binning, scales flux and apodizes edges

Returns

np.array fully pre-processed flux array

augprocess(*self*, *augz*, *sig_wA*)
process spectrum with augmentation

perturbs wavelength by multiplicative (1 + augz), drops ends randomly up to a maximum of aug_drop_frac, modify signal_window_angstroms to sig_wA, and then perform pre-processing according to process method

Returns

np.array augmented and fully pre-processed flux array

plot(*self*, *show='processed'*)
plot original or processed spectrum for quick inspection

Parameters

show [str, optional] type of plot to show ('processed' or 'original')

SNID(*self*, ***kwargs*)
run SNID on spectrum using pySNID (if available)

Parameters

****kwargs** arbitrary keyword arguments for pySNID

Returns

pySNID outputs

class deepSIP.preprocessing.**EvaluationSpectra**(*spectra*, *savefile='eval.spectra.sav'*)
Bases: object

class for preparing spectra for evaluation

Parameters

spectra [pd.DataFrame] spectra to prepare for evaluation; must have columns columns of [SN, filename, z] and optionally obsframe as bool

savefile [str, optional] name of save file

Attributes

X [np.ndarray with dimensions (number of spectra, lwave_n_bins)] processed spectra (attribute set by process method)

SNID_results [pd.DataFrame] pySNID results for each spectrum (attribute set by SNID method)

Methods

SNID(self[, selection, status])	run SNID on selected (via boolean array) spectra in dataset

Continued on next page

Table 3 – continued from previous page

SNID_to_csv(self)	write SNID results to csv file
load(self)	load from savefile
process(self[, status])	process all loaded spectra with no augmentation
save(self)	save current state to savefile
to_npy(self)	write processed spectra to .npy file

save(*self*)
save current state to savefile

load(*self*)
load from savefile

process(*self*, *status=True*)
process all loaded spectra with no augmentation

Parameters

status [bool, optional] show status bars

SNID(*self*, *selection='all'*, *status=True*, ***kwargs*)
run SNID on selected (via boolean array) spectra in dataset

Parameters

selection [boolean array] spectra from data set to run SNID on via pySNID

status [bool, optional] show status bars

****kwargs** arbitrary keyword arguments for pySNID

SNID_to_csv(*self*)
write SNID results to csv file

to_npy(*self*)
write processed spectra to .npy file

class deepSIP.preprocessing.**TVTSpectra**(*spectra*, *savefile='tvt.spectra.sav'*, *prep=1*, *val_frac=0.1*, *test_frac=0.1*, *phase_bounds=(-10, 18)*, *dm15_bounds=(0.85, 1.55)*, *phase_binsize=4*, *dm15_binsize=0.1*, *aug_num=5000*, *aug_z_bounds=(-0.004, 0.004)*, *aug_signal_window_angstroms=(50, 150)*, *random_state=100*)
Bases: *deepSIP.preprocessing.EvaluationSpectra*

class for preparing Training, Validation, and Testing sets

Parameters

spectra [pd.DataFrame] spectra to prepare; must have columns columns of [SN, filename, z] and optionally obsframe as bool

savefile [str, optional] name of save file

prep [int, optional] preparation mode (1 for all spectra, 2 for domain-restricted subset)

val_frac [float, optional] fraction of full set to split for validation

test_frac [float, optional] fraction of full set to split for testing

phase_bounds [tuple, list, or other iterable of length 2, optional] lower and upper phase limits of domain

dm15_bounds [tuple, list, or other iterable of length 2, optional] lower and upper dm15 limits of domain

Other Parameters

phase_binsize [int or float, optional] phase bin size for pseudo-stratified splitting

dm15_binsize [int or float, optional] dm15 bin size for pseudo-stratified splitting

aug_num [int, optional] final size of training set after augmentation

aug_z_bounds [tuple, list, or other iterable of length 2, optional] lower and upper limits of randomly selected redshifts for augmented spectra

aug_signal_window_angstroms [tuple, list, or other iterable of length 2] lower and upper limits of randomly selected signal windows for augmented spectra

random_state [int, optional] seed for random number generator

Attributes

spectra_[out,in] [pd.DataFrame] subset of spectra that are [out,in] selected domain

spectra_aug [pd.DataFrame] augmented spectra

Ycol [list] columns in spectra that correspond to labels

[train,aug,val,test]X [np.ndarray] processed spectra (attribute set by [aug]process method)

[train,aug,val,test]Y [np.ndarray] targets (attribute set by [aug]process method)

Methods

SNID(self[, selection, status])	run SNID on selected (via boolean array) spectra in dataset
SNID_to_csv(self)	write SNID results to csv file
augprocess(self[, status])	process all loaded spectra with augmentation
load(self)	load from savefile
process(self[, status])	process all loaded spectra with no augmentation
save(self)	save current state to savefile
split(self[, force, method])	split spectra into train/val/test subsets
to_npy(self)	write processed spectra and targets to .npy files

split (*self*, *force=False*, *method='stratified'*)
split spectra into train/val/test subsets

two splitting methods are available:

1. **'stratified' - split in-domain spectra in pseudo-stratified** fashion by randomly selected subsets from bins
2. 'randomized' - random selection

Parameters

force [bool, optional] overwrite pre-existing splits if True

method [str, optional] splitting method ('stratified' or 'randomized')

process (*self*, *status=True*)
process all loaded spectra with no augmentation

Parameters

status [bool, optional] show status bars

augprocess(*self*, *status=True*)
process all loaded spectra with augmentation

Parameters

status [bool, optional] show status bars

to_npy(*self*)
write processed spectra and targets to .npy files

3.3 deepSIP.dataset

class deepSIP.dataset.**NumpyXDataset**(*X*, *device='auto'*)
Bases: torch.utils.data.dataset.Dataset

create torch tensor dataset from X numpy ndarray

Parameters

X [np.ndarray of shape (number of spectra, number of wavelength bins)] pre-processed spectra

device [str, optional] device to push tensors to, defaults to GPU if available

Attributes

X [torch.tensor of shape (number of spectra, 1, number of wavelength bins)]

class deepSIP.dataset.**NumpyXYDataset**(*X*, *Y*, ***kwargs*)
Bases: *deepSIP.dataset.NumpyXDataset*

create torch tensor dataset from X, Y numpy ndarrays

Parameters

X [np.ndarray of shape (number of spectra, number of wavelength bins)] pre-processed spectra

Y [np.ndarray of shape (number of spectra, number of targets)] targets

****kwargs** arbitrary keyword arguments for NumpyXDataset constructor

Attributes

X [torch.tensor of shape (number of spectra, 1, number of wavelength bins)]

Y [torch.tensor of shape (number of spectra, number of targets)]

3.4 deepSIP.architecture

class deepSIP.architecture.**DropoutCNN**(*spec_len*, *kernel=15*, *filters=8*, *fc_size=32*, *drop_rate=0.1*)
Bases: torch.nn.modules.module.Module

core model architecture

4 x (Conv + ReLU + Max Pooling + Dropout) + (Linear + ReLU + Dropout) + Linear

Parameters

spec_len [int,] number of wavelength bins for pre-processed spectra

kernel [odd int, optional] convolutional kernel size

filters [int, optional] number of filters in first convolution layer

fc_size [int, optional] number of neurons in fully connected layer

drop_rate [float, optional] dropout probability

Attributes

conv[1-4] [torch.nn.Sequential] convolution block [1-4]

fc [torch.nn.Sequential] fully connected block

out [toch.nn.Sequential] output block

Methods

`__call__`(self, *input, **kwargs)	Call self as a function.
`add_module`(self, name, module)	Adds a child module to the current module.
`apply`(self, fn)	Applies `fn` recursively to every submodule (as returned by `.children()`) as well as self.
`buffers`(self[, recurse])	Returns an iterator over module buffers.
`children`(self)	Returns an iterator over immediate children modules.
`cpu`(self)	Moves all model parameters and buffers to the CPU.
`cuda`(self[, device])	Moves all model parameters and buffers to the GPU.
`double`(self)	Casts all floating point parameters and buffers to `double` datatype.
`eval`(self)	Sets the module in evaluation mode.
`extra_repr`(self)	Set the extra representation of the module
`float`(self)	Casts all floating point parameters and buffers to float datatype.
`forward`(self, x)	forward pass
`half`(self)	Casts all floating point parameters and buffers to `half` datatype.
`load_state_dict`(self, state_dict[, strict])	Copies parameters and buffers from `state_dict` into this module and its descendants.
`modules`(self)	Returns an iterator over all modules in the network.
`named_buffers`(self[, prefix, recurse])	Returns an iterator over module buffers, yielding both the name of the buffer as well as the buffer itself.
`named_children`(self)	Returns an iterator over immediate children modules, yielding both the name of the module as well as the module itself.
`named_modules`(self[, memo, prefix])	Returns an iterator over all modules in the network, yielding both the name of the module as well as the module itself.
`named_parameters`(self[, prefix, recurse])	Returns an iterator over module parameters, yielding both the name of the parameter as well as the parameter itself.
`parameters`(self[, recurse])	Returns an iterator over module parameters.
`register_backward_hook`(self, hook)	Registers a backward hook on the module.
`register_buffer`(self, name, tensor)	Adds a persistent buffer to the module.
`register_forward_hook`(self, hook)	Registers a forward hook on the module.

Continued on next page

Table 5 – continued from previous page

register_forward_pre_hook(self, hook)	Registers a forward pre-hook on the module.
register_parameter(self, name, param)	Adds a parameter to the module.
state_dict(self[, destination, prefix, ...])	Returns a dictionary containing a whole state of the module.
to(self, *args, **kwargs)	Moves and/or casts the parameters and buffers.
train(self[, mode])	Sets the module in training mode.
type(self, dst_type)	Casts all parameters and buffers to dst_type.
zero_grad(self)	Sets gradients of all model parameters to zero.

share_memory	

forward(*self*, *x*)
forward pass

Parameters

x [torch.tensor of shape (batch size, 1, number of wavelength bins)] inputs to the network

3.5 deepSIP.training

class deepSIP.training.**Train**(*trainX, trainY, valX, valY, testX=None, testY=None, Ylim=[0.0, 1.0], seed=100, threshold=0.5, regression=True, mcnum=75, kernel=15, filters=16, fc_size=32, drop_rate=0.1, epochs=75, early_stop=[0.0], lr_decay_steps=[45, 60, 70], lr=0.001, batch_size=16, weight_decay=0.0001, verbose=True, wandb=None, save=True, savedir='./'*)

Bases: object

network training

Parameters

train[X,Y] [np.ndarray] training [inputs,outputs]

val[X,Y] [np.ndarray] validation [inputs,outputs]

test[X,Y] [np.ndarray, optional] testing [inputs,outputs]

Ylim [tuple, list, or other iterable of length 2, optional] lower and upper limits for for utils.LinearScaler on outputs (Y)

seed [int, optional] seed for random number generator

threshold [float, optional] minimum threshold for 'in' classification by Domain model

regression [bool, optional] toggle for regression (determines scalers used)

mcnum [int, optional] number of stochastic forward passes to perform

kernel [odd int, optional] convolutional kernel size

filters [int, optional] number of filters in first convolution layer

fc_size [int, optional] number of neurons in fully connected layer

drop_rate [float, optional] dropout probability

epochs [int, optional] number of training epochs

lr [float, optional] initial learning rate

batch_size [int, optional] batch size for training

weight_decay [float, optional] weight decay for training

verbose [bool, optional] show network summary and status bars

save [bool, optional] flag for saving training history and trained model

savedir [str, optional] directory for save files

Other Parameters

early_stop [length-1 array_like, optional] early stopping threshold on validation RMSE for regression mode

lr_decay_steps [array_like, optional] epochs at which to decay learning rate by factor 10

wandb [wandb instance, optional] wandb instance for run tracking

Attributes

device [torch.device] device type being used (GPU if available, else CPU)

network [DropoutCNN] network to train (may be wrapped in DataParallel if on GPU)

Yscaler [VoidScaler or LinearScaler] scaler for Y labels

optimizer [torch optimizer (Adam)] optimizer for training

scheduler [VoidLRScheduler or MultiStepLR] learning rate scheduler

loss [torch loss] loss for training

Methods

test_epoch(self, X, Y[, label])	perform validation or testing steps for single epoch
train(self)	train network
train_epoch(self)	perform training steps for single epoch

train_epoch(*self*)
perform training steps for single epoch

Returns

dict training metrics for epoch (loss and lr-current)

test_epoch(*self*, *X*, *Y*, *label=''*)
perform validation or testing steps for single epoch

Parameters

X [torch.tensor] inputs

Y [torch.tensor] outputs

label [str, optional] label to prepend output dict keys with (e.g. 'val' or 'test')

Returns

metrics [dict] validation or testing metrics for epoch

train(*self*)
train network

class deepSIP.training.**Sweep**(*trainX, trainY, valX, valY, entity, project, kernels, filters, fc_sizes, drop_rates, batch_sizes, lrs, weight_decays, seed=100, regression=True, mcnum=75, epochs=75, early_stop=[0.0], Ylim=[0.0, 1.0], sweep_method='random', testX=None, testY=None*)

Bases: object

sweep (search) through hyperparameters using wandb

Parameters

train[X,Y] [np.ndarray] training [inputs,outputs]

val[X,Y] [np.ndarray] validation [inputs,outputs]

entity [str] wandb entity

project [str] wandb project

kernels [array_like] convolutional kernel sizes to sweep over

filters [array_like] numbers of filters in first convolution layer to sweep over

fc_sizes [array_like] numbers of neurons in fully connected layer to sweep over

drop_rates [array_like] dropout probabilities to sweep over

batch_sizes [array_like] batch sizes to sweep over

lrs [array_like] initial learning rates to sweep over

weight_decays [array_like] weight decays to sweep over

seed [int, optional] seed for random number generator

regression [bool, optional] toggle for regression (determines scalers used)

mcnum [int, optional] number of stochastic forward passes to perform

epochs [int, optional] number of training epochs

Ylim [tuple, list, or other iterable of length 2, optional] lower and upper limits for for utils.LinearScaler on outputs (Y)

sweep_method [str, optional] method for sweep ('random' or 'grid')

test[X,Y] [np.ndarray, optional] testing [inputs,outputs]

Other Parameters

early_stop [length-1 array_like, optional] early stopping threshold on validation RMSE for regression mode

Attributes

sweep_config [dict] wandb sweep configurations

Methods

sweep(self[, tags, saveroot])	run sweep

sweep(*self*, *tags=[]*, *saveroot=None*)

run sweep

Parameters

tags [list, optional] list of strings to add as tags to sweep runs

saveroot [str, optional] root name to use for saving

3.6 deepSIP.utils

deepSIP.utils.**load_txt_spectrum**(*filename*)
load spectrum from txt file with first two columns as wave flux

Parameters

filename [str] name of text file to read spectrum from

Returns

wave [np.array] wavelength grid

flux [np.array] fluxes on wavelength grid

deepSIP.utils.**savenet**(*network*, *filename*)
save network parameters

Parameters

network [DropoutCNN or torch network] network to save

filename [str] name of file to save network to

deepSIP.utils.**loadnet**(*network*, *filename*, *device='cpu'*)
load network parameters

Parameters

network [DropoutCNN] network to load weights and biases into

filename [str] name of file to load network from

device [str, optional] device to load network onto ('cpu' by default)

deepSIP.utils.**deredshift**(*wave*, *z*)
transform wavelength to rest frame

Parameters

wave [np.array] wavelength grid

z [float] redshift of SN

Returns

np.array rest-frame wavelength grid

deepSIP.utils.**smooth**(*flux*, *window*, *order*)
smooth using a savitzky-golay filter

Parameters

flux [np.array] fluxes

window [int] window size in angstroms for smoothing

smoothing [int] polynomial order for smoothing

Returns

np.array smoothed spectrum

deepSIP.utils.**get_continuum**(*flux*, *window*, *order*)
model continuum from *heavily* smoothed spectrum

Notes

see documentiation for smooth, which this function simply wraps

deepSIP.utils.**log_wave**(*bounds*, *n_bins*)
compute logarithmic wavelength grid

Parameters

bounds [tuple, list, or other iterable of length 2] lower and upper limit of logarithmic wavelength array

n_bins [int] number of bins in logarithmic wavelength grid

Returns

np.array logarithmic wavelength grid

deepSIP.utils.**log_bin**(*wave*, *flux*, *bounds*, *n_bins*)
rebin onto logarithmic wavelength grid (adapted from astrodash/SNID)

Parameters

wave [np.array] wavelength grid

flux [np.array] fluxes on wavelength grid

bounds [tuple, list, or other iterable of length 2] lower and upper limit of logarithmic wavelength array

n_bins [int] number of bins in logarithmic wavelength grid

Returns

lwave [np.array] logarithmic wavelength grid

lflux [np.array] rebinned fluxes on grid

valid_mask [boolean array] selection mask for signal on grid

deepSIP.utils.**normalize_flux**(*flux*)
normalize flux range of 1 centered at 0

Parameters

flux [np.array] fluxes

Returns

np.array normalized flux

deepSIP.utils.**apodize**(*flux*, *end_pct*)
taper spectrum at each end to zero using hanning window

Parameters

flux [np.array] fluxes

end_pct [float] percentage to taper at each end

Returns

np.array apodized flux

deepSIP.utils.**redshift**(*wave*, *z*)
transform wavelength to observer frame

Parameters

wave [np.array] rest wavelength grid

z [float] redshift of SN relative to observer

Returns

np.array observer-frame wavelength grid

deepSIP.utils.**drop_ends**(*wave*, *flux*, *max_drop_frac*)
drop random fraction of spectrum from ends

Parameters

wave [np.array] wavelength grid

flux [np.array] fluxes on wavelength grid

max_drop_frac [float] maximum fraction to be dropped from each end

Returns

np.array, np.array truncated wave and flux arrays

deepSIP.utils.**reset_state**(*seed=100*)
set all seeds for reproducibility

Parameters

seed [int, optional] seed for random number generator

Notes

seeds are set for torch (including cuda), numpy, random, and PHYTHONHASH; this appears to be sufficient to ensure reproducibility

class deepSIP.utils.**VoidScaler**(**args*, ***kwargs*)
Bases: object

sklearn conforming scaler that has no effect (i.e. identity transformation)

Parameters

***args** arbitrary arguments with no effect

****kwargs** arbitrary keyword arguments with no effect

Methods

fit	
fit_transform	
inverse_transform	
transform	

fit(*self*, *X*)

transform(*self*, *X*)

inverse_transform(*self*, *X*)

fit_transform(*self*, *X*)

class deepSIP.utils.**LinearScaler**(*minimum=0.0*, *maximum=1.0*)
Bases: *deepSIP.utils.VoidScaler*

sklearn conforming that behaves as MinMaxScaler but min and max are fixed

Parameters

[min,max]imum [float or np.array] [min,max]imum value(s) used for transformation

Attributes

[min,max] [float or np.array] see Parameters

Methods

fit	
fit_transform	
inverse_transform	
transform	

transform(*self*, *X*)

inverse_transform(*self*, *X*)

deepSIP.utils.**torch2numpy**(**tensors*)
convert input torch tensors to numpy ndarrays

Parameters

***tensors** torch tensors to convert to corresponding numpy ndarrays

Returns

ndarrays: np.ndarrays numpy ndarrays corresponding to input torch tensors

deepSIP.utils.**count_params**(*network*)
count trainable parameters in network

Parameters

network [torch network] network to count trainable parameters for

Returns

int number of trainable parameters in input network

deepSIP.utils.**init_weights**(*network_component*)
initialize weights (normal) and biases (slight positive)

Parameters

network_component [component torch network] component to initialize

deepSIP.utils.**stochastic_predict**(*network*, *X*, *mcnum=75*, *sigmoid=False*, *seed=None*, *scaler=None*, *status=False*)
perform mcnum stochastic forward passes, return mean and std np.arrays

Parameters

network [torch network] network to generate predictions with

X [torch tensor] inputs to network

mcnum [int, optional] number of stochastic forward passes to perform

sigmoid [bool, optional] apply sigmoid to outputs

seed [int, optional] seed for random number generator

scaler [sklearn conforming scaler] scaler to inverse transform outputs with

status [bool, optional] show status bars

Returns

mean [np.ndarray] mean prediction per input

std [np.ndarray] std of predictions per input

Notes

Does mcnum forward passes on each row of input, one at a time, with re-seeding between each row. This ensures that predictions on a given input will be the same regardless of the order of inputs. It is possible the algorithm could be optimized for speed, but in tests it performs comparably to the previous implementation that suffered from modest reproducibility issues.

class deepSIP.utils.**WrappedModel**(*network*)

Bases: torch.nn.modules.module.Module

wrapper to seamlessly load network on cpu from dicts saved with DataParallel

Parameters

network [torch network] network to wrap

Attributes

module [torch network] see Parameters

Methods

__call__(self, *input, **kwargs)	Call self as a function.
add_module(self, name, module)	Adds a child module to the current module.
apply(self, fn)	Applies fn recursively to every submodule (as returned by .children()) as well as self.
buffers(self[, recurse])	Returns an iterator over module buffers.
children(self)	Returns an iterator over immediate children modules.
cpu(self)	Moves all model parameters and buffers to the CPU.
cuda(self[, device])	Moves all model parameters and buffers to the GPU.
double(self)	Casts all floating point parameters and buffers to double datatype.
eval(self)	Sets the module in evaluation mode.
extra_repr(self)	Set the extra representation of the module
float(self)	Casts all floating point parameters and buffers to float datatype.
forward(self, *x)	pass arguments to forward of module
half(self)	Casts all floating point parameters and buffers to half datatype.

Continued on next page

Table 8 – continued from previous page

load_state_dict(self, state_dict[, strict])	Copies parameters and buffers from state_dict into this module and its descendants.
modules(self)	Returns an iterator over all modules in the network.
named_buffers(self[, prefix, recurse])	Returns an iterator over module buffers, yielding both the name of the buffer as well as the buffer itself.
named_children(self)	Returns an iterator over immediate children modules, yielding both the name of the module as well as the module itself.
named_modules(self[, memo, prefix])	Returns an iterator over all modules in the network, yielding both the name of the module as well as the module itself.
named_parameters(self[, prefix, recurse])	Returns an iterator over module parameters, yielding both the name of the parameter as well as the parameter itself.
parameters(self[, recurse])	Returns an iterator over module parameters.
register_backward_hook(self, hook)	Registers a backward hook on the module.
register_buffer(self, name, tensor)	Adds a persistent buffer to the module.
register_forward_hook(self, hook)	Registers a forward hook on the module.
register_forward_pre_hook(self, hook)	Registers a forward pre-hook on the module.
register_parameter(self, name, param)	Adds a parameter to the module.
state_dict(self[, destination, prefix, ...])	Returns a dictionary containing a whole state of the module.
to(self, *args, **kwargs)	Moves and/or casts the parameters and buffers.
train(self[, mode])	Sets the module in training mode.
type(self, dst_type)	Casts all parameters and buffers to dst_type.
zero_grad(self)	Sets gradients of all model parameters to zero.

share_memory

forward(*self*, **x*)
pass arguments to forward of module

class deepSIP.utils.**VoidLRScheduler**
Bases: object

torch conforming LR scheduler with no effect

Methods

step

step(*self*, **args*)

deepSIP.utils.**rmse**(*y*, *yhat*)
root-mean-square error

Parameters

y [np.ndarray] true labels for each unique target

yhat [np.ndarray] predictions for each unique target

Returns

rmse [np.ndarray] root-mean-square error for each unique target

deepSIP.utils.**wrmse**(*y*, *yhat*, *sigma_hat*)
inverse predicted variance weighted root-mean-square error

Parameters

y [np.ndarray] true labels for each unique target

yhat [np.ndarray] predictions for each unique target

sigma_hat [np.ndarray] predicted uncertainties for each unique target

Returns

wrmse [np.ndarray] weighted root-mean-square error for each unique target

deepSIP.utils.**outlier**(*y*, *yhat*)
largest absolute residual

Parameters

y [np.ndarray] true labels for each unique target

yhat [np.ndarray] predictions for each unique target

Returns

mr [np.ndarray] maximum absolute residual for each unique target

deepSIP.utils.**slope**(*y*, *yhat*)
slope of linear fit to yhat as a function of y

Parameters

y [np.array] true labels

yhat [np.array] predictions

Returns

slope [float] slope of fitted line to yhat vs y

deepSIP.utils.**regression_metrics**(*y*, *yhat*, *sigma_hat*, *key_prepend=''*)
compute all regression-related metrics

Parameters

y [np.ndarray] true labels for each unique target

yhat [np.ndarray] predictions for each unique target

sigma_hat [np.ndarray] predicted uncertainties for each unique target

key_prepend [str, optional] label to prepend output dict keys with

Returns

metrics [dict] all computed regression metrics for each unique target

deepSIP.utils.**classify**(*p*, *threshold=0.5*)
given probabilities, perform binary classification using threshold

Parameters

p [np.array] probabilities for binary classification

threshold [float, optional] decision threshold for binary classification

Returns

np.array binary classifications

deepSIP.utils.**classification_metrics**(*y*, *p*, *threshold=0.5*, *key_prepend=''*)

compute all classification related metrics

Parameters

y [np.array] true labels

p [np.array] probabilities for binary classification

threshold [float, optional] decision threshold for binary classification

key_prepend [str, optional] label to prepend output dict keys with

Returns

metrics [dict] all computed classification metrics

CHAPTER 4

Contributing

We welcome community involvement in the form of bug fixes, architecture improvements and new trained models, additional high-quality data (spectra and photometry), and expanded functionality. To those wishing to participate, we ask that you fork the deepSIP repository and issue a pull request with your changes (along with unit tests and a description of your contribution).

deepSIP is developed and maintained by Benjamin Stahl under the supervision of Prof. Alex Filippenko at UC Berkeley.

Python Module Index

d

Index

S

T

V

W