DeepClean Documentation

Release 0.1.0

Rich Ormiston, Michael Coughlin, Rana Adhikari, Gabriele Vajent

Jun 14, 2018

CONTENTS:

This repository takes a deep learning approach to solving nonlinear regression problems. Specifically, it aims to filter the nonlinear noise which couples into DARM.

This code is built on top of Tensorflow using Keras. For questions, bug reporting or requests about functionalities to include, email Rich Ormiston at rich.ormiston@ligo.org

CHAPTER

ONE

TUTORIALS

The sections below give a quick overview of the installation and how to use some of the tools within the DeepClean package

1.1 Installation

Installing the DeepClean repository is a simple task. First, clone the repository.

```
$ git clone git@git.ligo.org:rich.ormiston/DeepClean.git
```

Next, cd to the base directory and run the install script.

```
$ cd DeepClean
$ chmod +x install.sh
$ ./install.sh
```

And that's it! The virtual environment will be created for you and all of the dependencies will be installed within it. NOTE: Before the scipts may be run, you need to source the new environment

```
$ source $HOME/deepclean_ve/bin/activate
```

Additionally, you must edit the config file, namely, change `basedir` to the root directory of DeepClean (the rest of the defaults should work well).

Warning: If you have libraries installed in your `$HOME/.local` directory, those repositories will be not be installed. They will also not be found once you source the virtual environment, In that case, `cd` to the base directory of the repository and do `pip install -r requirements.txt && pip install -e .`

1.2 Retrieving Data with `getRegressionData.py`

Before the network can be trained, there must be data to run on. To acquire this data, use the `getRegressionData.py` script.

This script will connect to nds2 and therefore requires you to first deactivate the virtual environment and then to run

```
$ kinit albert.einstein
```

There are many command line options associated that may be seen by using the `-h` flag

```
$ python getRegressionData.py -h
usage: getRegressionData.py [-h] [--duration DURATION] [--fname FNAME]
                            [--fsup FSUP] [--ifo IFO] [--output OUTPUT]
                            [--portNumber PORTNUMBER]
                            [--time TIMES [TIMES ...]]
optional arguments:
  -h, --help            show this help message and exit
  --duration DURATION, -d DURATION
                        data segment duration
  --fname FNAME, -f FNAME
                        channel list file name
  --fsup FSUP, -fs FSUP
                        sample frequency
  --ifo IFO, -i IFO     interferometer: L1 or H1
  --output OUTPUT, -o OUTPUT
                        output file name
  --portNumber PORTNUMBER, -p PORTNUMBER
                        port to connect to
  --time TIMES [TIMES ...], -t TIMES [TIMES ...]
                        start time. Ex. 2017-01-04 11:40:00
```

The output is a mat file. Suppose you want to collect data from Hanford during the O2 run. Specifically, you'd like 2048 seconds of data starting from Augist 14, 2017 02:00:00 using the channel list `ChanList_H1.txt` and want the outfile named `H1_data_August.mat`. Then you would run

```
$ python getRegressionData.py -i H1 -t 2017-08-14 02:00:00 -o H1_data_August.mat -f␣
↪ChanList_H1.txt
```

The data will be saved to `deepclean/Data.`

1.3 Running the Network

Now that data has been collected, the network can be trained and evaluated. Make sure that you are sourced and have edited the config file in the way you wish. If you do not wish to run a particular "loop", then set that loop to `False` under the section `To_Run`.

```
$ dc-run-network -i path/to/configs.ini
```

Or, assuming that the config file being used is `configs/configs.ini` (i.e., with the default location/name), we may equivalently do

```
$ dc-run-network
```

The output plots are stored in `deepclean/Plots.` An example of output generated after running `dc-run-network` (with no arguments) on the data collected in the above example is shown below

1.4 Generating Webpages

After plots have been created, webpages can be generated to help to visualize the results and to collect all of the parameters used in that network evaluation. These html pages are saved in `html/day/` and are not written over. Thus this provides a simple way of tracking model progress.

Building the webpages is simple.

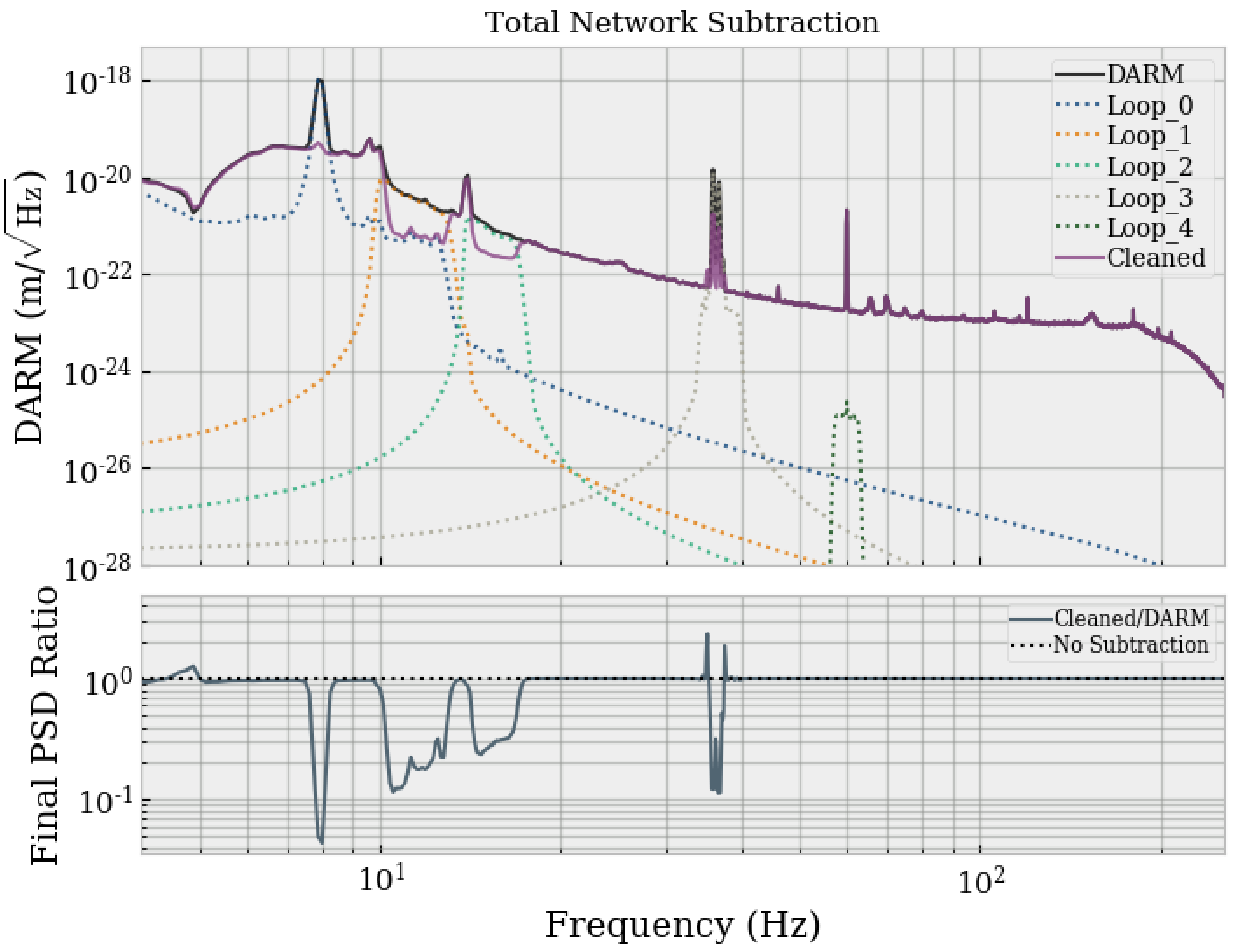

Fig. 1: PSD of DARM, the predictions for each network iteration and the total subtraction progress. There is improvement below ~38Hz.

```
$ dc-webpage -i path/to/configs.ini
```

Again, the flag may be left off if the config file being used is `configs/configs.ini`. **A sample webpage output can be viewed** here

1.5 Visualizing Data

In machine learning, it is often helpful to plot the data to see if correlations exist between input data streams. The command-line function `spearman` is for just that purpose. This script reads in the supplied data and calculates the product of every permutation of the dataset channels and outputs those results to a csv if the Spearman "rho" coefficient or the Pearson's coefficient are above the set threshold.

The data can also be plotted against itself to look for correlations. The flags available are found by using the help flag

```
usage: dc-spearman [-h] [--data_type DATA_TYPE] [--ifo IFO]
                   [--outputDir OUTPUTDIR] [--output OUTPUT] [--rho RHO]
                   [--pearson PEAR] [--threshold THRESHOLD]
optional arguments:
  -h, --help            show this help message and exit
  --data_type DATA_TYPE, -d DATA_TYPE
                        real or mock data set
  --ifo IFO, -ifo IFO   L1 or H1
  --outputDir OUTPUTDIR, -dir OUTPUTDIR
                        directory in which to store results
  --output OUTPUT, -o OUTPUT
                        output file name
  --rho RHO, -r RHO     spearman's rho threshold value
  --pearson PEAR, -p PEAR
                        pearson's coefficient threshold value
  --threshold THRESHOLD, -t THRESHOLD
                        chose from: pearson, rho, and, or
```

A sample output plot is below

1.6 Config File Parameters

There are many network configurations and hyperparameters available for wasy modification through the configuration file located in `configs/configs.ini`. The available parameters are in the config file and must remain there, so **do not delete them**. The parameters available are:

```
[Data]
datafile  = ../deepclean/Data/H1_data_array.mat
data_type = real

[Webpage]
basedir = /home/richard.feynman/git_repositories/DeepClean/

[To_Run]
Loop_0 = True
Loop_1 = True
Loop_2 = True
Loop_3 = True
Loop_4 = True
```

(continues on next page)

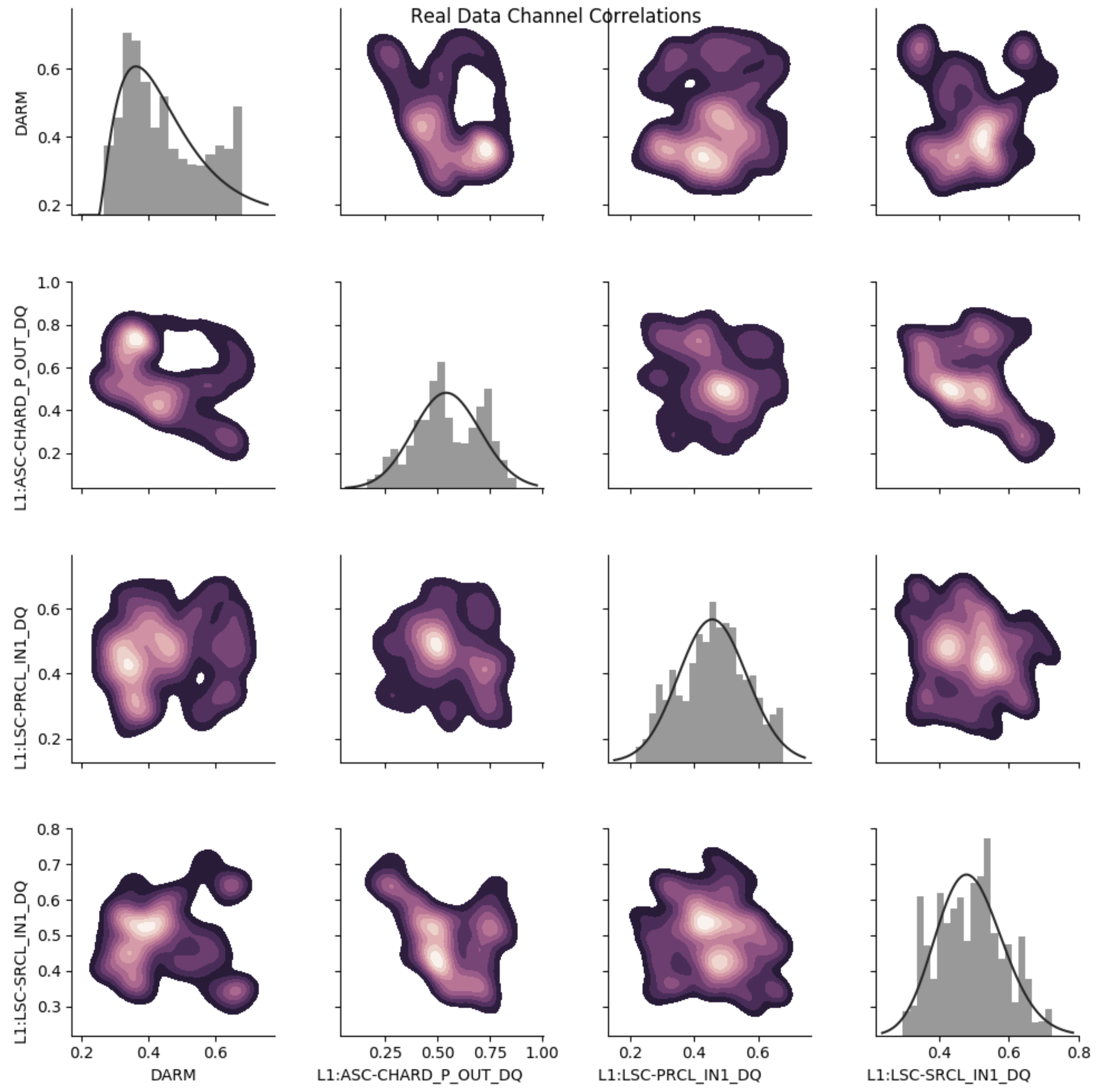

Fig. 2: 3D channel correlations of the supplied data channels

(continued from previous page)

```
[Loop_0]
beta_1      = 0.9
beta_2      = 0.999
decay       = None
epochs      = 3
epsilon     = 1e-8
fmin        = 4
fmax        = 256
hc_offset   = 0
highcut     = 12.0
loss        = mse
lowcut      = 3.0
lr          = None
momentum    = 0.0
nesterov    = False
N_bp        = 8
optimizer   = adam
plotDir     = ../deepclean/Plots
postFilter  = True
preFilter   = True
rho         = None
subsystems  = all
tfrac       = 0.5
ts          = 2
```

The parameters listed under `Loop_0` are also available to every subsequent "Loop" section. Currently, the code can handle up to 6 independent network iterations.

CHAPTER

TWO

DEEPCLEAN MODULES AND TOOLS

`deepclean.preprocessing`, `deepclean.analysis` and `deepclean.models` contain the functions useful for data preparation, analysis and visualization, and model construction respectively. There is also a separate exceptions module `deepclean.exceptions`.

2.1 `analysis` Module

`deepclean.analysis.`**`plot_channel_correlations`**(*datafile*, *plotNum=4*, *data_type=None*, *seconds=15*, *plotDir='.'*)

plot_channel_correlations calculates comparisons between channels in order to show which channels may contain 'features' of DARM.

Parameters

datafile [*str*] mat file to analyze

plotNum [*int*] Number of plots per image

data_type [*str*] use either 'real' or 'mock' data

seconds [*int*] How many seconds of data to query. NOTE: using times longer than ~30 seconds start to take a really long time to compute. If possible, use <= 30 seconds unless you're particularly patient :)

plotDir [*str*] path to store plots

`deepclean.analysis.`**`plot_progress`**(*darm*, *predictions*, *fs=512*, *nfft=4096*, *loops=['Loop_0', 'Loop_1', 'Loop_2']*, *plotDir='Plots'*, *title='Total Network Subtraction'*, *saveas='total_subtraction'*, *savepdf=False*, *fmin=4*, *fmax=256*)

plot the psd of the DARM (testing) timeseries, the prediction timeseries calculated by the network, and the residual.

Parameters

target [*numpy.ndarray*] validation DARM timeseries

prediction: 'numpy.ndarray' validation prediction timeseries

fs [*int*] data sample rate

nfft [*int*] overlapping windows

loops [*list*] list of loop iterations trained and tested on

plotDir [*str*] path to store plots

title [*str*] plot title

saveas [*str*] output filename for plot

savepdf [*bool*] when set to True, both png and pdf filetypes will be saved

fmin [*float*] x-axis min

fmax [*float*] x-axis max

`deepclean.analysis.`**`plot_psd`**`(`*target*, *prediction*, *fs=512*, *nfft=4096*, *plotDir='Plots'*, *title='Neural Network Validation PSD'*, *saveas='validation_psd'*, *savepdf=False*, *fmin=4*, *fmax=256*`)`

plot the psd of the DARM (testing) timeseries, the prediction timeseries calculated by the network, and the residual

Parameters

target [*numpy.ndarray*] validation DARM timeseries

prediction: 'numpy.ndarray' validation prediction timeseries

fs [*int*] data sample rate

nfft [*int*] overlapping windows

plotDir [*str*] path to store plots

title [*str*] plot title

saveas [*str*] output filename for plot

savepdf [*bool*] when set to True, both png and pdf filetypes will be saved

fmin [*float*] x-axis min

fmax [*float*] x-axis max

`deepclean.analysis.`**`set_plot_style`**`()`

provide matplotlib plotting style

`deepclean.analysis.`**`split_dict`**`(`*d*, *elements*`)`

split_dict is used in plot_density in order to take a dict *d* of length *n* and split it into a list of dicts of length *elements*

Parameters

d [*dict*] input dictionary

elements [*int*] number of elements in each sub-dict

Returns

output [*list*] list containing dicts of length *elements*

2.2 `preprocessing` Module

`deepclean.preprocessing.`**`get_dataset`**`(`*datafile*, *subsystems='all'*, *data_type='real'*, *chanlist='all'*`)`

get_dataset reads in a datafile and returns the dataset used during training. Optionally, particular subsystems may be given as witness channels

Parameters

datafile [*string*] full path to mat file

subsystems [*list*] subsystems to include in dataset e.g. subsystems = ['ASC', 'CAL', 'HPI', 'SUS']

data_type [*str*] use either "real", "mock" or "scatter"

Returns

dataset [*numpy.ndarray*] test data. includes all channels except darm

fs [*int*] sample rate of data

`deepclean.preprocessing.`**`get_run_params`**(*ini_file*, *section*)

function for reading parameters from the config file

Parameters

ini_file [*str*] path to config file

section [*str*] config file section to read from

Returns

run_params [*dict*] dict of params from supplied config file and section

`deepclean.preprocessing.`**`lstm_lookback`**(*data*, *n_in=1*, *n_out=1*)

create lookback in the dataset

Parameters

data [*numpy.ndarray*] dataset for training and testing

n_in [*int*] number of timesteps to lookback

n_out [*int*] number of timesteps to forecast

Returns

combined [*numpy.ndarray*] dataset with lookback

`deepclean.preprocessing.`**`phase_filter`**(*dataset*, *lowcut=4.0*, *highcut=20.0*, *order=8*, *btype='bandpass'*, *fs=512*)

phase preserving bandpass filter

Parameters

btype [*str*] filter type

dataset [*numpy.ndarray*] dataset for training and testing

fs [*int*] data sample rate

highcut [*float*] stop frequency for filter

lowcut [*float*] start frequency for filter

order [*int*] bandpass filter order

Returns

dataset [*numpy.ndarray*] bandpassed dataset for training and testing

2.3 `models` Module

`deepclean.models.`**`get_optimizer`**(*opt*, *decay=None*, *lr=None*, *momentum=0.0*, *nesterov=False*, *beta_1=0.9*, *beta_2=0.999*, *epsilon=1e-08*, *rho=None*)

get_optimizer is a wrapper for Keras optimizers.

Parameters

beta_1 [*float*] adam optimizer parameter in range [0, 1) for updating bias first moment estimate

beta_2 [*float*] adam optimizer parameter in range [0, 1) for updating bias second moment estimate

decay [*None* or *float*] learning rate decay

epsilon [*float*] parameter for numerical stability

opt [*str*] Keras optimizer. Options: "sgd", "adam", "nadam", "rmsprop", "adagrad", "adamax" and "adadelta"

lr [*None* or *float*] optimizer learning rate

momentum [*float*] accelerate the gradient descent in the direction that dampens oscillations

nesterov [*bool*] use Nesterov Momentum

rho [*None* or *float*] gradient history

Returns

optimizer [`keras.optimizer`] keras optimizer object

2.4 `exceptions` Module

exception `deepclean.exceptions.`**`DataNotFound`**(*datafile*)
FileNotFound is used for catching exceptions when the file that is attempting to be opened does not exist.

exception `deepclean.exceptions.`**`FileNotFound`**(*FILE*)
FileNotFound is used for catching exceptions when the file that is attempting to be opened does not exist.

exception `deepclean.exceptions.`**`TemplateNotFound`**(*template*)
TemplateNotFound is used for catching exceptions when jinja templates are not found

`deepclean.exceptions.`**`checkFileExists`**(*ini_file*)
checkFileExists throws an exception if the file does not exist. This prevents misleading errors when trying to read from secitons in config files that do not exist.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX